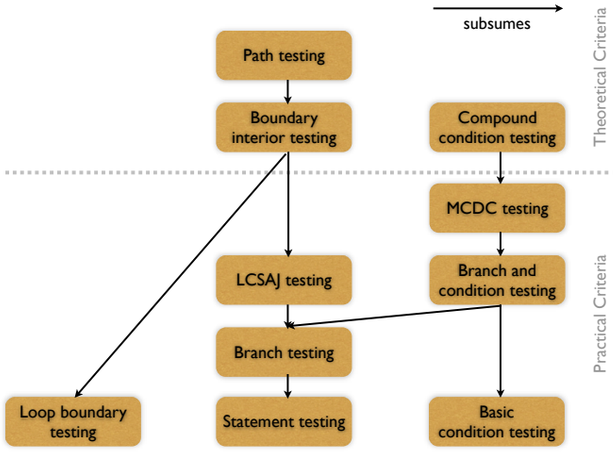
Search-based Testing

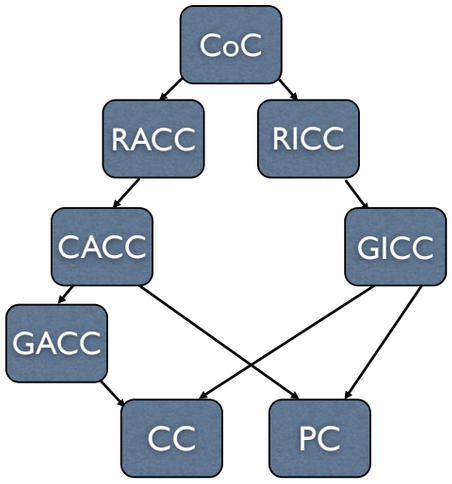
Software Engineering

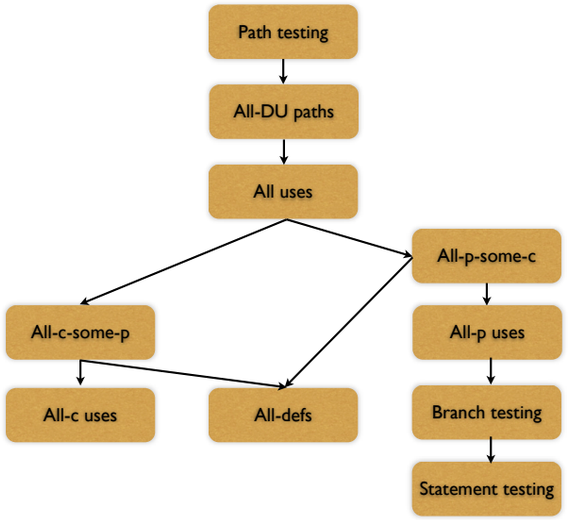
Gordon Fraser • Saarland University

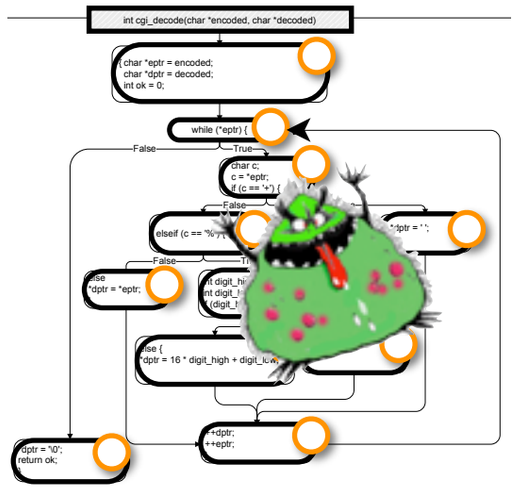
So many different test objectives...

So many different test objectives...

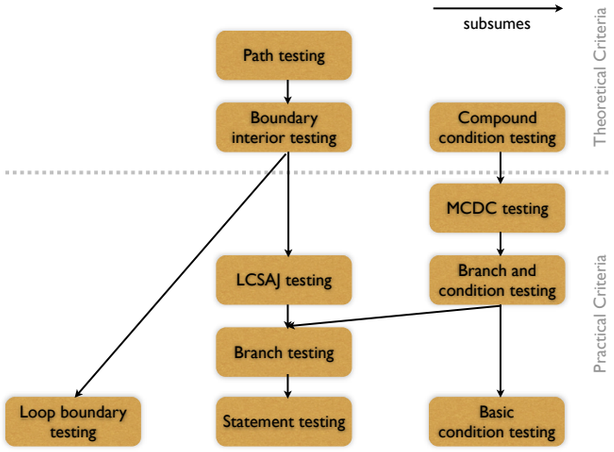


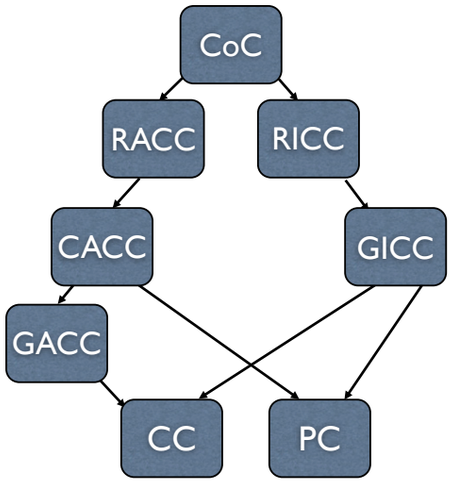


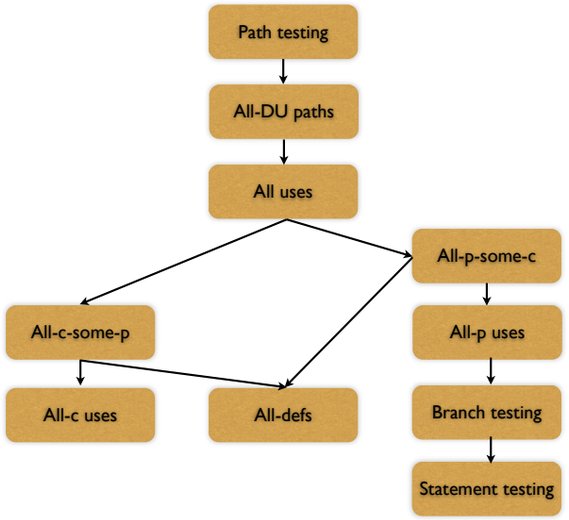


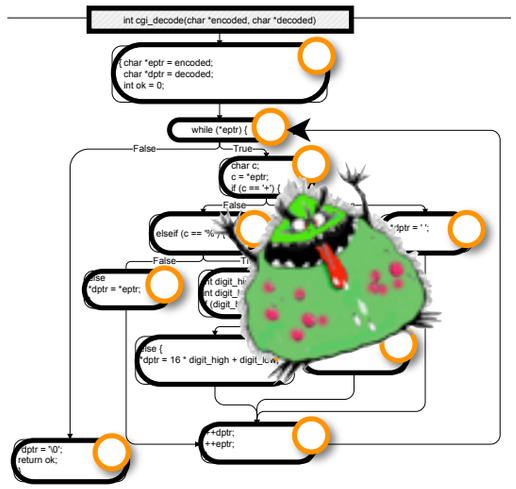


So many different test objectives...





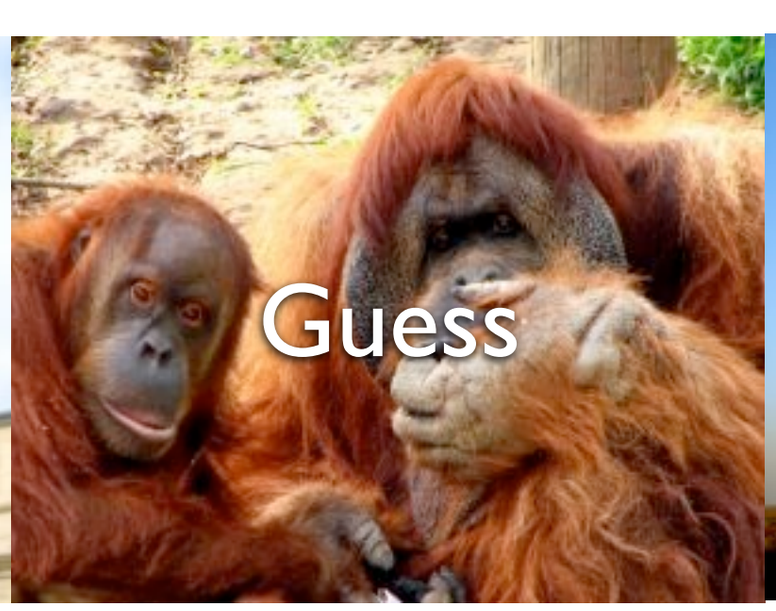


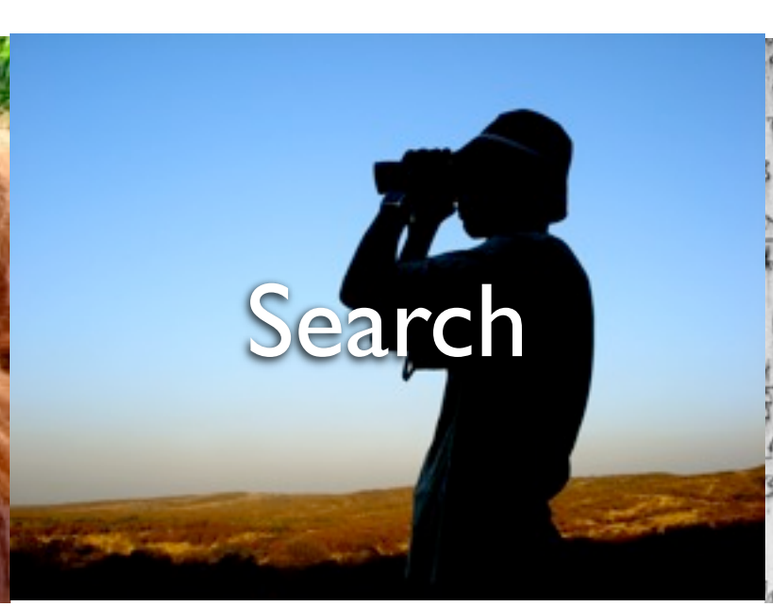


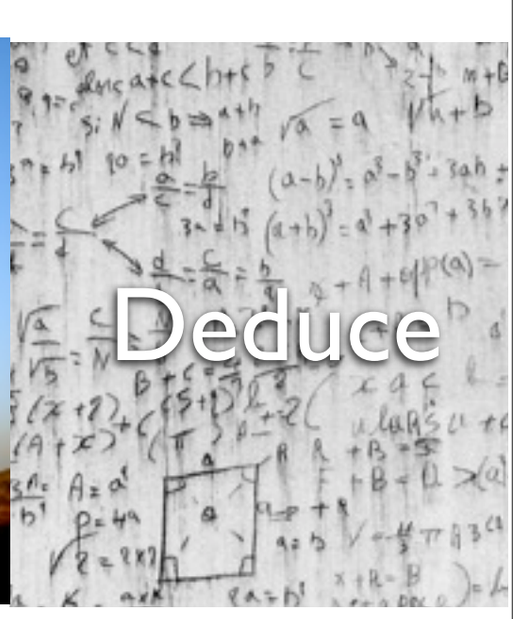
But how to create the tests?

Test Data Generation

Given a function and a location we want to reach, how do we derive inputs to the function that lead the control flow to the desired statement?







Search-based Software Engineering

- Cast problems of software engineering as optimization problems

- Search problems in software engineering are BIG
- Apply meta-heuristic search algorithms to solve these problems

Search Algorithms

Random Testing

- (Totally) uninformed search
- If a systematic approach is not better than random testing, it is not useful
- Cheap & easy to implement
- Works pretty well in many cases

Heuristics

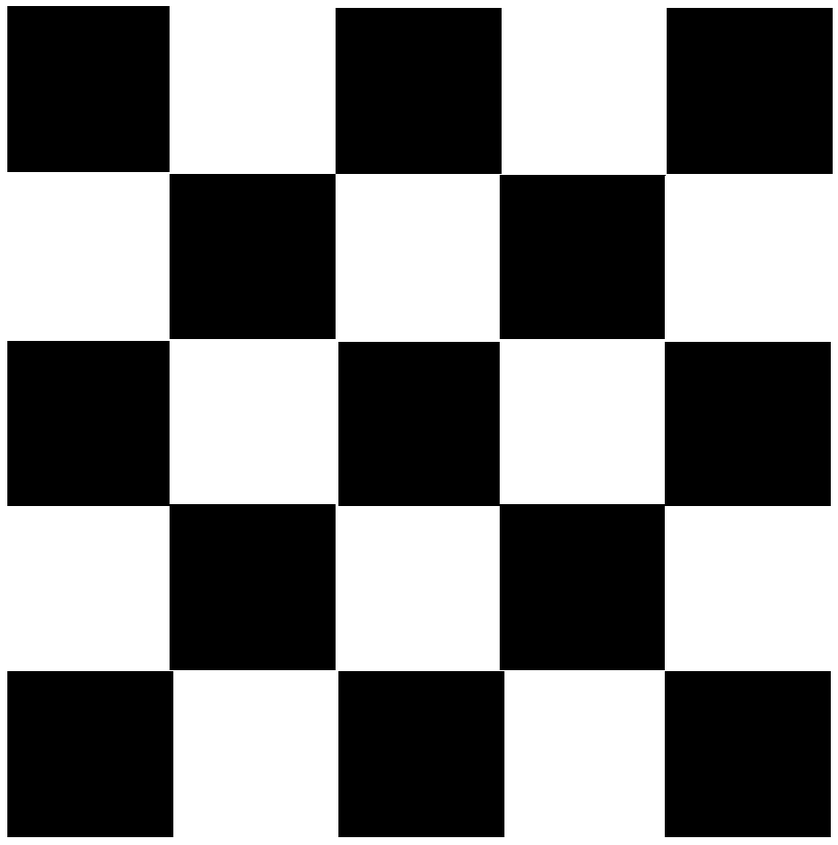
- Might not always find the best solution
- But finds a good solution in reasonable time
- Sacrifices completeness but increases efficiency
- Useful in solving tough problems
  Which could not be solved any other
  Which would take very long time to compute
- Use problem specific metrics

Meta-Heuristics

- Problem-independent algorithms to solve optimization problems
- Problem-specific implementation of certain parts (as heuristic)

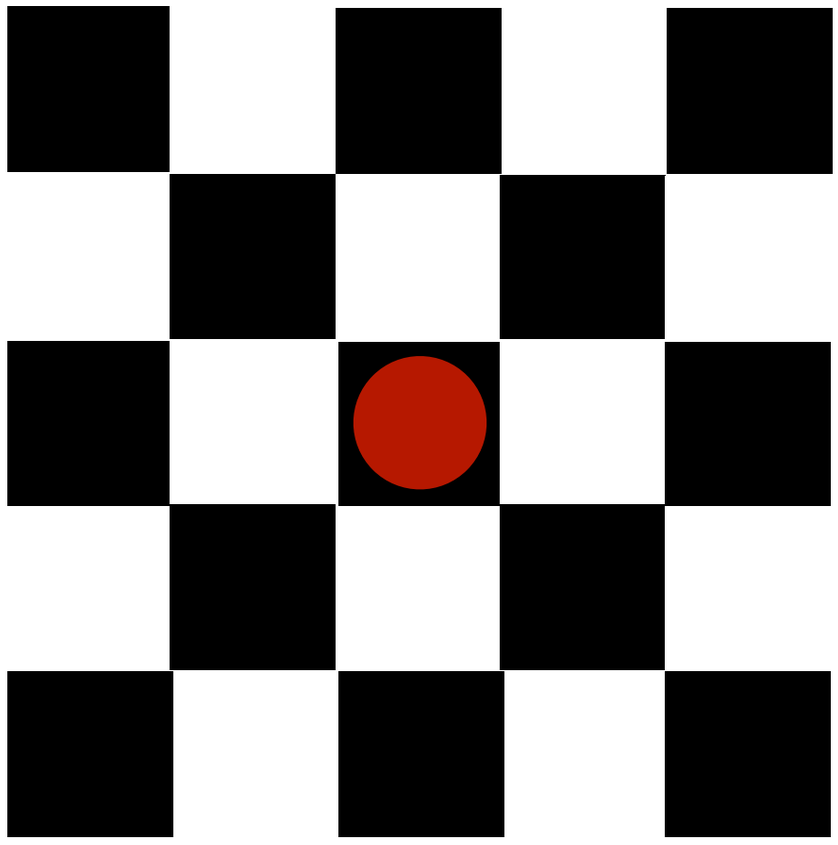
Search Space

Search space every
possible position (x,y)



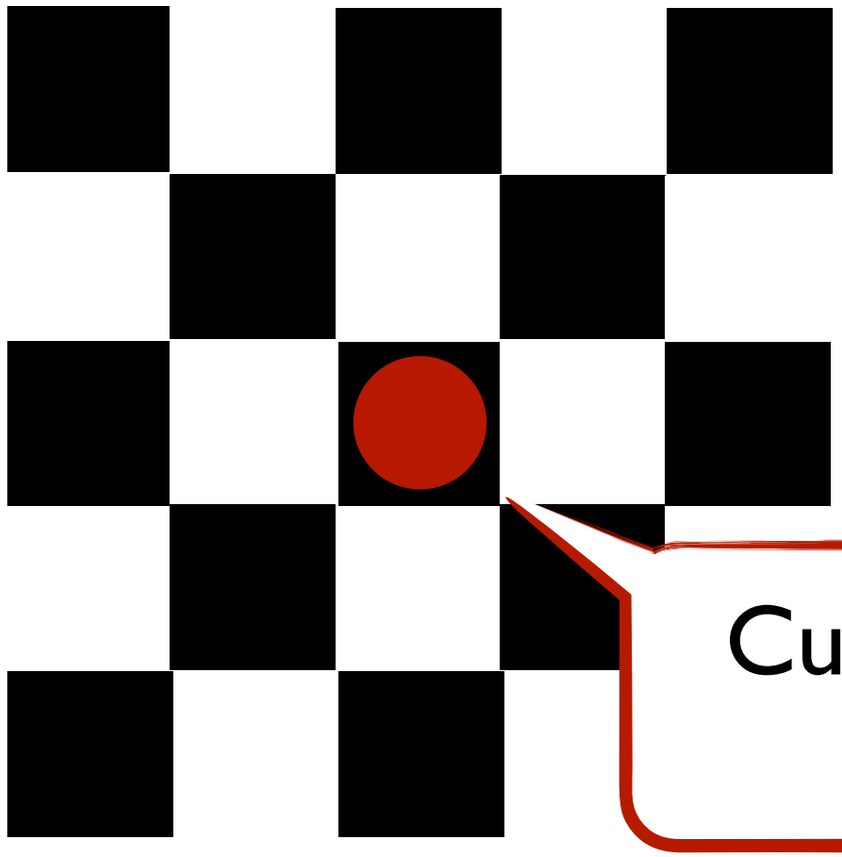
Search Space

Search space every
possible position (x,y)



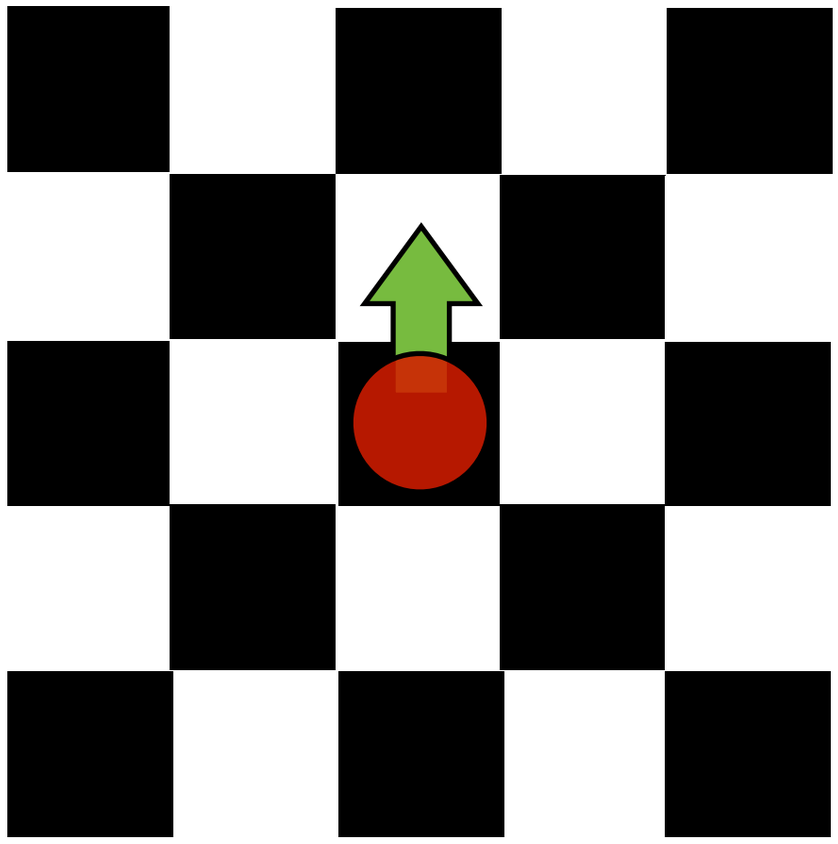
Search Space

Search space every
possible position (x,y)



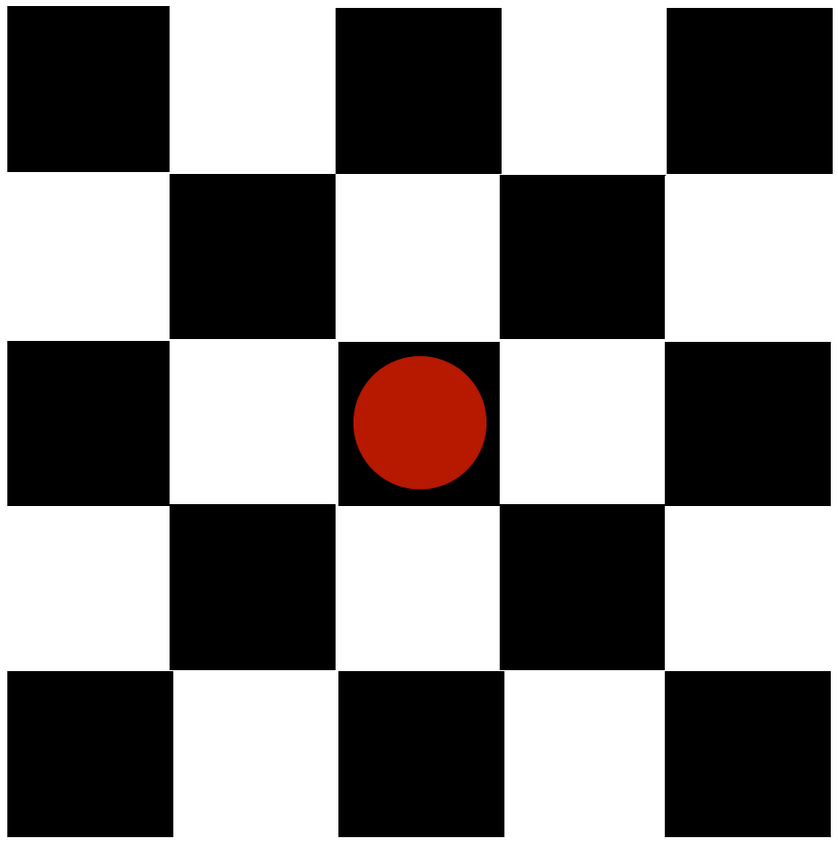
Search Space

Possible move:
(3,3), (3,4)



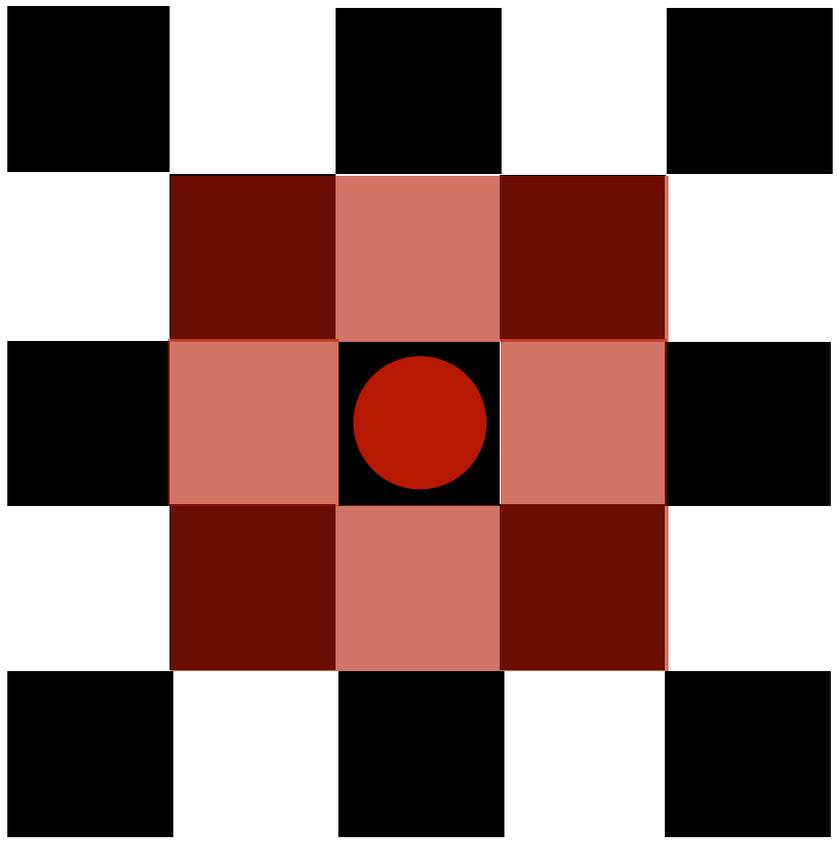
Neighborhood

Any possible move

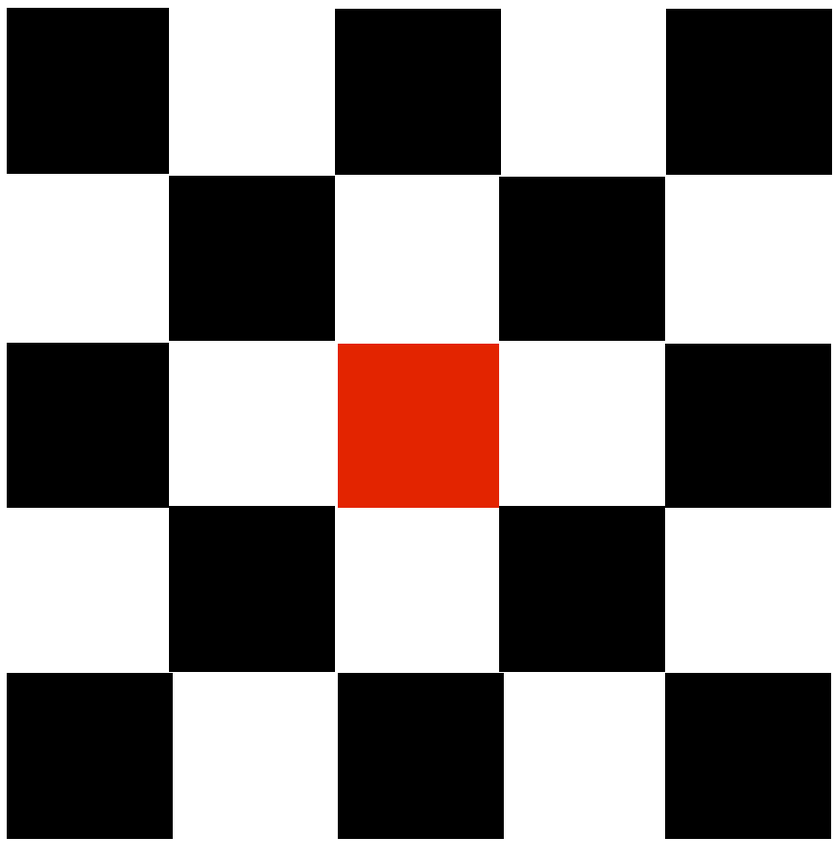


Neighborhood

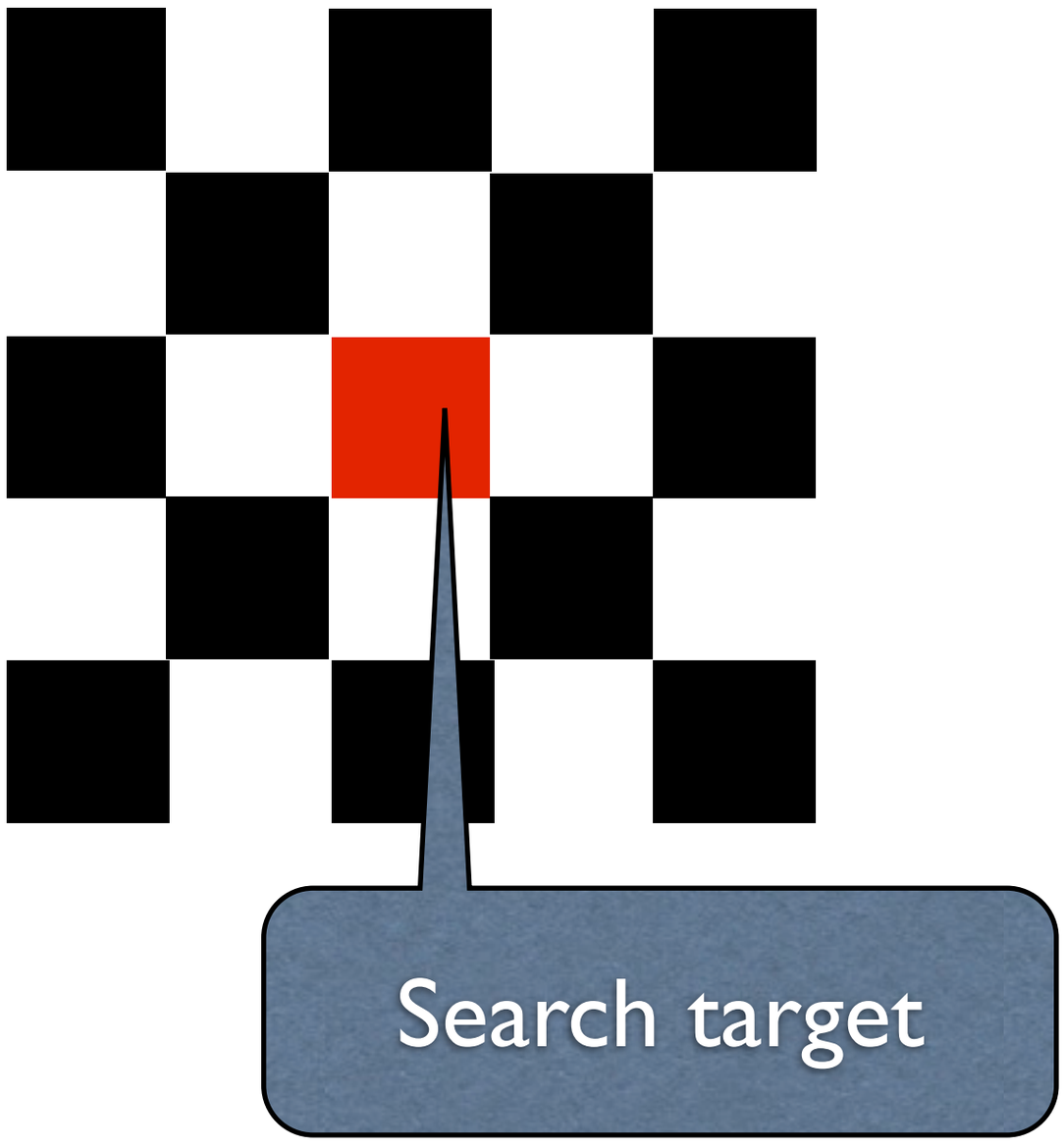
Any possible move



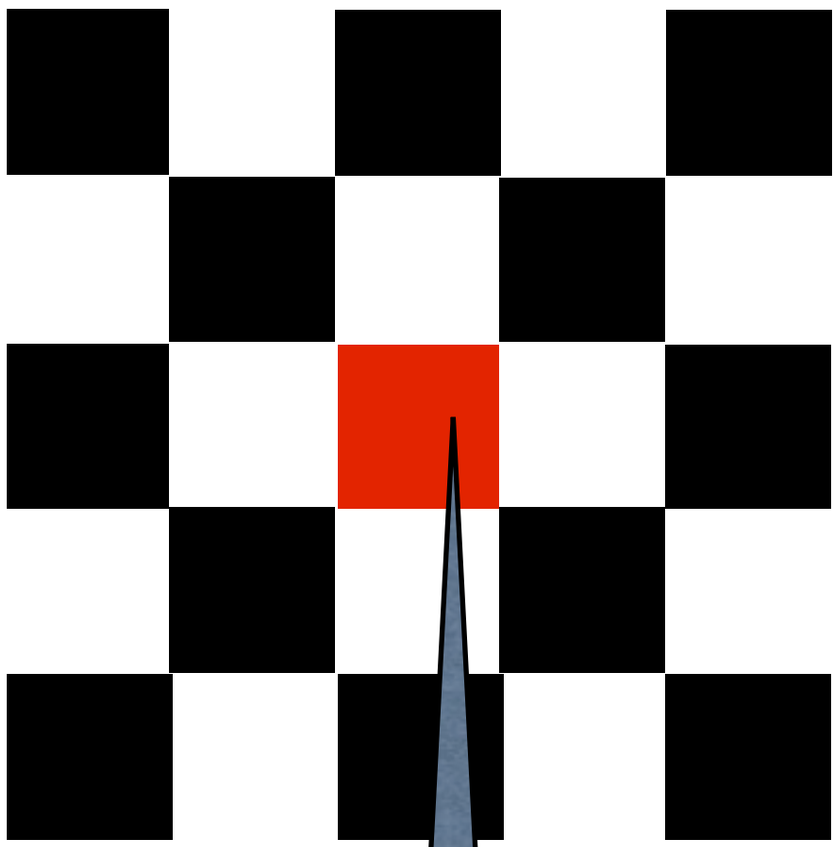
Search Space

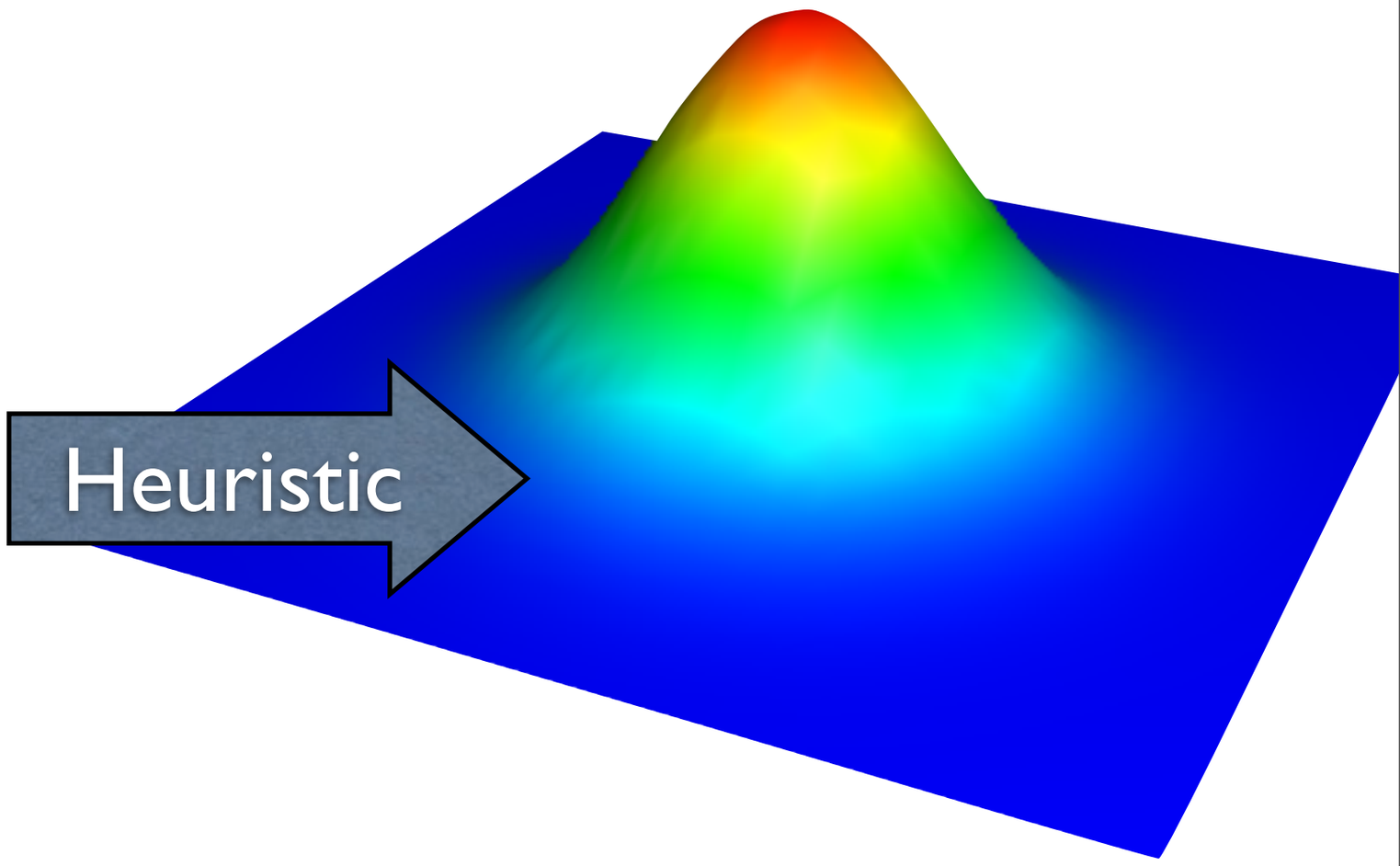


Search Space



Search Space

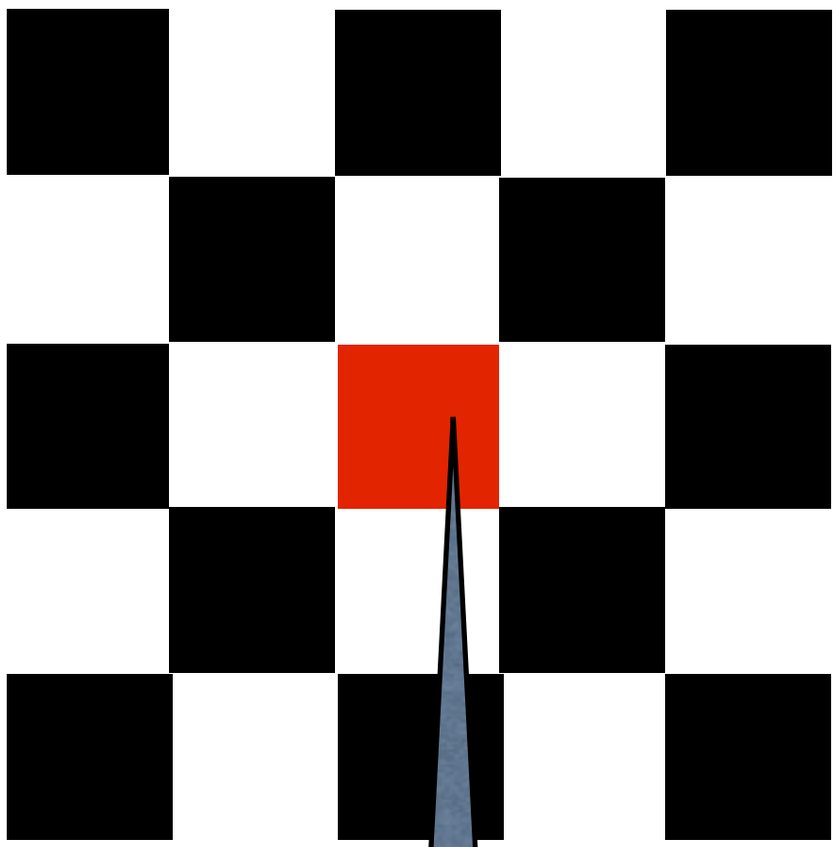


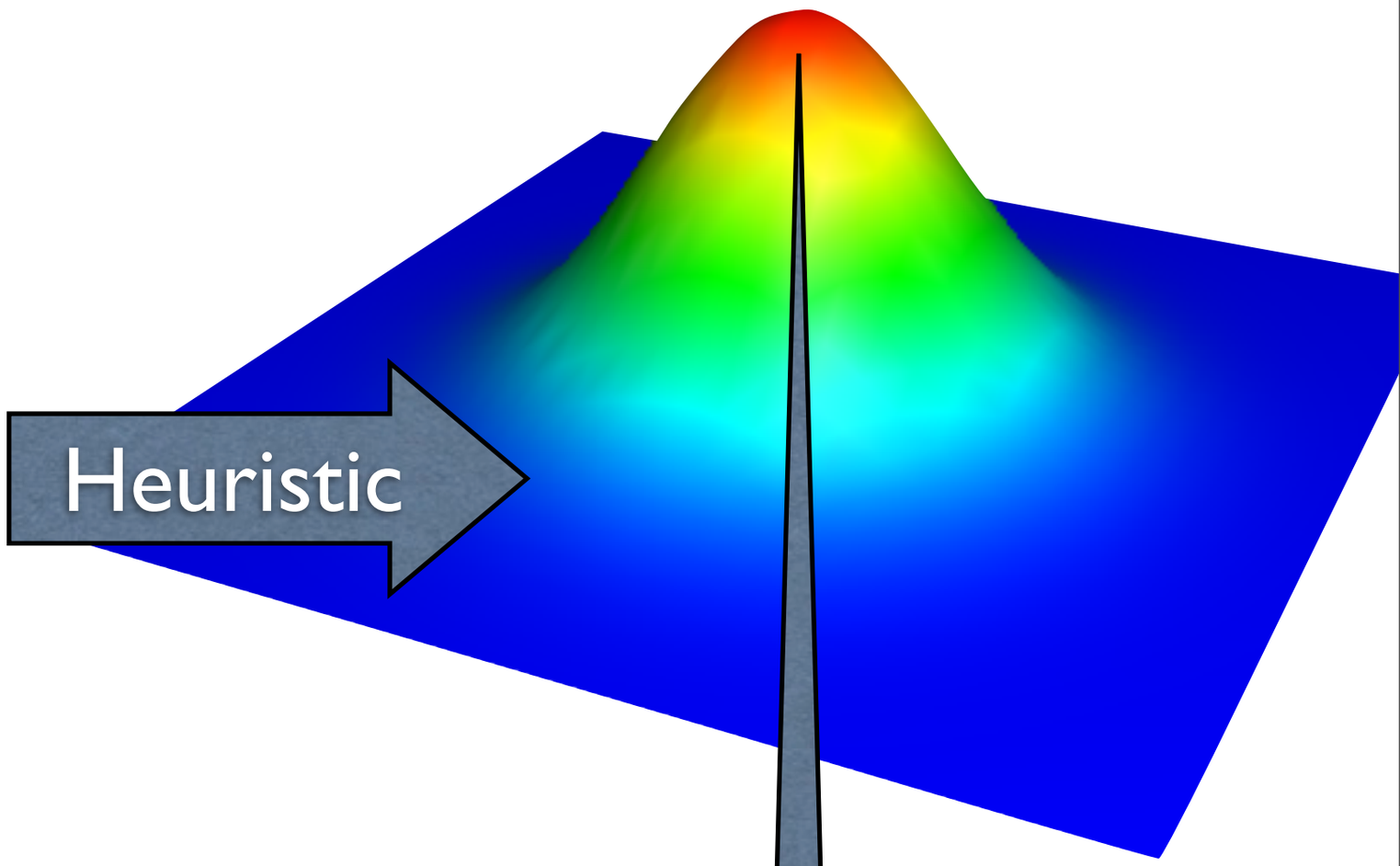


Heuristic

Search target

Search Space

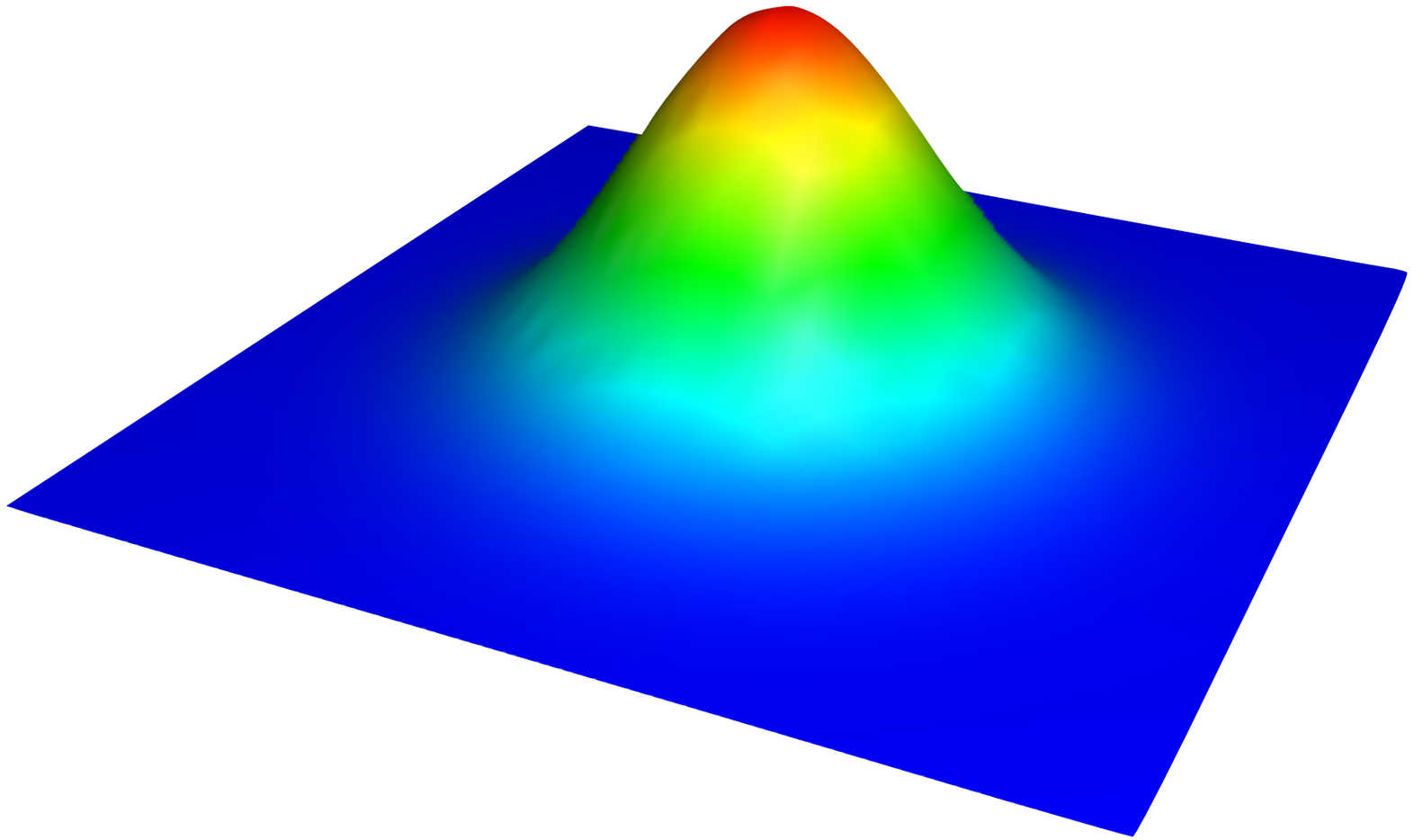




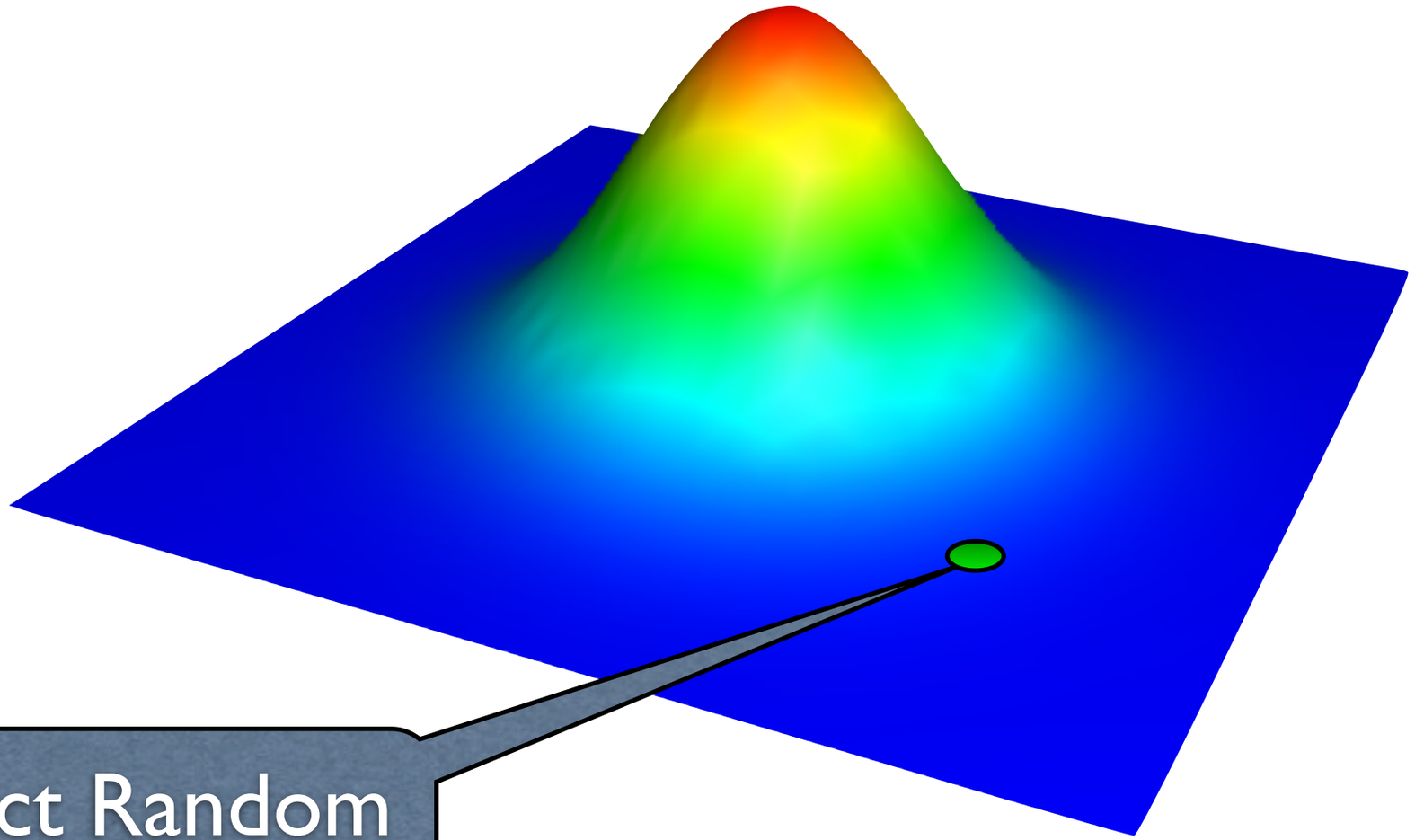
Search target

Optimum

Hill Climbing

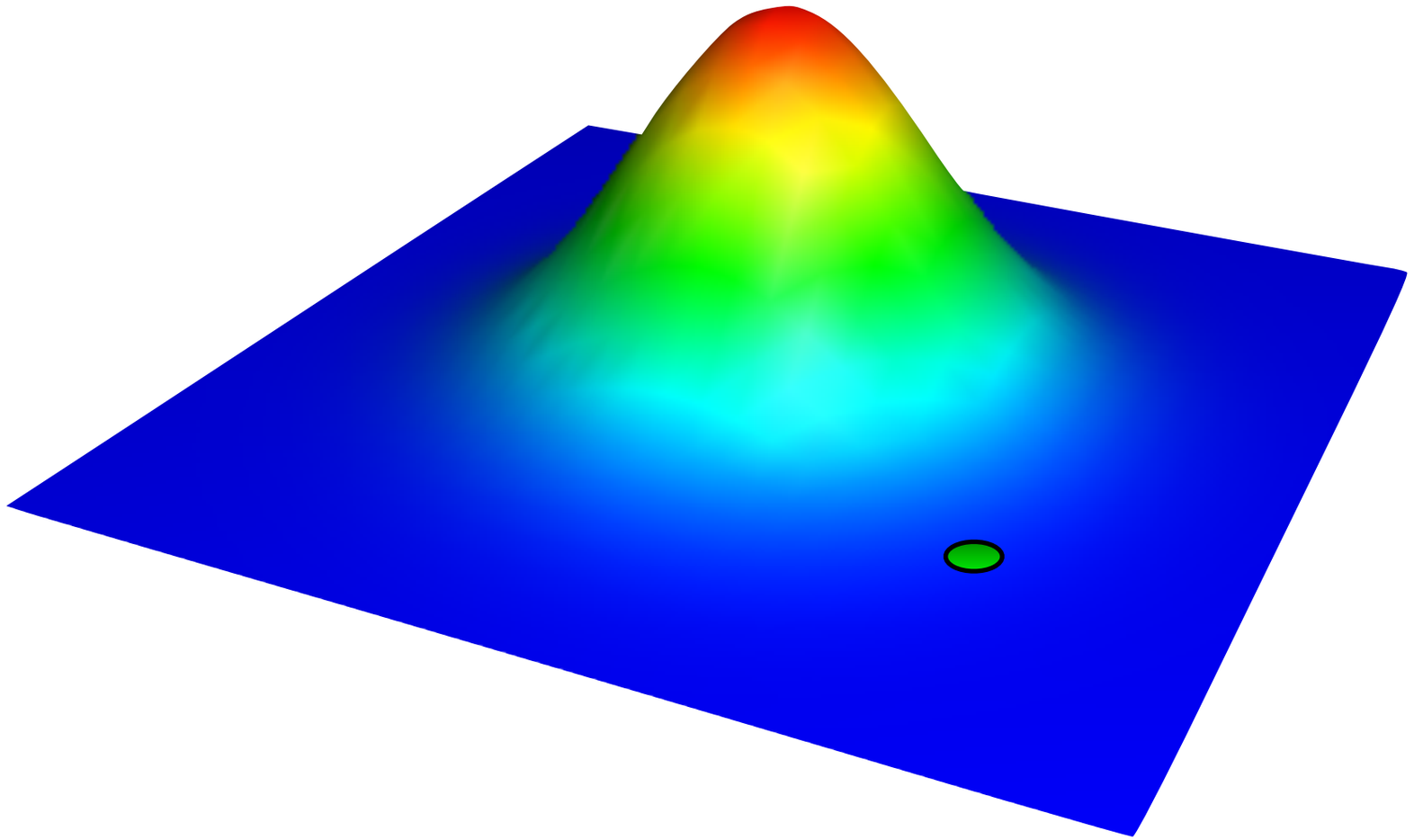


Hill Climbing

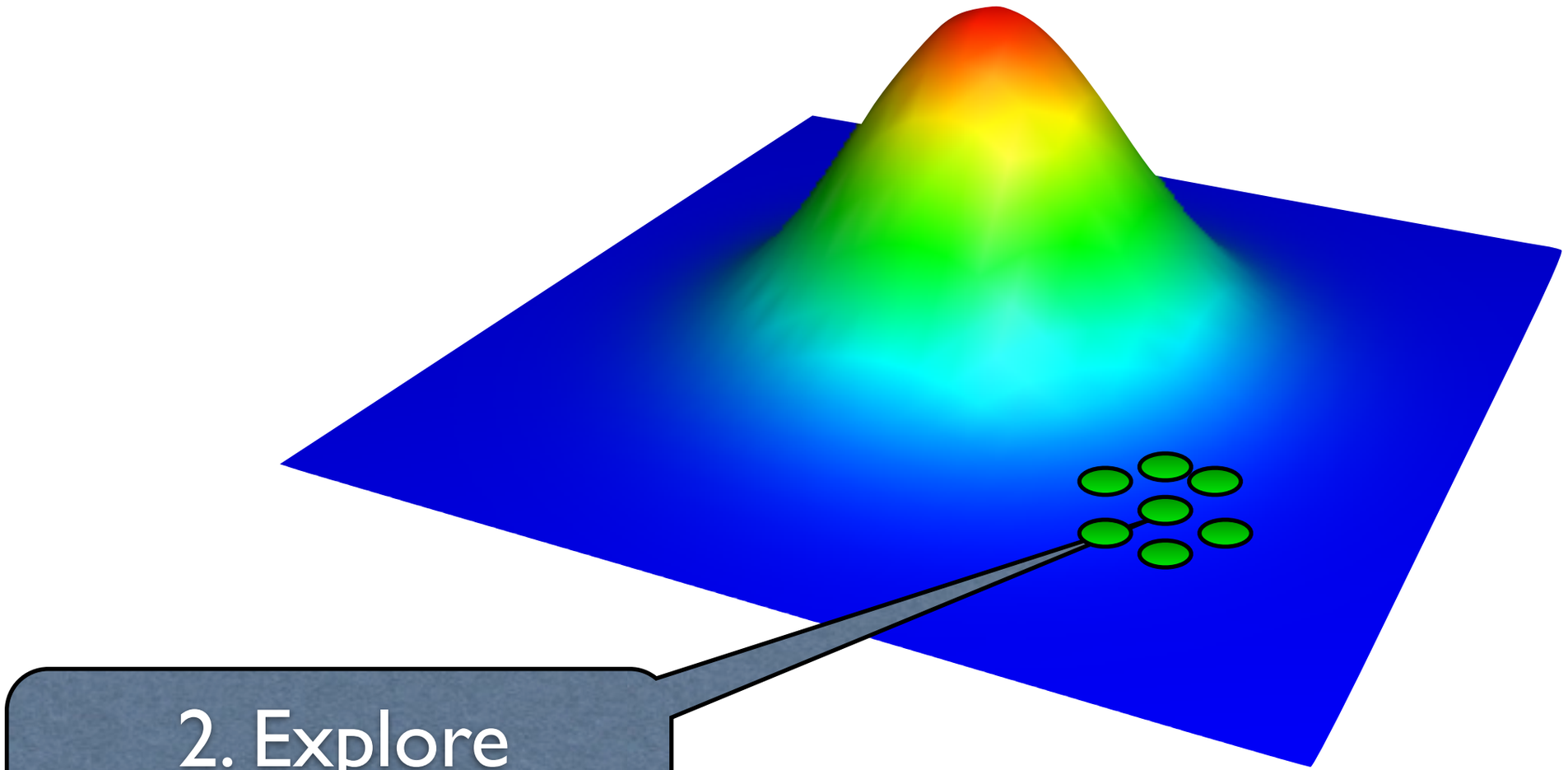


1. Select Random Value

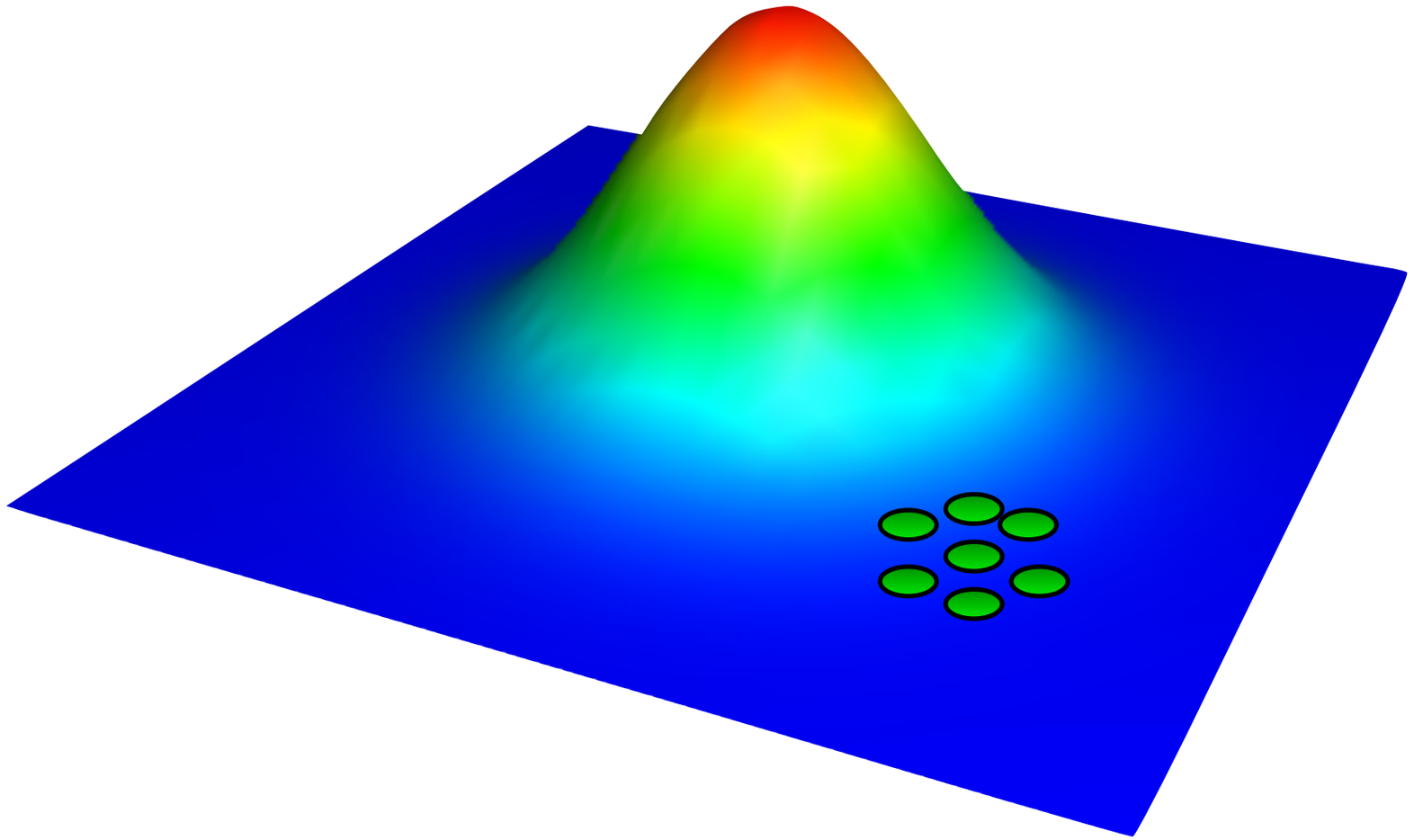
Hill Climbing



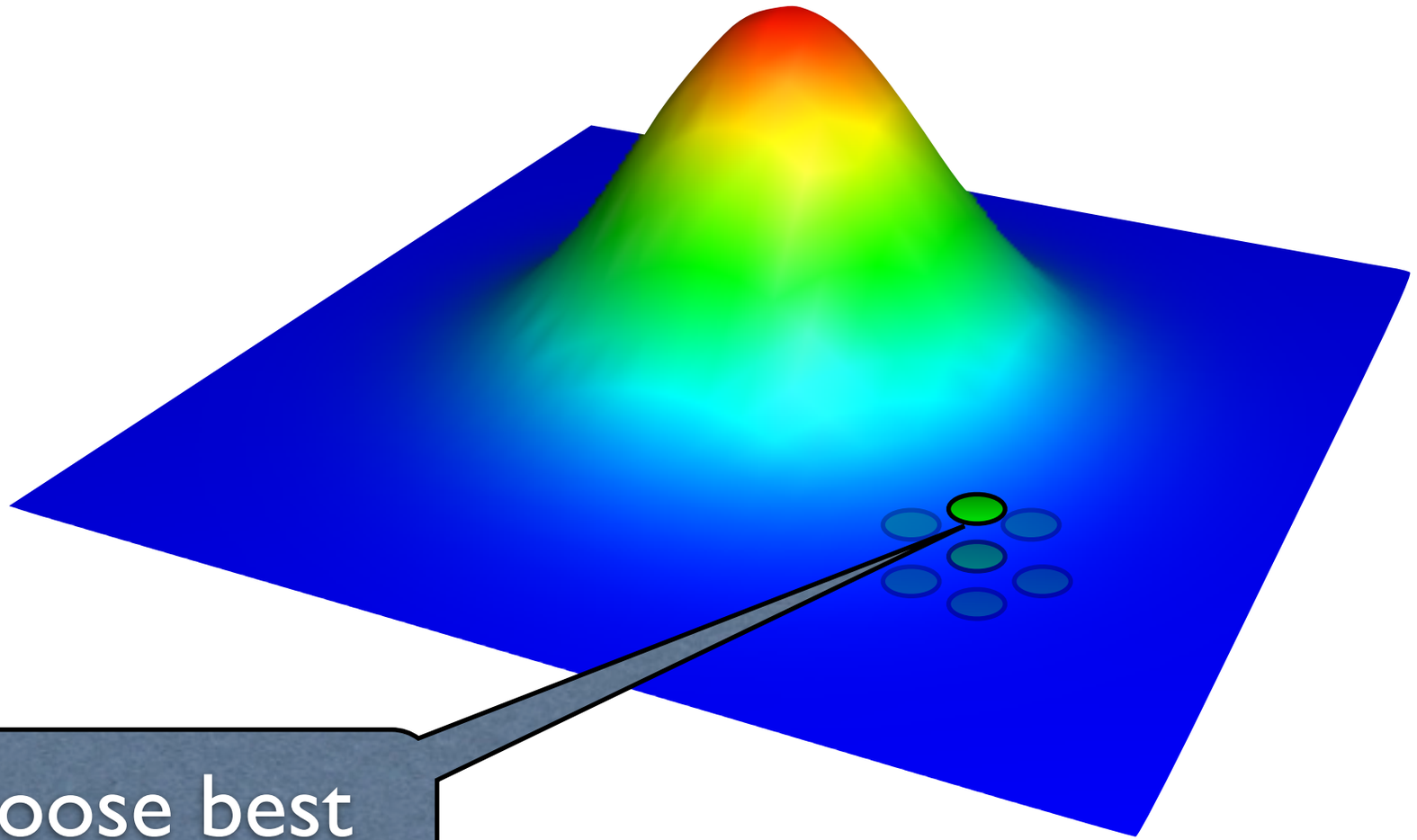
Hill Climbing



Hill Climbing

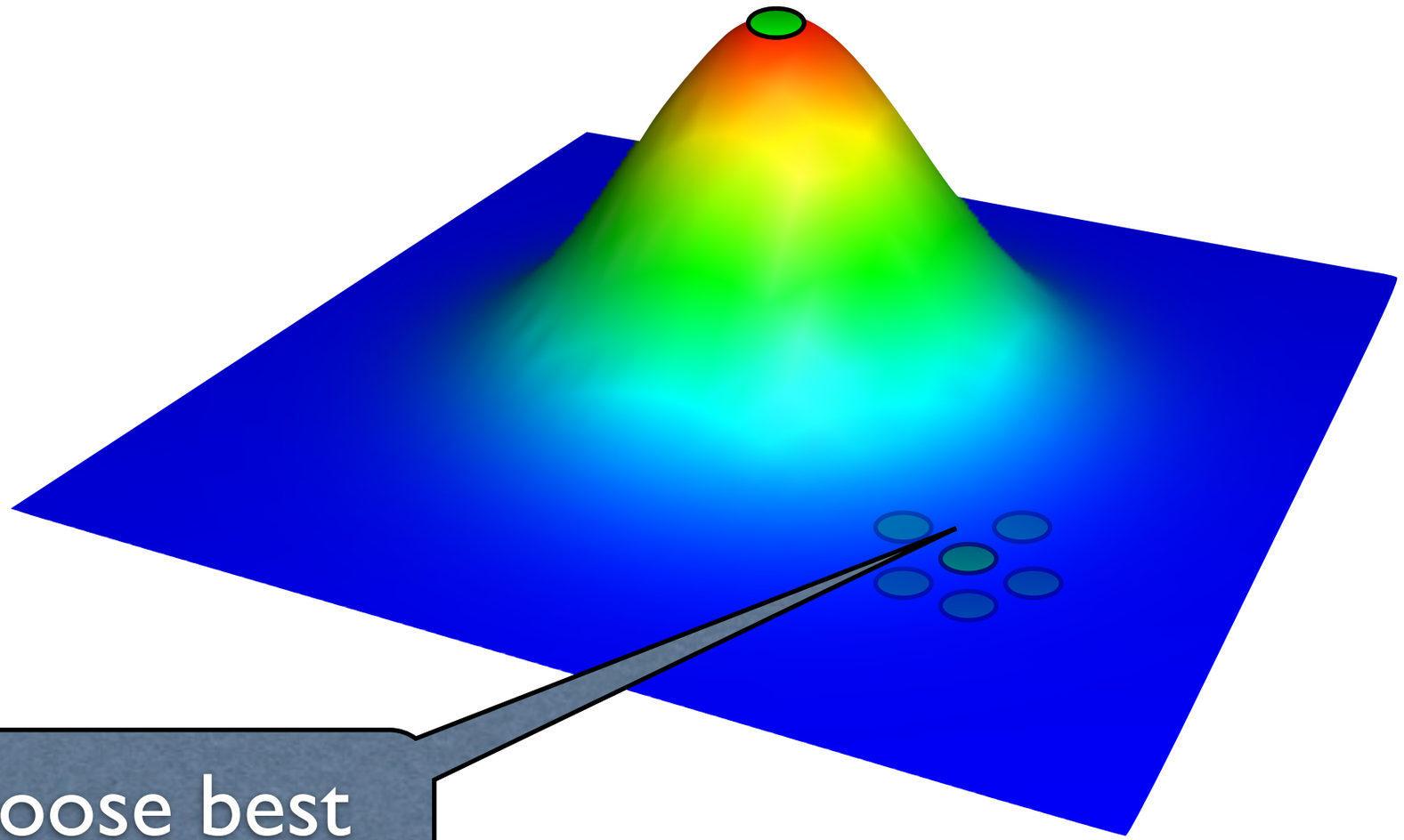


Hill Climbing



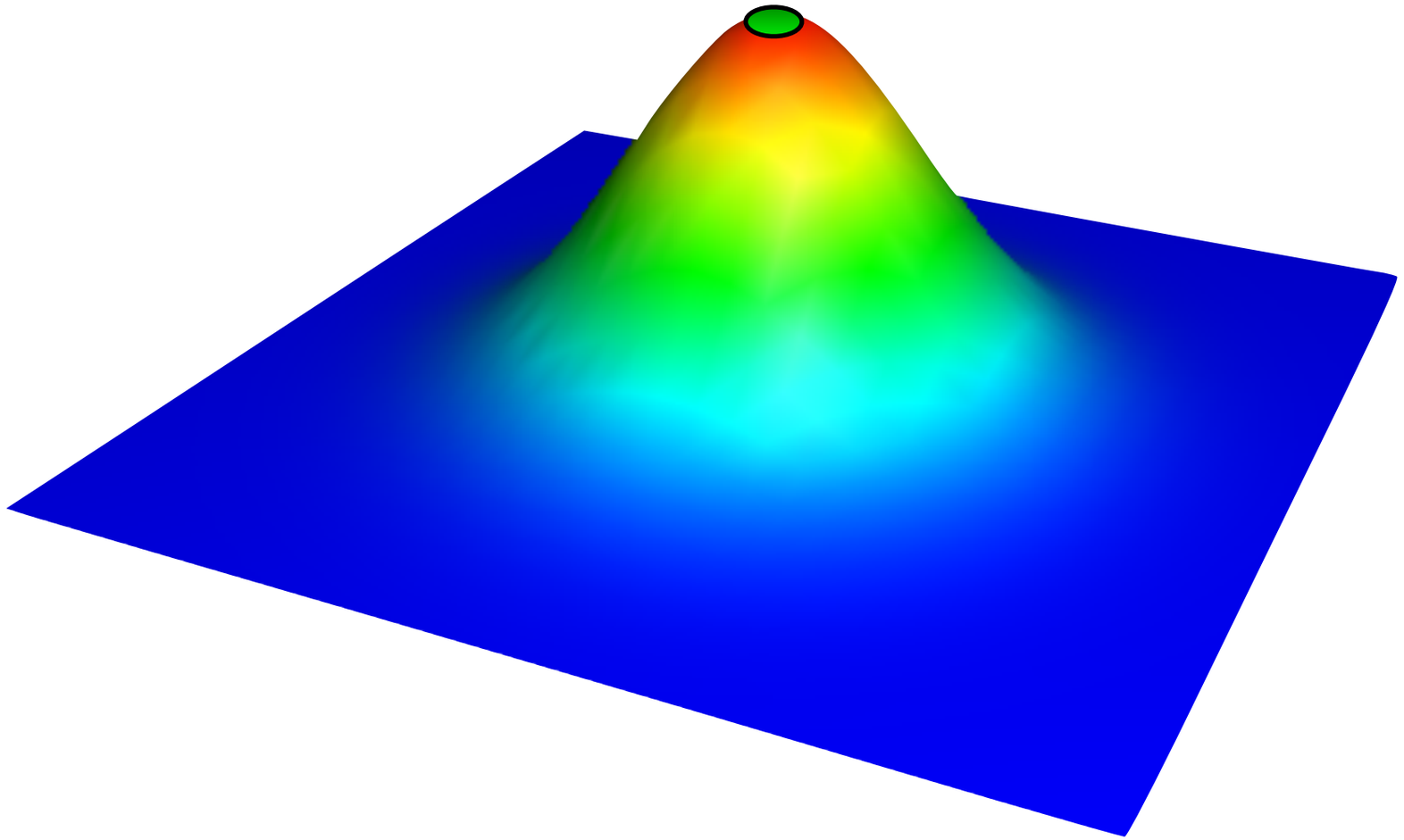
3. Choose best neighbor

Hill Climbing

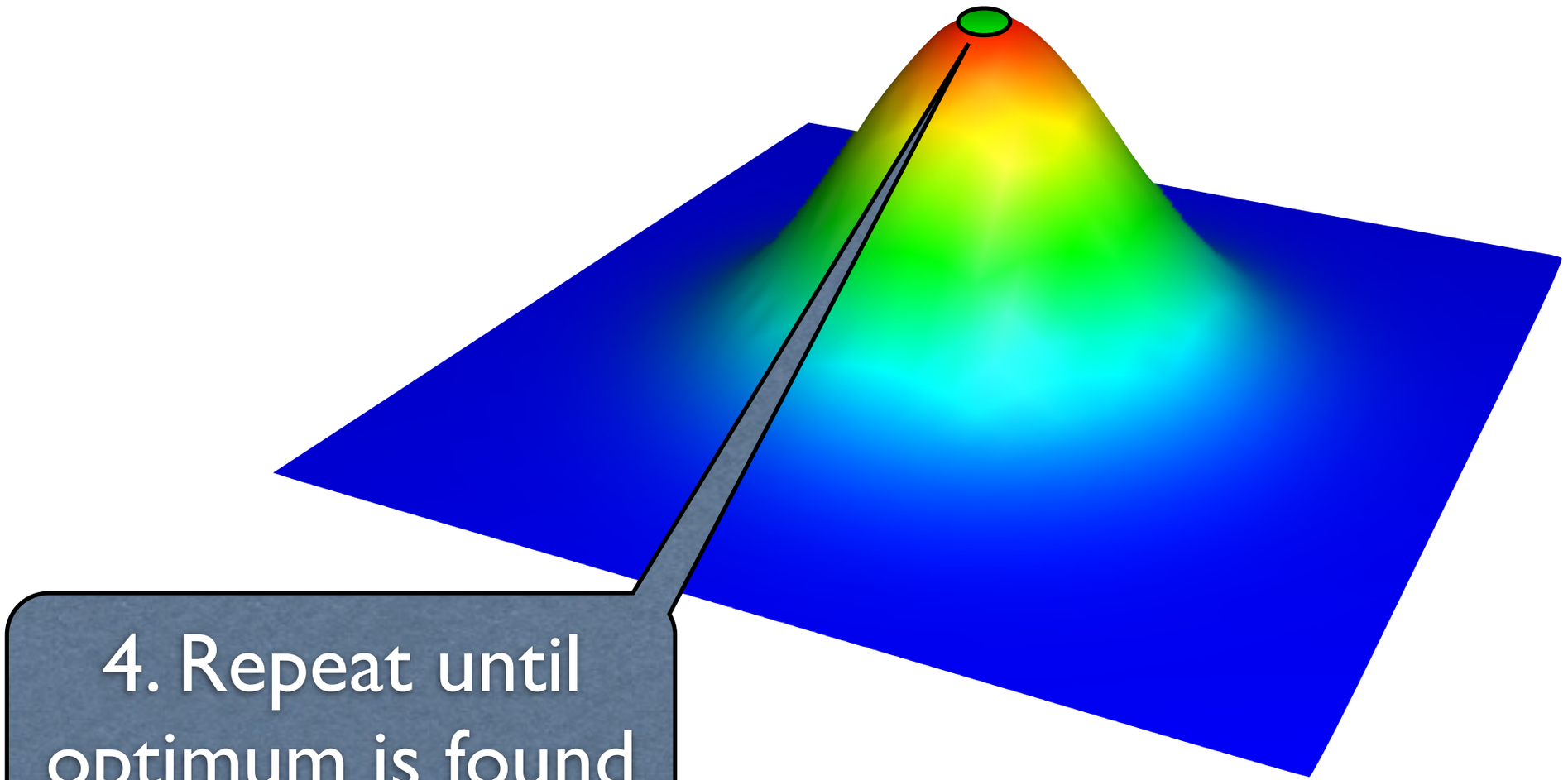


3. Choose best neighbor

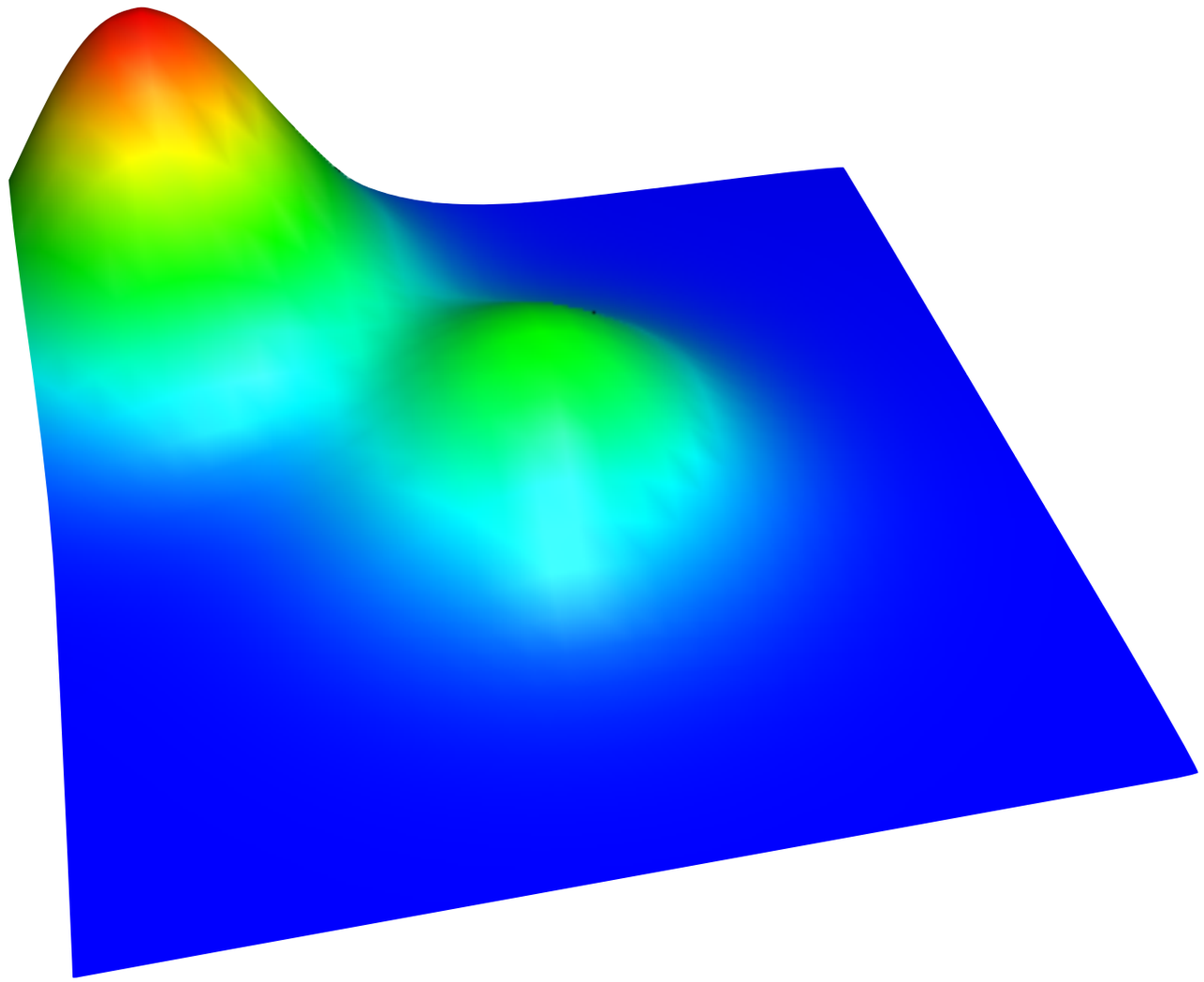
Hill Climbing



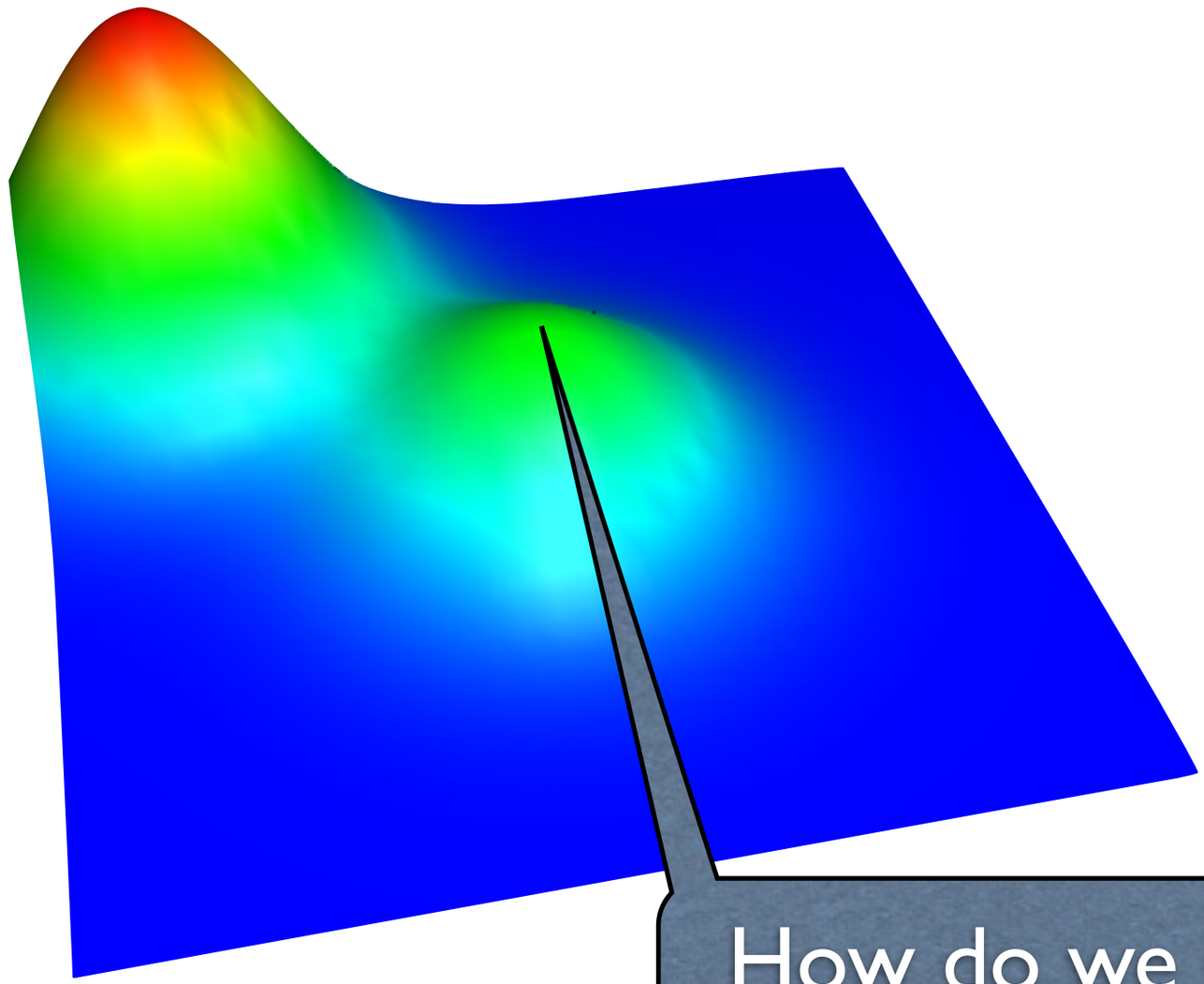
Hill Climbing



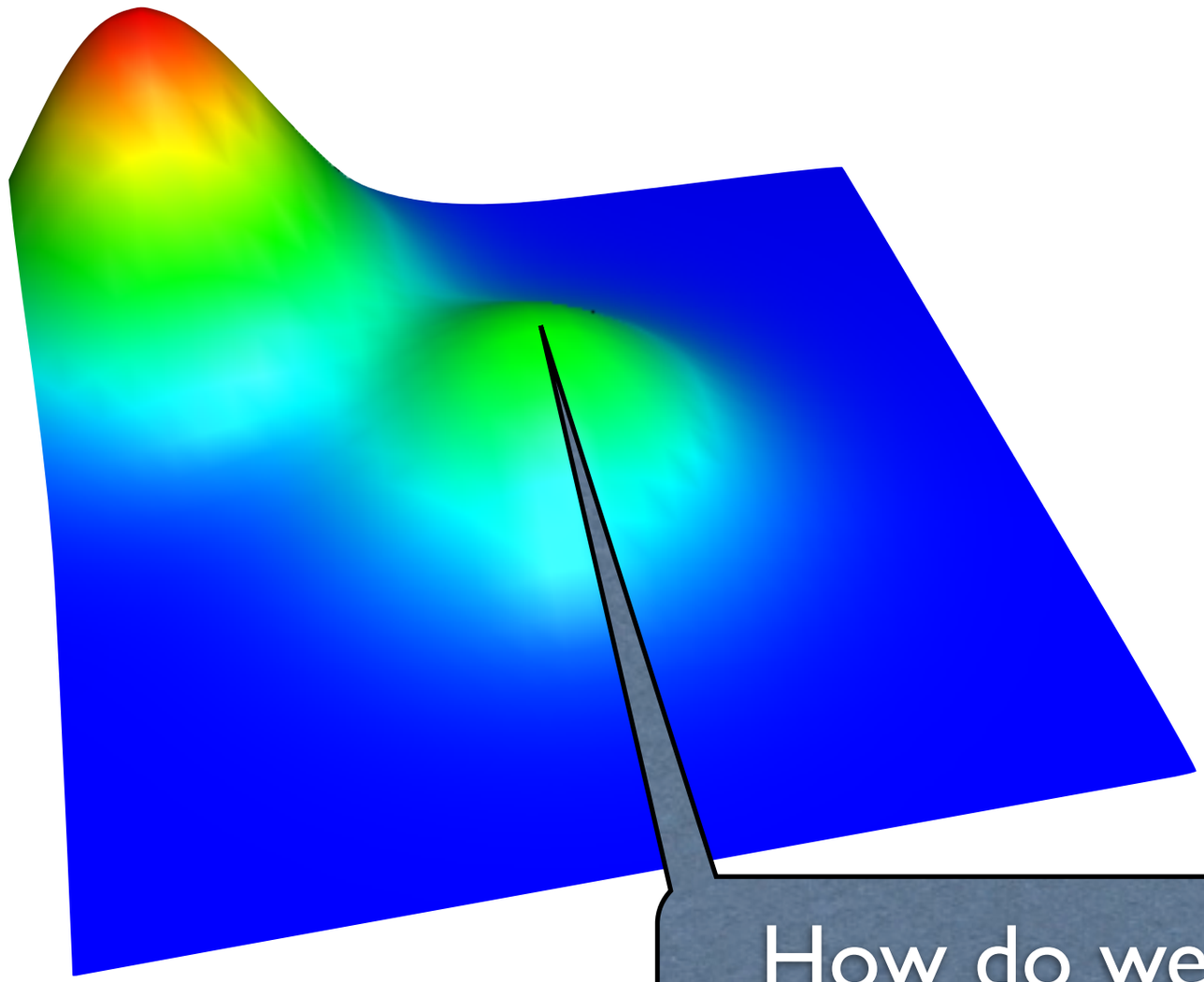
Local Optima



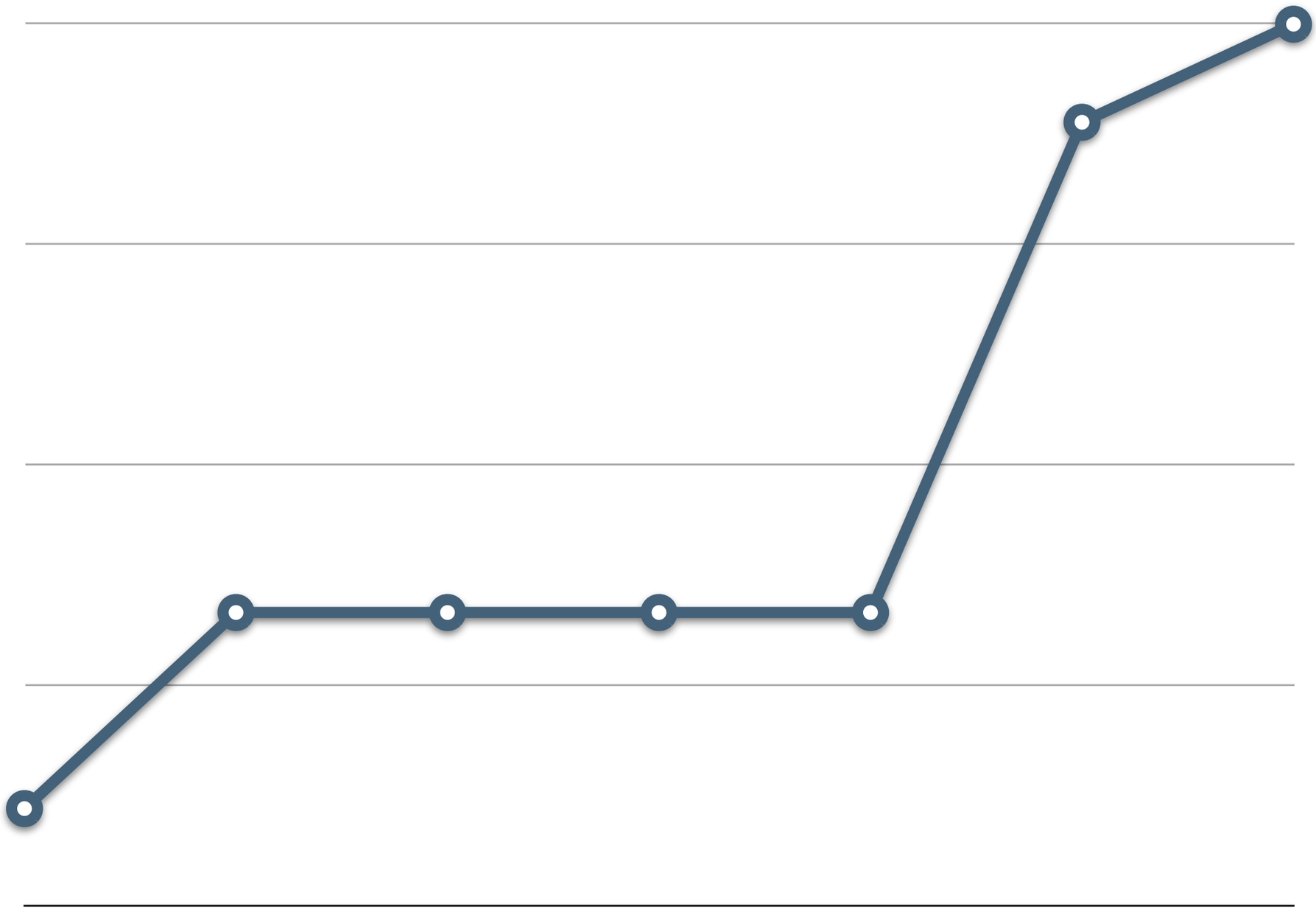
Local Optima



Local Optima



Plateaux

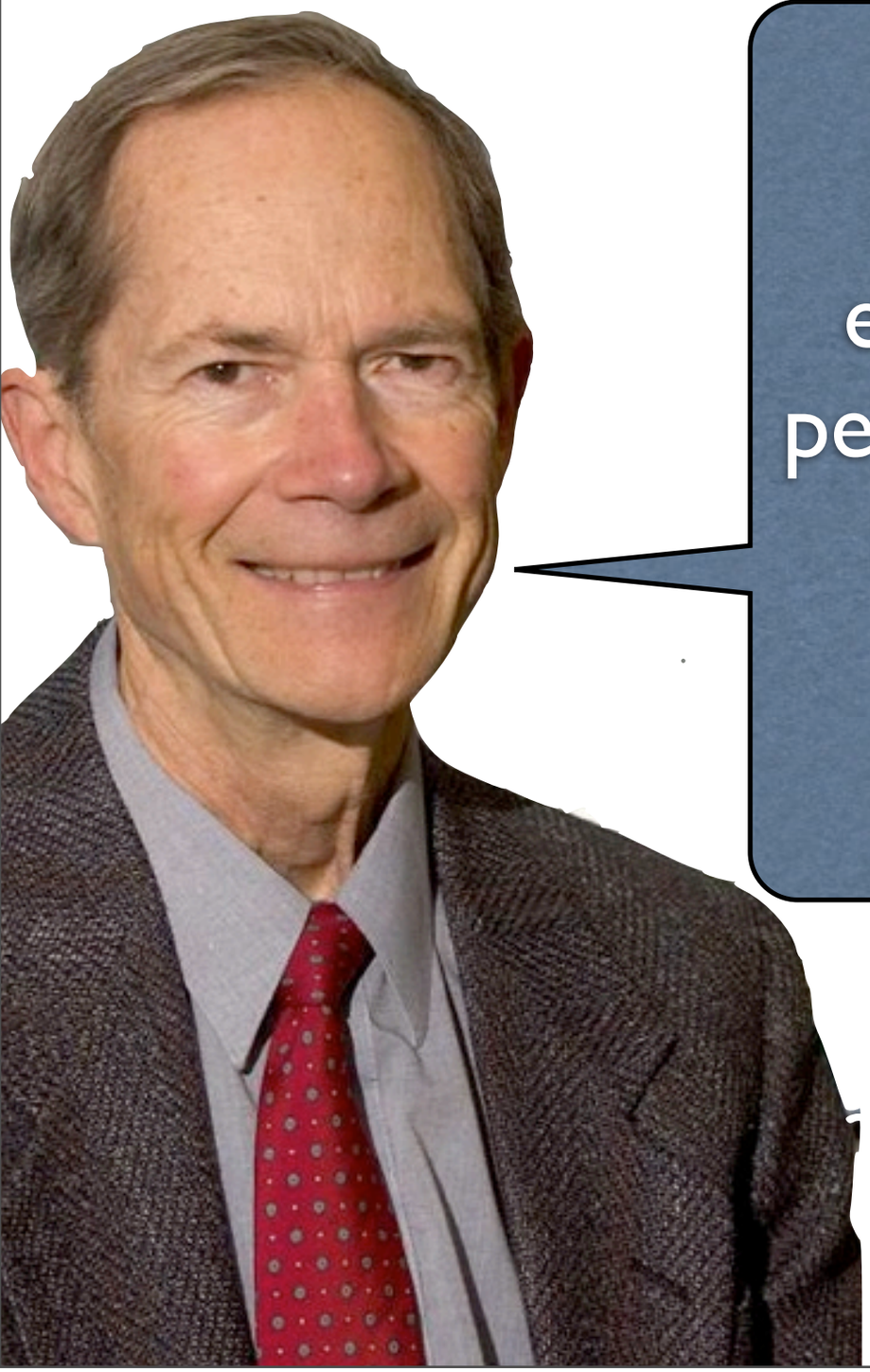


Plateaux

Hill Climbing

- Simple hill climbing
  First better neighbor is chosen
- Steepest ascent
  Best neighbor is chosen
- Random-restart
  Randomly reset and start with a different random value
- Resets can lead out of local optima

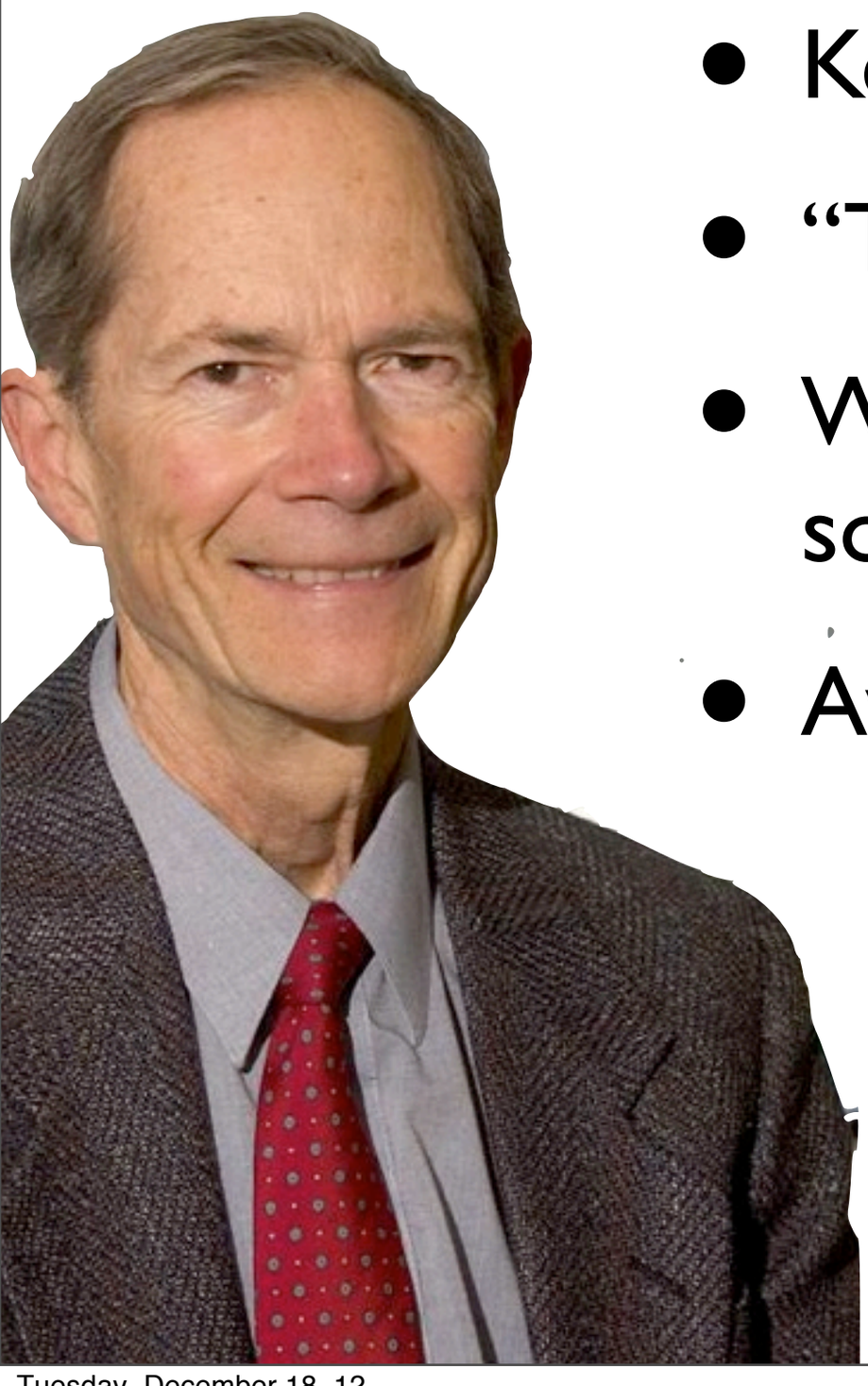
Tabu Search



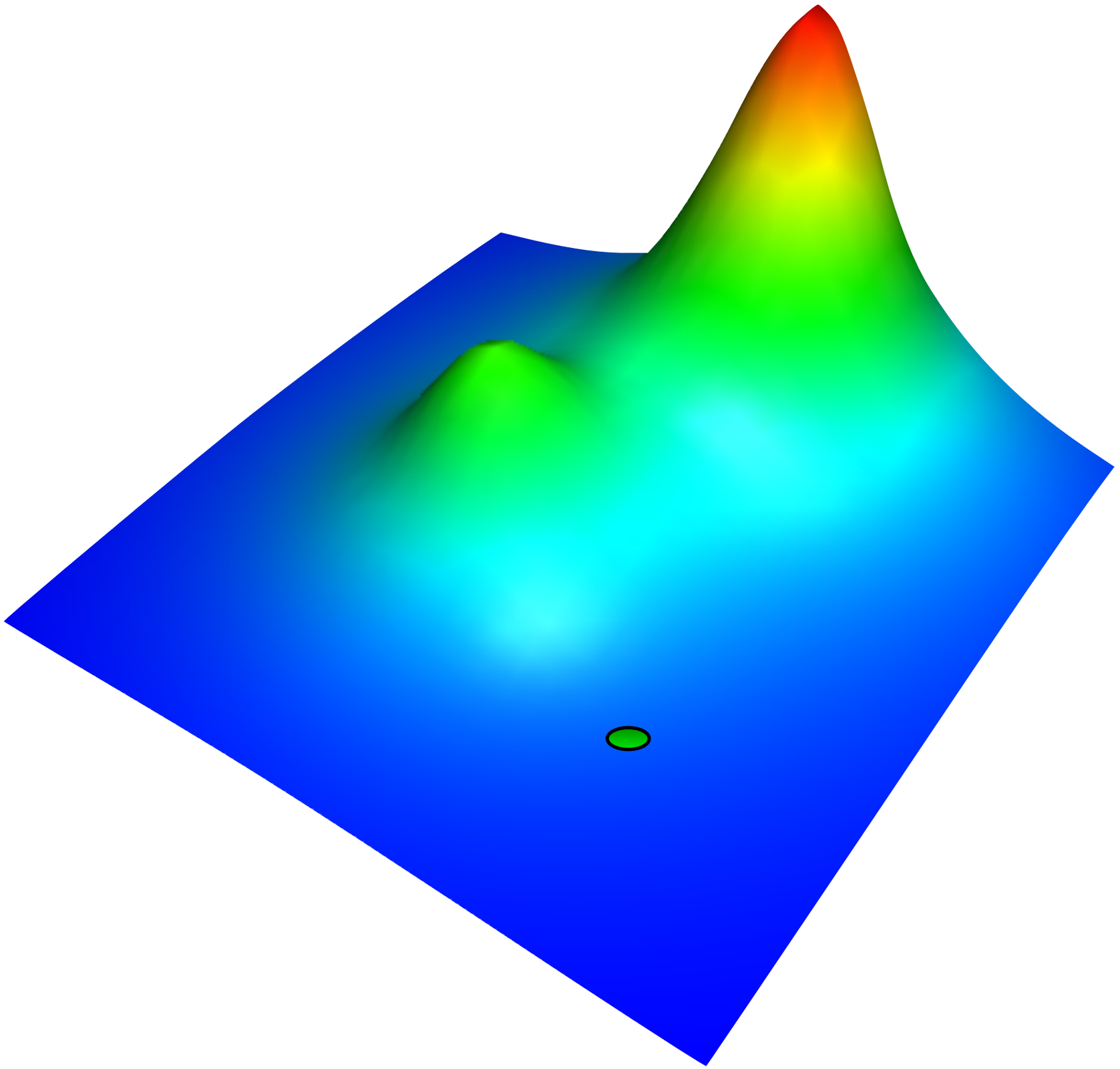
The overall approach is to avoid entrainment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited.

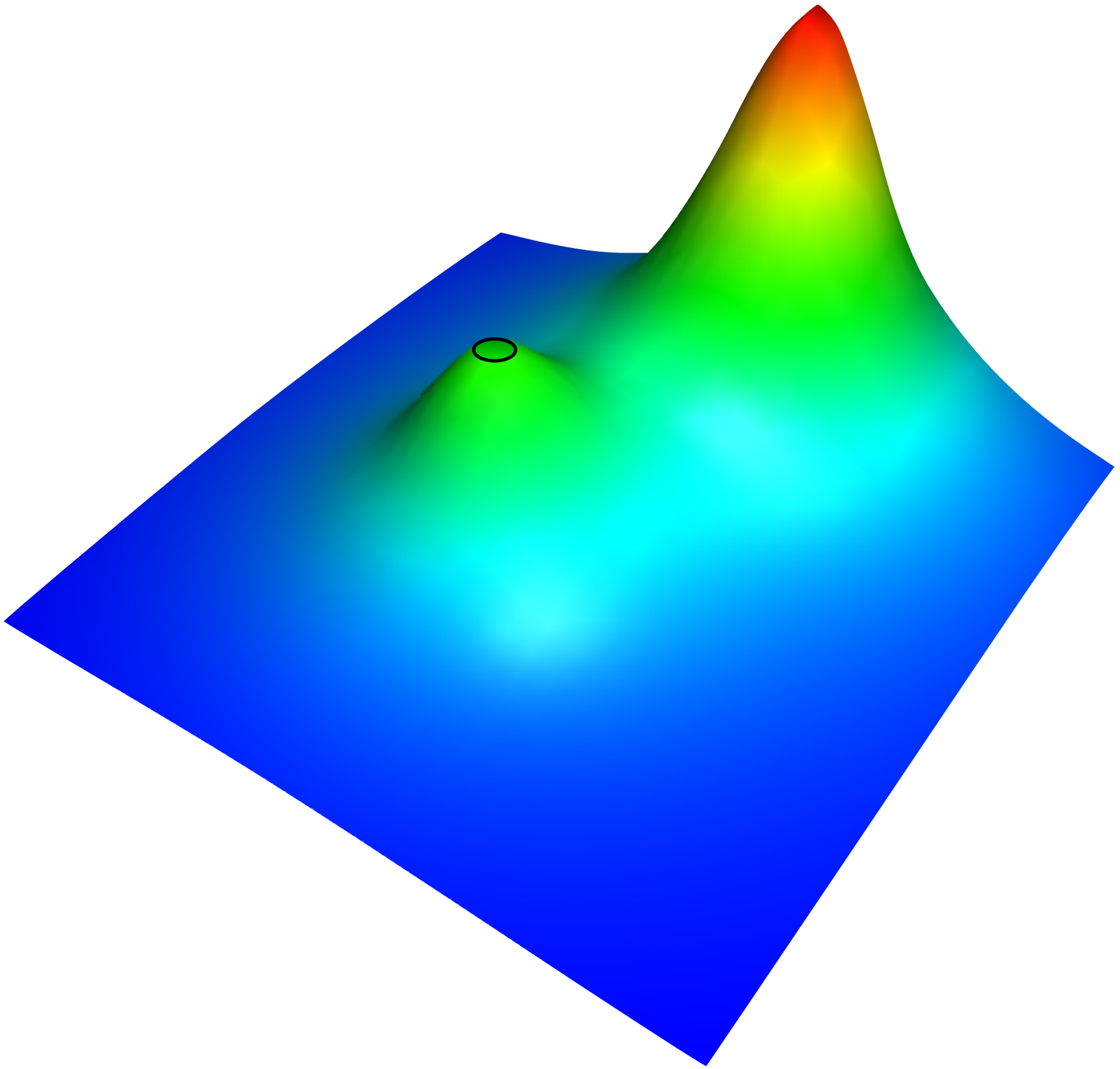
(Fred Glover 1987)

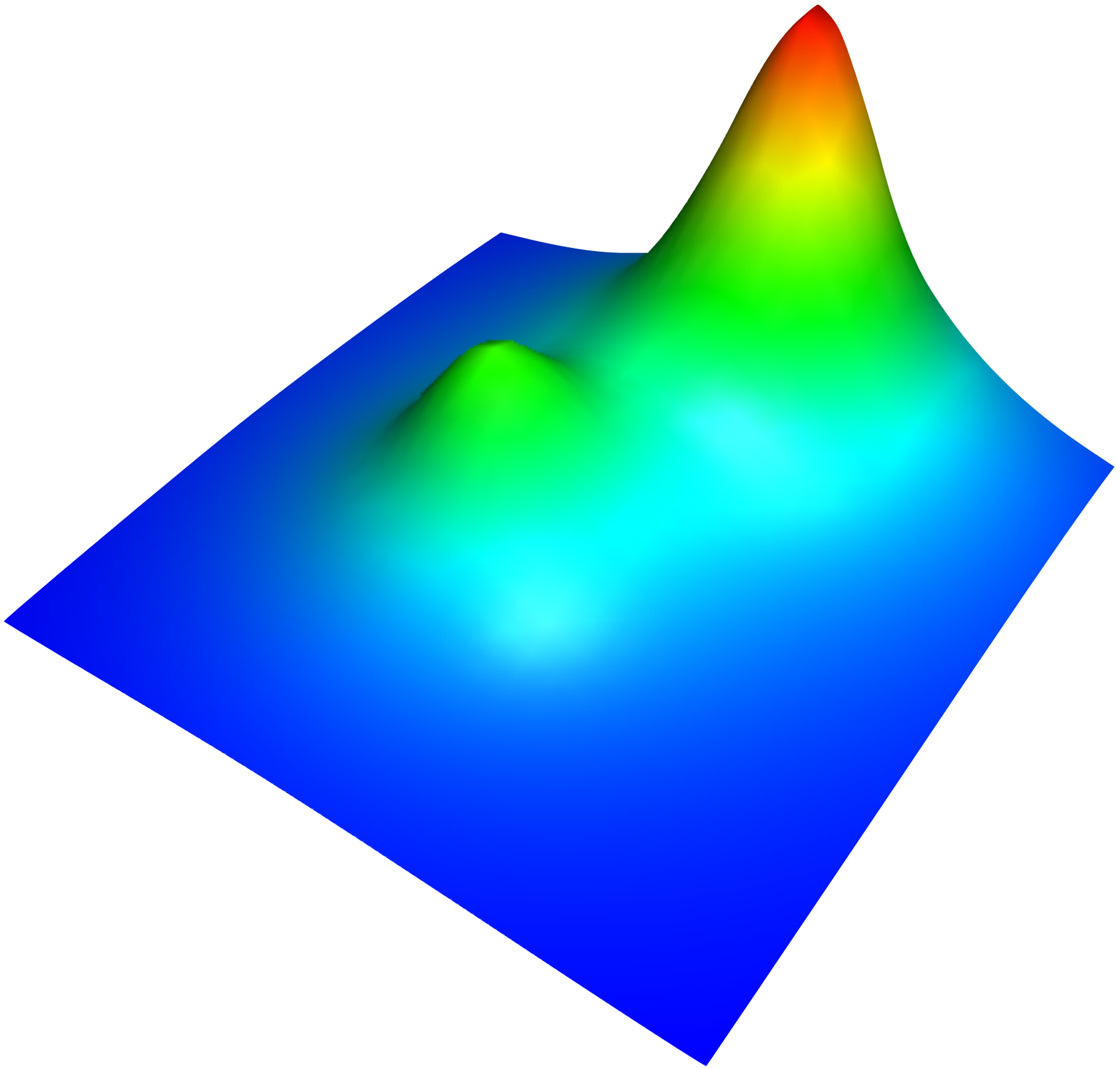
Tabu Search

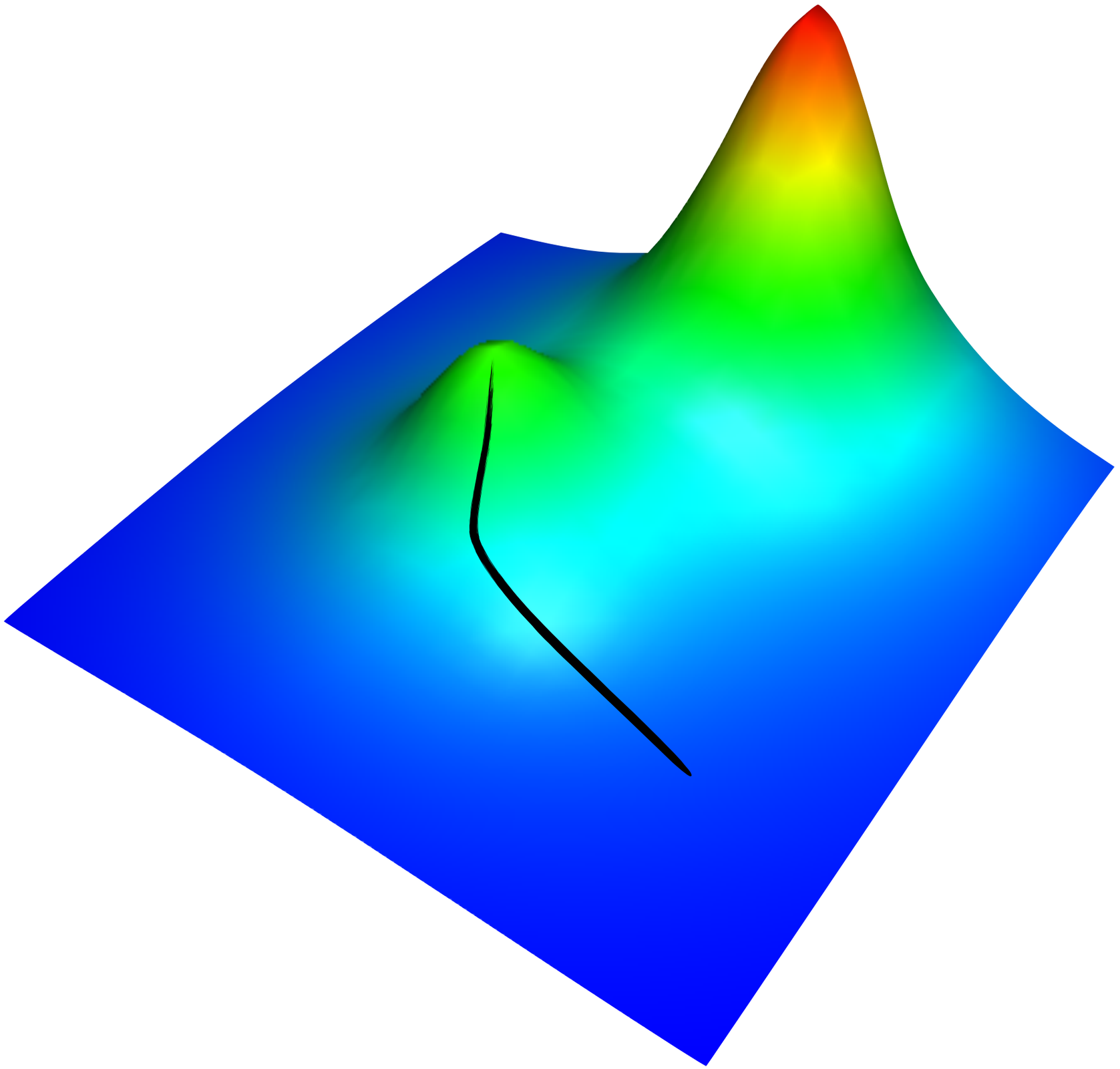


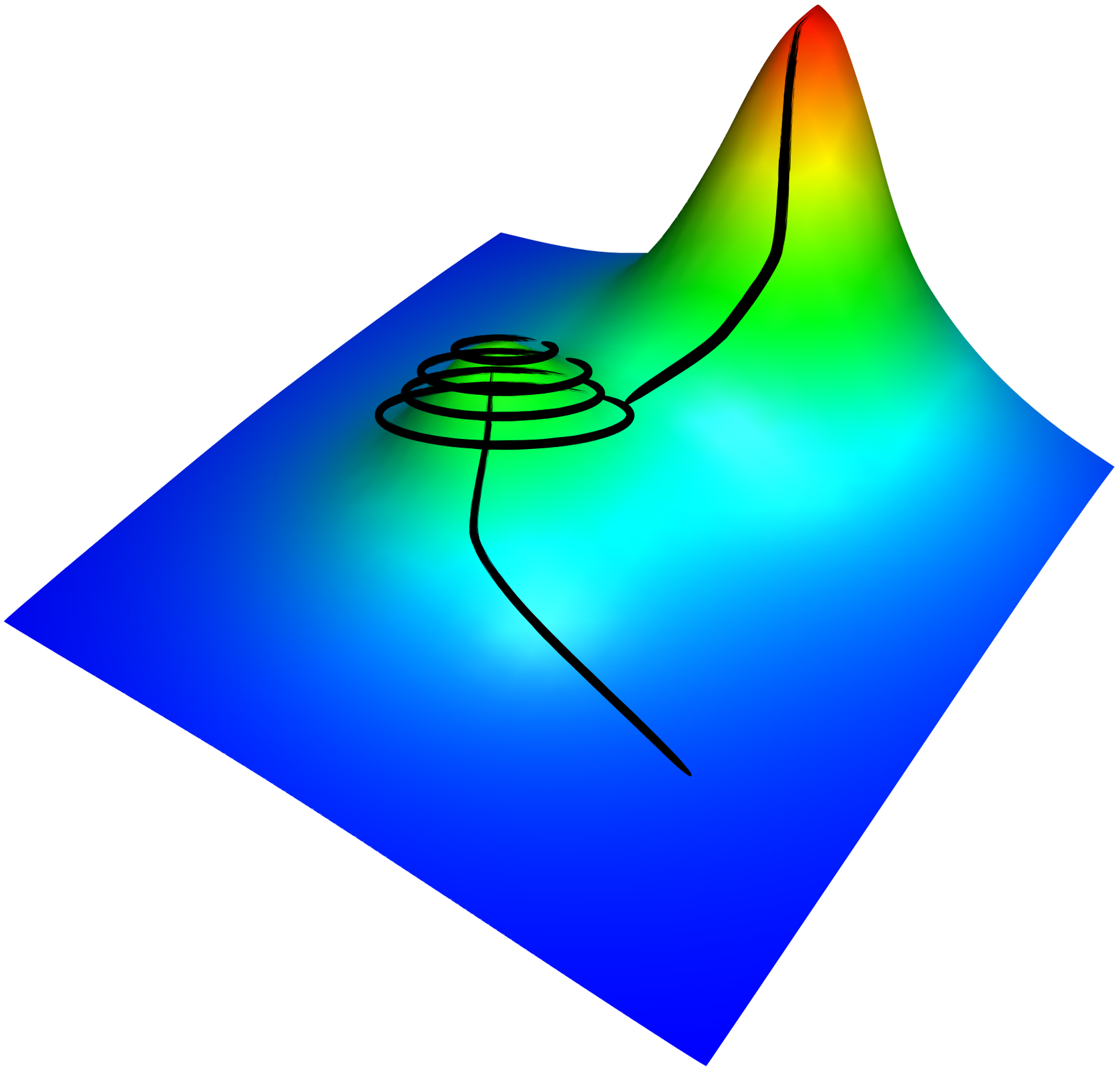
- Keep track of last k moves
- "Tabu list"
- When choosing next move, take no solution that is on tabu list
- Avoids cycles of length k



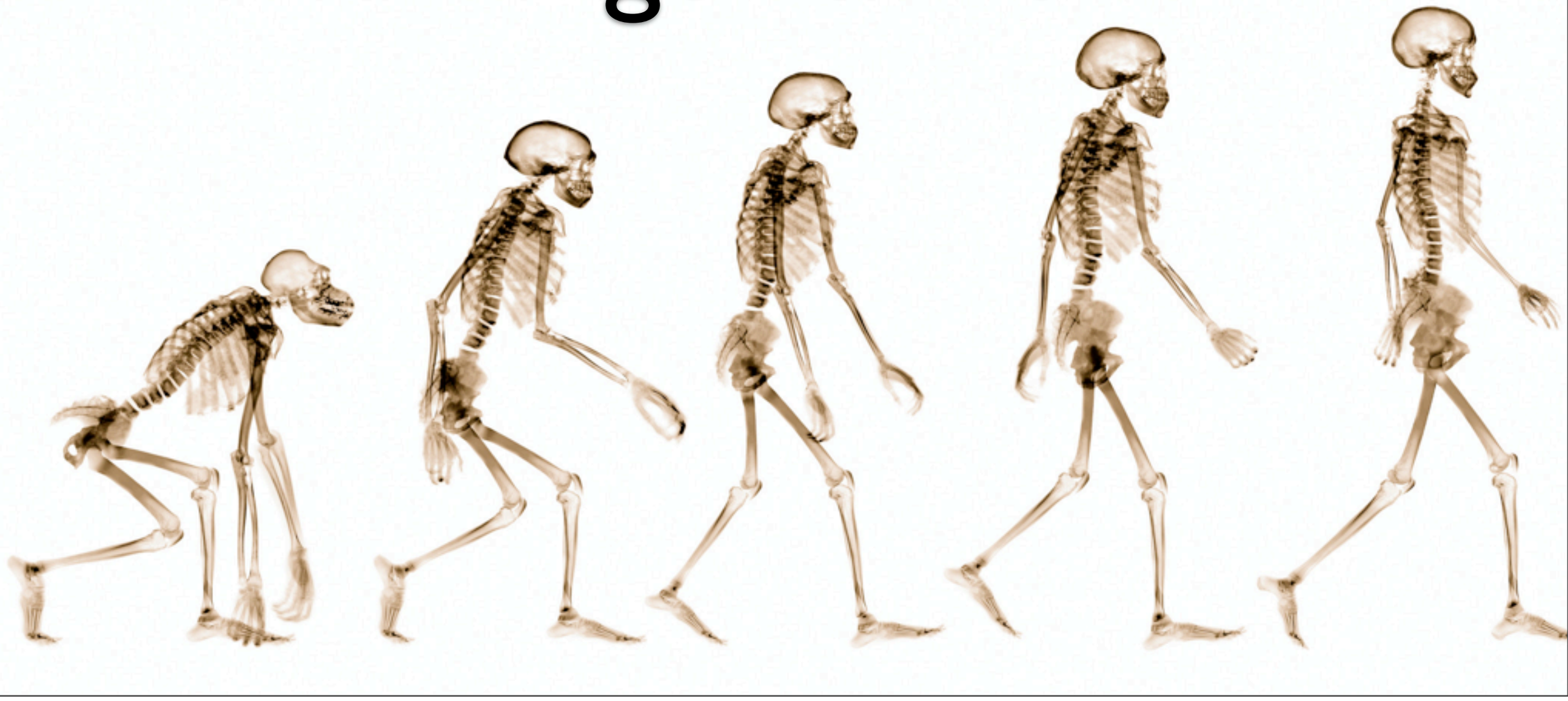






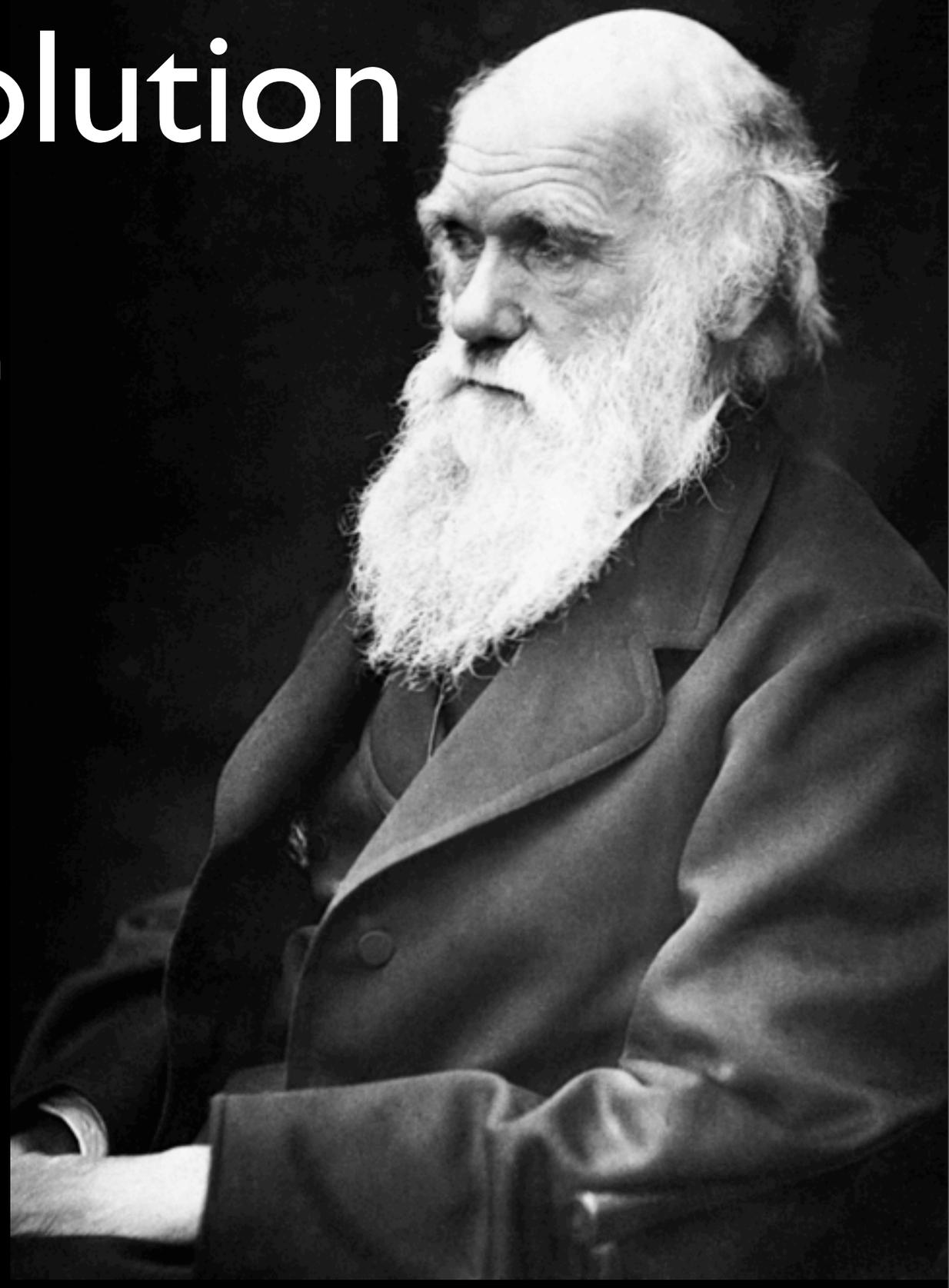


Evolutionary Algorithms



Biological Evolution

- Gene
  Unit of information • passed from one generation to another
- Natural Selection
- Survival of The Fittest
- Origin of New Species

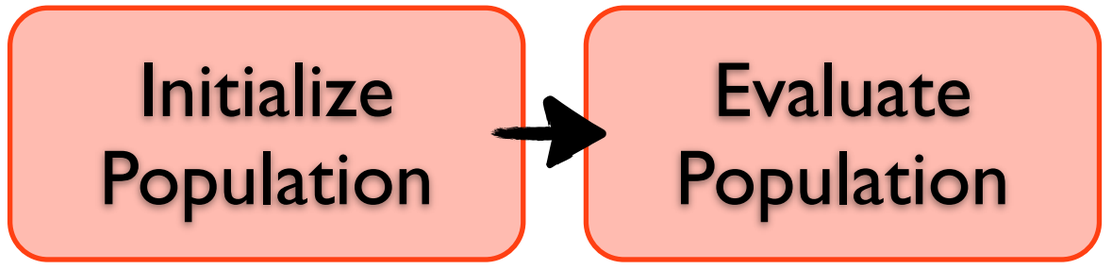


Genetic Algorithms

Genetic Algorithms

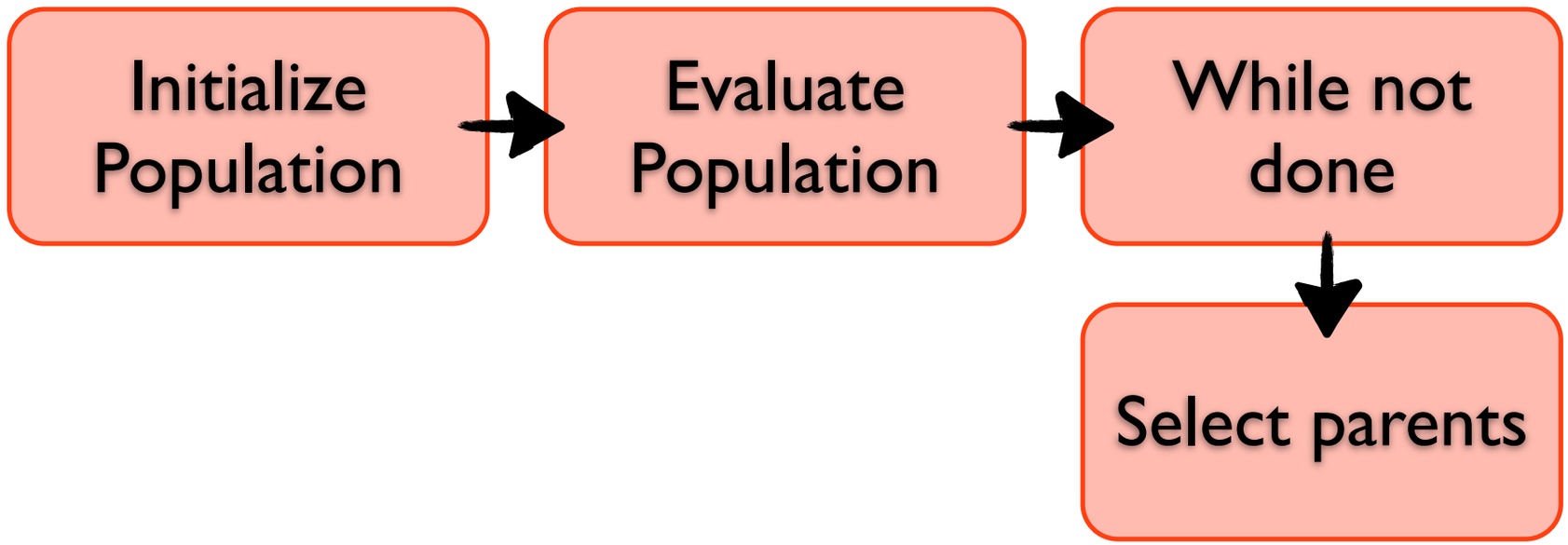
Initialize Population

Genetic Algorithms

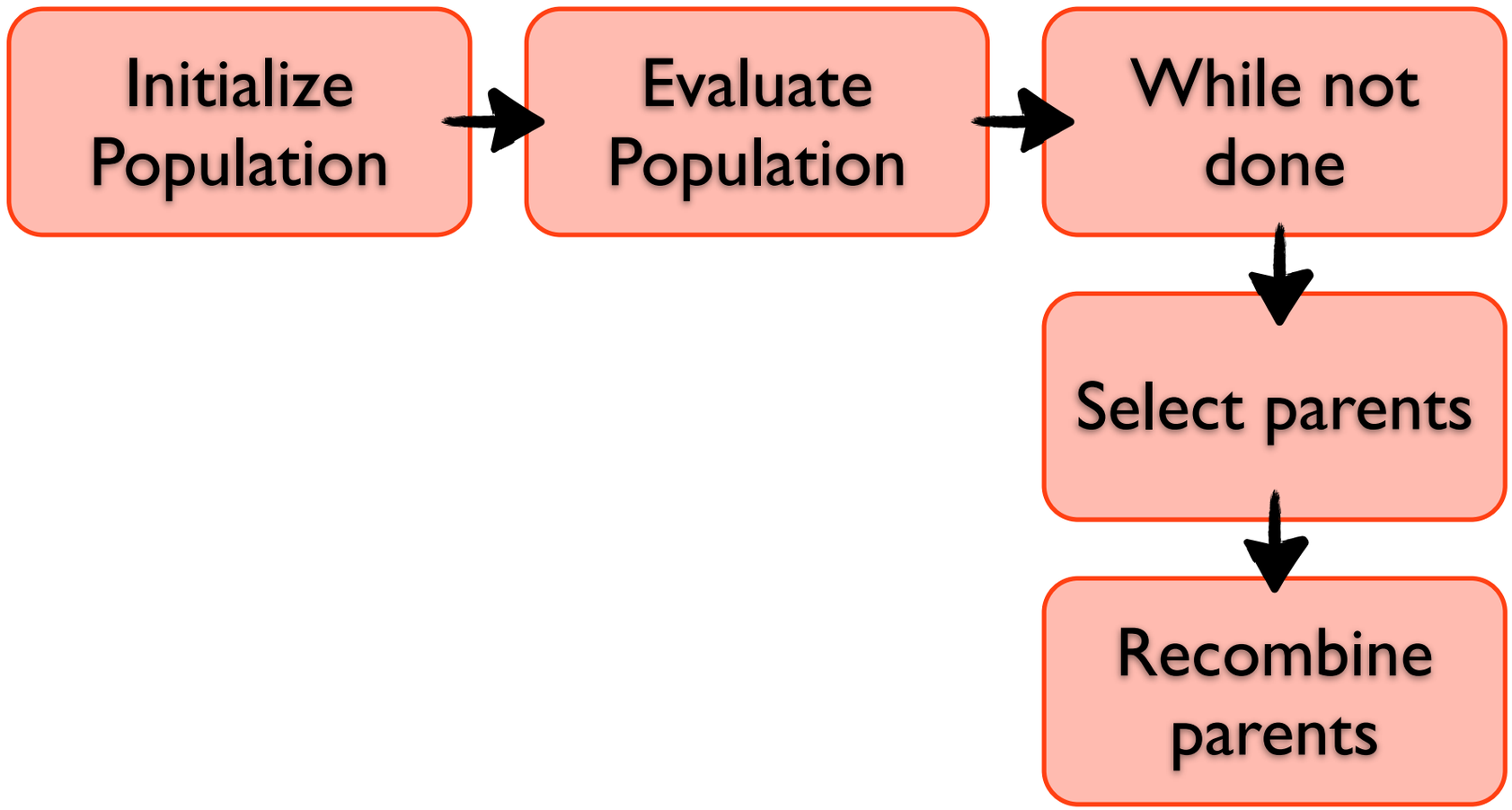


Genetic Algorithms

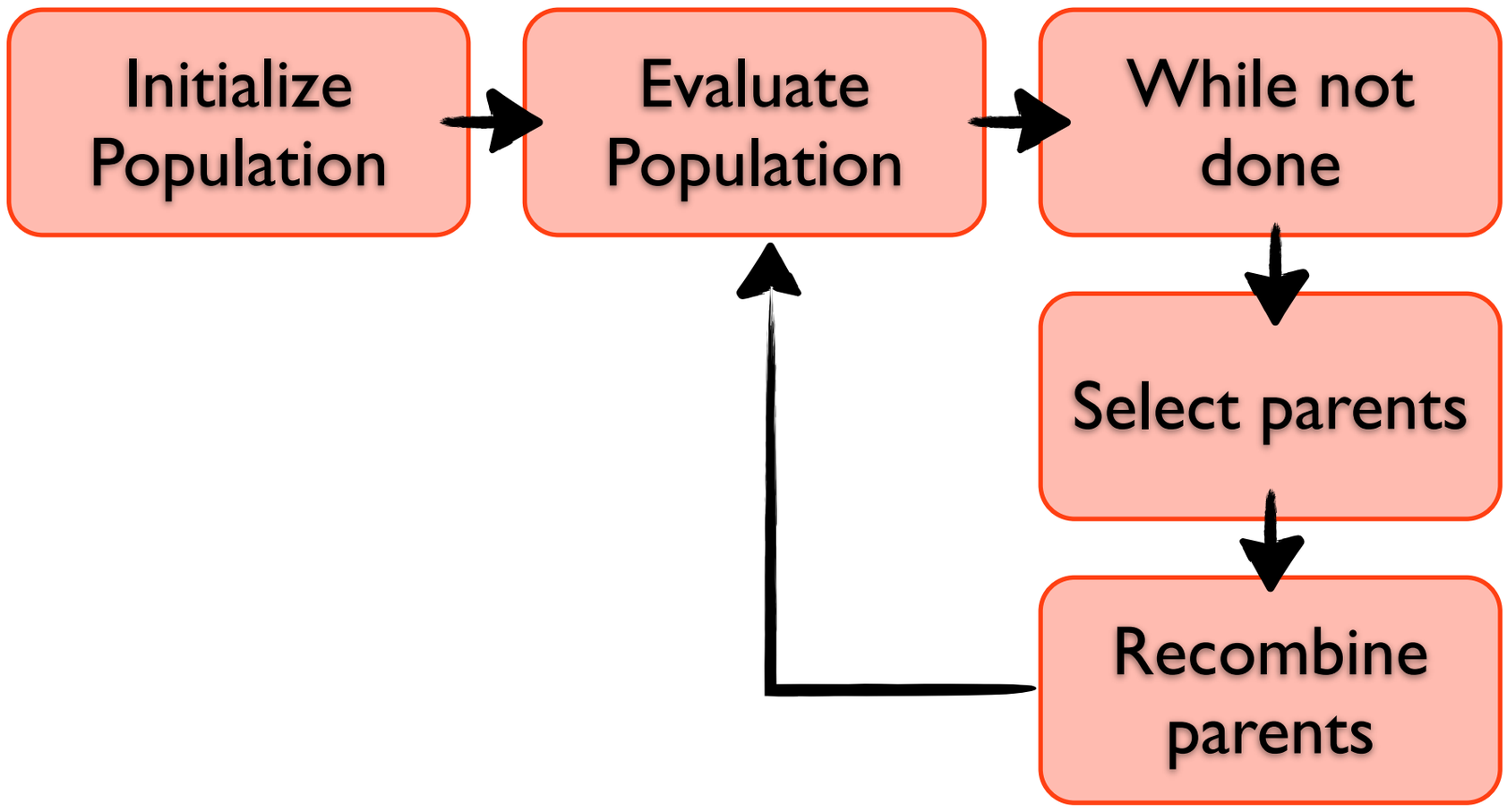
Genetic Algorithms



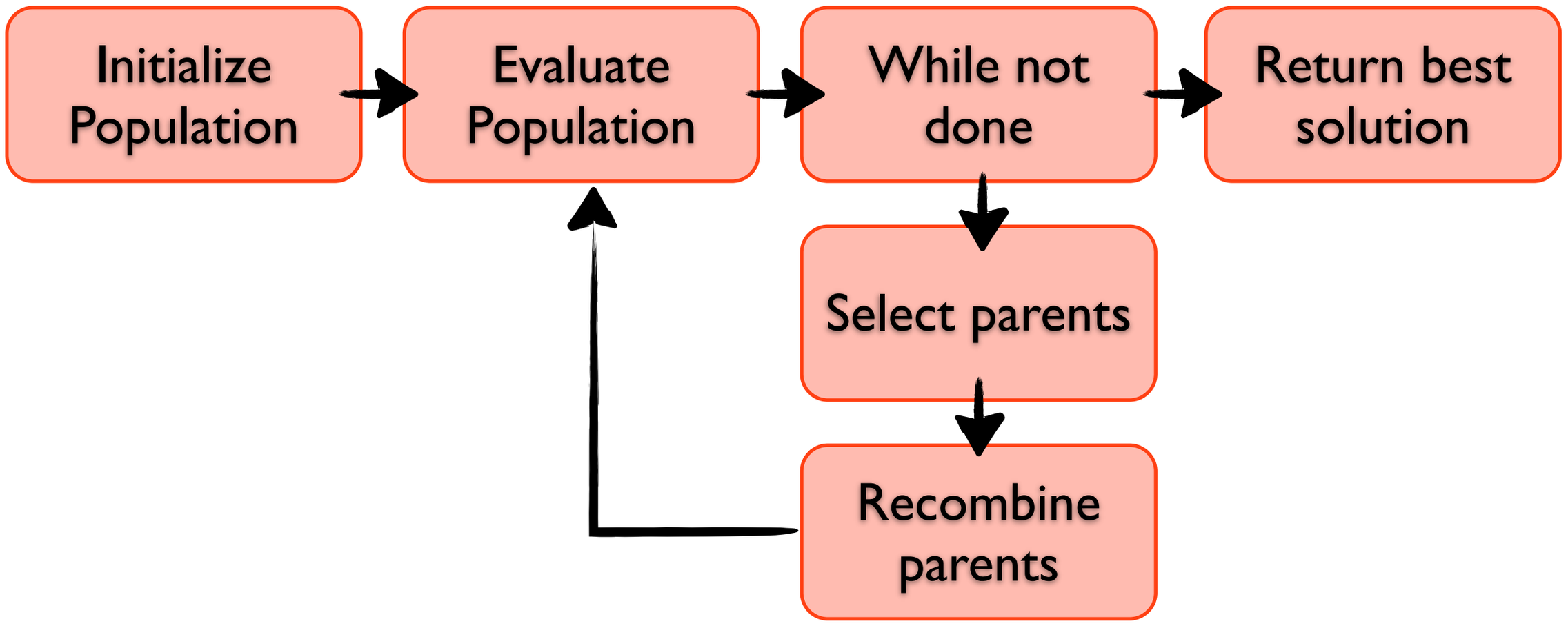
Genetic Algorithms

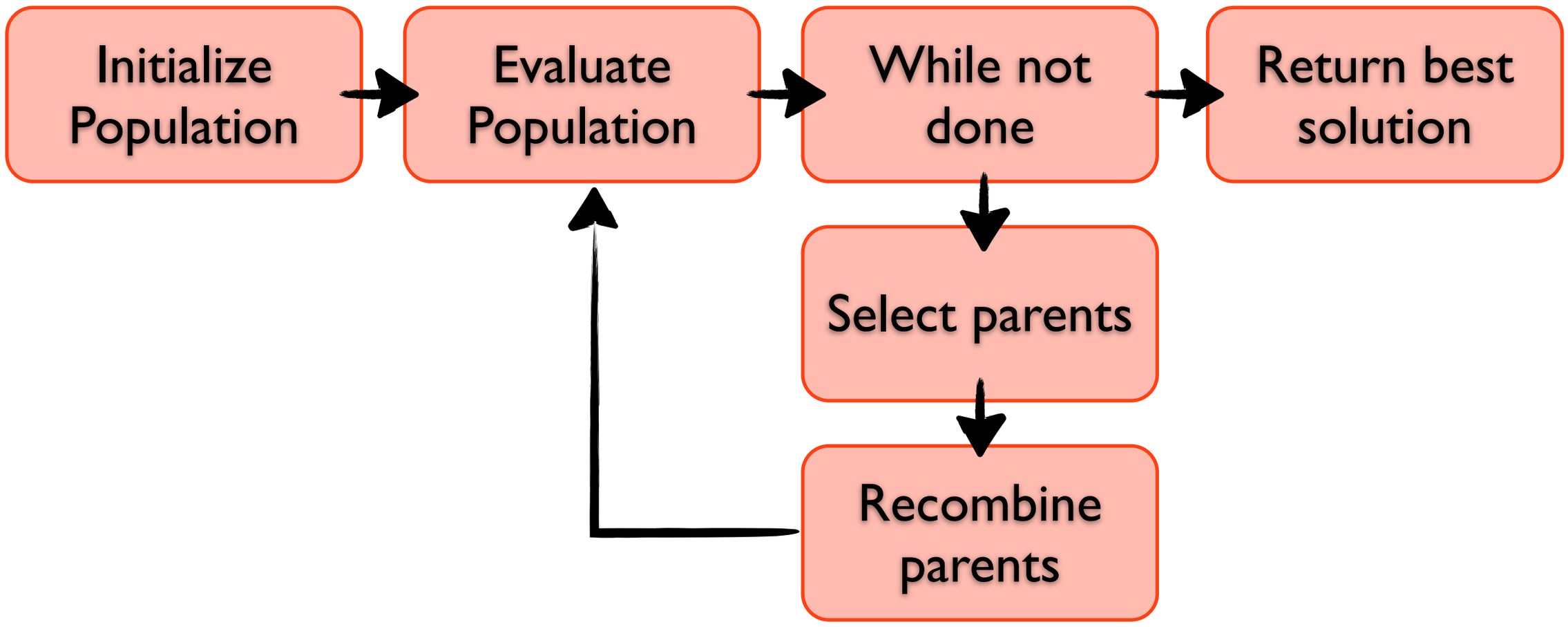


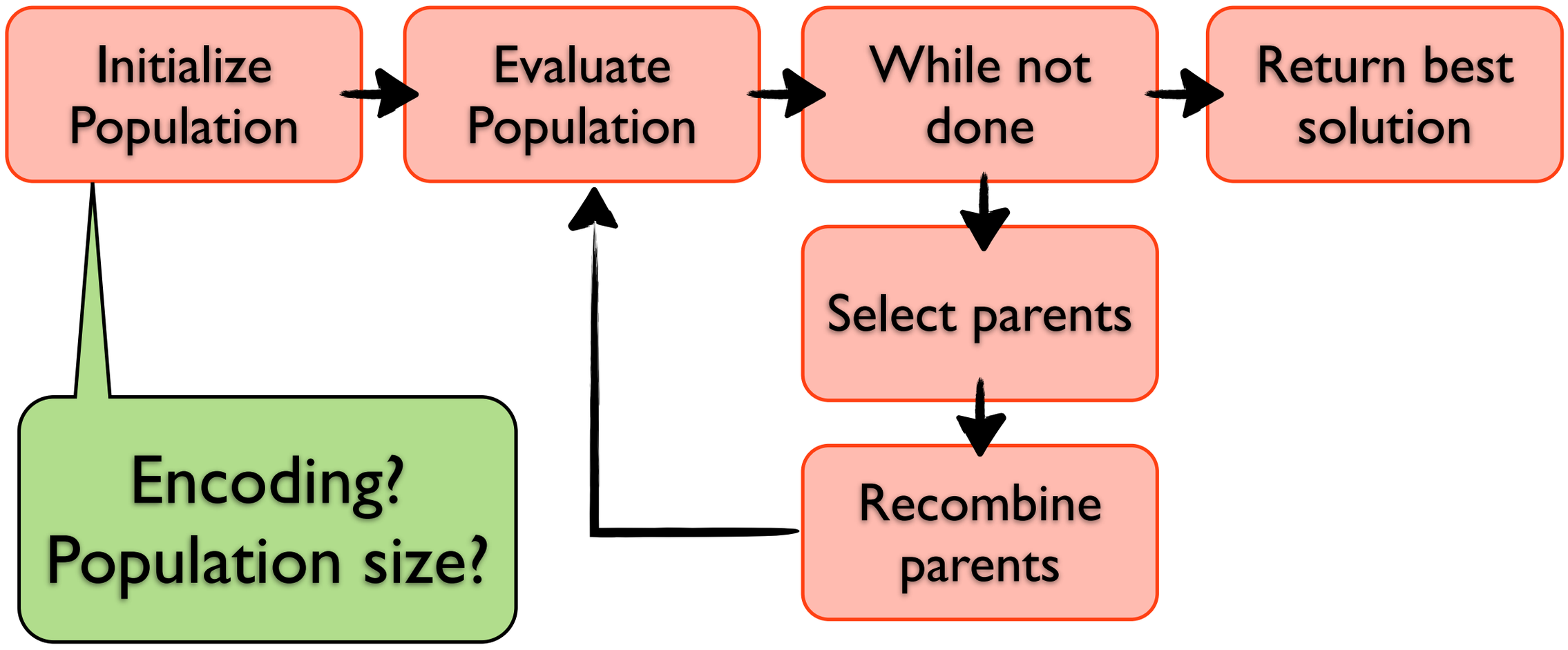
Genetic Algorithms

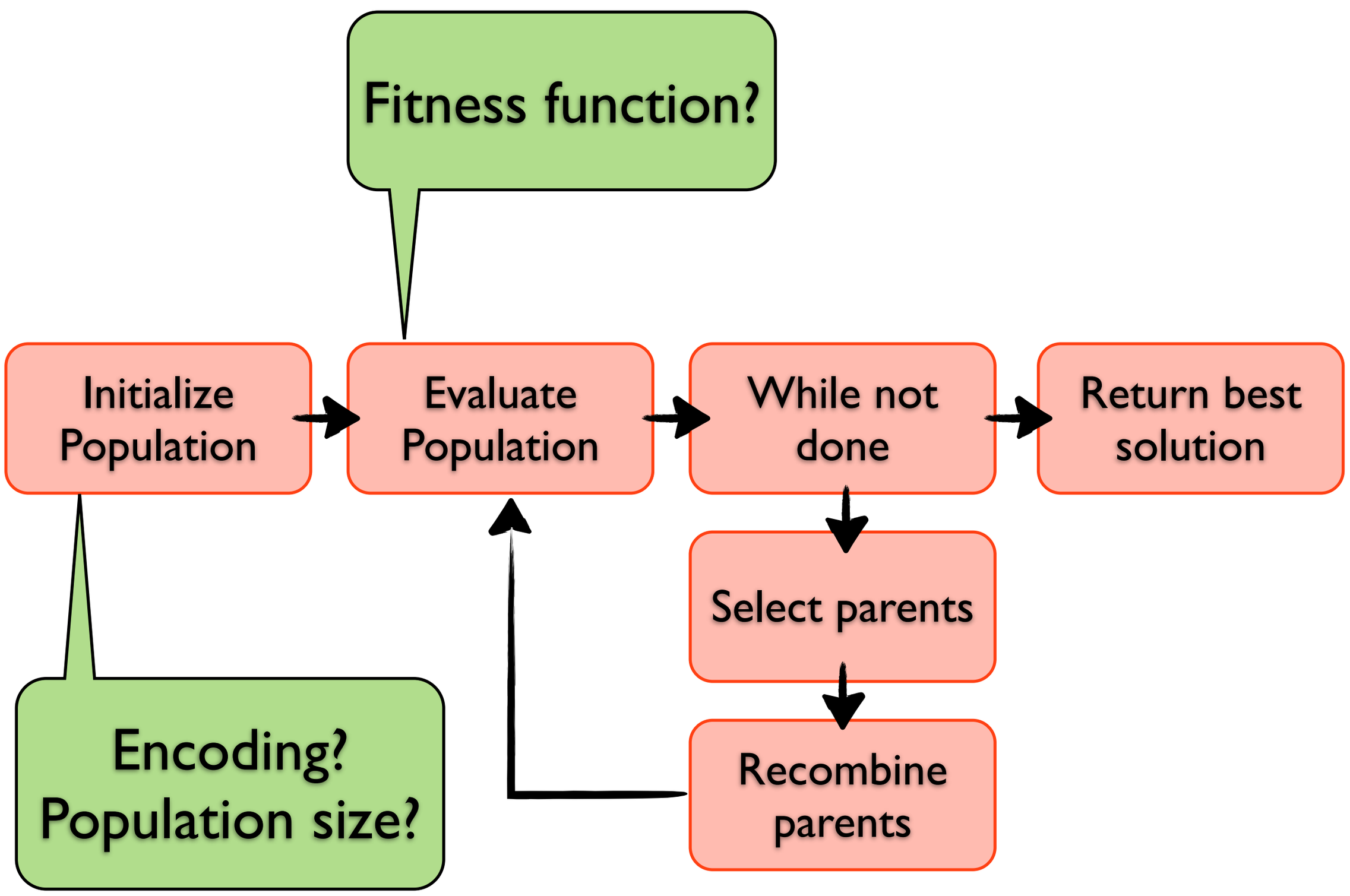


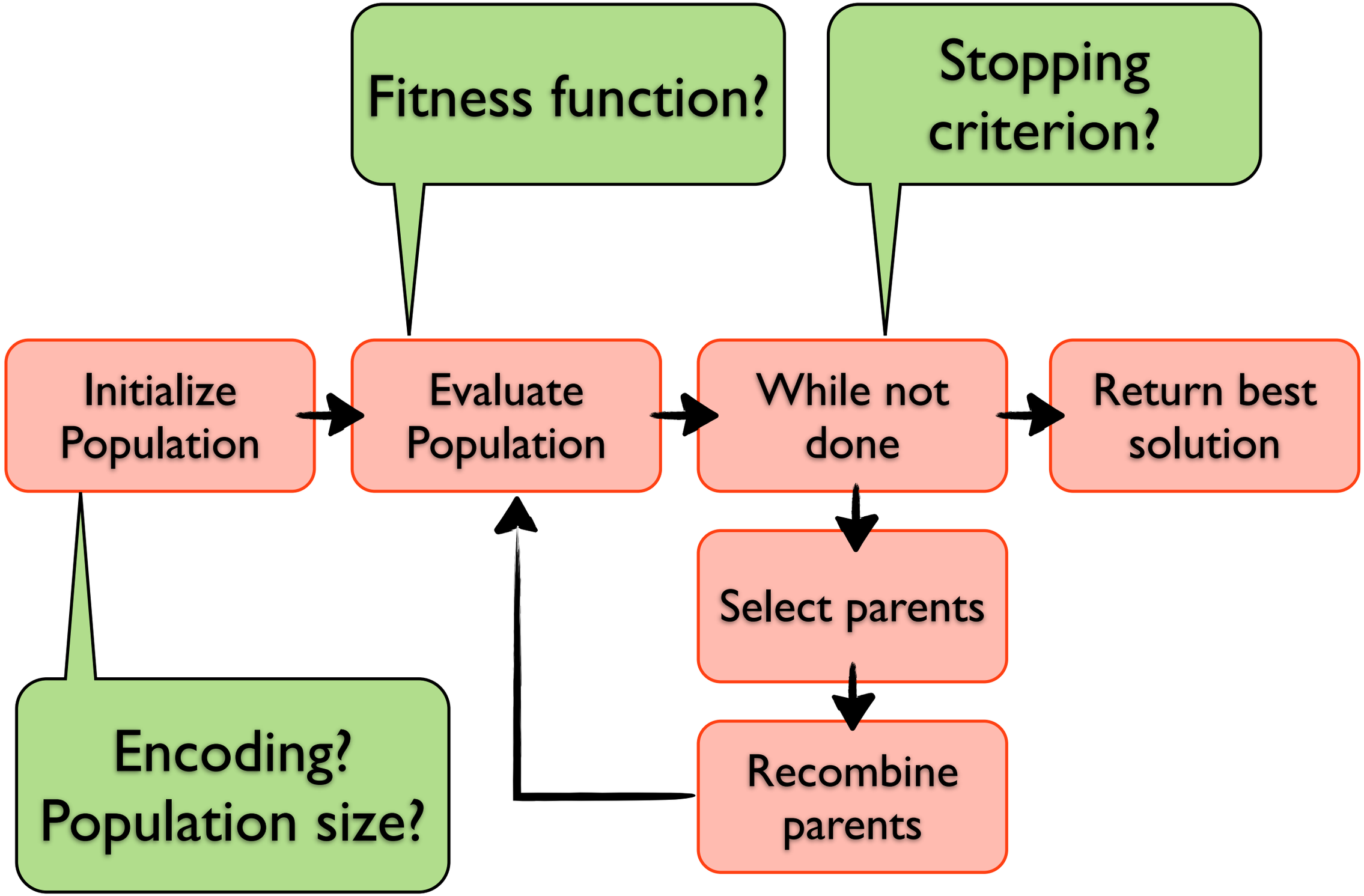
Genetic Algorithms

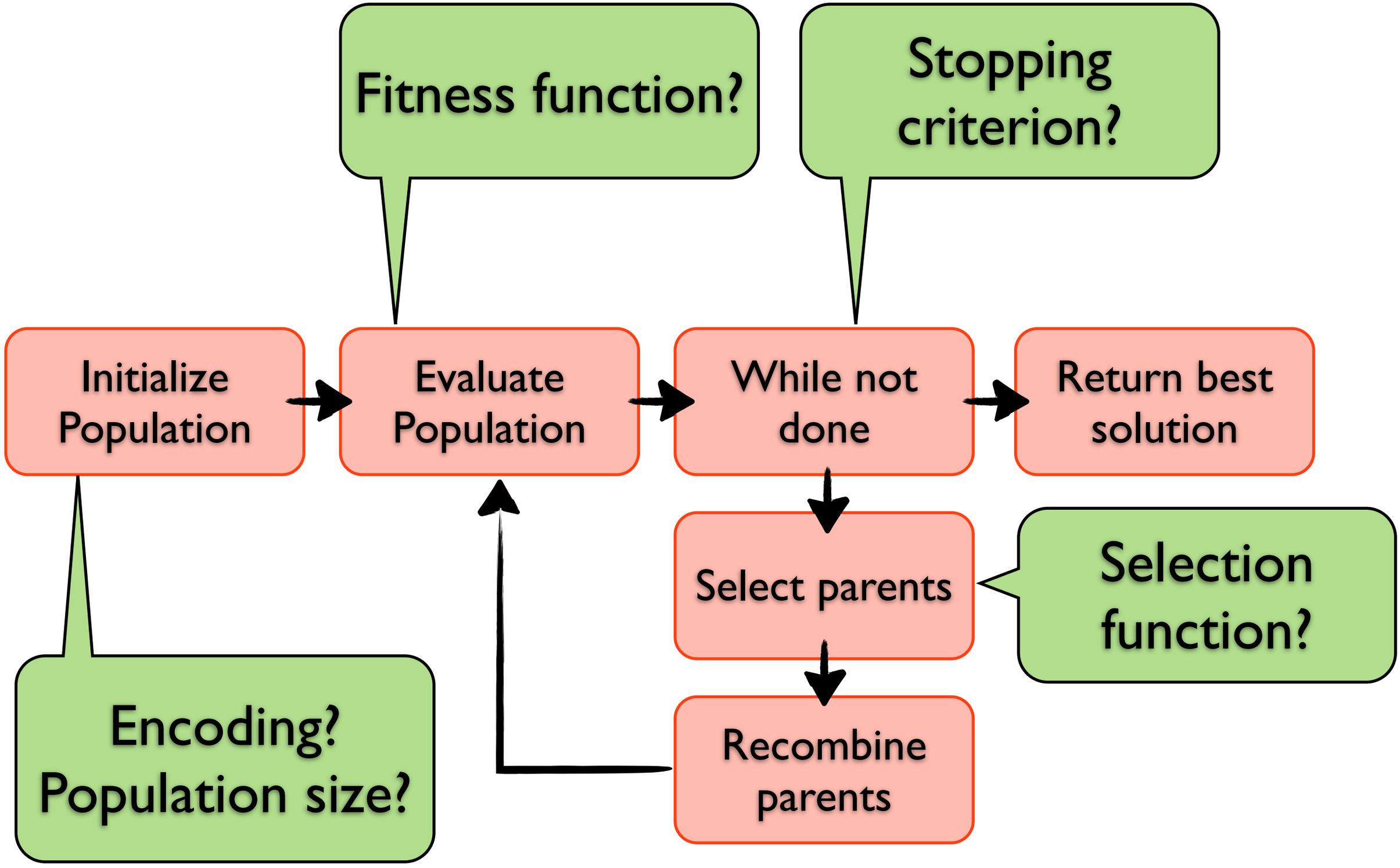


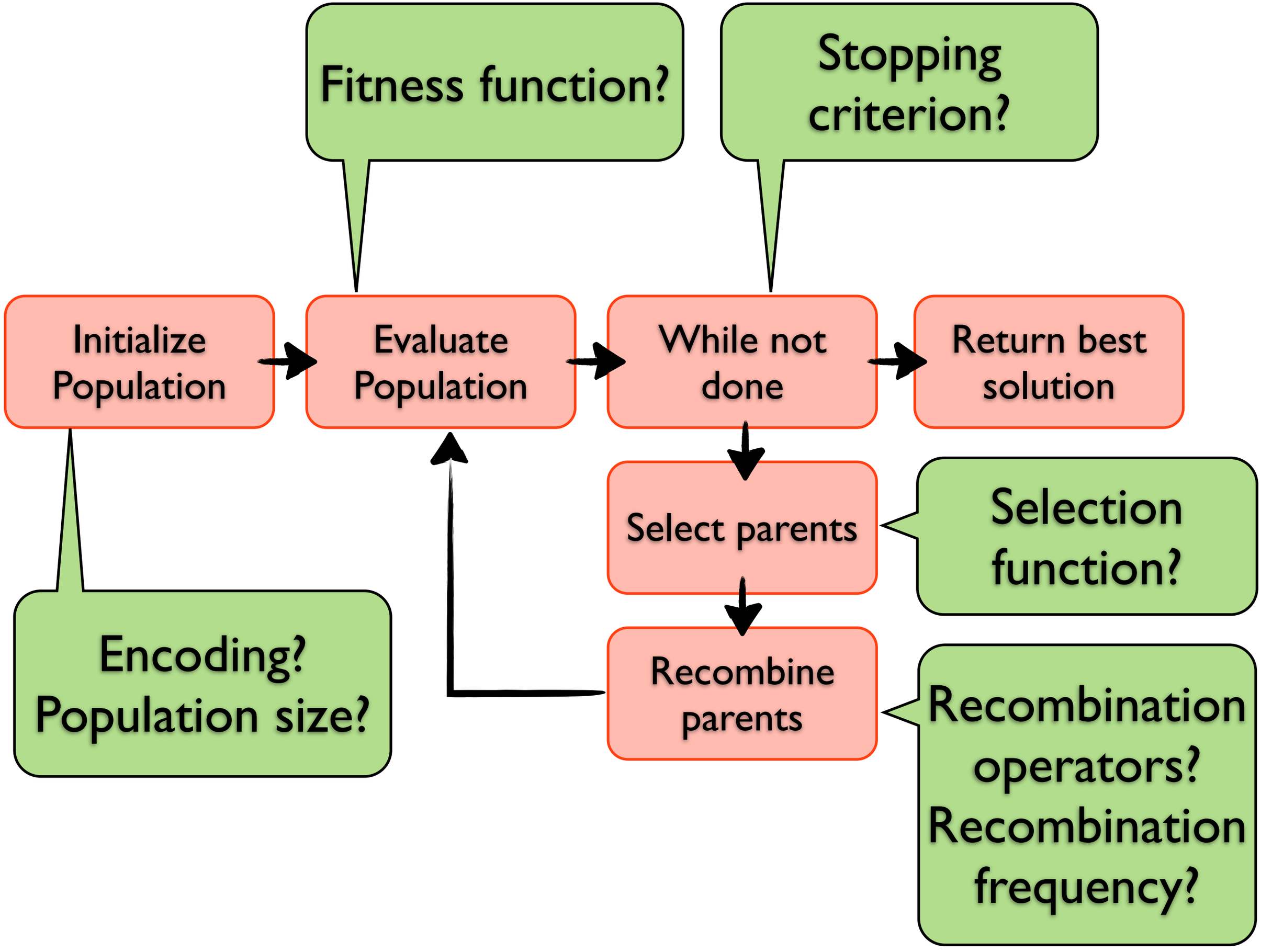

Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Recombine parents


Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Recombine parents
Encoding?
Population size?


Fitness function?
Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Recombine parents
Encoding?
Population size?


Fitness function?
Stopping criterion?
Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Recombine parents
Encoding?
Population size?


Fitness function?
Stopping criterion?
Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Selection function?
Encoding? Population size?
Recombine parents


Fitness function?
Stopping criterion?
Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Selection function?
Encoding? Population size?
Recombine parents
Recombination operators? Recombination frequency?

Fitness function?

Stopping criterion?

Initialize Population

Evaluate Population

While not done

Return best solution

Select parents

Selection function?

Recombine parents

Recombination operators? Recombination frequency?

Encoding? Population size?

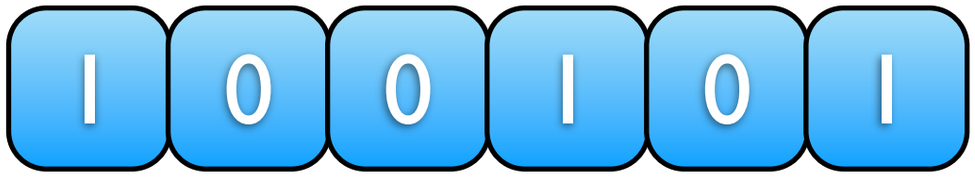
Encoding

Encoding

- Binary encoding

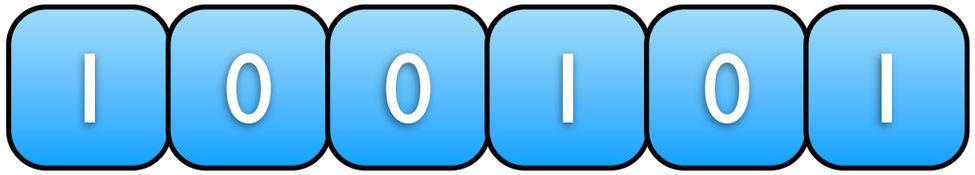
Encoding

- Binary encoding



Encoding

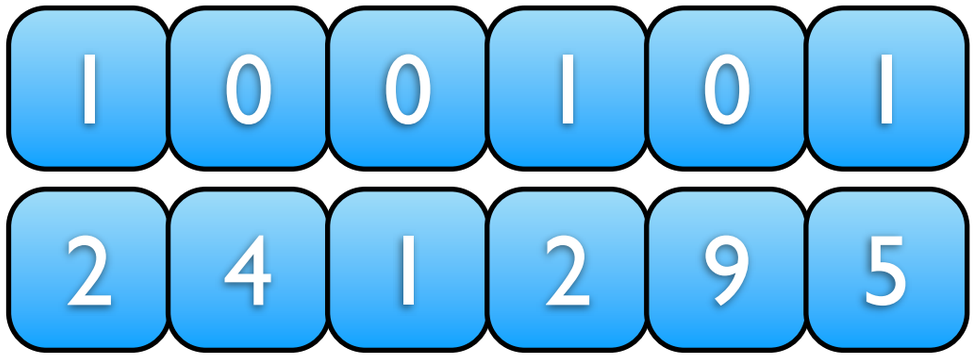
- Binary encoding



- Number encoding

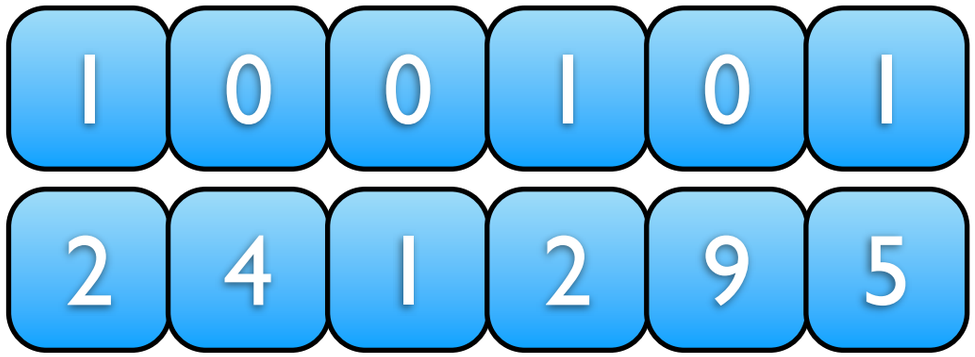
Encoding

- Binary encoding
- Number encoding



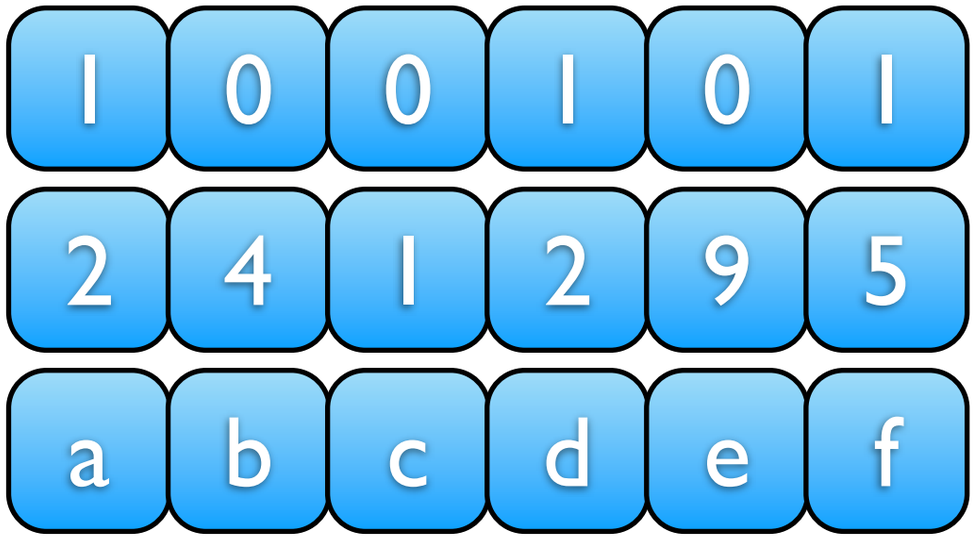
Encoding

- Binary encoding
- Number encoding
- Any encoding really...



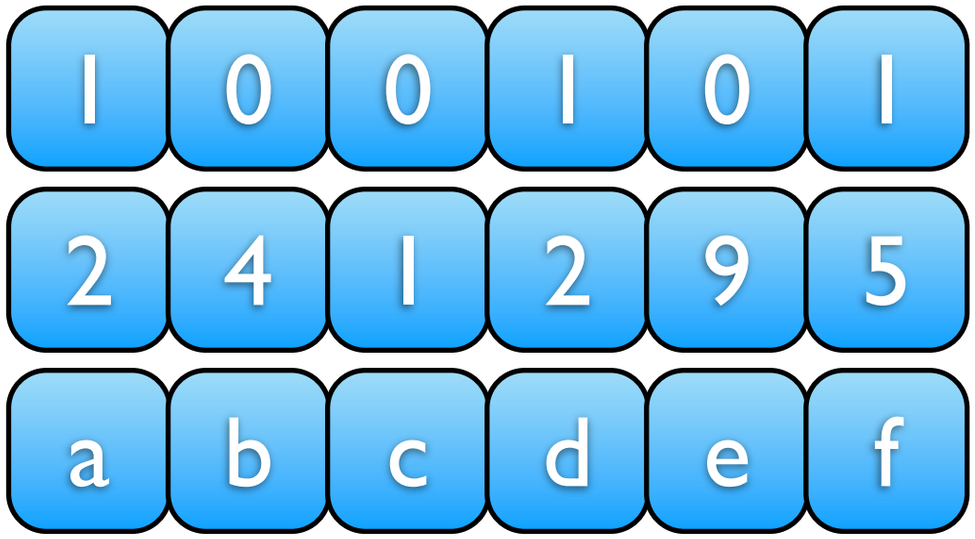
Encoding

- Binary encoding
- Number encoding
- Any encoding really...



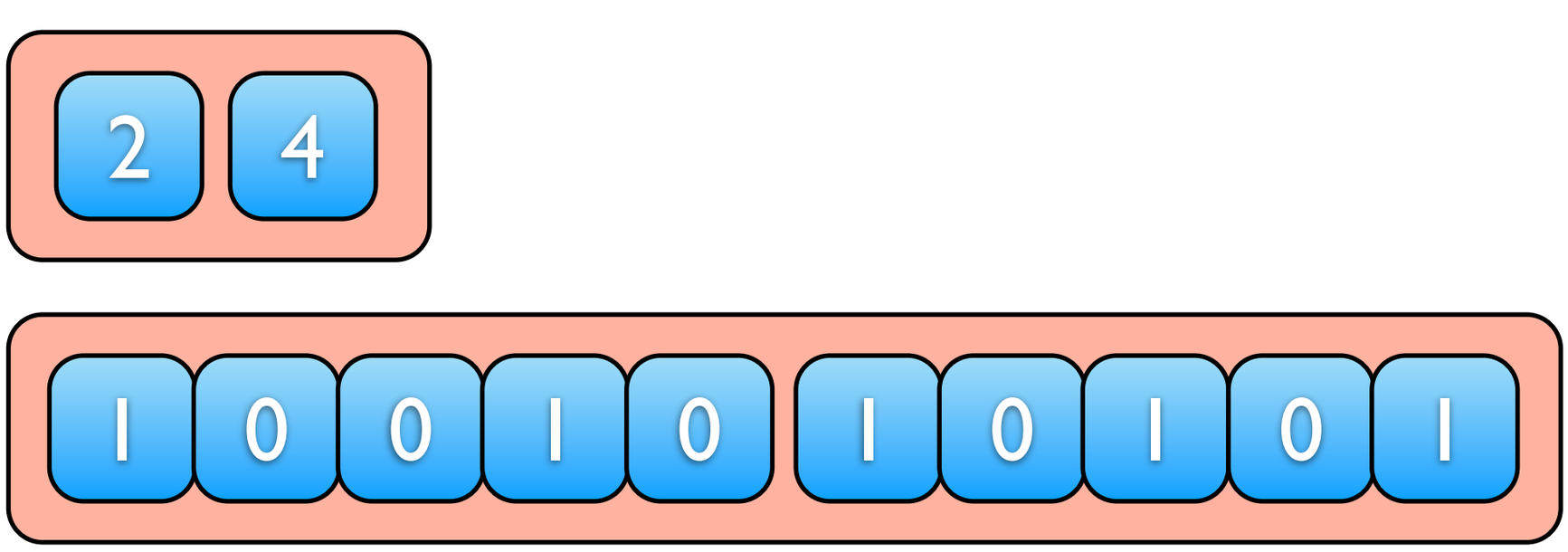
Encoding

- Binary encoding
- Number encoding
- Any encoding really...
- Encoding is problem specific



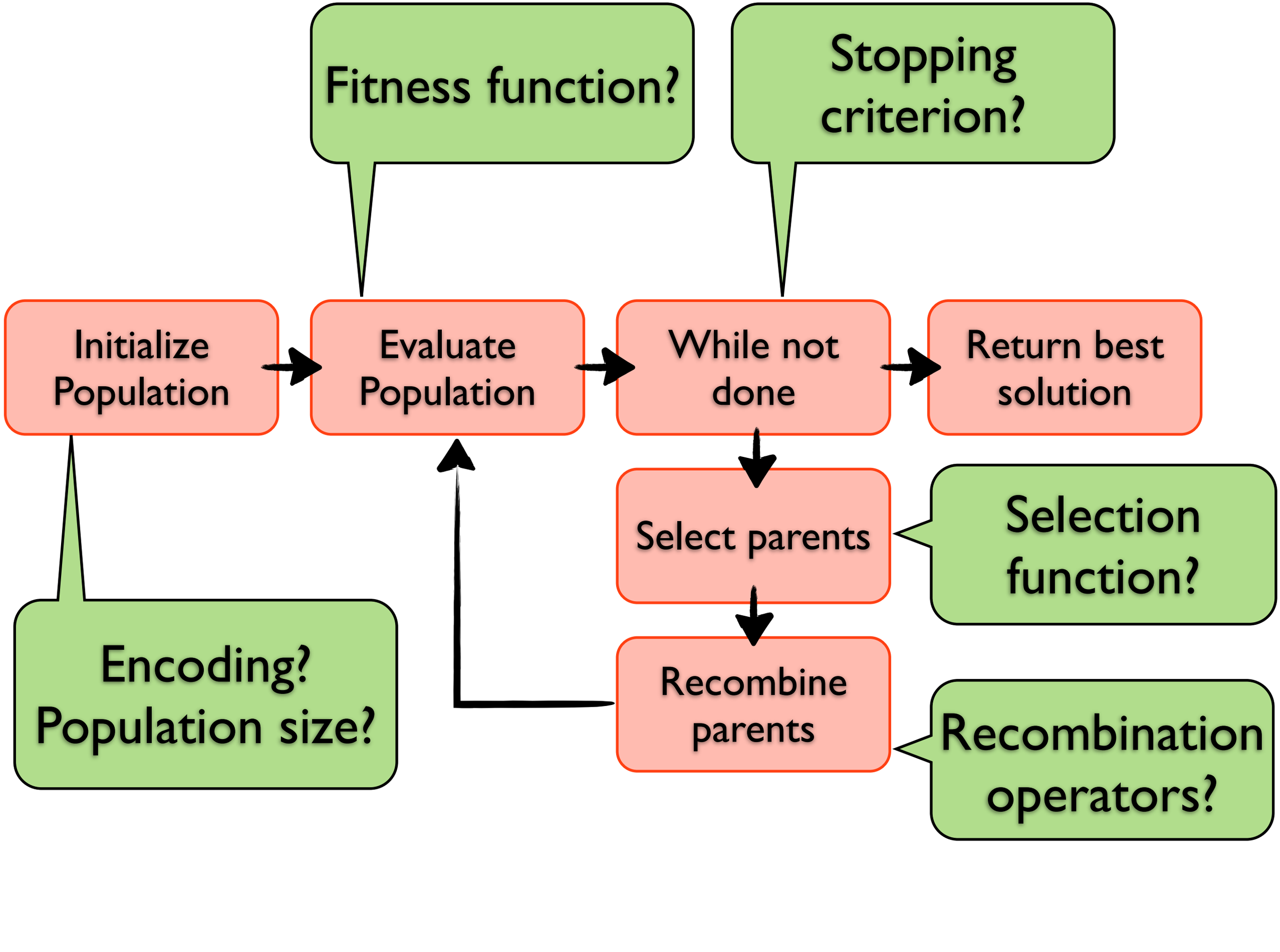
Encoding

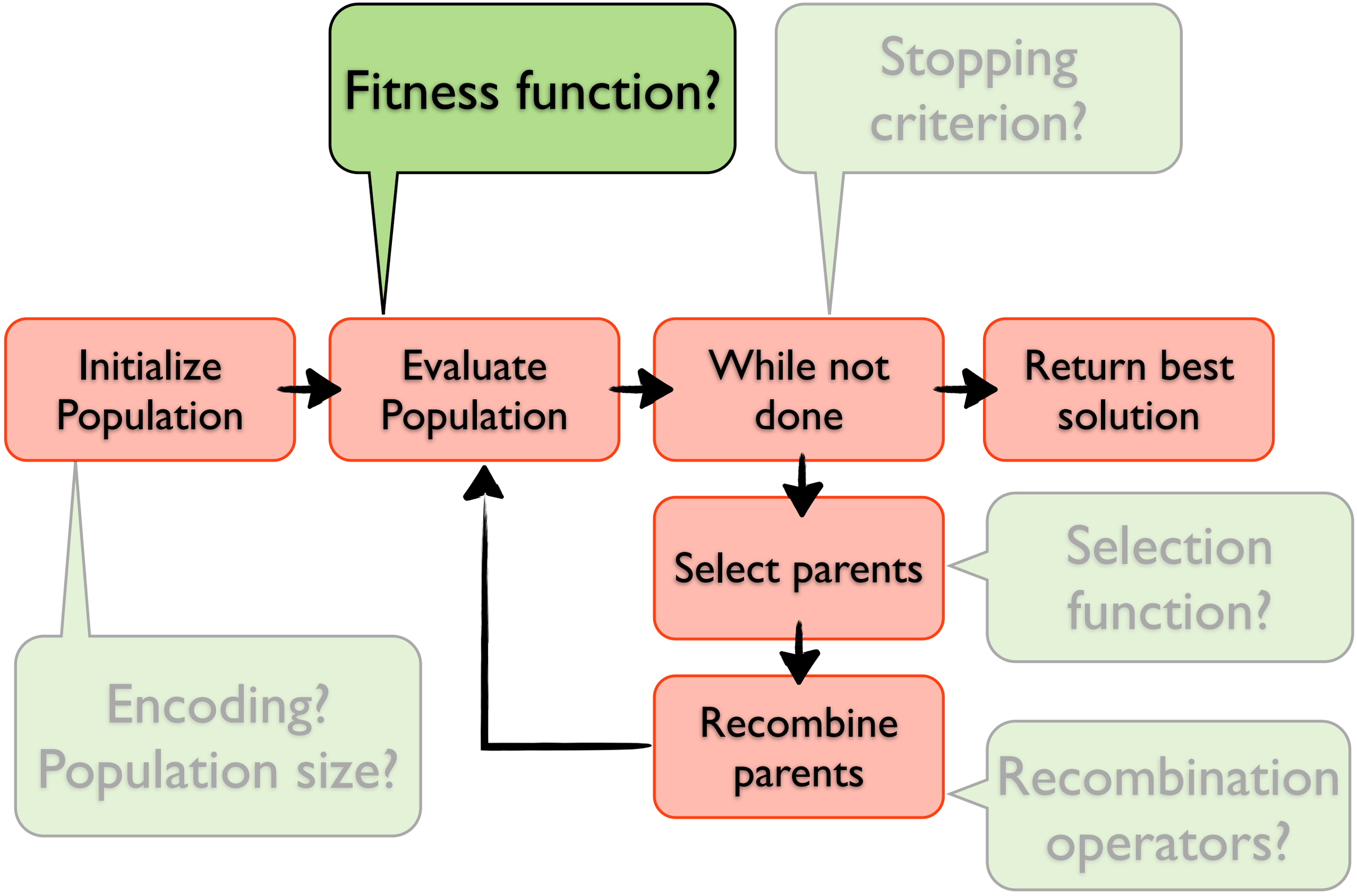
- Search space: (x,y)



Initial Population

- Randomly chosen
- Existing solutions
- Coverage test suite for weaker criterion
- Manually generated test cases


Fitness function?
Stopping criterion?
Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Selection function?
Encoding? Population size?
Recombine parents
Recombination operators?

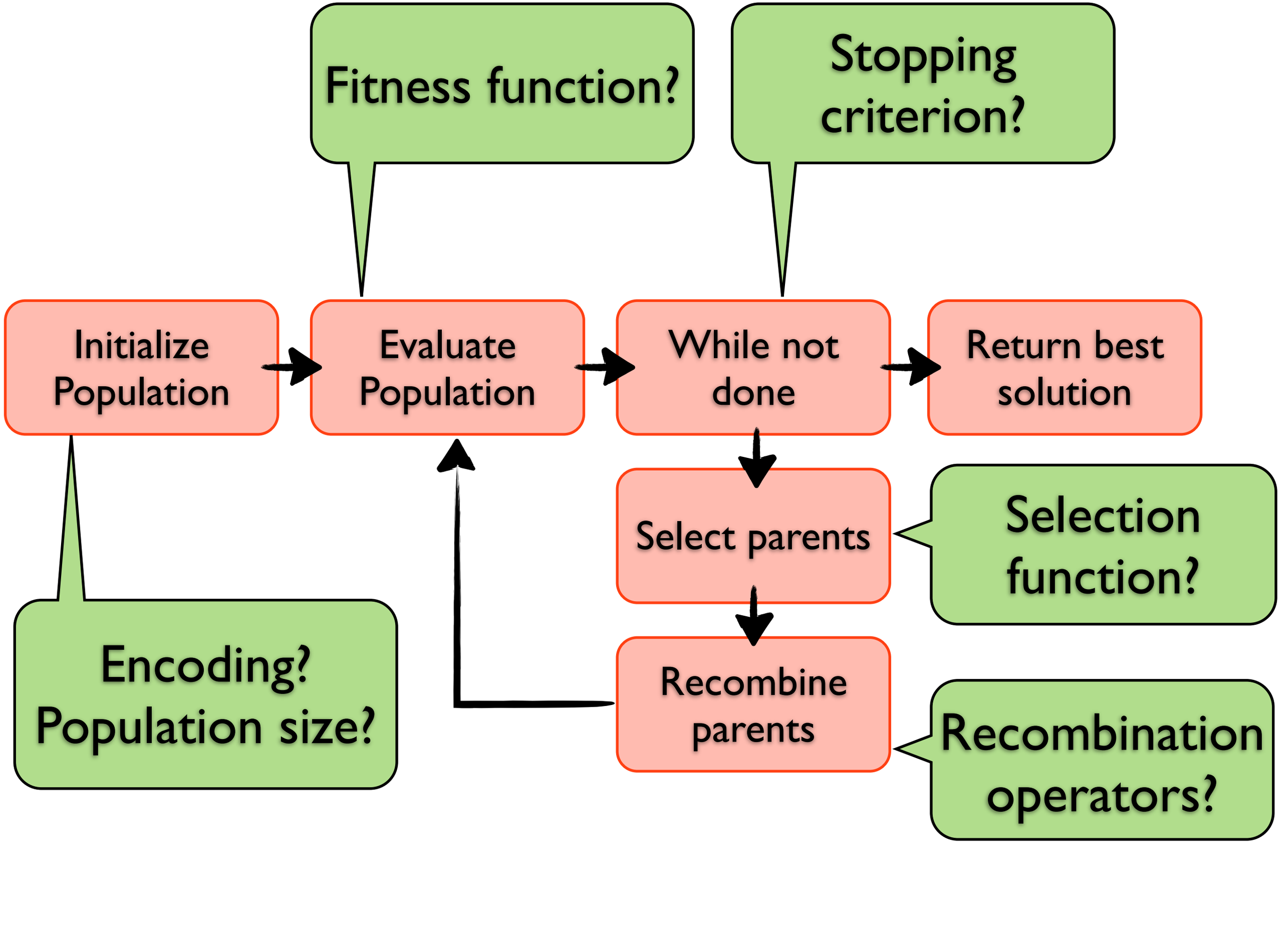

Fitness function?
Stopping criterion?
Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Selection function?
Encoding?
Population size?
Recombine parents
Recombination operators?

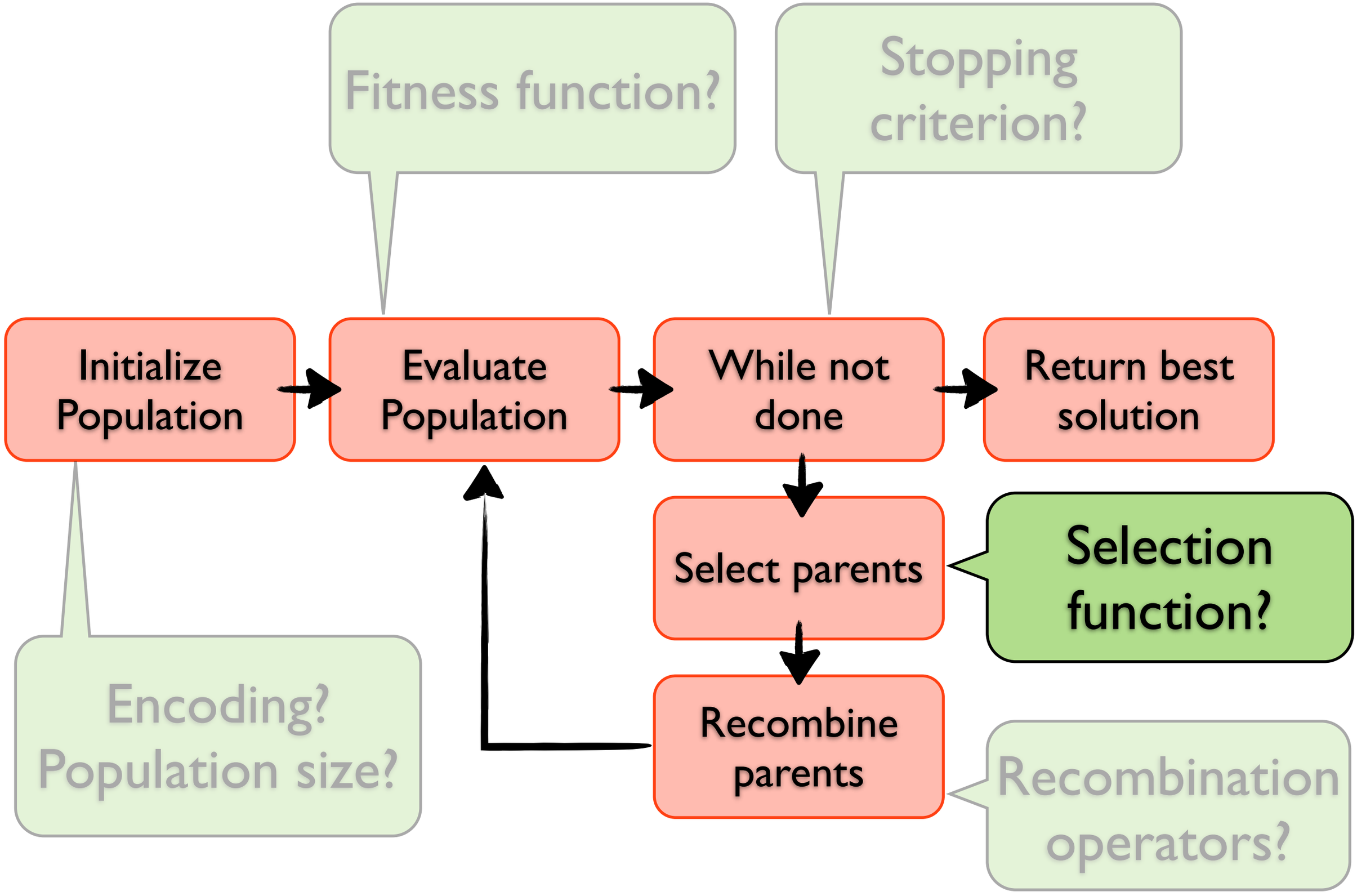
Fitness

- How good is a candidate solution?
- Is solution A better than solution B?
- Fitness quantifies the "goodness" of a solution
- Fitness is problem specific
- Fitness function determines the search landscape

Kin Compensation

- Diversity in the population is key to successful search
- Kin compensation:
  - Fitness penalty if identical individual already exists


Fitness function?
Stopping criterion?
Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Selection function?
Encoding? Population size?
Recombine parents
Recombination operators?

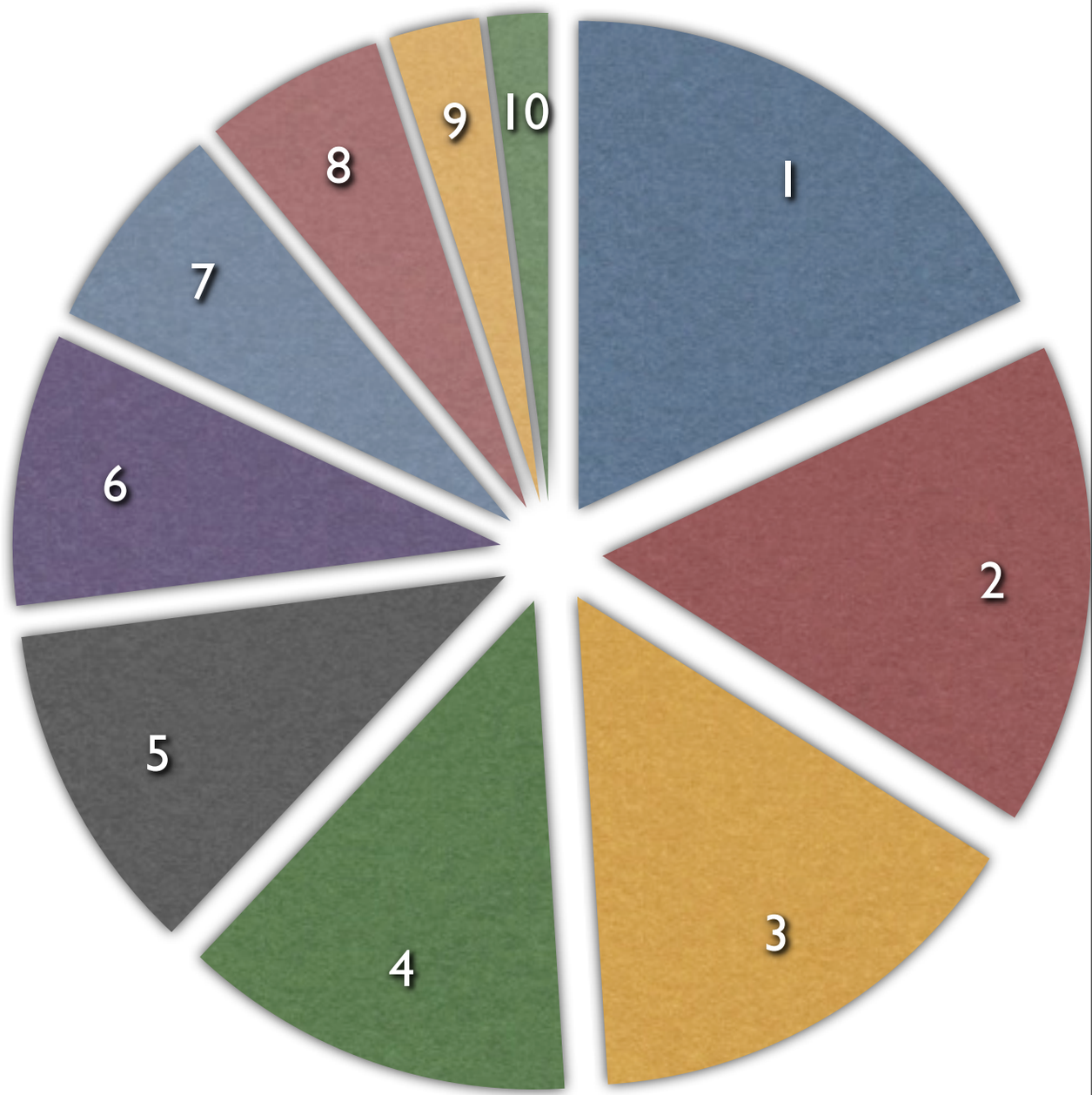

Fitness function?
Stopping criterion?
Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Selection function?
Encoding?
Population size?
Recombine parents
Recombination operators?

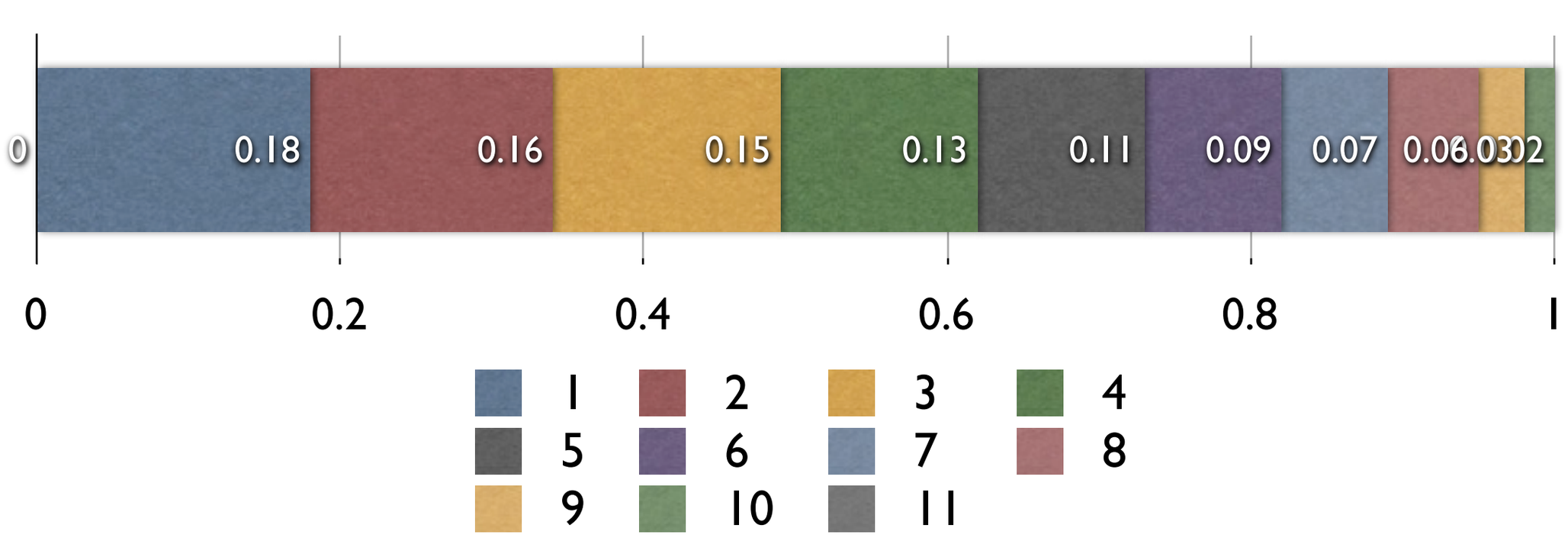
Fitness Proportionate Selection

- A.k.a. Roulette wheel selection
- Probability of choosing an individual is proportional to its fitness

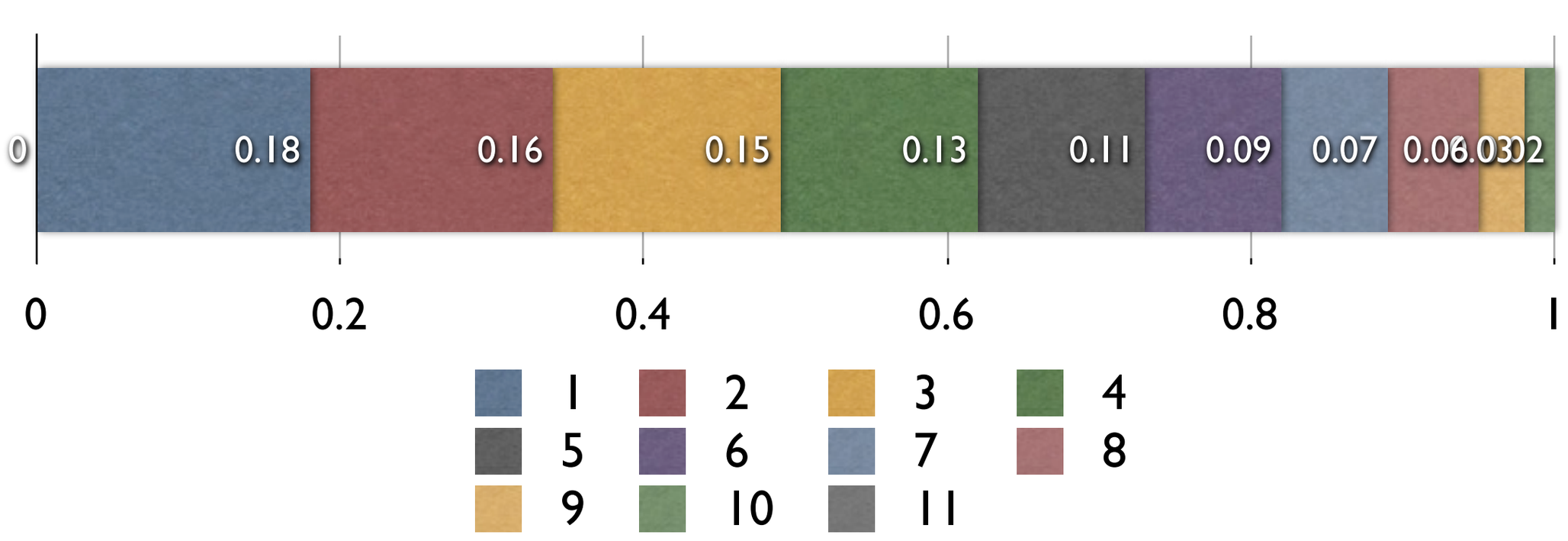
Individual	Fitness
1	2.0
2	1.8
3	1.6
4	1.4
5	1.2
6	1.0
7	0.8
8	0.6
9	0.4
10	0.2
11	0.0

Individual	Fitness
1	2.0
2	1.8
3	1.6
4	1.4
5	1.2
6	1.0
7	0.8
8	0.6
9	0.4
10	0.2
11	0.0

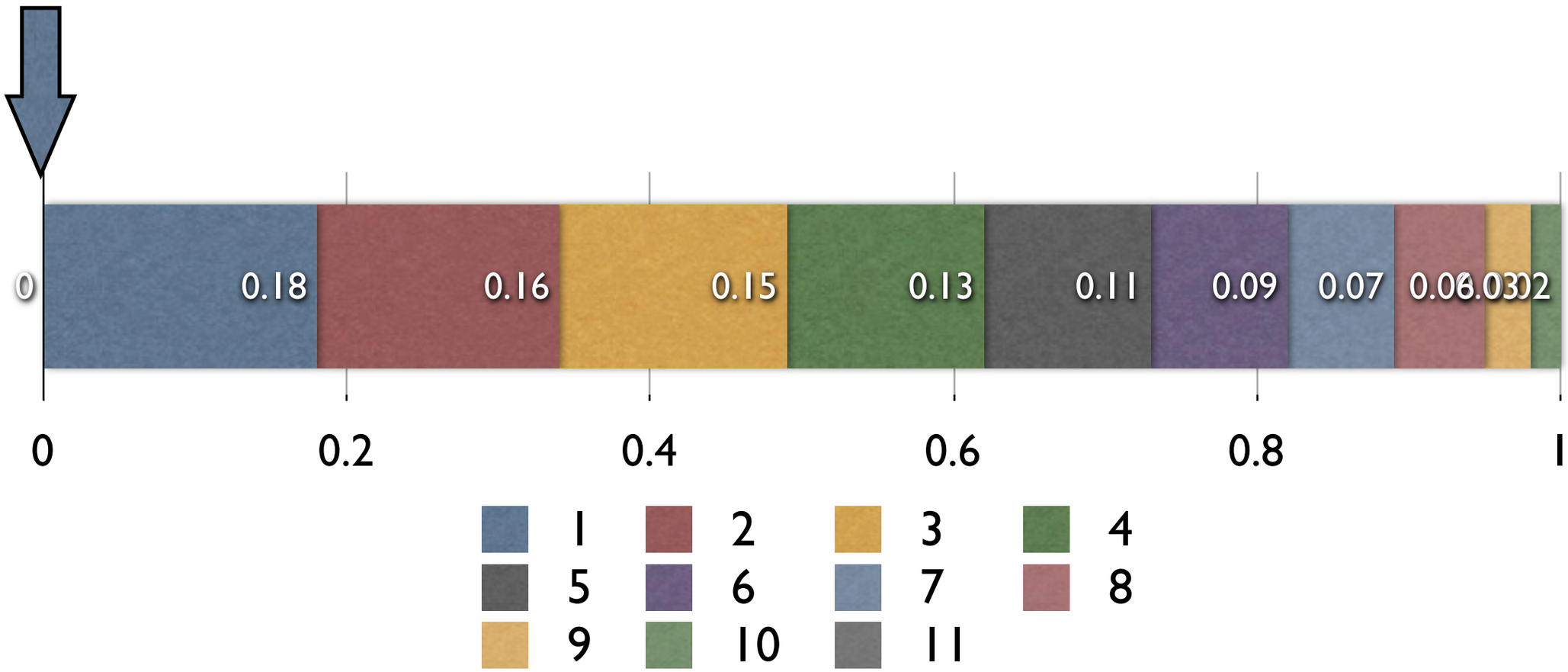



0
0.18
0.16
0.15
0.13
0.11
0.09
0.07
0.06
0.03
0.02
0
0.2
0.4
0.6
0.8
1
1
2
3
4
5
6
7
8
9
10
11

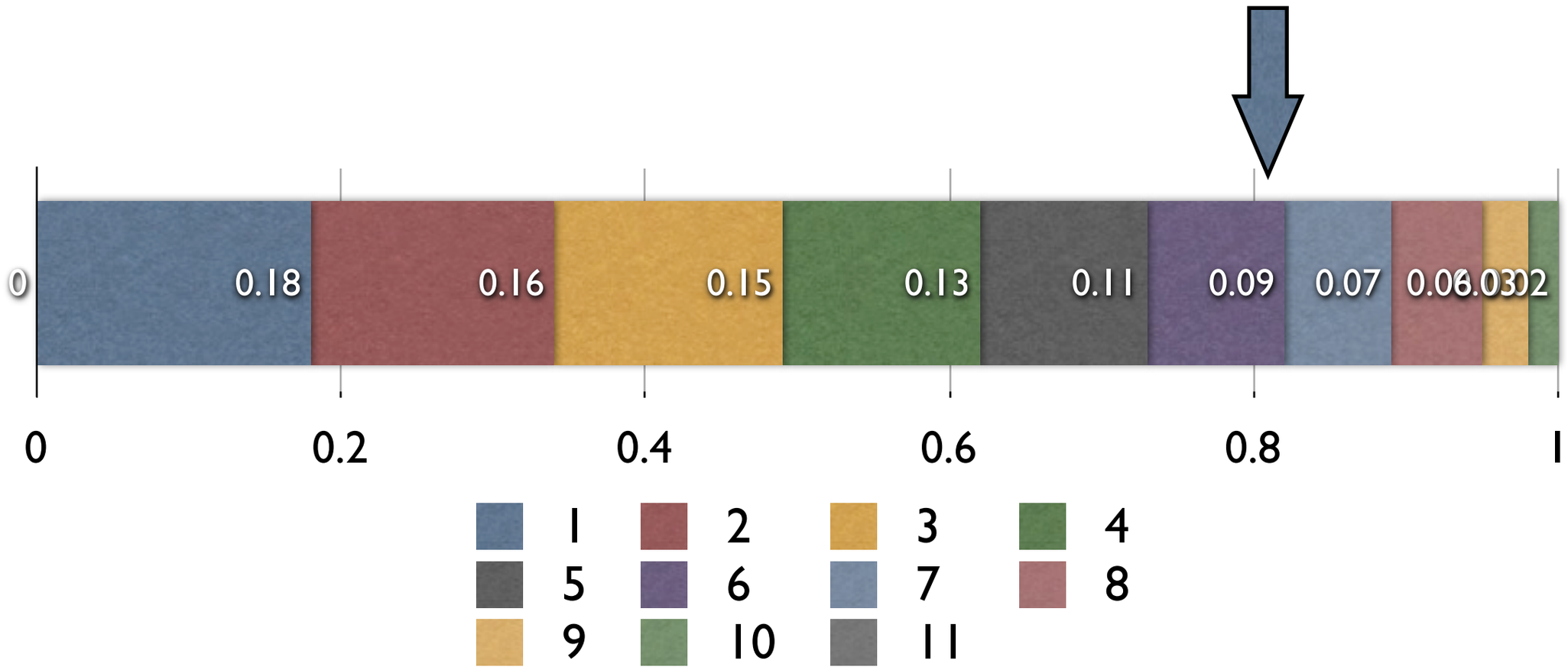
- Chosen value: 0,81



- Chosen value: 0,81

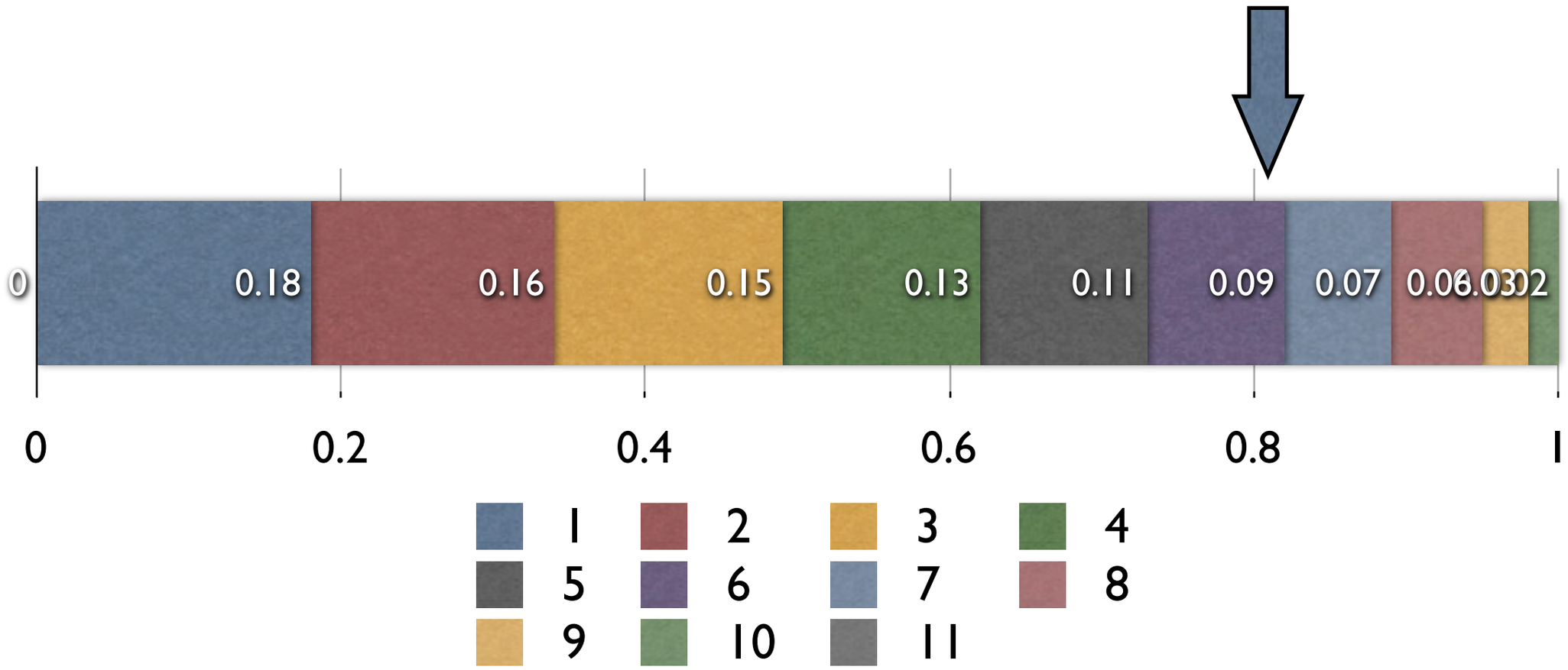


- Chosen value: 0,81



- Chosen value: 0,81

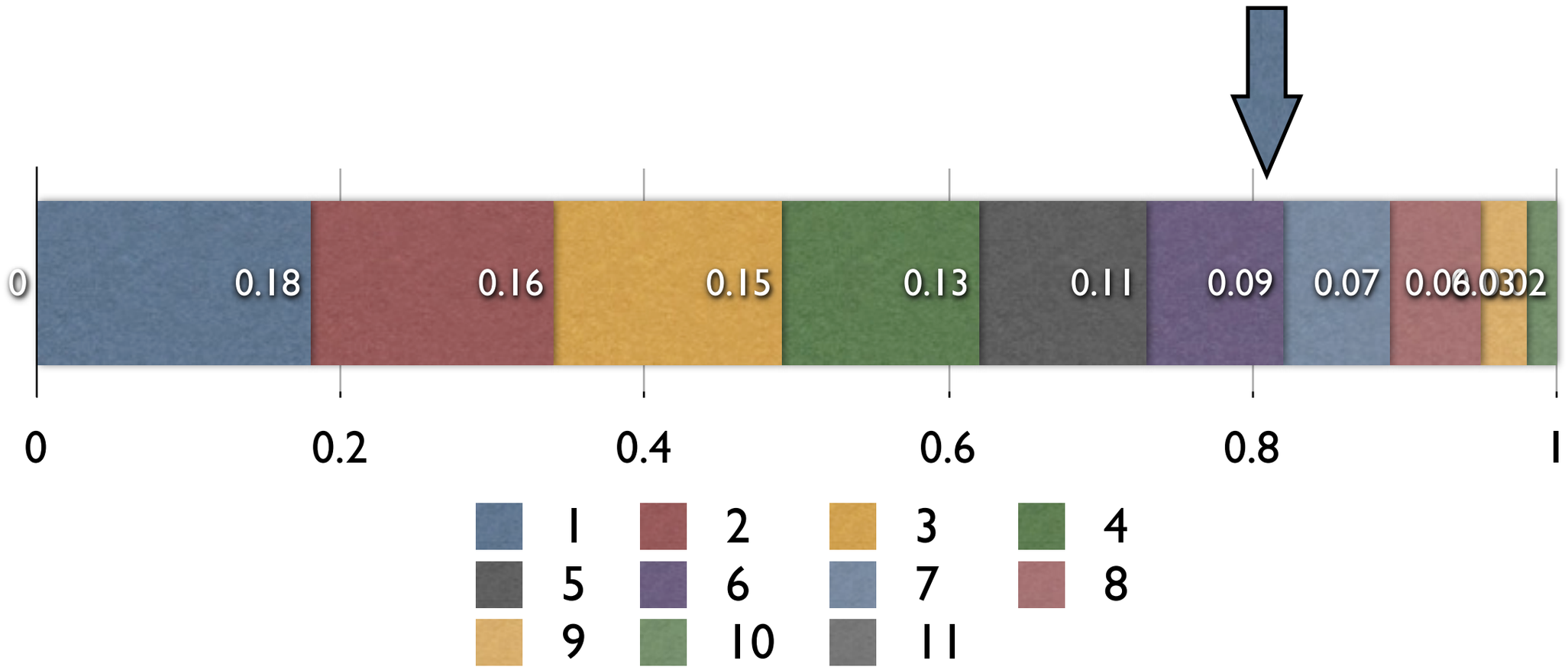
6



- Chosen value: 0,81

6

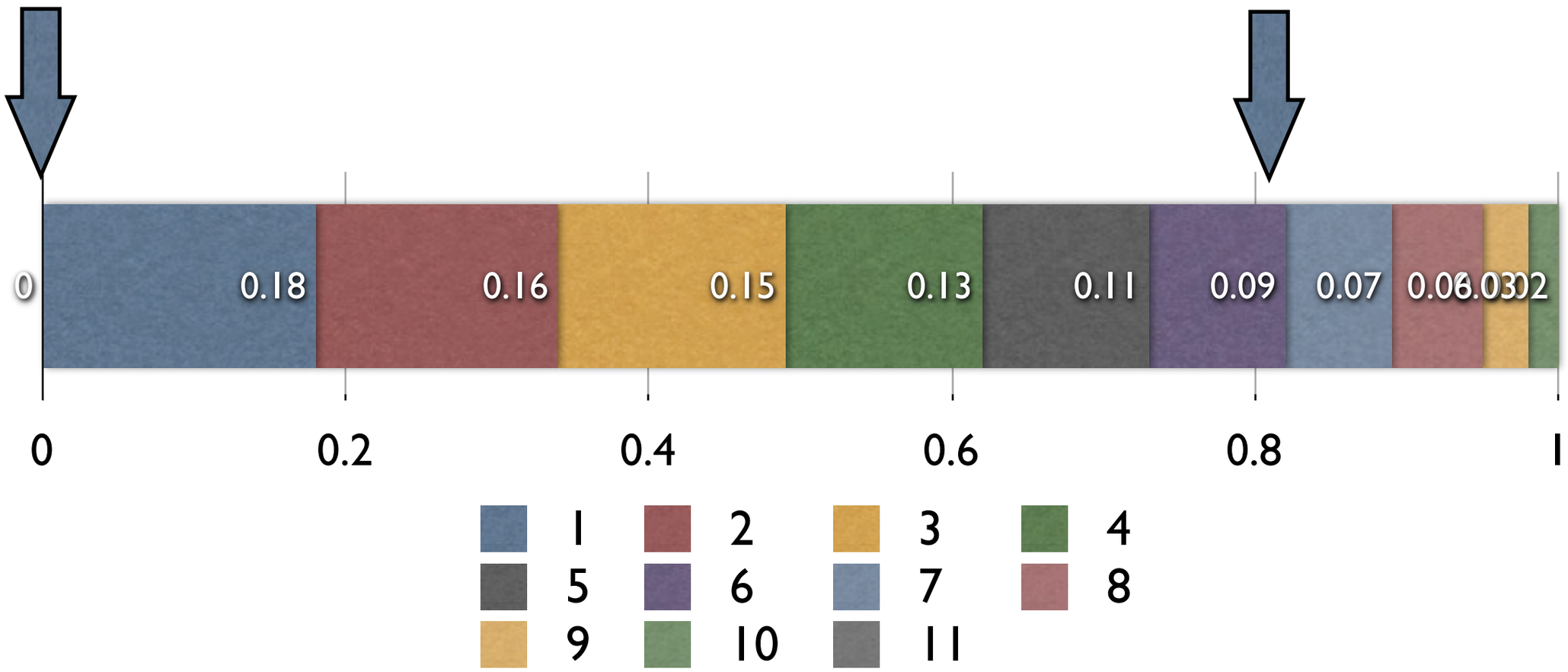
- Chosen value: 0,32



- Chosen value: 0,81

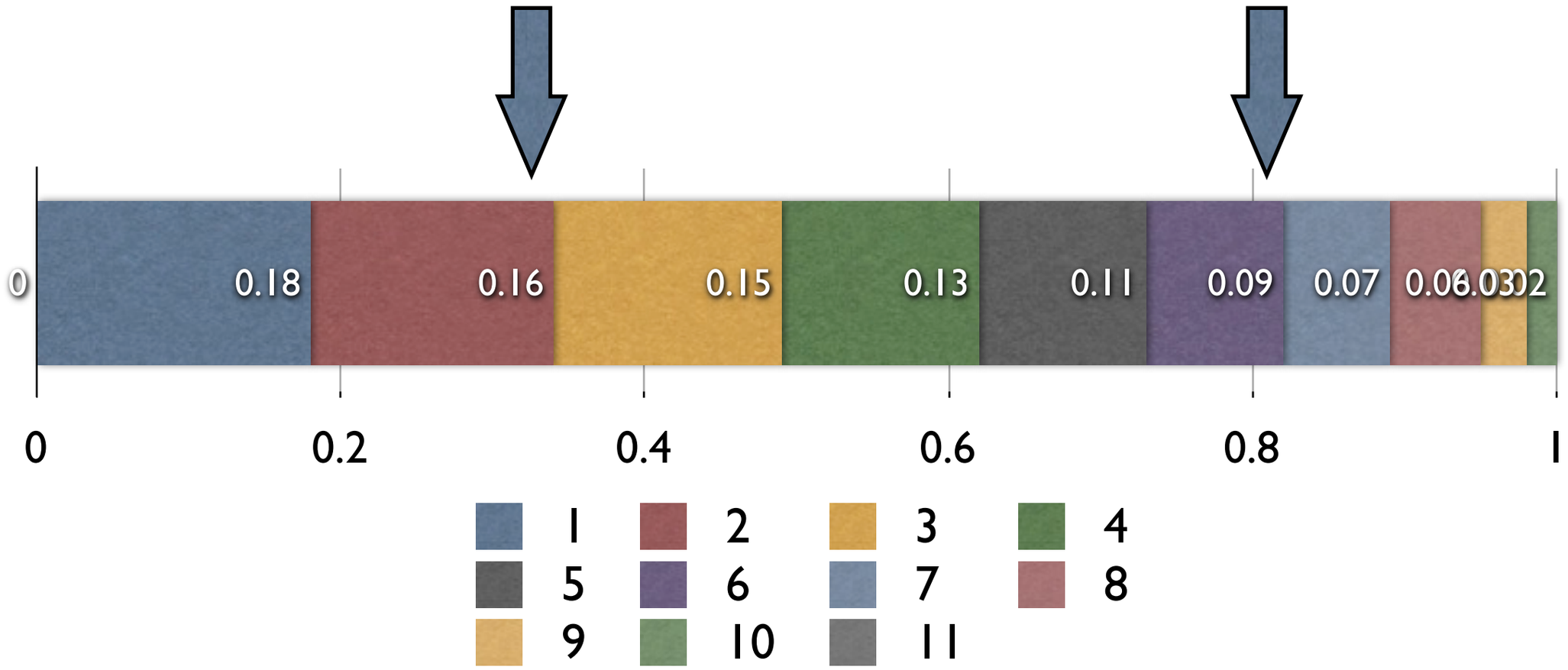
6

- Chosen value: 0,32

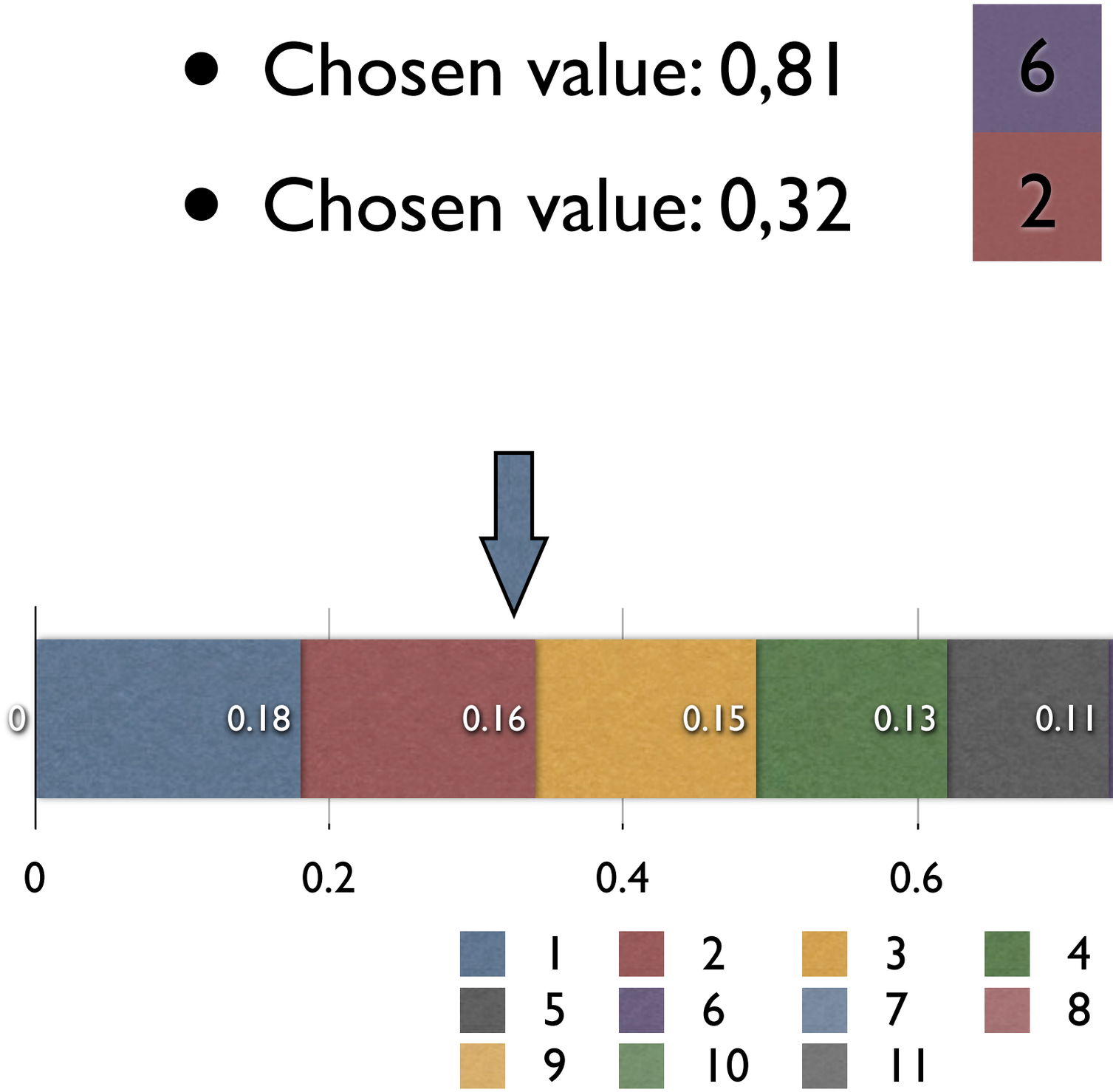


- Chosen value: 0,81
- Chosen value: 0,32

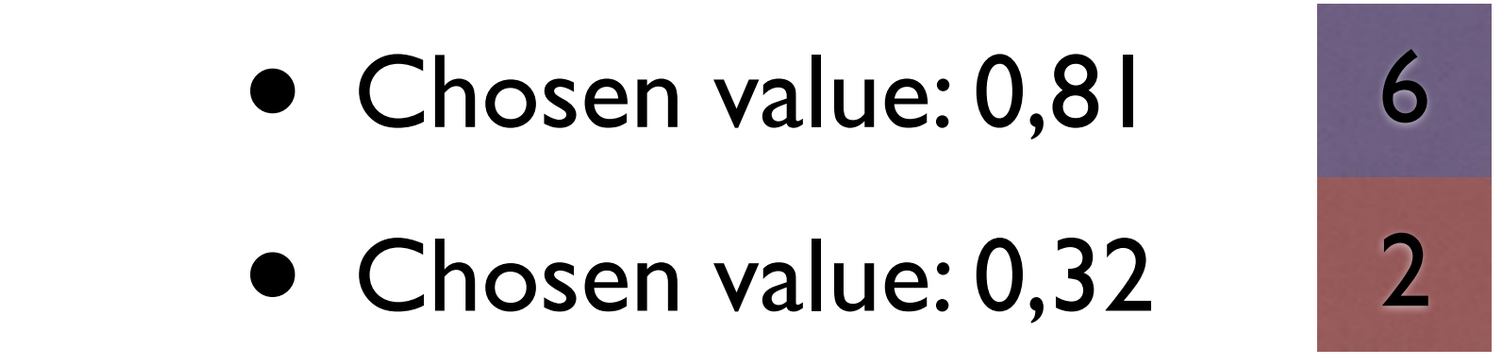
6

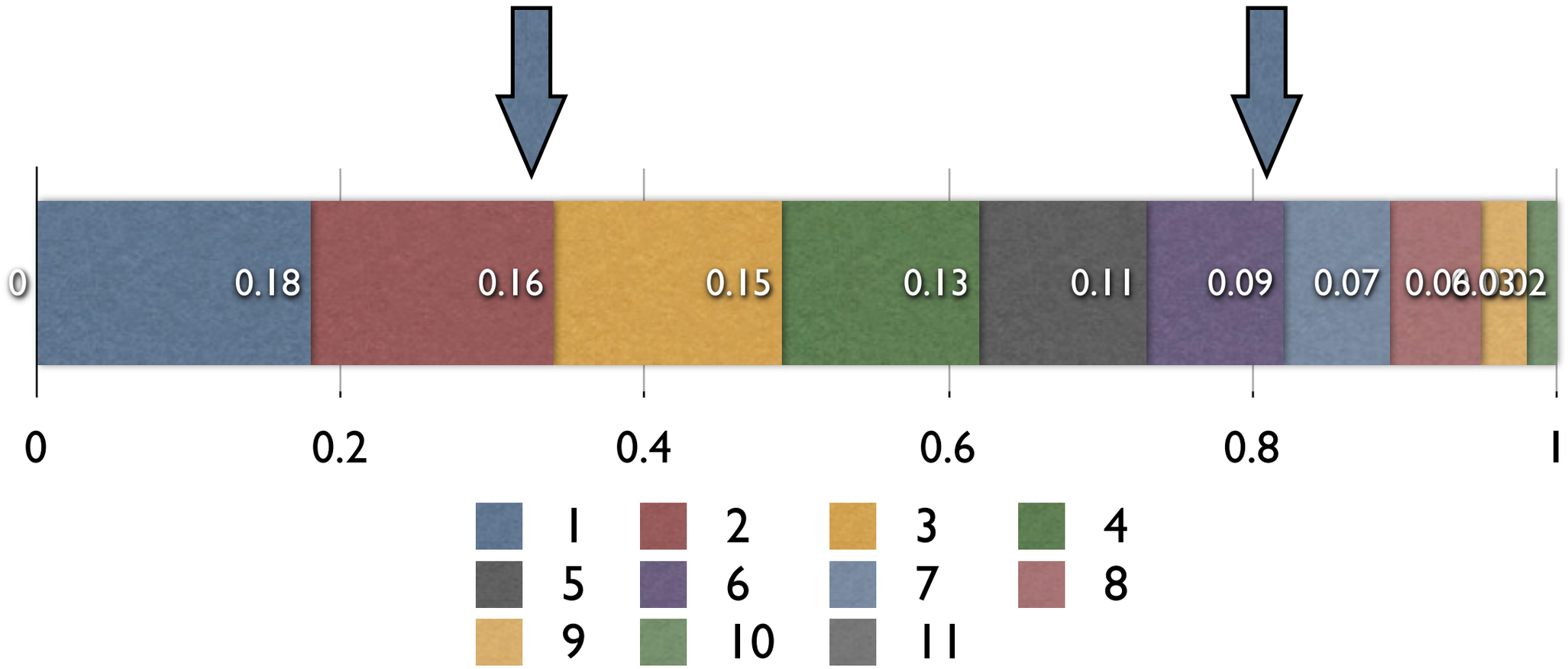


- Chosen value: 0,81
- Chosen value: 0,32

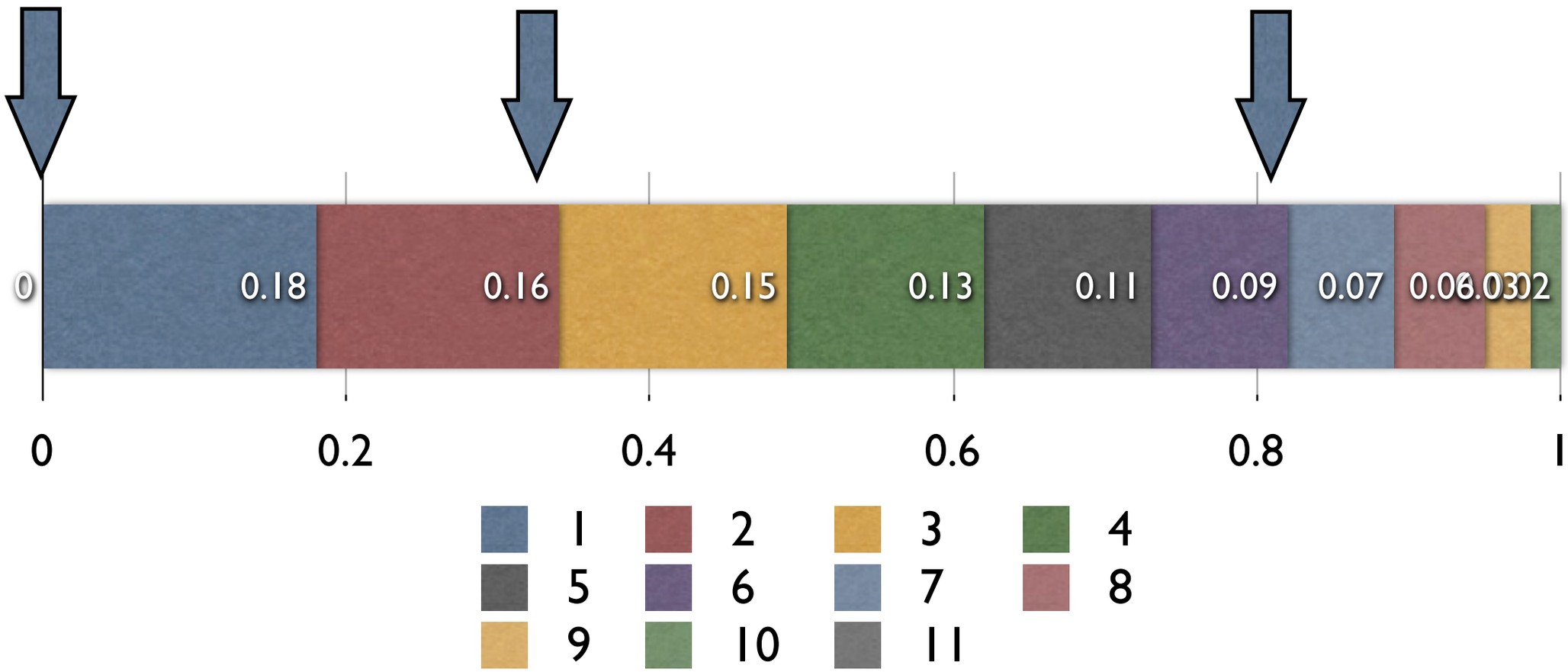


- Chosen value: 0,81
- Chosen value: 0,32
- Chosen value: 0,01

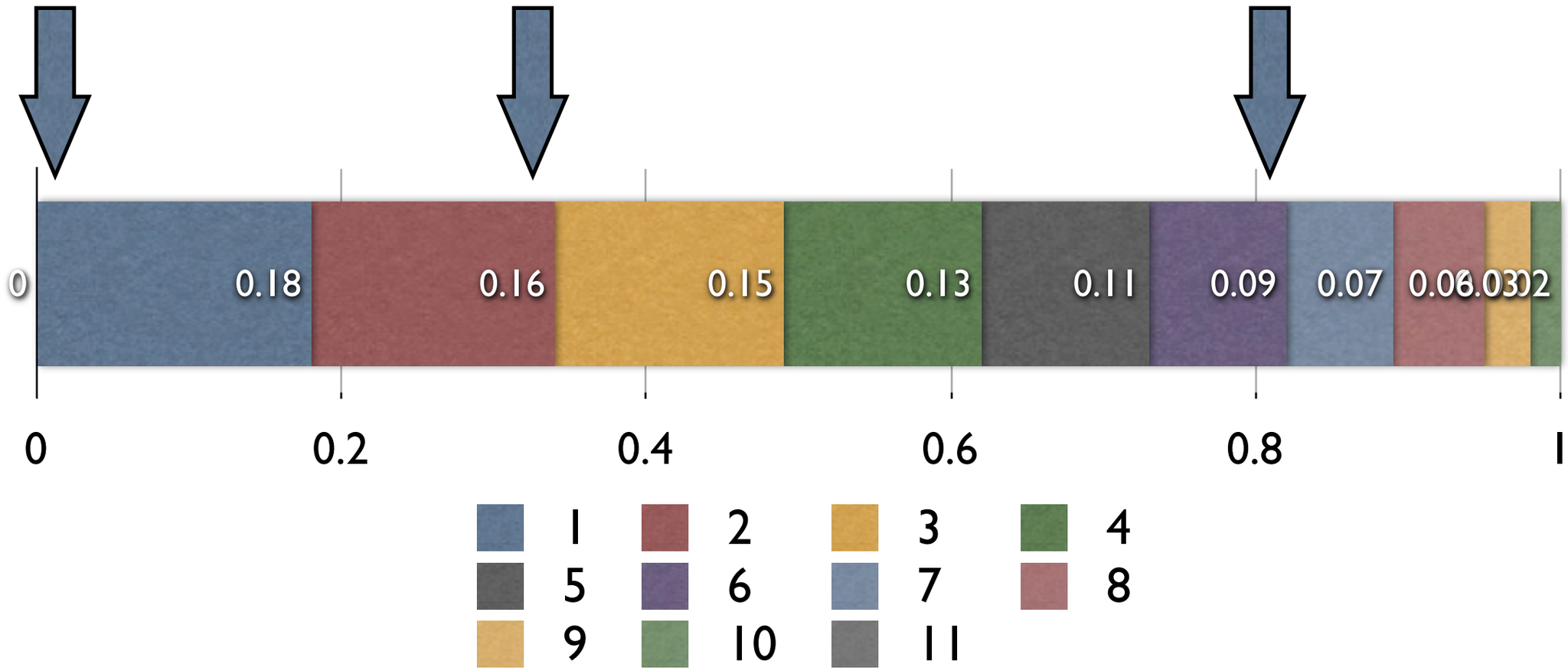




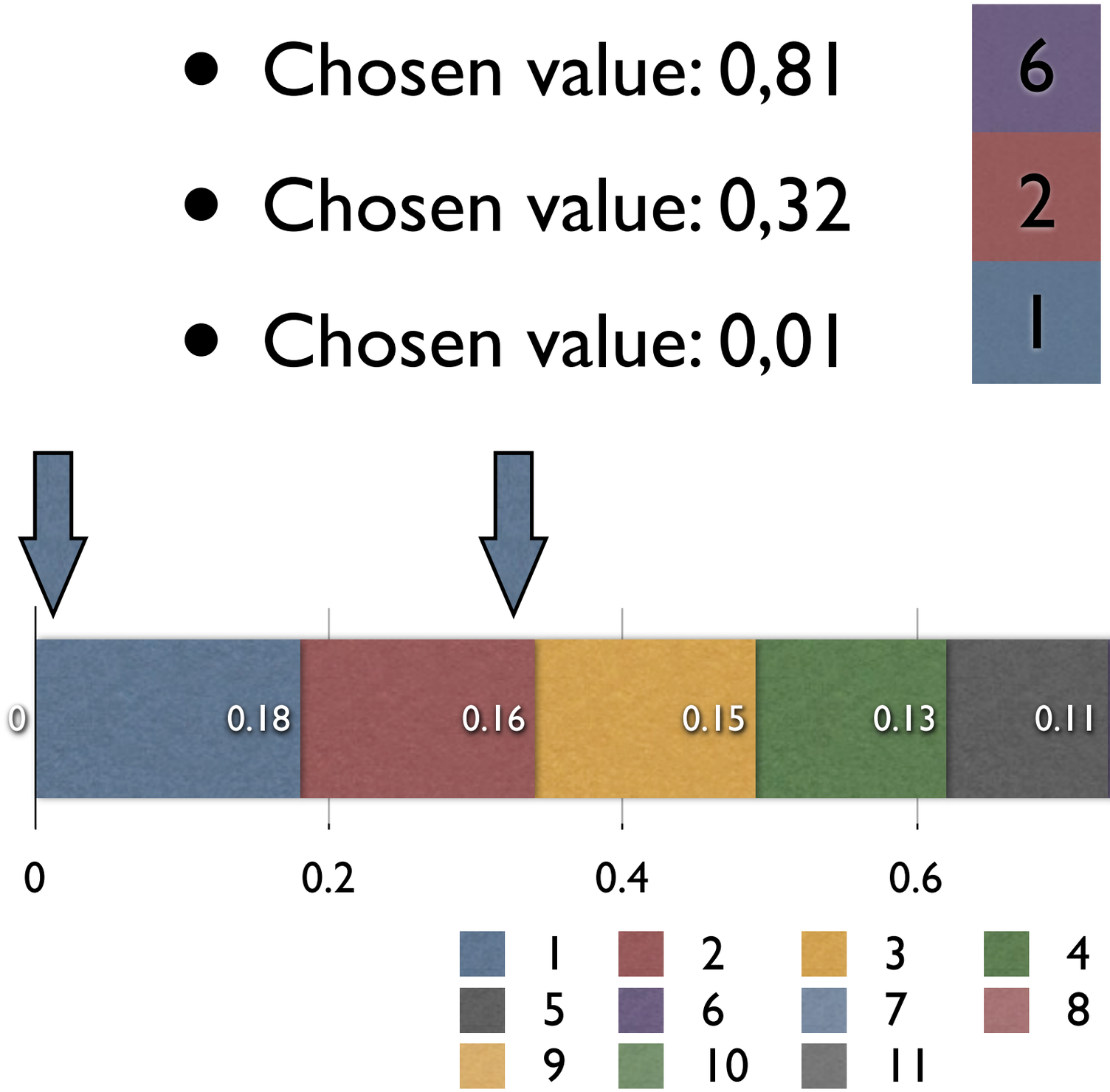
- Chosen value: 0,81
- Chosen value: 0,32
- Chosen value: 0,01

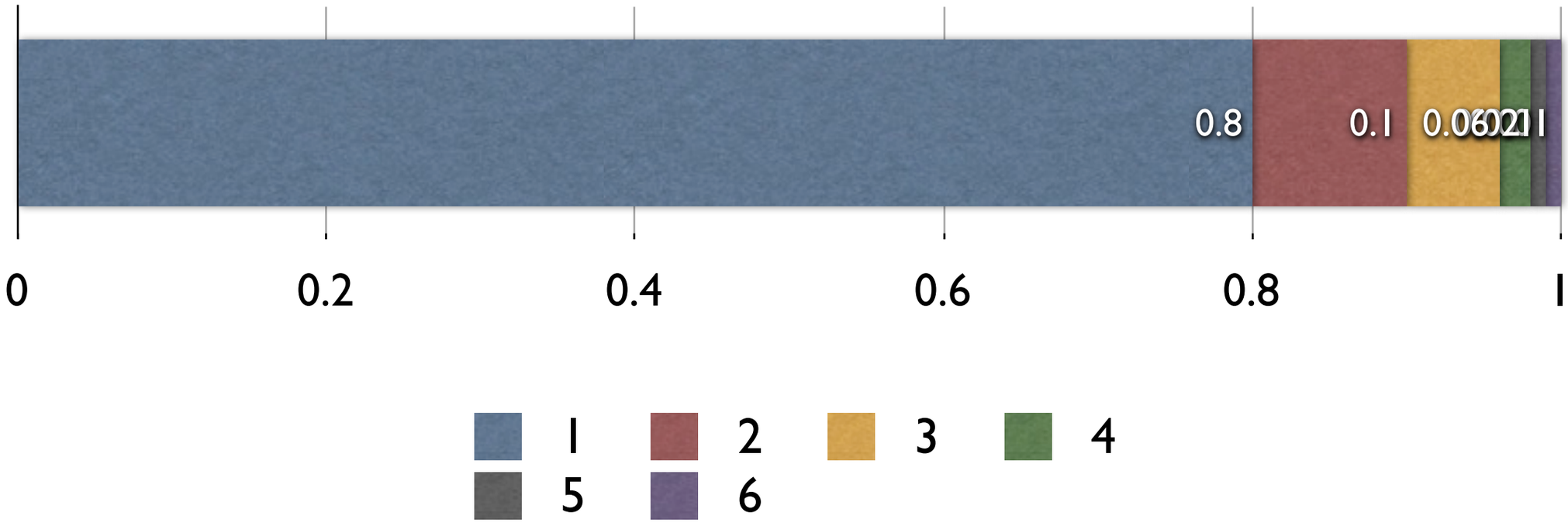


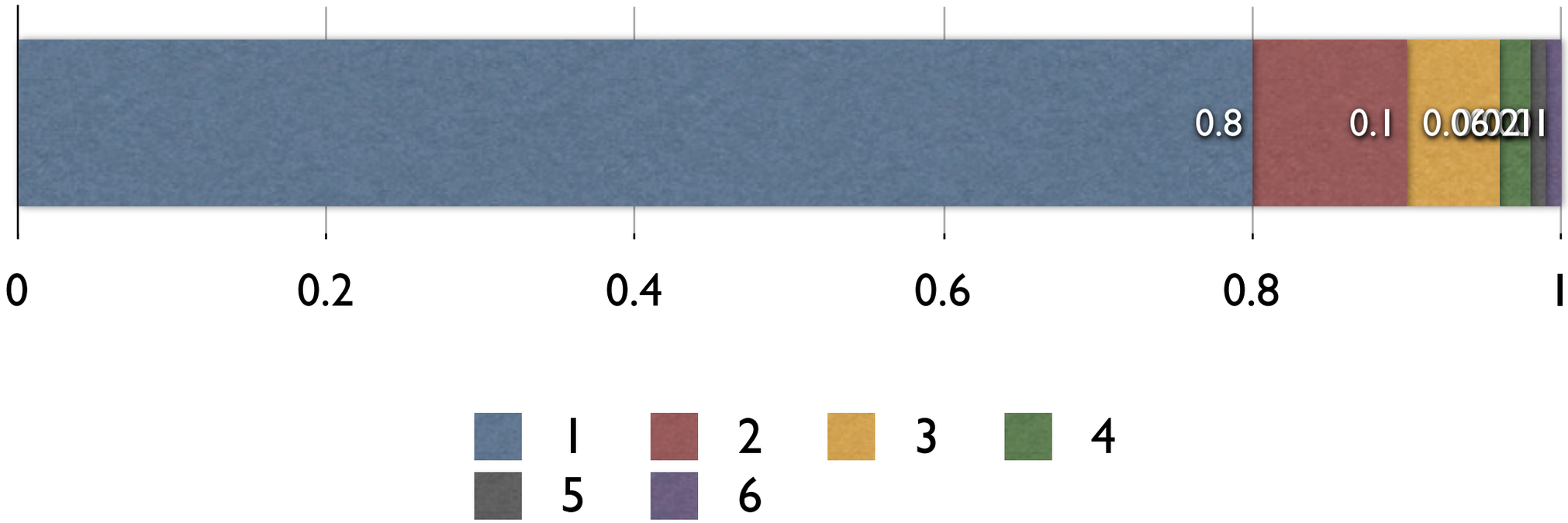
- Chosen value: 0,81
- Chosen value: 0,32
- Chosen value: 0,01



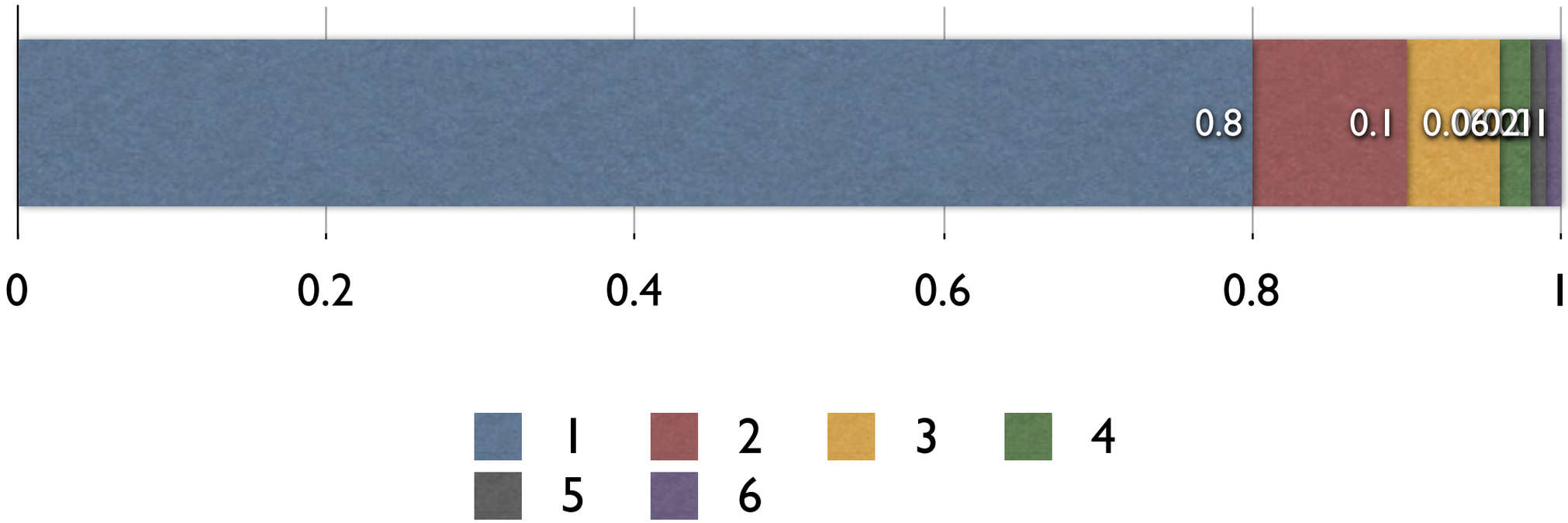
- Chosen value: 0,81
- Chosen value: 0,32
- Chosen value: 0,01




0.8
0.1
0.06
0.02
0.01
0
0.2
0.4
0.6
0.8
1
1
2
3
4
5
6



- Individual 1 will dominate selection



- Individual 1 will dominate selection
- **Don’t use roulette wheel selection!**

Problems with Selection

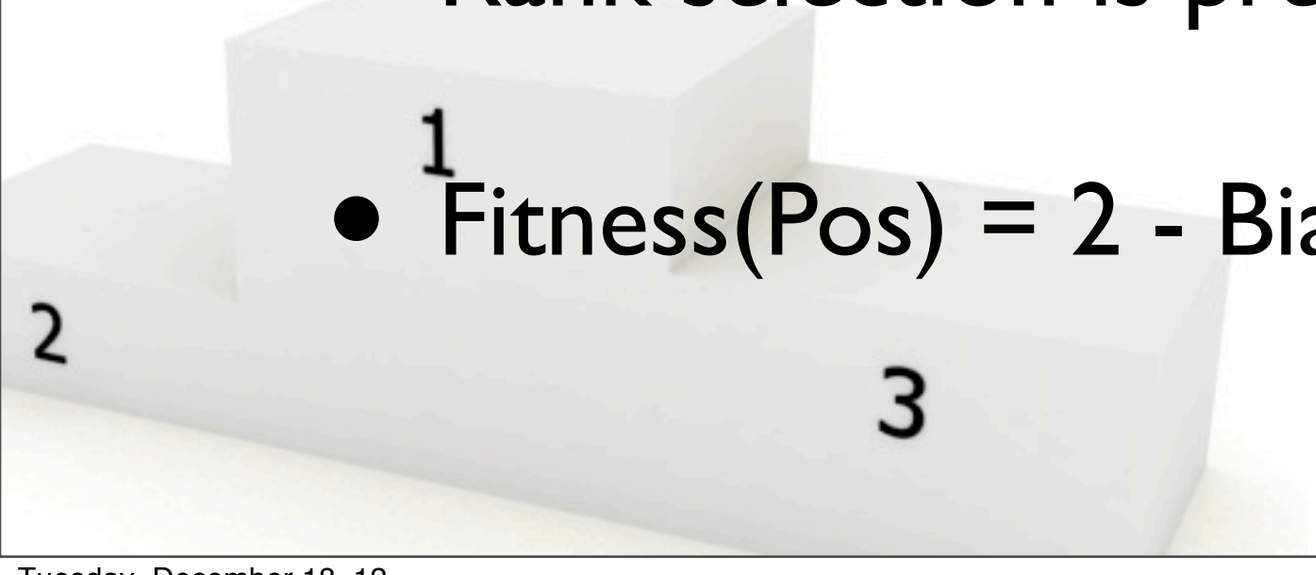
- Selective pressure:
  The higher, the more likely the fittest are chosen

- Stagnation:
  Selective pressure too small

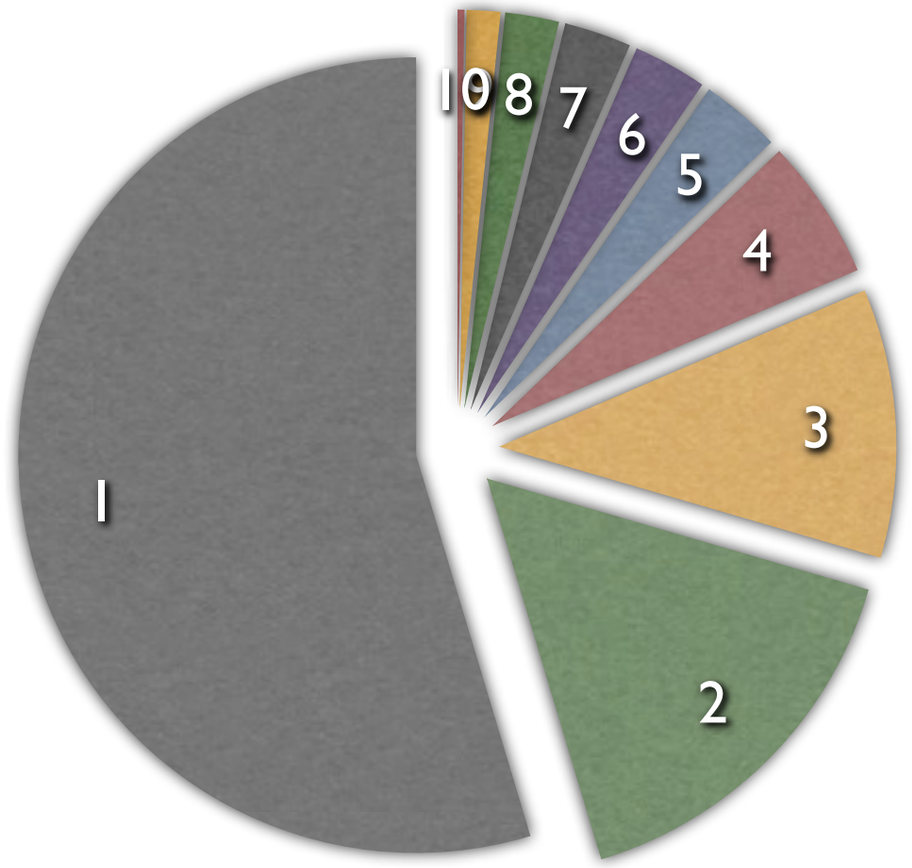
- Premature convergence:
  Selective pressure too high

Rank Selection

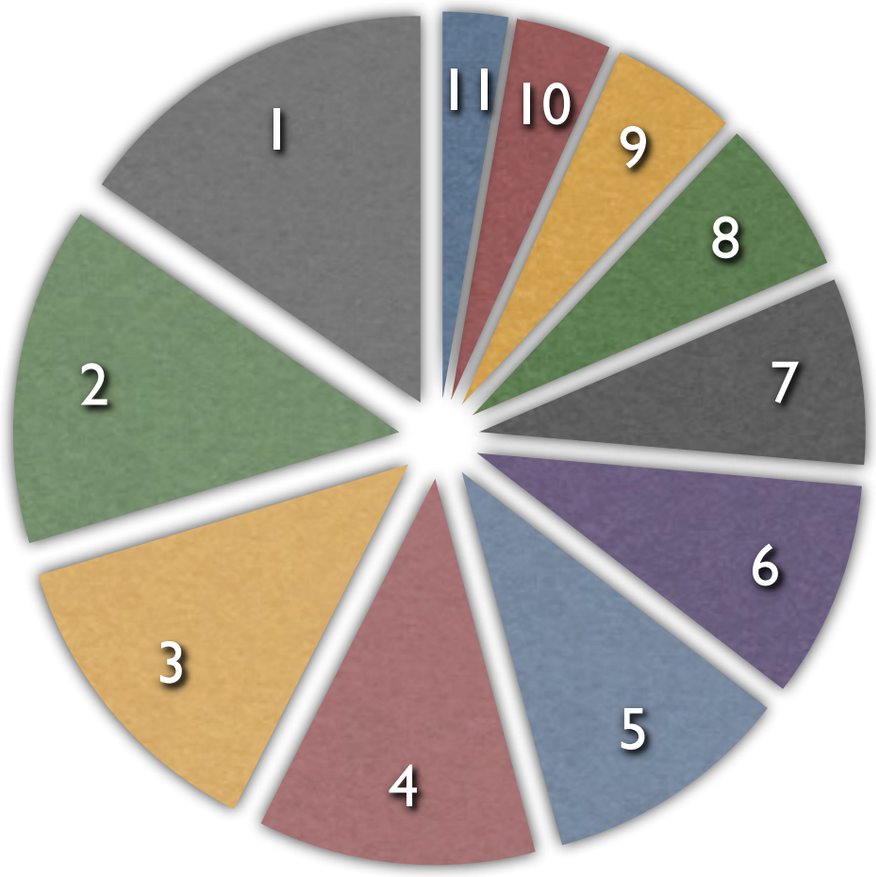
- Rank individuals according to their fitness
- No difference whether fittest candidate is ten times fitter than the next fittest or 0.001% fitter
- Rank selection is preferable in practice
- $\text{Fitness}(\text{Pos}) = 2 - \text{Bias} + 2*(\text{Bias} - 1)*\frac{\text{Pos} - 1}{\text{Num} - 1}$



Individual	Fitness
1	2.0
2	0.58
3	0.4
4	0.21
5	0.12
6	0.11
7	0.1
8	0.08
9	0.05
10	0.01
11	0.0



Individual	Fitness
1	2.0
2	0.58
3	0.4
4	0.21
5	0.12
6	0.11
7	0.1
8	0.08
9	0.05
10	0.01
11	0.0



Tournament Selection

- N = Tournament size
- Select N individuals randomly
- Best of the N individuals is selected
- Tournament size defines selective pressure
- A worse individual can win with a given probability

Individual	Fitness
1	2.0
2	1.8
3	1.6
4	1.4
5	1.2
6	1.0
7	0.8
8	0.6
9	0.4
10	0.2
11	0.0

Individual	Fitness
1	2.0
2	1.8
3	1.6
4	1.4
5	1.2
6	1.0
7	0.8
8	0.6
9	0.4
10	0.2
11	0.0

Tournament size: 4

Individual	Fitness
1	2.0
2	1.8
3	1.6
4	1.4
5	1.2
6	1.0
7	0.8
8	0.6
9	0.4
10	0.2
11	0.0

Tournament size: 4

4

6

2

8

Individual	Fitness
1	2.0
2	1.8
3	1.6
4	1.4
5	1.2
6	1.0
7	0.8
8	0.6
9	0.4
10	0.2
11	0.0

Tournament size: 4

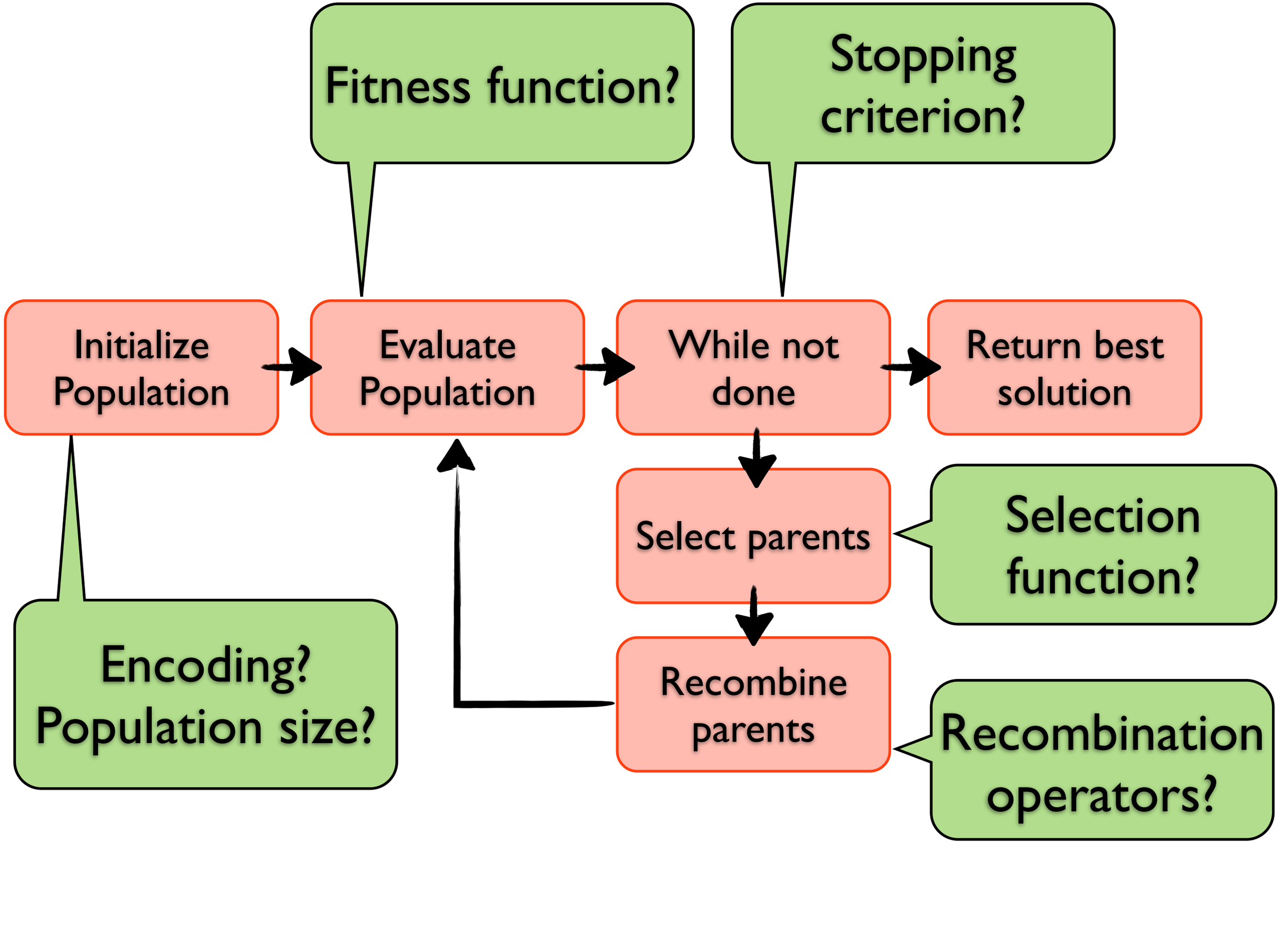
4

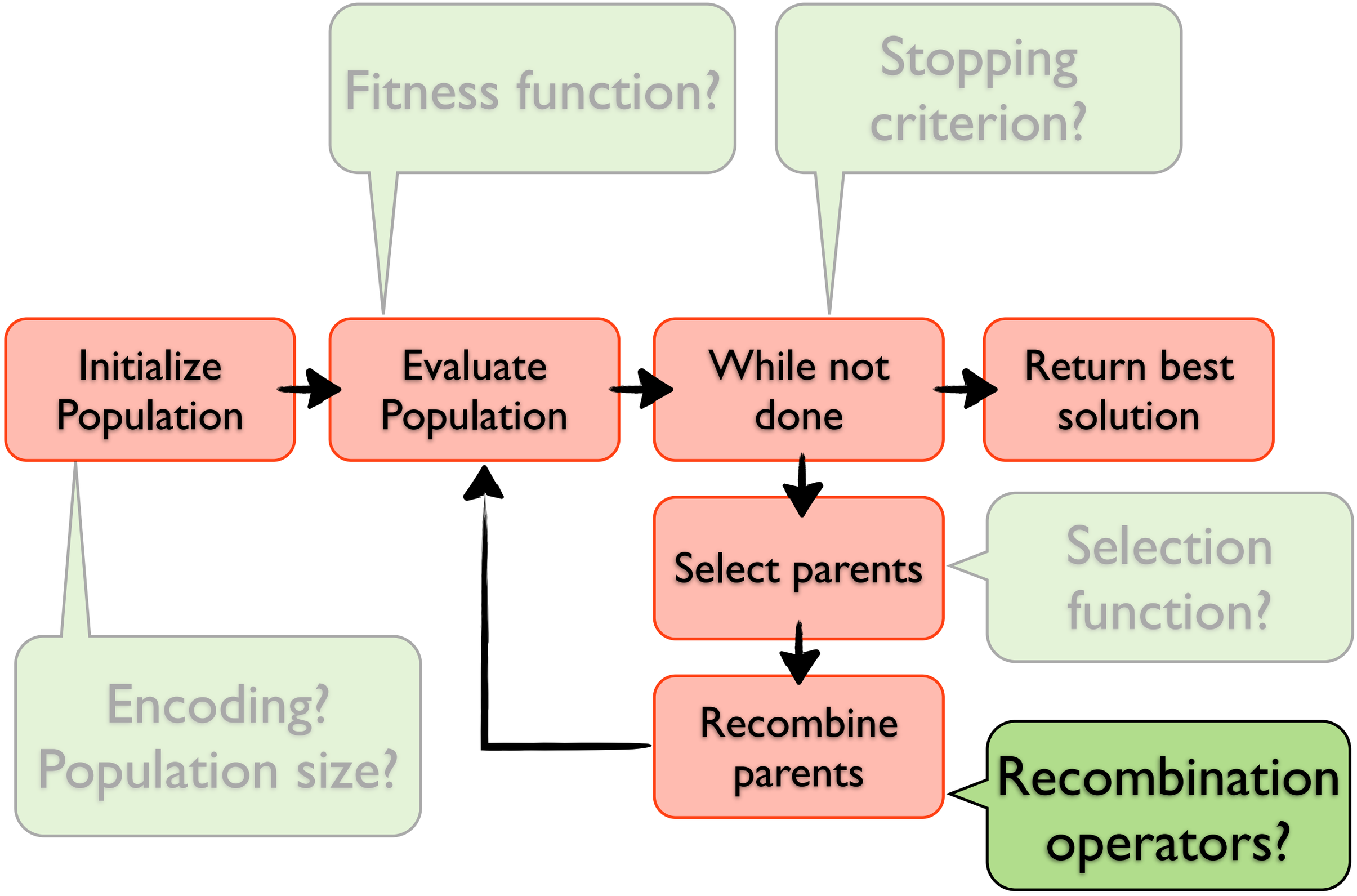
6

2

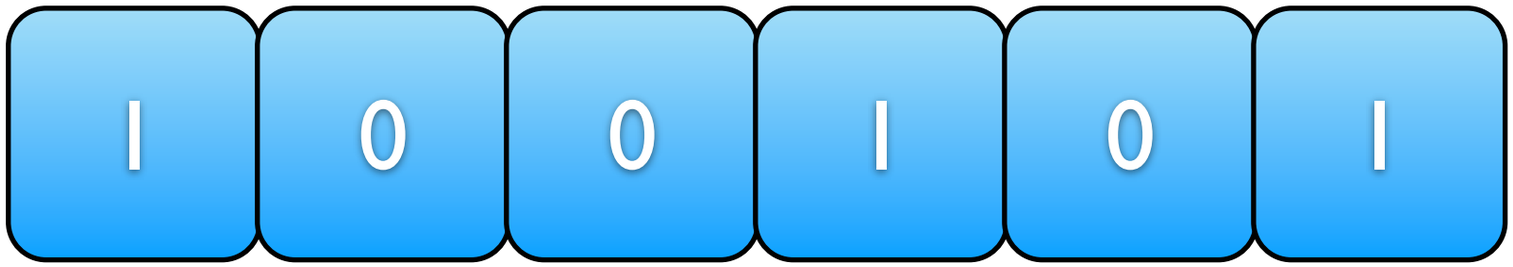
8

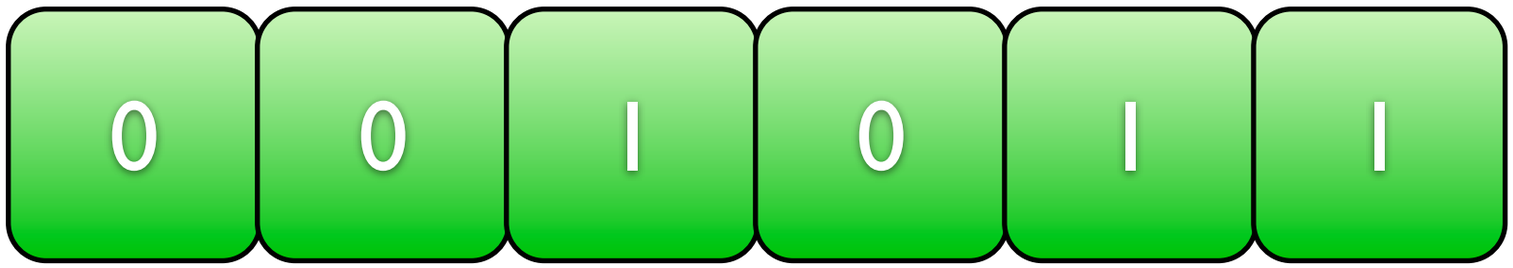
2 (1,8)


Fitness function?
Stopping criterion?
Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Selection function?
Encoding? Population size?
Recombine parents
Recombination operators?

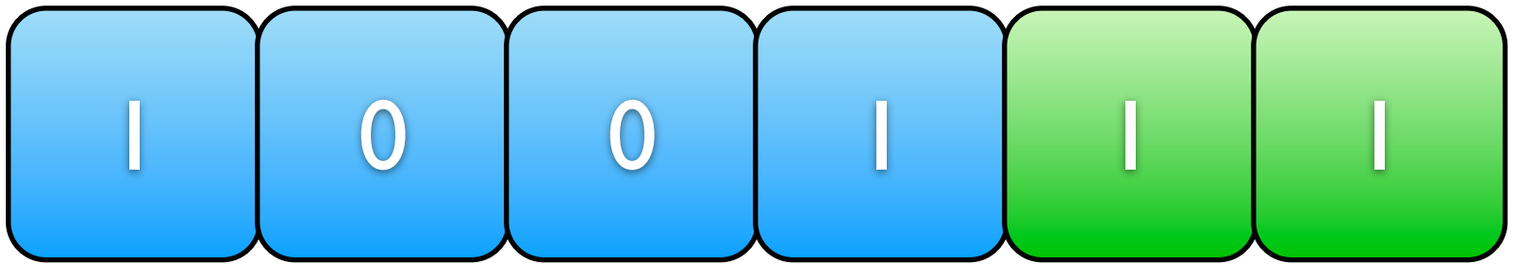

Fitness function?
Stopping criterion?
Initialize Population
Evaluate Population
While not done
Return best solution
Select parents
Selection function?
Encoding?
Population size?
Recombine parents
Recombination operators?

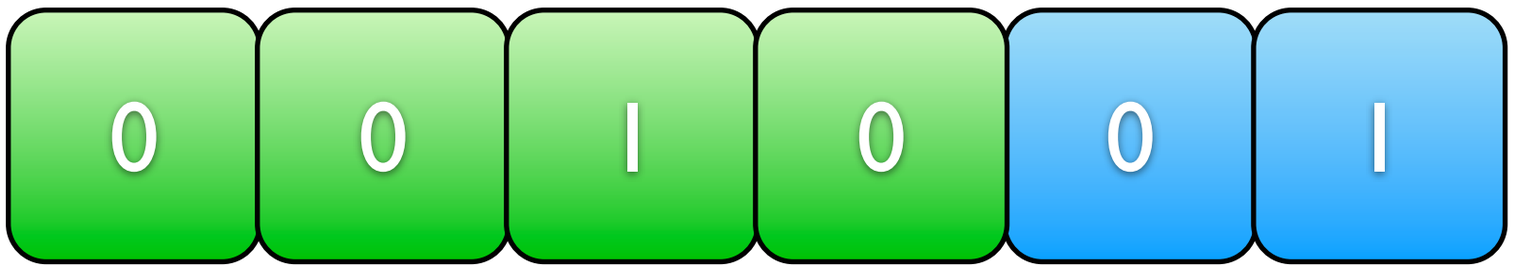
Crossover





Crossover





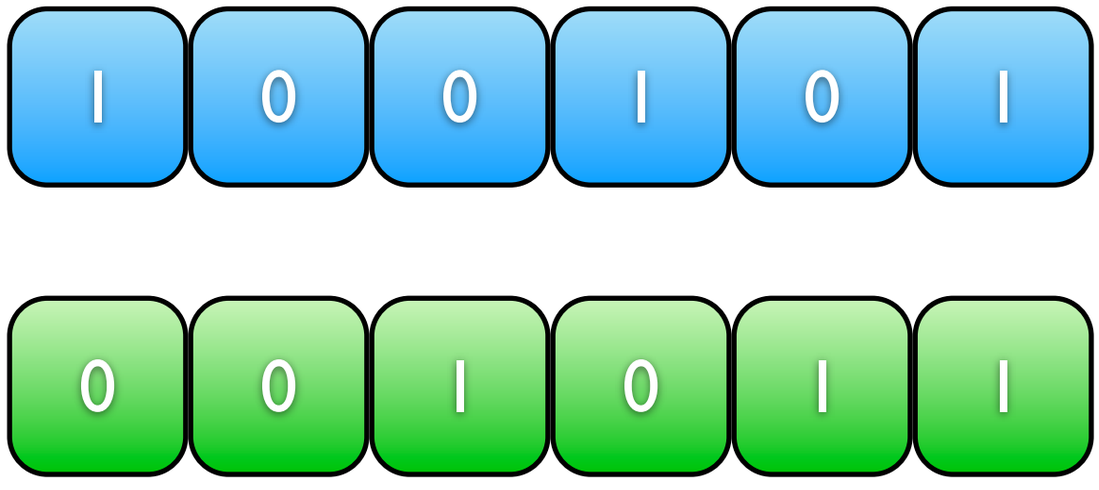
Crossover

Examples with illustration

- Single point crossover
  Choose one point in parents and split/merge at that point
- Two point crossover
  Choose two points in parents and exchange middle parts
- Fixed vs. variable length
  Same crossover point in both parents - constant length
- Uniform crossover
  Genes are randomly chosen from either parent chromosome

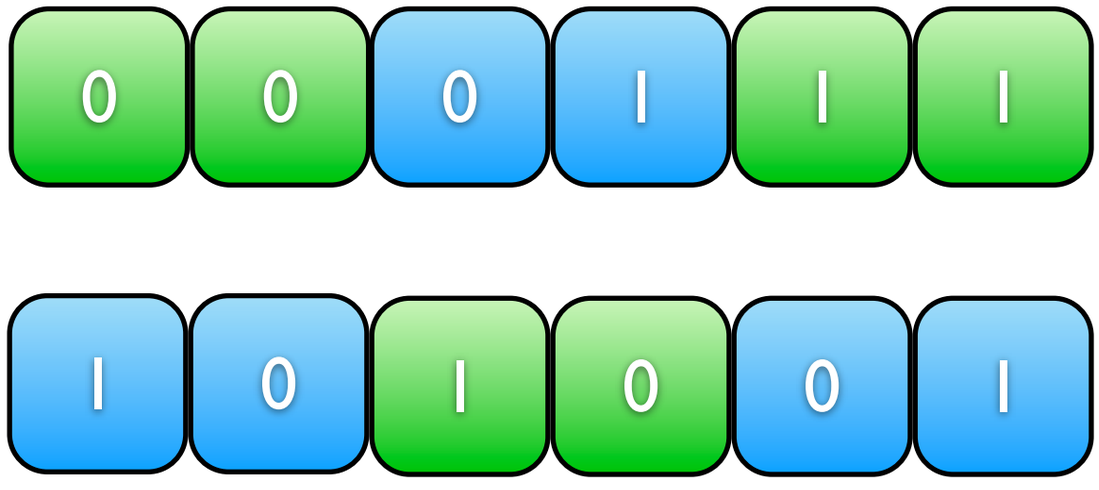
Crossover

Two-point crossover



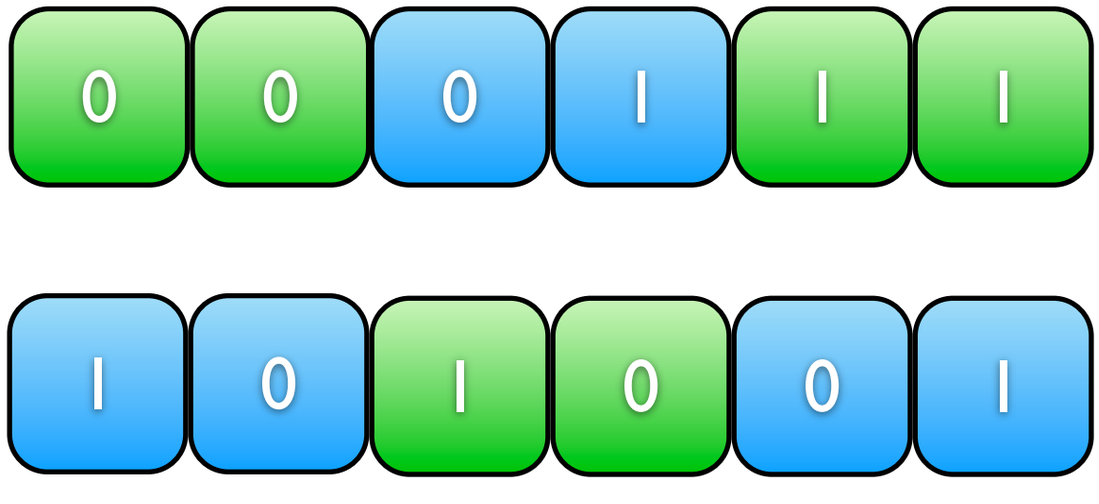
Crossover

Two-point crossover

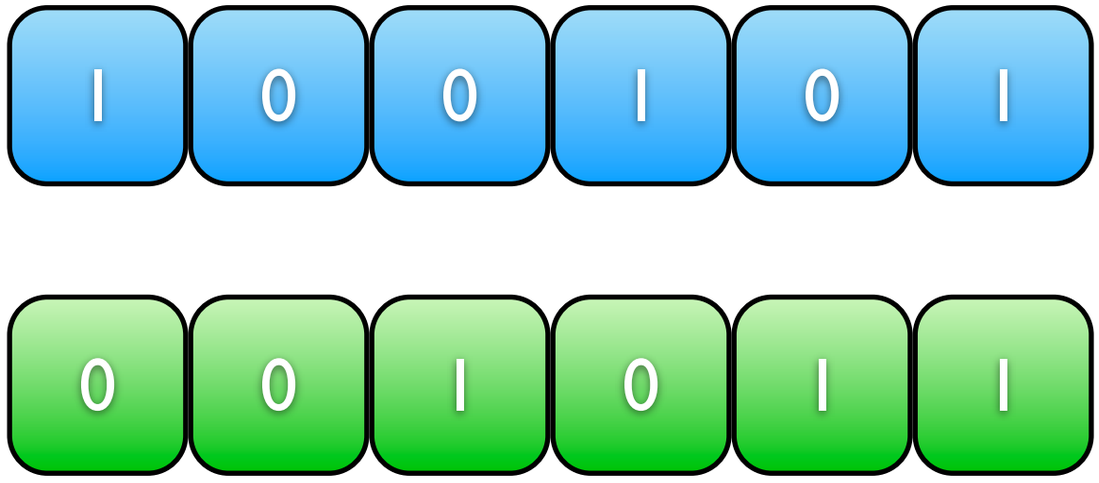


Crossover

Two-point crossover

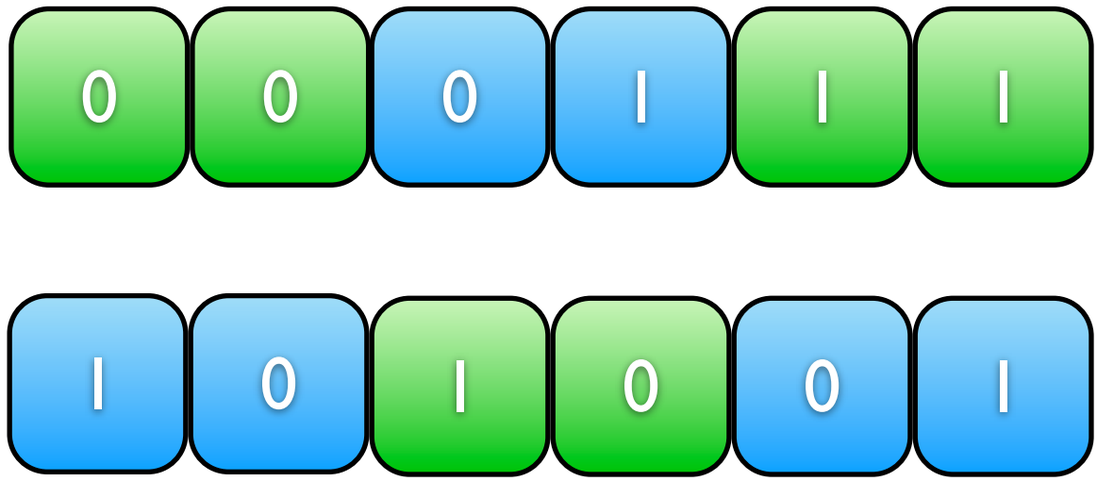


Variable length crossover

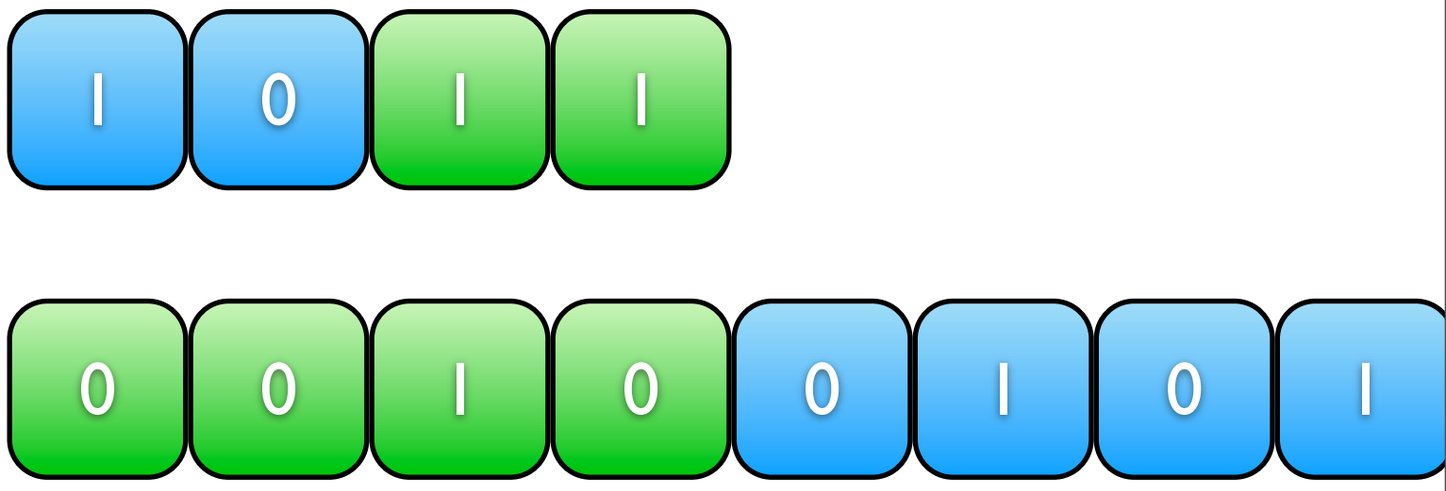


Crossover

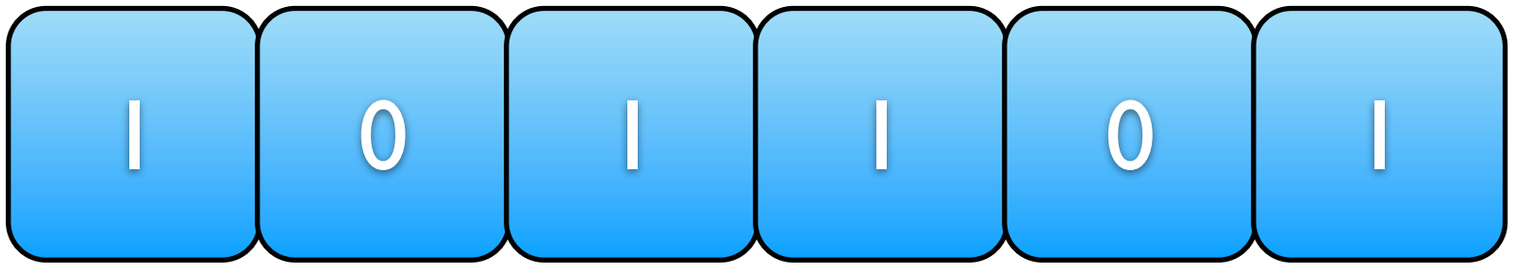
Two-point crossover



Variable length crossover



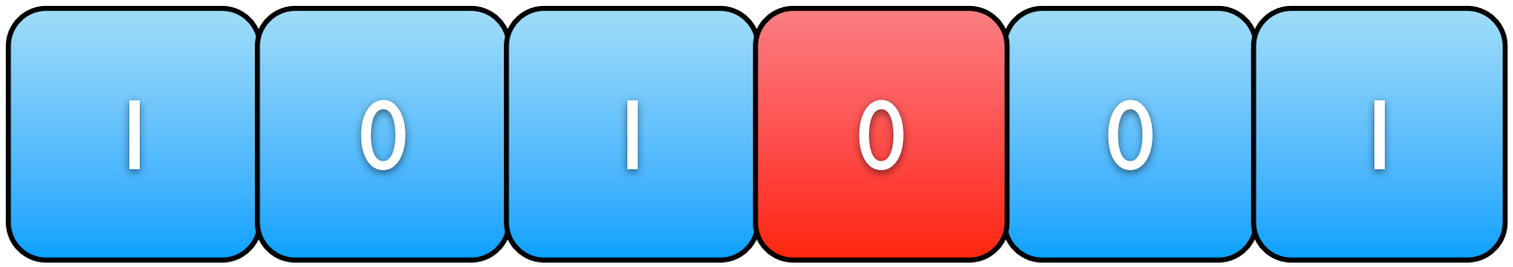
Mutation



- Change of genes

- Mutation depends on genetic operators
- Binary representation - bit flip
- For binary encoding - use Gray code
  Guarantees Hamming distance of 1 for bit flips

Mutation



- Change of genes

- Mutation depends on genetic operators
- Binary representation - bit flip
- For binary encoding - use Gray code
  Guarantees Hamming distance of 1 for bit flips

Gray Code

Gray Code

- Binary code: 0111

Gray Code

- Binary code: 0111
- Right shift inverted code: 0011

Gray Code

- Binary code: 0111
- Right shift inverted code: 0011
- 0111 xor 0011

Gray Code

- Binary code: 0111
- Right shift inverted code: 0011
- 0111 xor 0011
- 0100

Gray Code

- Binary code: 0111
- Right shift inverted code: 0011
- 0111 xor 0011
- 0100

0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Determining Parameters

- Which cross over function?
- Which mutation operators?
- Which selection function?
- Selection bias?
- Tournament size?
- Encoding?
- How often does crossover occur?
- How often does mutation occur?
- How many generations?
- Population size?
- Elite size?

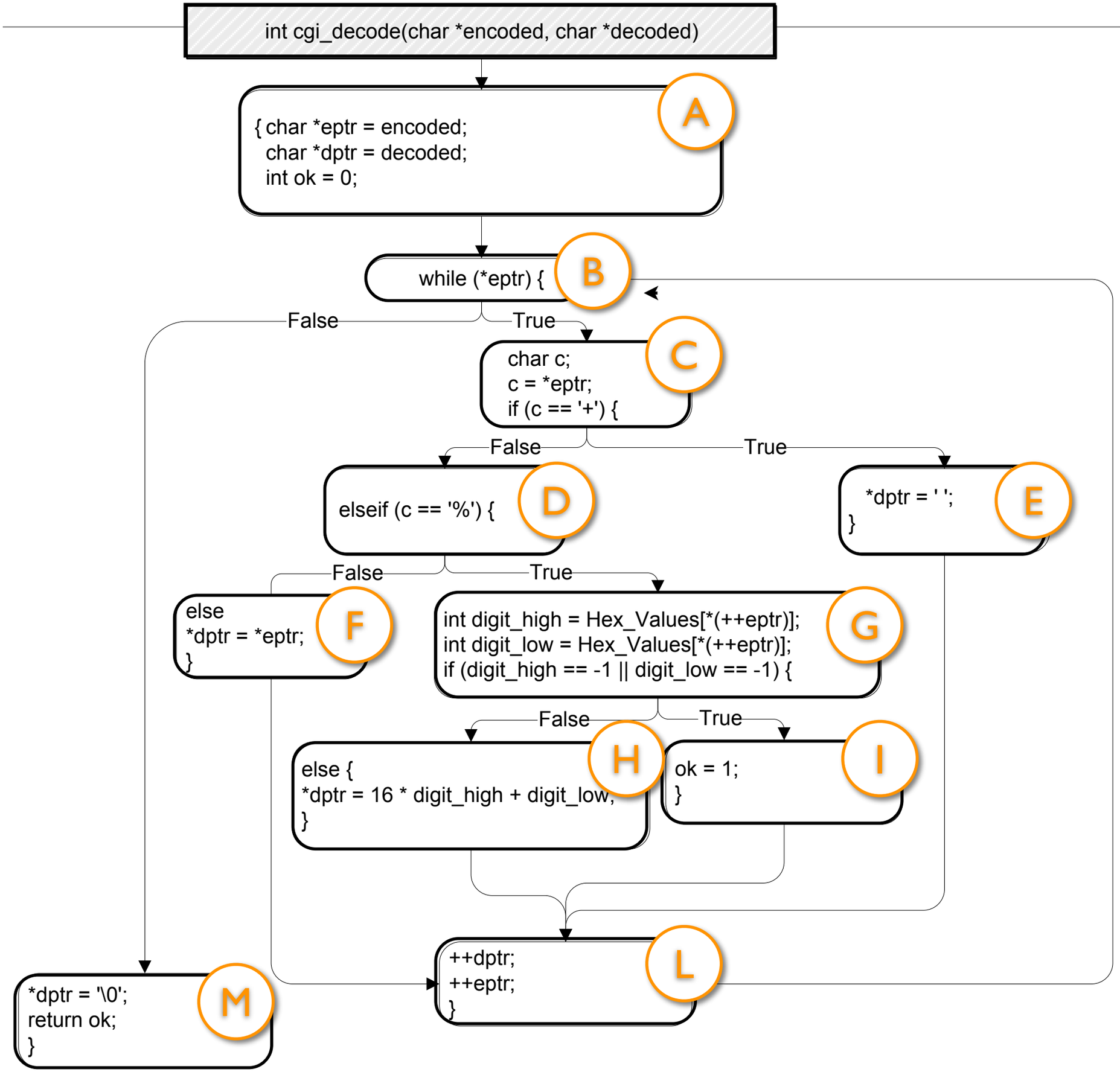
No Free Lunch Theorem

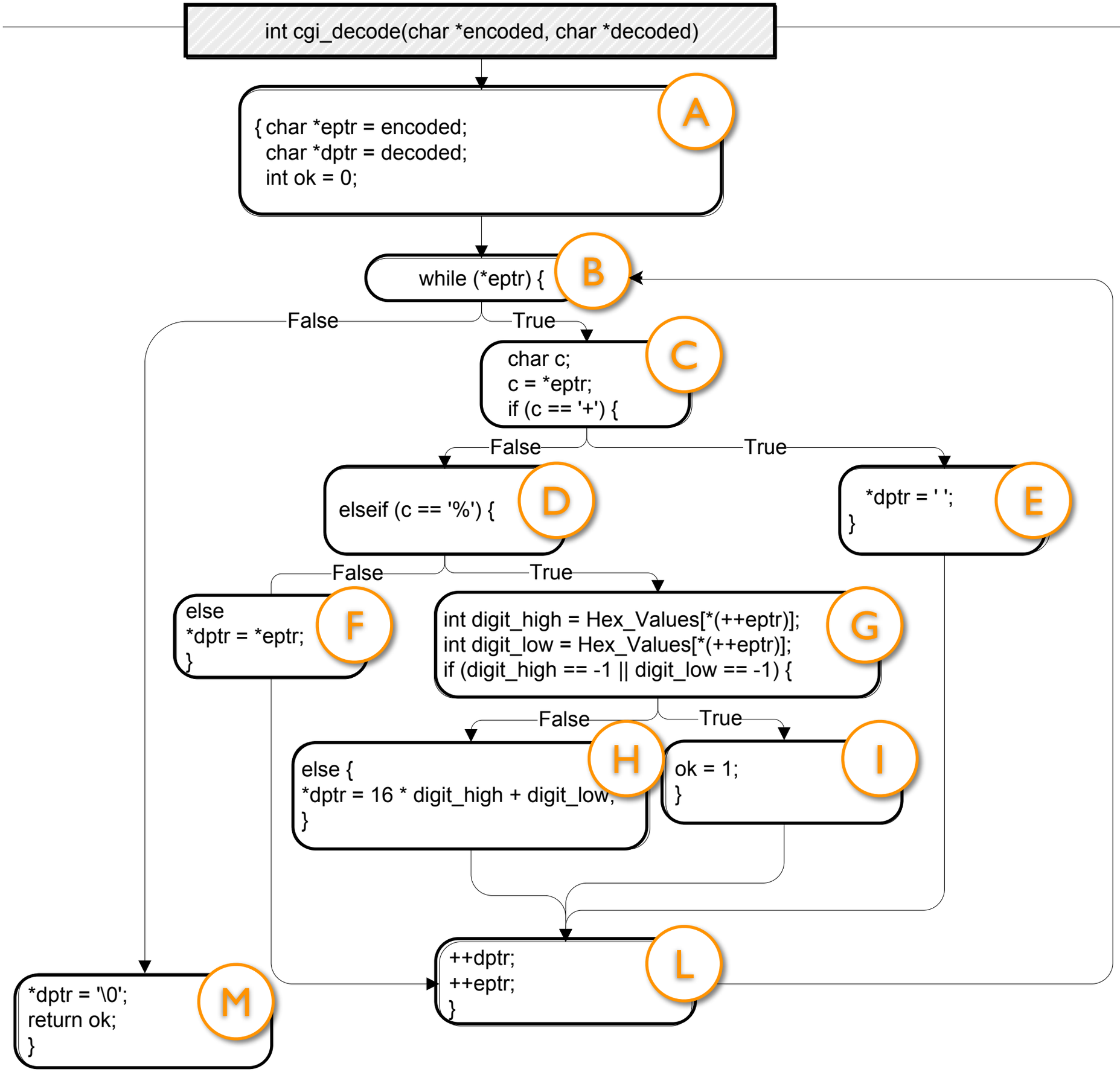
- In the entire domain of search problem, all search algorithms perform equally on average
- Each algorithm makes assumptions on the underlying problem
- Choose / adapt algorithms to specific domain

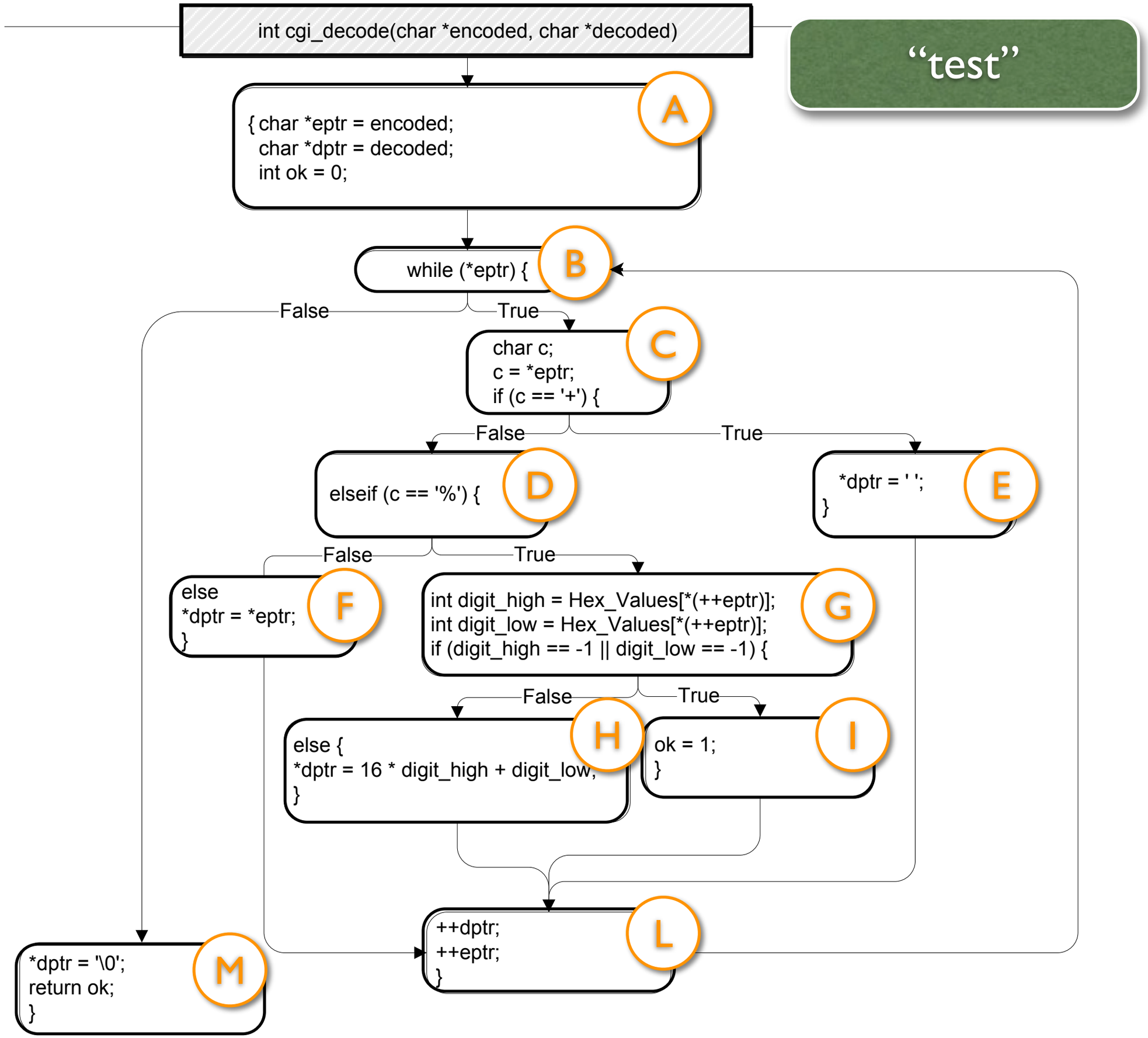
Test Data Generation

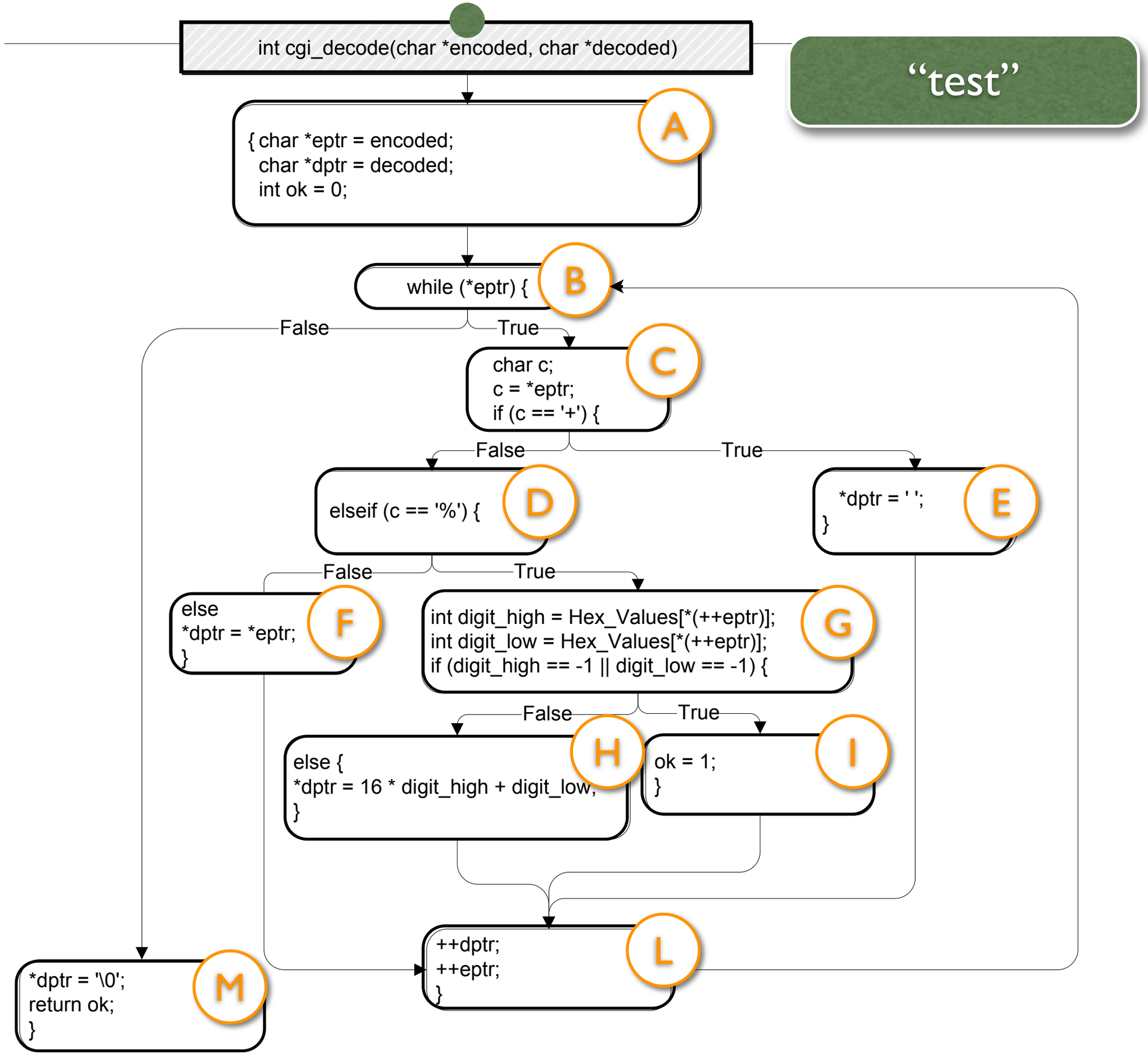
Objectives

- Structural testing
- Given: Entry function / function to test
- Test function has parameters
- Determine values for parameters to satisfy a chosen test requirement
- Search space: Values of input parameters


int cgi_decode(char *encoded, char *decoded)
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
A
while (*eptr) {
B
False
True
char c;
c = *eptr;
if (c == '+') {
C
False
True
elseif (c == '%') {
D
*dptr = ' ';
}
E
False
True
else
*dptr = *eptr;
}
F
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
}
L
*dptr = '\0';
return ok;
}
M


int cgi_decode(char *encoded, char *decoded)
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
A
while (*eptr) {
B
False
True
char c;
c = *eptr;
if (c == '+') {
C
False
True
elseif (c == '%') {
D
*dptr = ' ';
}
E
False
True
else
*dptr = *eptr;
}
F
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
}
L
*dptr = '\0';
return ok;
}
M


int cgi_decode(char *encoded, char *decoded)
"test"
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
A
while (*eptr) {
B
False
True
char c;
c = *eptr;
if (c == '+') {
C
False
True
elseif (c == '%') {
D
*dptr = ' ';
}
E
False
True
else
*dptr = *eptr;
}
F
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
}
L
*dptr = '\0';
return ok;
}
M


int cgi_decode(char *encoded, char *decoded)
"test"
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
A
while (*eptr) {
B
False
True
char c;
c = *eptr;
if (c == '+') {
C
False
True
elseif (c == '%') {
D
*dptr = ' ';
}
E
False
True
else
*dptr = *eptr;
}
F
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
}
L
*dptr = '\0';
return ok;
}
M

int cgi_decode(char *encoded, char *decoded)

"test"

{ char *eptr = encoded;
  char *dptr = decoded;
  int ok = 0;

B

while (*eptr) {

False

True

C

char c;
c = *eptr;
if (c == '+') {

False

True

D

elseif (c == '%') {

E

*dptr = ' ';
}

False

True

F

else
*dptr = *eptr;
}

G

int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {

False

True

H

else {
*dptr = 16 * digit_high + digit_low;
}

I

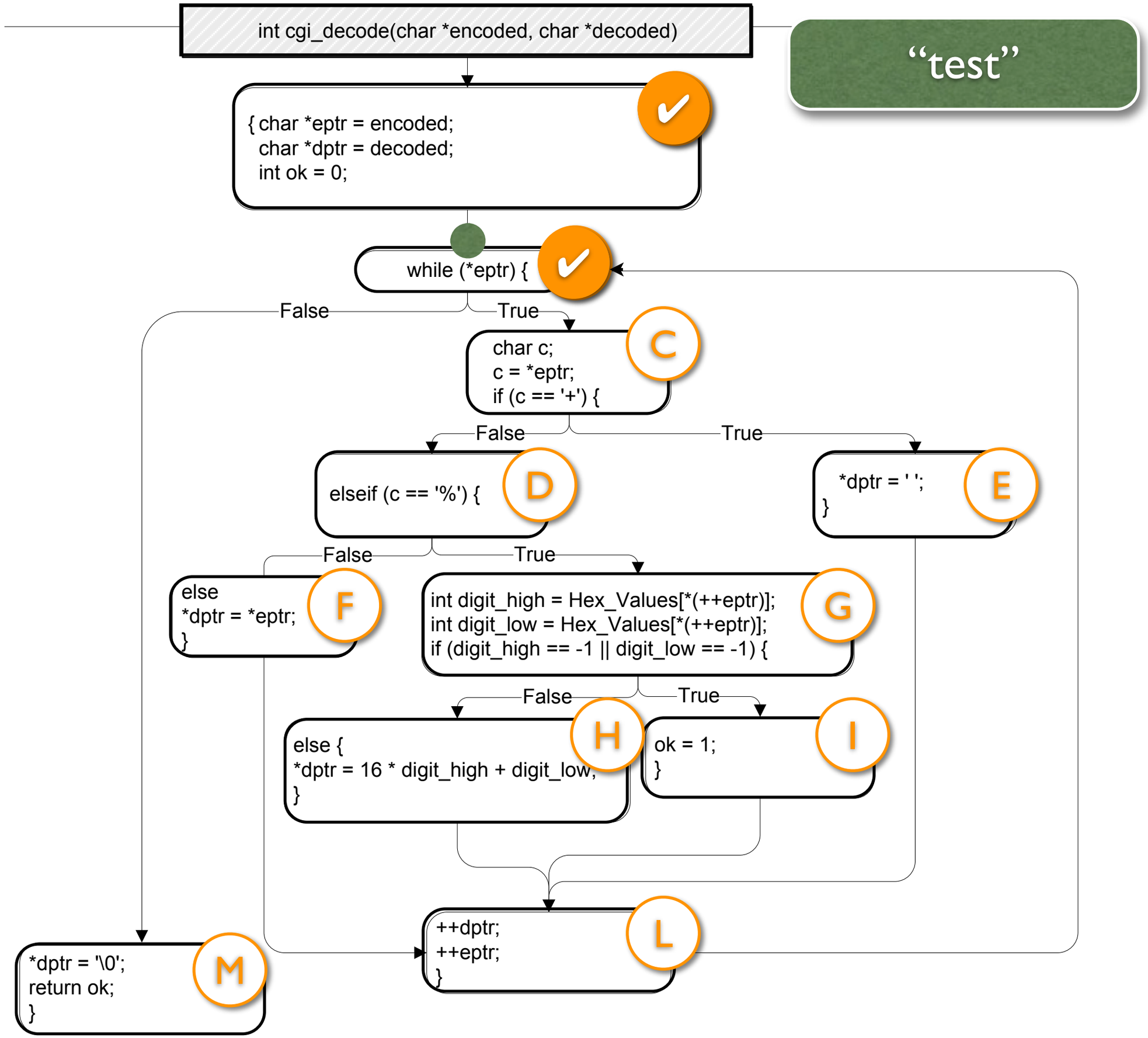
ok = 1;
}

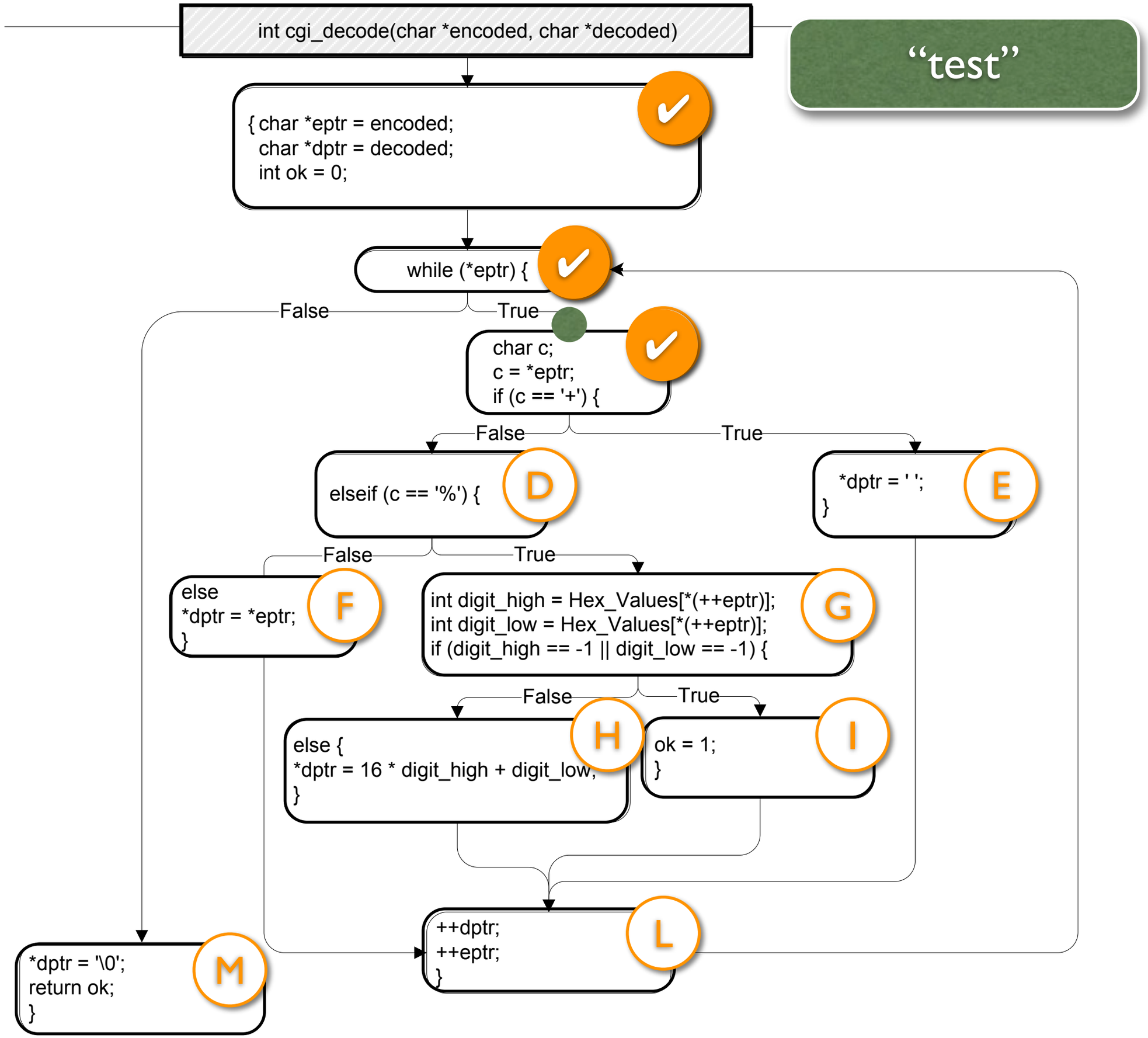
L

++dptr;
++eptr;
}

M

*dptr = '\0';
return ok;
}


int cgi_decode(char *encoded, char *decoded)
"test"
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
while (*eptr) {
False
True
char c;
c = *eptr;
if (c == '+') {
C
False
True
elseif (c == '%') {
D
*dptr = ' ';
}
E
False
True
else
*dptr = *eptr;
}
F
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
}
L
*dptr = '\0';
return ok;
}
M


int cgi_decode(char *encoded, char *decoded)
"test"
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
while (*eptr) {
False
True
char c;
c = *eptr;
if (c == '+') {
False
True
elseif (c == '%') {
D
*dptr = ' ';
}
E
False
True
else
*dptr = *eptr;
}
F
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
}
L
*dptr = '\0';
return ok;
}
M

int cgi_decode(char *encoded, char *decoded)

"test"

{ char *eptr = encoded;
  char *dptr = decoded;
  int ok = 0;

while (*eptr) {

False

True

char c;
c = *eptr;
if (c == '+') {

False

True

elseif (c == '%') {

*dptr = ' ';
}

E

False

True

else
*dptr = *eptr;
}

F

int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {

G

False

True

else {
*dptr = 16 * digit_high + digit_low;
}

H

ok = 1;
}

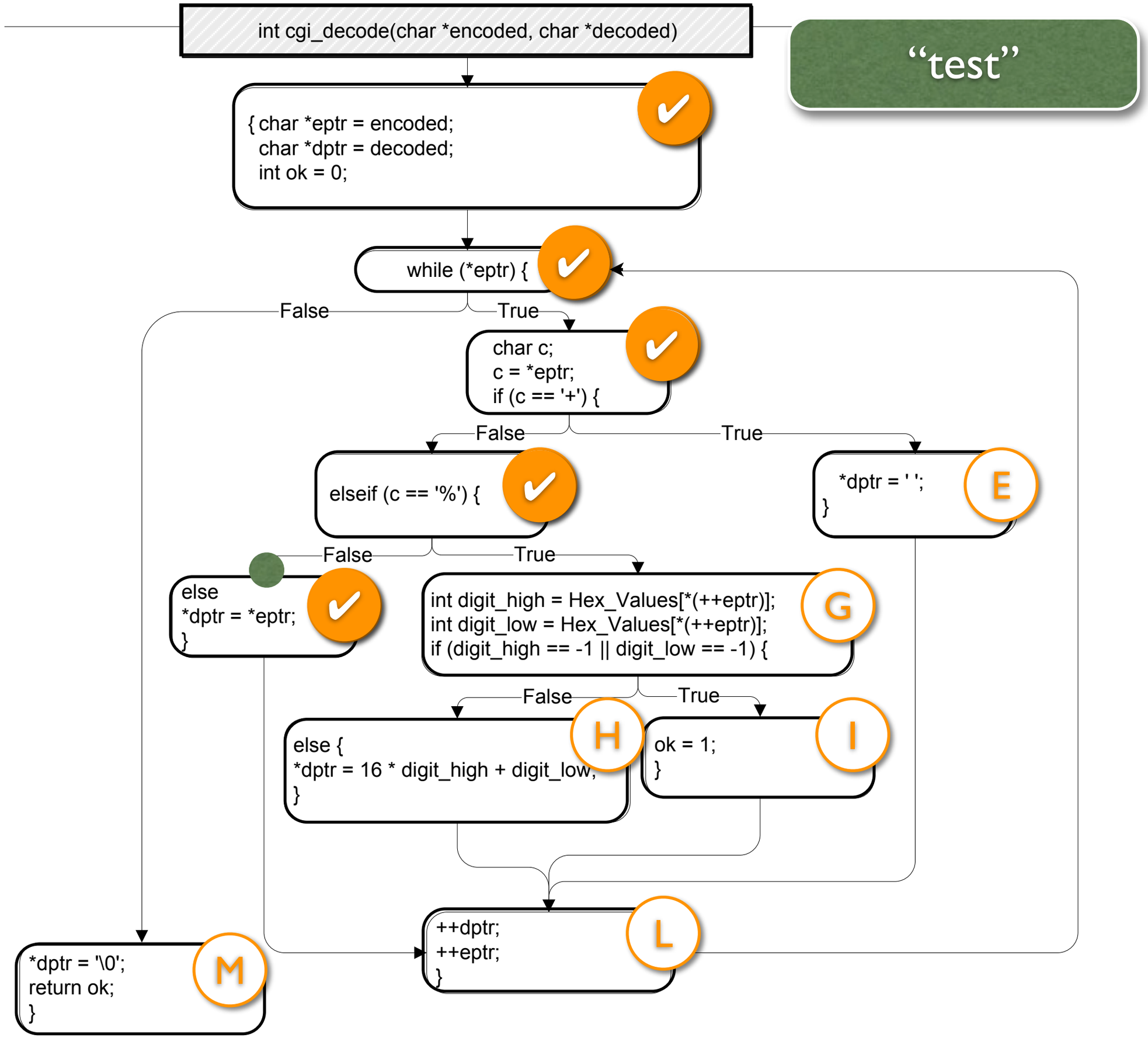
I

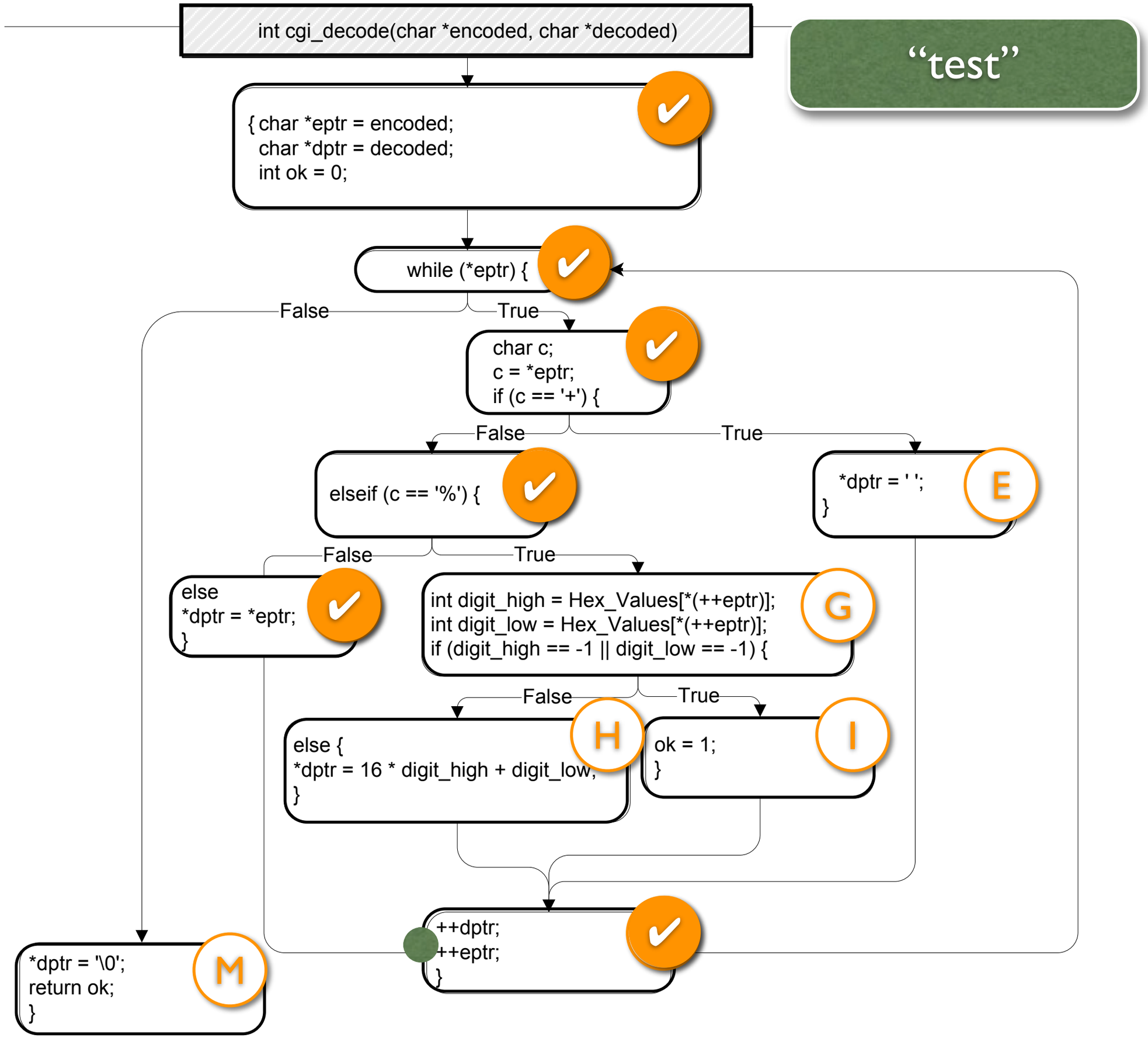
++dptr;
++eptr;
}

L

*dptr = '\0';
return ok;
}

M


int cgi_decode(char *encoded, char *decoded)
"test"
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
while (*eptr) {
False
True
char c;
c = *eptr;
if (c == '+') {
False
True
elseif (c == '%') {
*dptr = ' ';
}
E
False
True
else
*dptr = *eptr;
}
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
}
L
*dptr = '\0';
return ok;
}
M


int cgi_decode(char *encoded, char *decoded)
"test"
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
while (*eptr) {
False
True
char c;
c = *eptr;
if (c == '+') {
False
True
elseif (c == '%') {
*dptr = ' ';
}
E
False
True
else
*dptr = *eptr;
}
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
}
*dptr = '\0';
return ok;
}
M

int cgi_decode(char *encoded, char *decoded)

“test”

```
{ char *eptr = encoded;
  char *dptr = decoded;
  int ok = 0;
```

```
while (*eptr) {
```

False

True

```
char c;
c = *eptr;
if (c == '+') {
```

False

True

```
elseif (c == '%') {
```

```
*dptr = ' ';
}
```

E

False

True

```
else
*dptr = *eptr;
}
```

```
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
```

G

False

True

```
else {
*dptr = 16 * digit_high + digit_low;
}
```

H

```
ok = 1;
}
```

I

```
++dptr;
++eptr;
}
```

```
*dptr = '\0';
return ok;
}
```

M

int cgi_decode(char *encoded, char *decoded)

"test"

```
{ char *eptr = encoded;
  char *dptr = decoded;
  int ok = 0;
```

```
while (*eptr) {
```

False

True

```
char c;
c = *eptr;
if (c == '+') {
```

False

True

```
elseif (c == '%') {
```

```
*dptr = ' ';
}
```

E

False

True

```
else
*dptr = *eptr;
}
```

```
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
```

G

False

True

```
else {
*dptr = 16 * digit_high + digit_low;
}
```

H

```
ok = 1;
}
```

I

```
++dptr;
++eptr;
}
```

```
*dptr = '\0';
return ok;
}
```

M

int cgi_decode(char *encoded, char *decoded)

"test"

```
{ char *eptr = encoded;
  char *dptr = decoded;
  int ok = 0;
```

```
while (*eptr) {
```

False

True

```
char c;
c = *eptr;
if (c == '+') {
```

False

True

```
elseif (c == '%') {
```

```
*dptr = ' ';
}
```

E

False

True

```
else
*dptr = *eptr;
}
```

```
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
```

G

False

True

```
else {
*dptr = 16 * digit_high + digit_low;
}
```

H

```
ok = 1;
}
```

I

```
++dptr;
++eptr;
}
```

```
*dptr = '\0';
return ok;
}
```

M

int cgi_decode(char *encoded, char *decoded)

“test”

```
{ char *eptr = encoded;
  char *dptr = decoded;
  int ok = 0;
```

```
while (*eptr) {
```

False

True

```
char c;
c = *eptr;
if (c == '+') {
```

False

True

```
elseif (c == '%') {
```

```
*dptr = ' ';
}
```

E

False

True

```
else
*dptr = *eptr;
}
```

```
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
```

G

False

True

```
else {
*dptr = 16 * digit_high + digit_low;
}
```

H

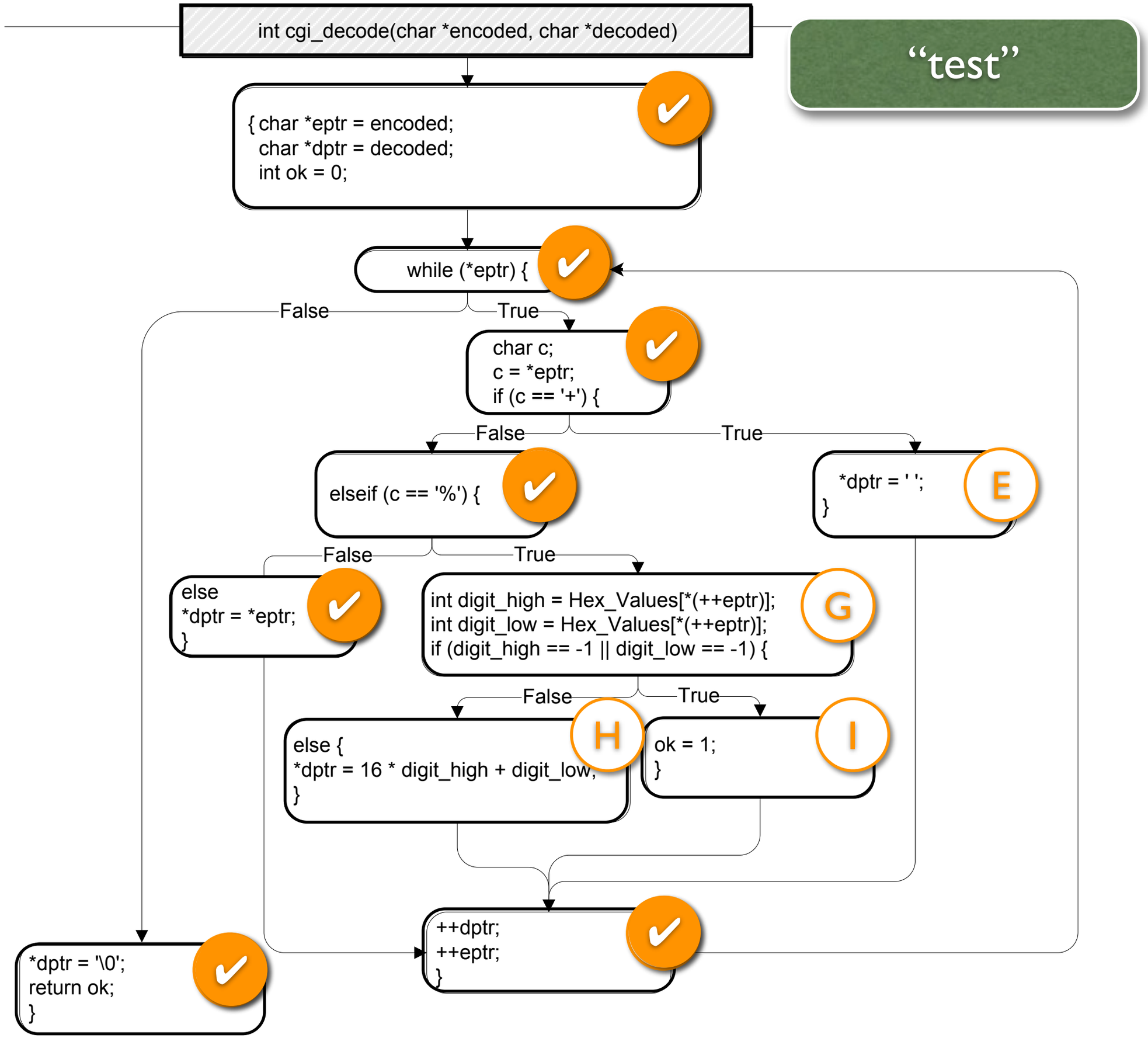
```
ok = 1;
}
```

I

```
++dptr;
++eptr;
}
```

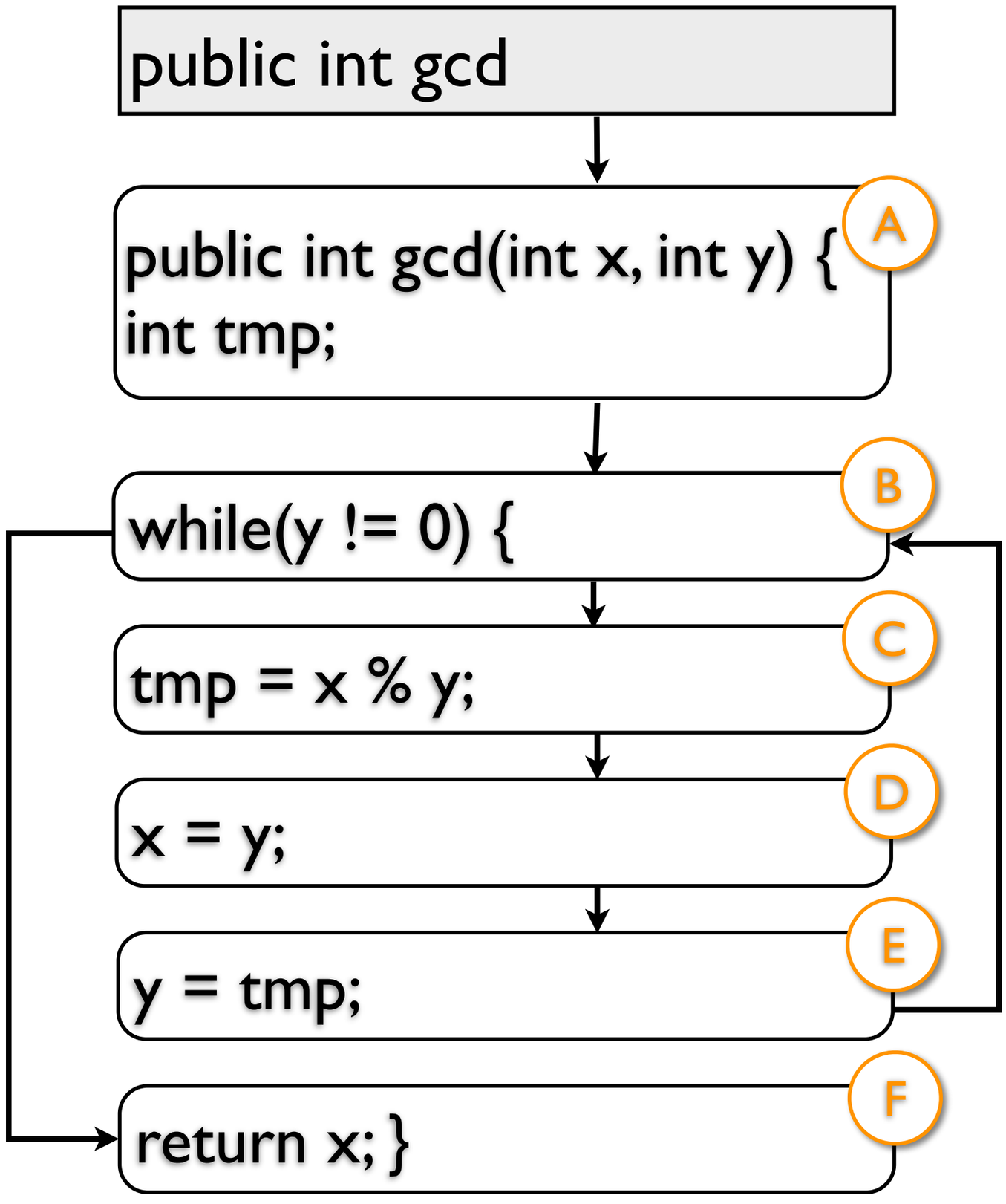
```
*dptr = '\0';
return ok;
}
```

M


int cgi_decode(char *encoded, char *decoded)
"test"
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
while (*eptr) {
False
True
char c;
c = *eptr;
if (c == '+') {
False
True
elseif (c == '%') {
*dptr = ' ';
}
E
False
True
else
*dptr = *eptr;
}
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
}
*dptr = '\0';
return ok;
}

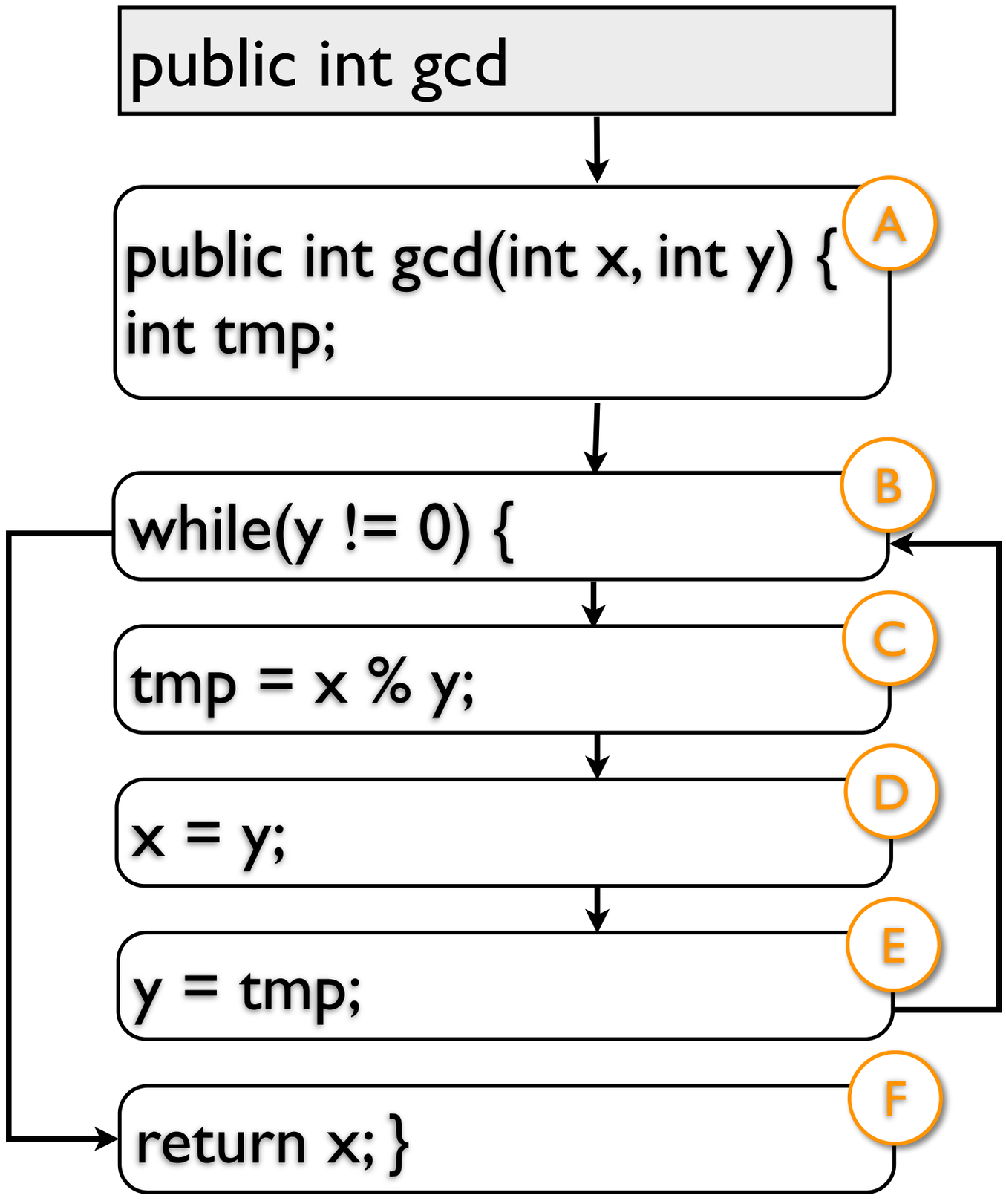

int cgi_decode(char *encoded, char *decoded)
"test"
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
while (*eptr) {
False
True
char c;
c = *eptr;
if (c == '+') {
False
True
elseif (c == '%') {
*dptr = ' ';
}
E
False
True
else
*dptr = *eptr;
}
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
G
How close were we to executing E?
++dptr;
++eptr;
}
*dptr = '\0';
return ok;
}

```
public int gcd(int x, int y) {
   int tmp;
   while (y != 0) {
      tmp = x % y;
      x = y;
      y = tmp;
   }
   return x;
}
```

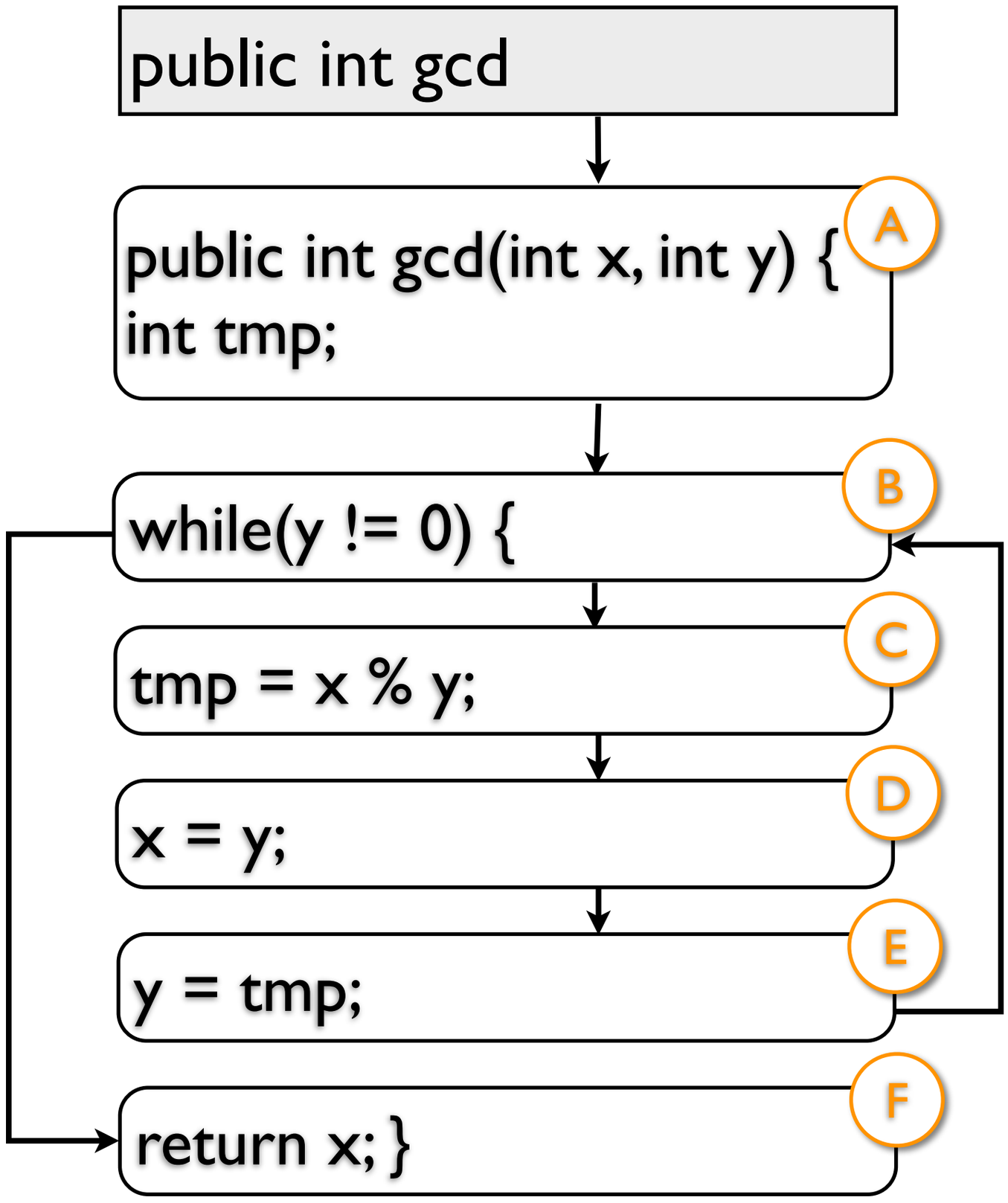

public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F

Dominators

- Node A dominates B if every path to B goes through A
- Immediate dominator: Closest dominator on any path from the root
- Root node has no immediate dominator

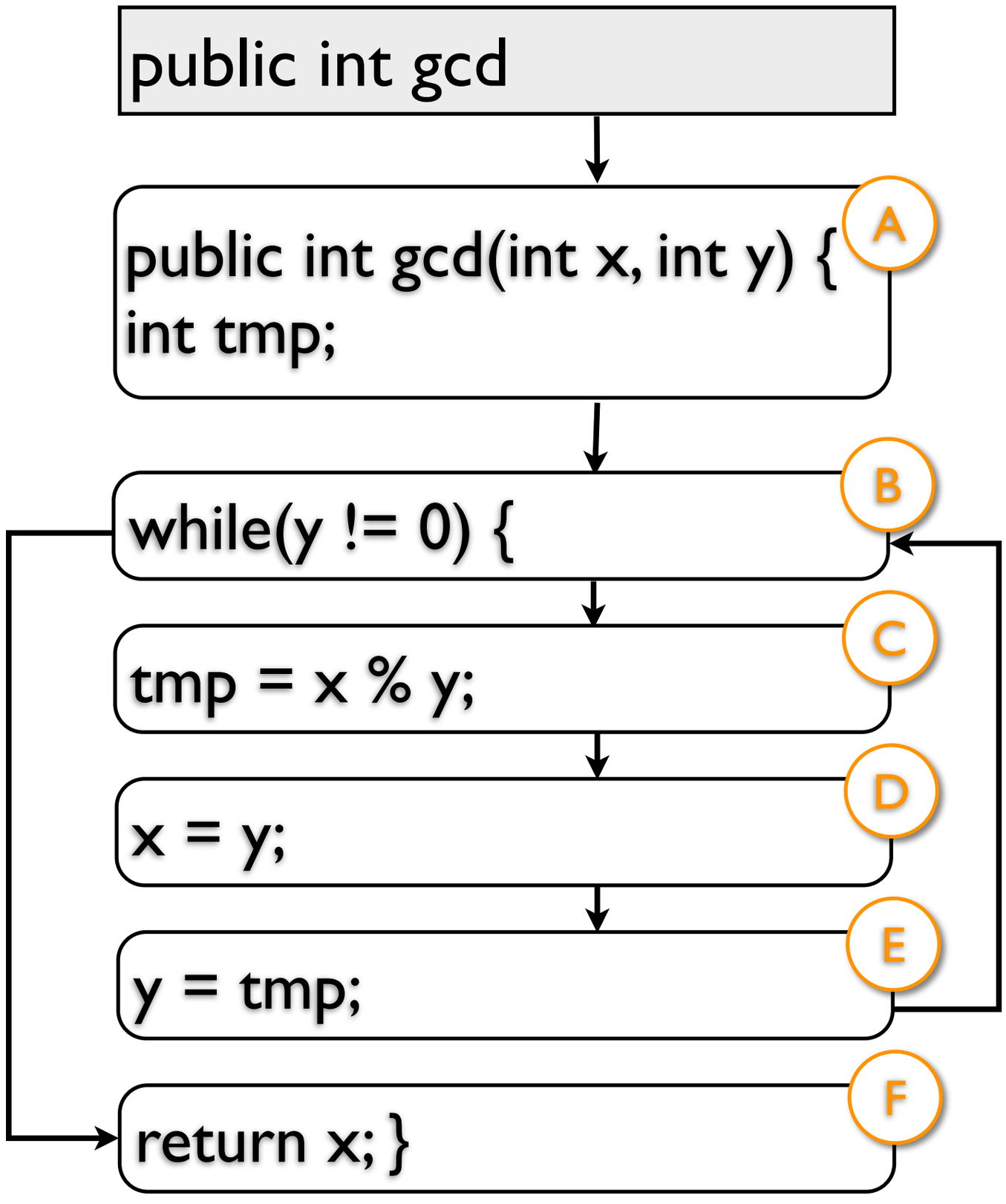

public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F

A


public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F

A

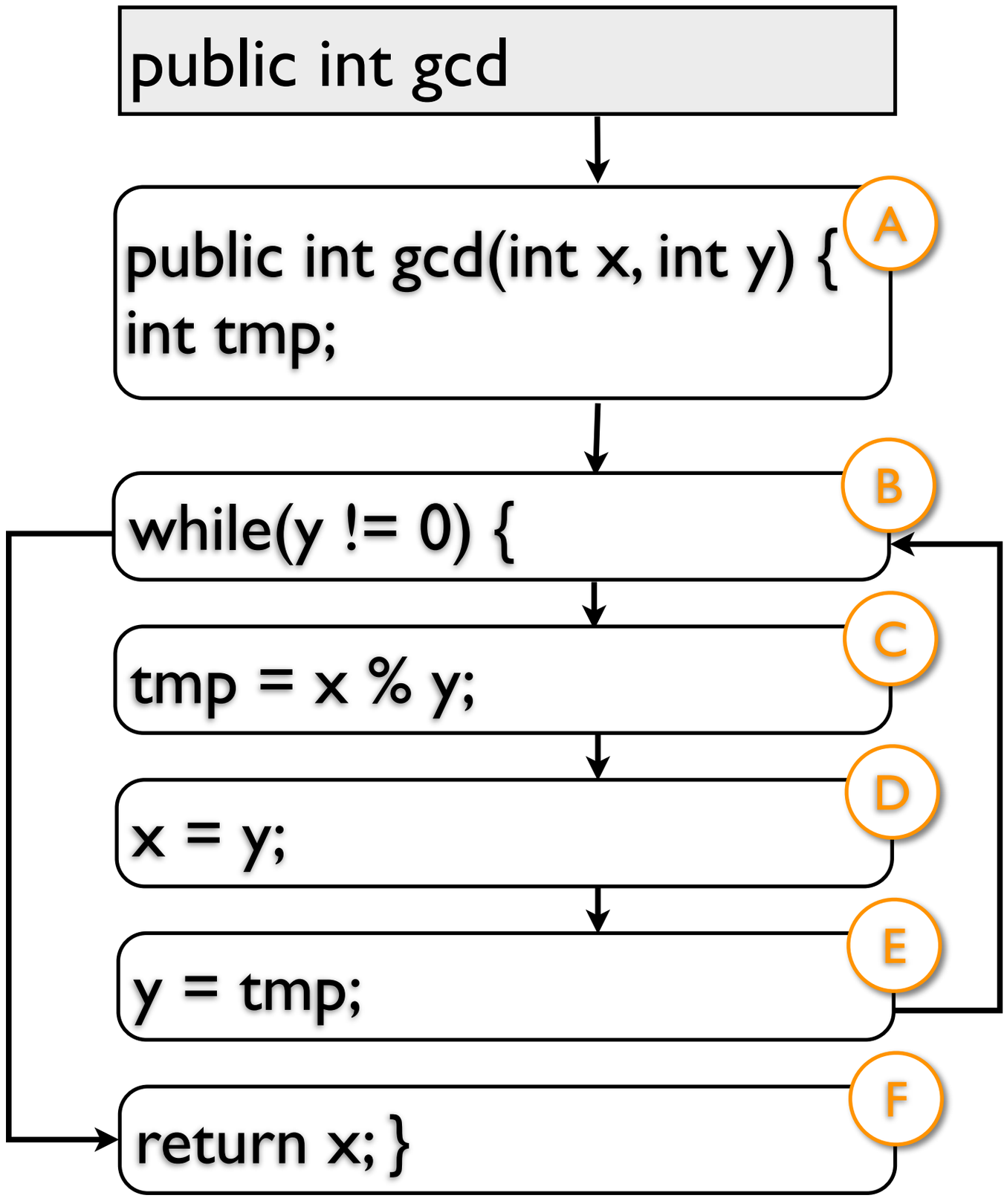
A,B


public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F

A

A,B

A,B,C

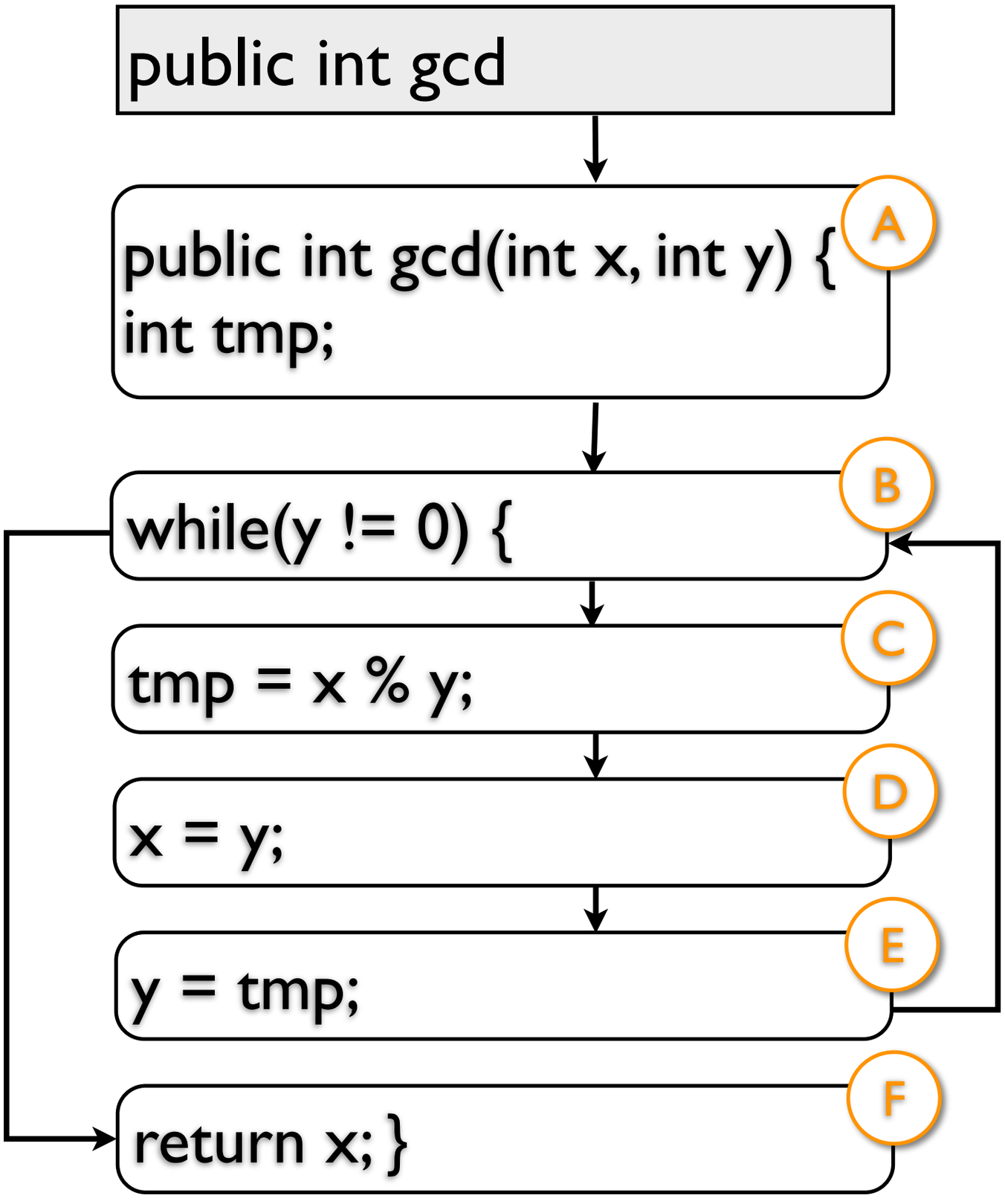

public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F

A

A,B

A,B,C

A,B,C,D


public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F

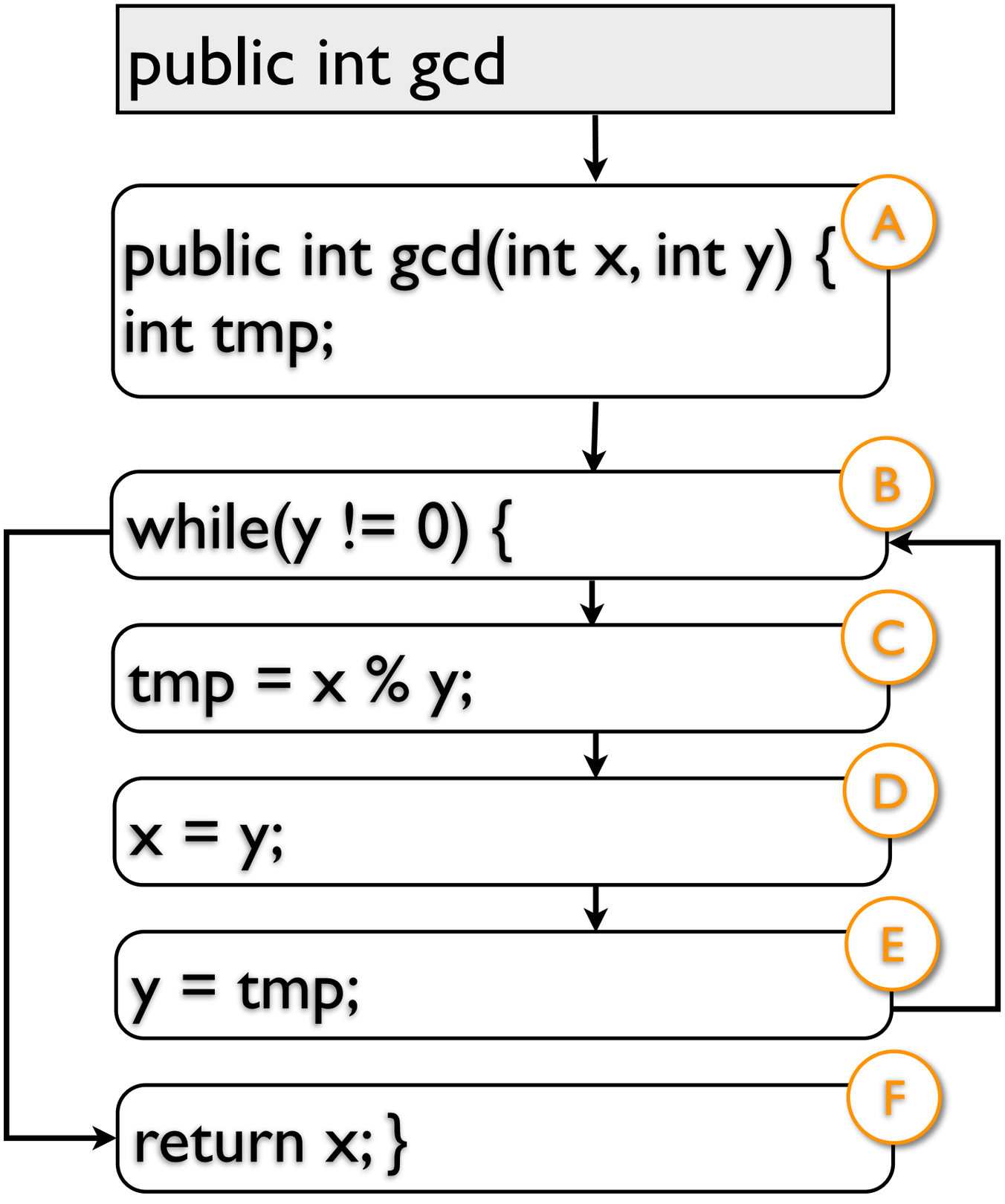
A

A,B

A,B,C

A,B,C,D

A,B,C,D,E



A

A,B

A,B,C

A,B,C,D

A,B,C,D,E

A,B,F

```
public int gcd
public int gcd(int x, int y) {
int tmp;
while(y != 0) {
tmp = x % y;
x = y;
y = tmp;
return x; }
```

A

B

C

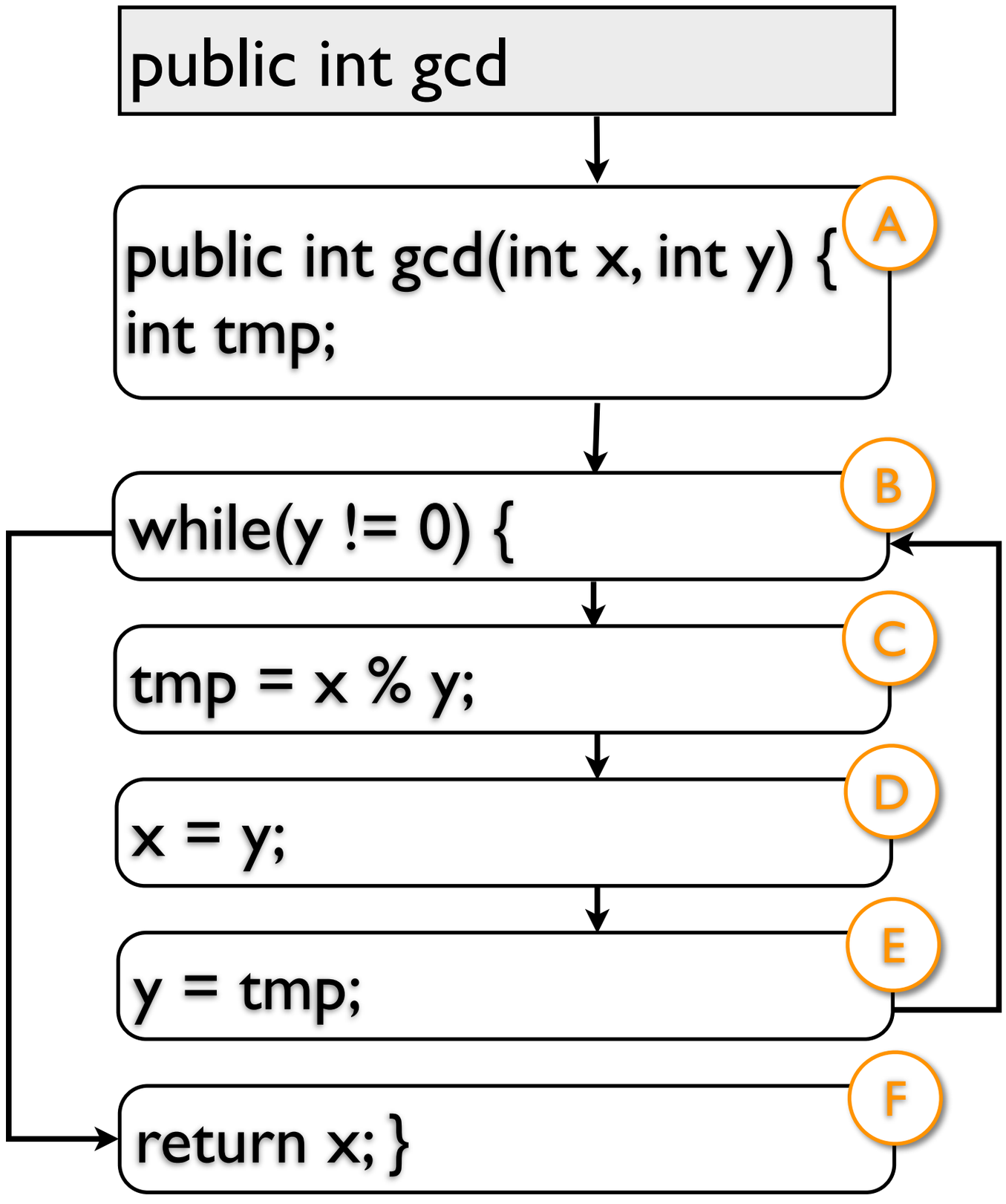
D

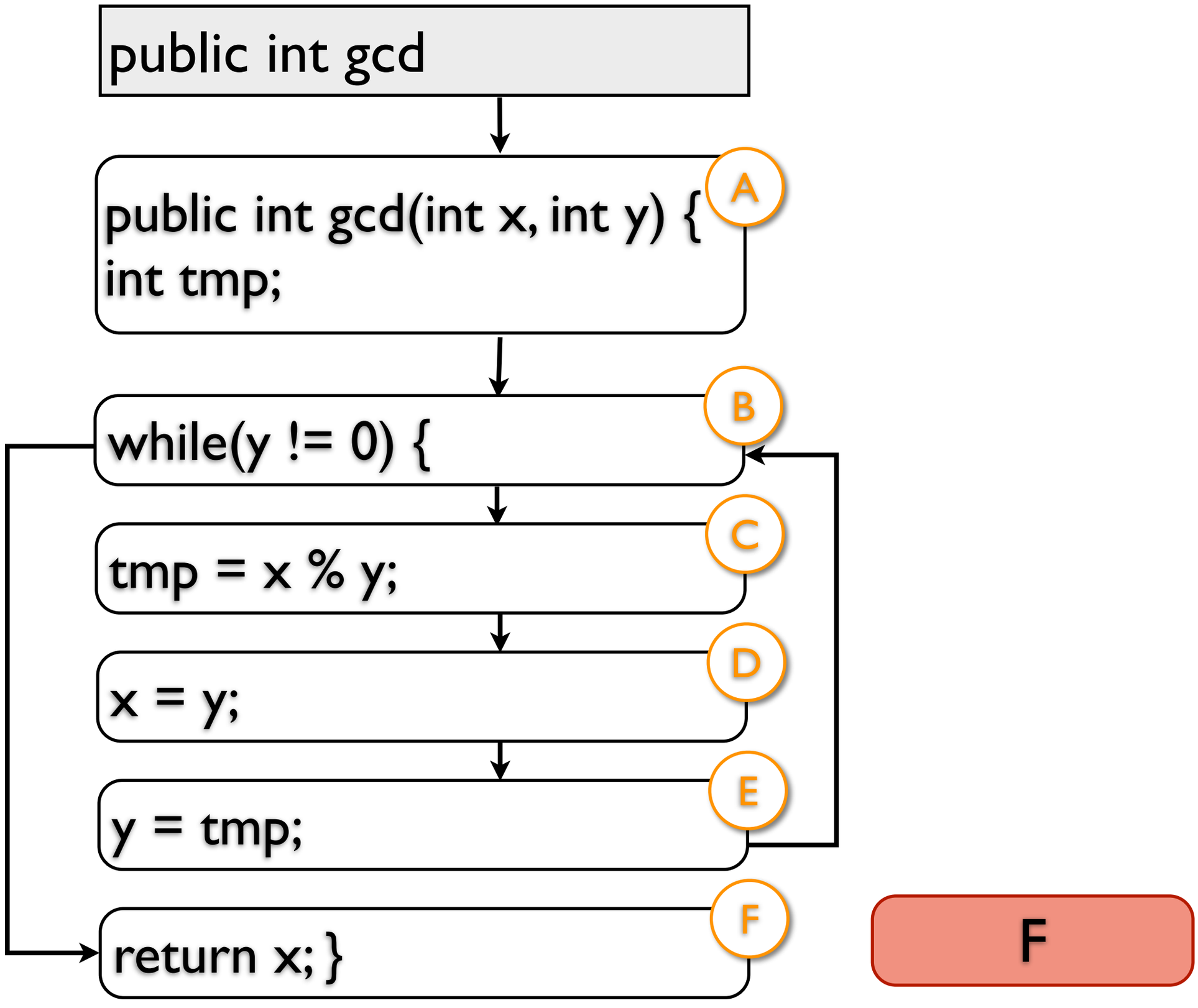
E

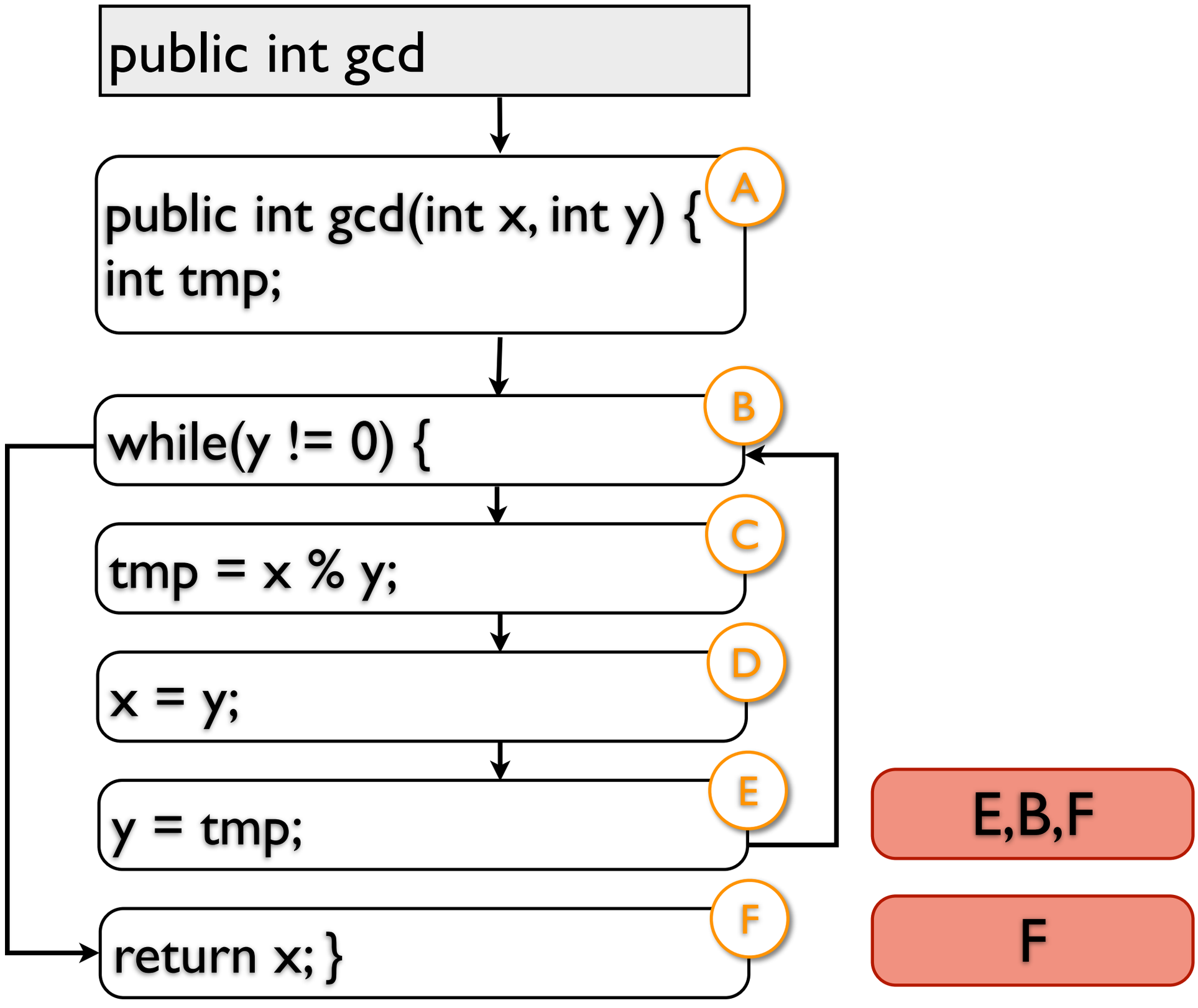
F

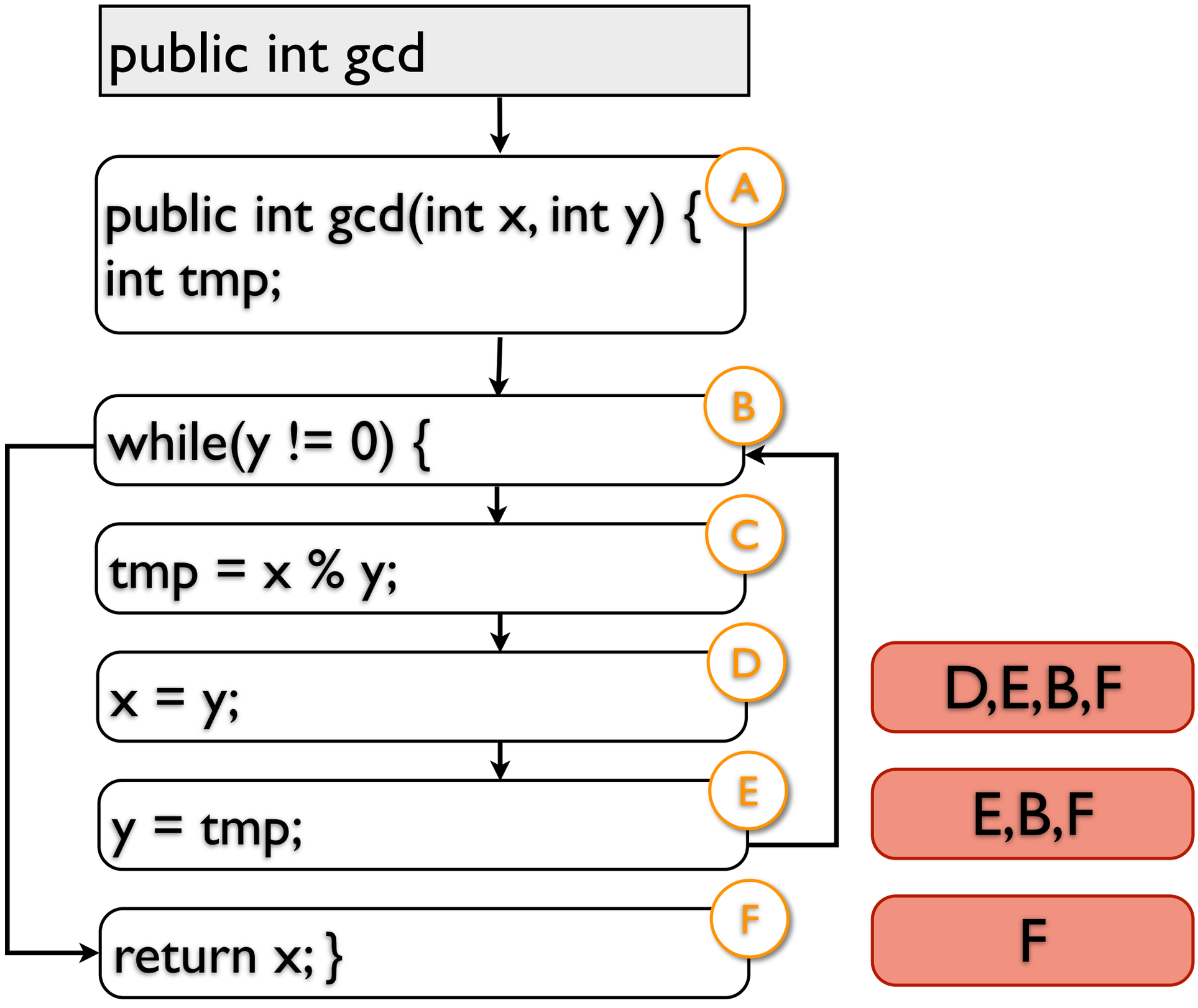
Post Dominators

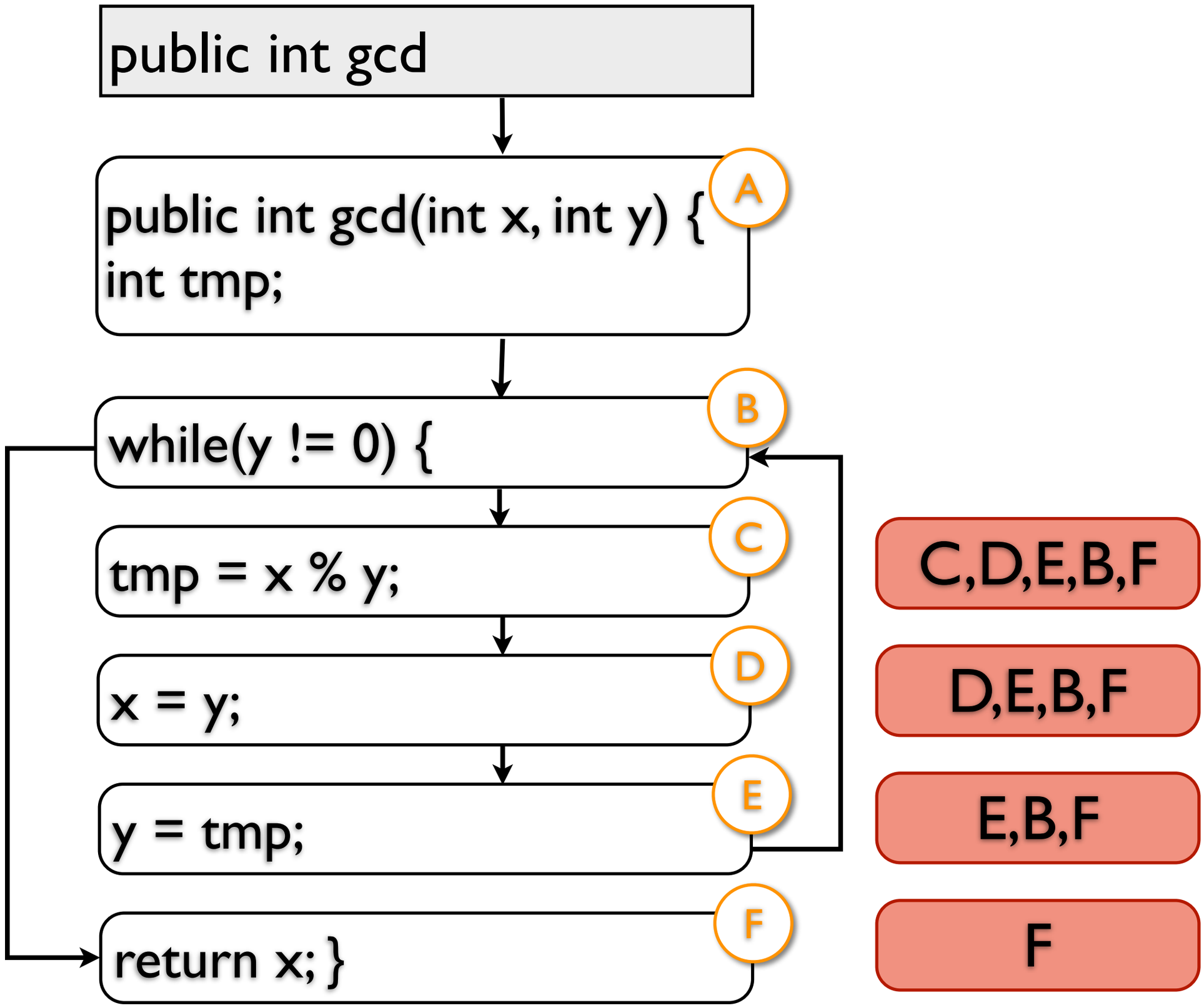
- Dominators viewed in reverse (paths from exit node)
- Node A post-dominates B if all paths from B to exit must go through A
- Immediate post dominator: Closest dominator on any path to the exit node

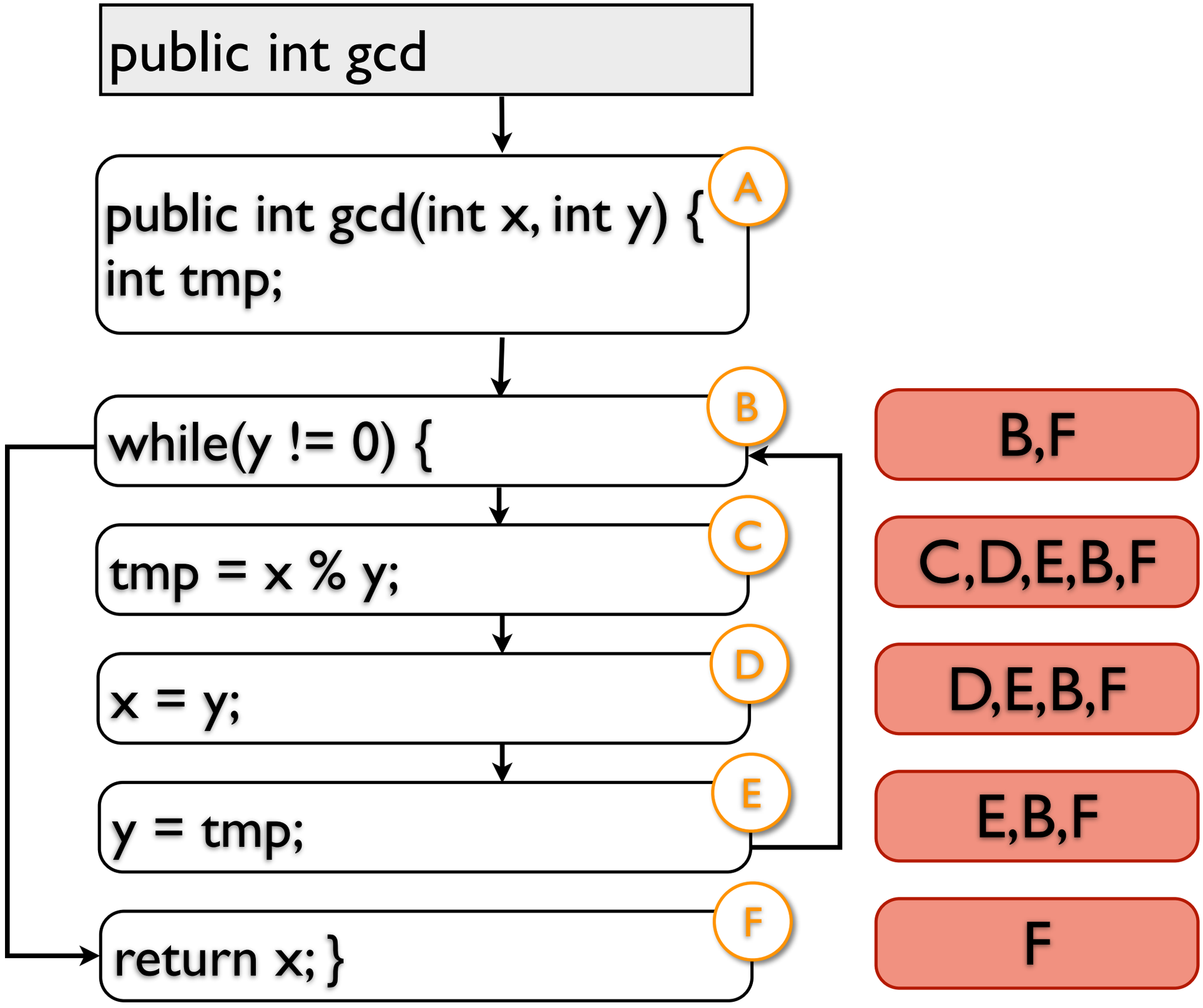

public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F

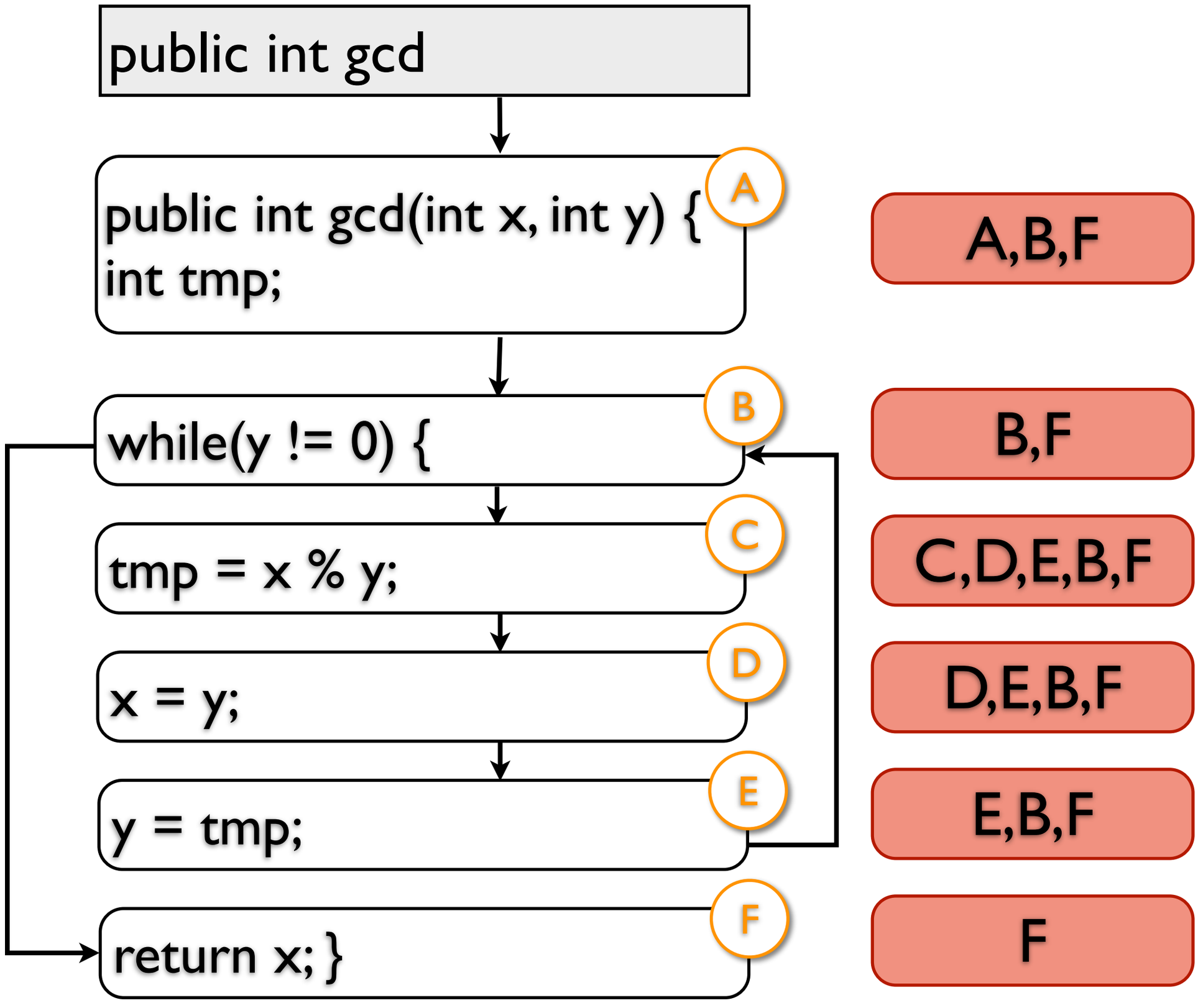

public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
F

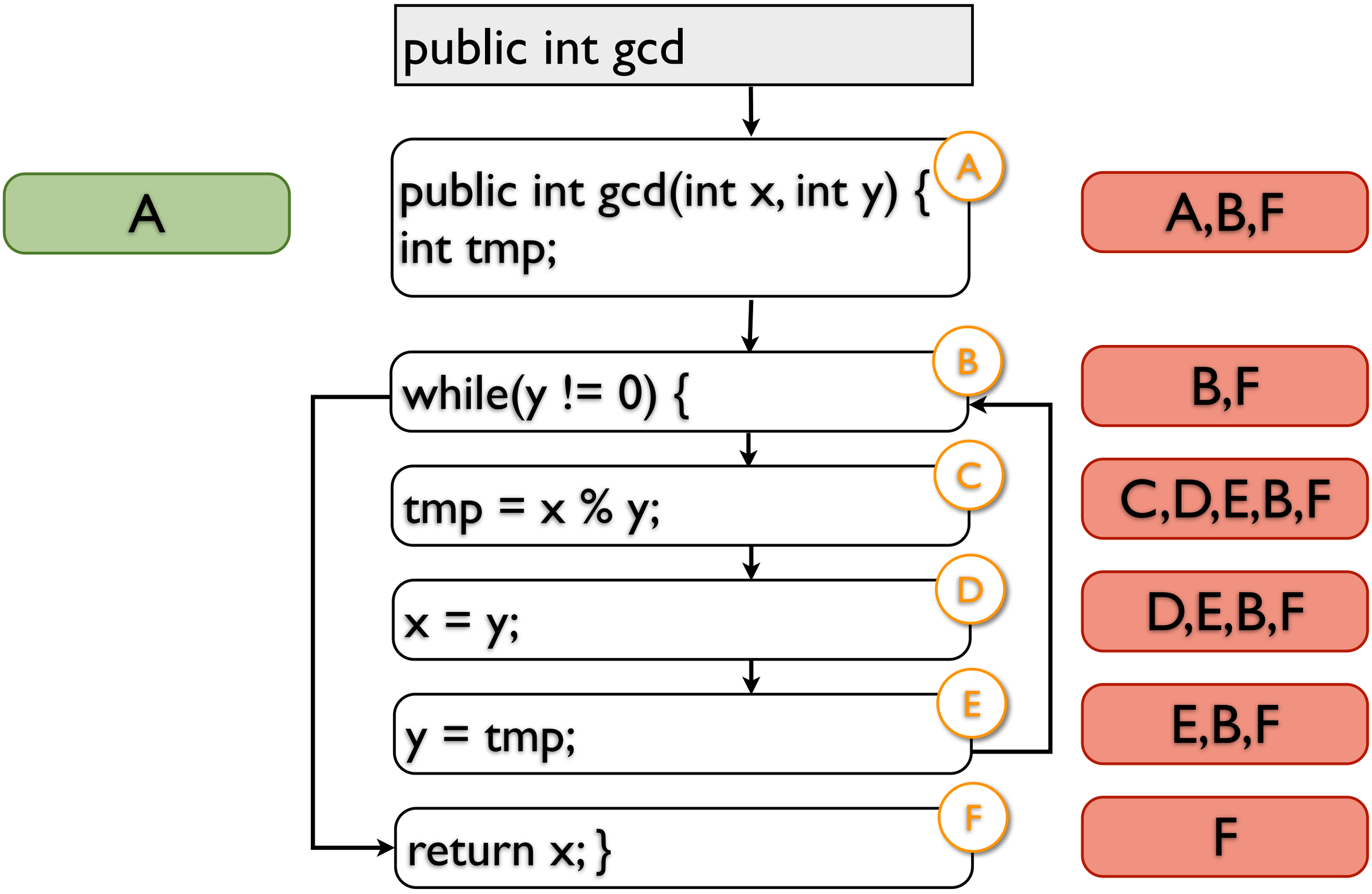

public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
E,B,F
F

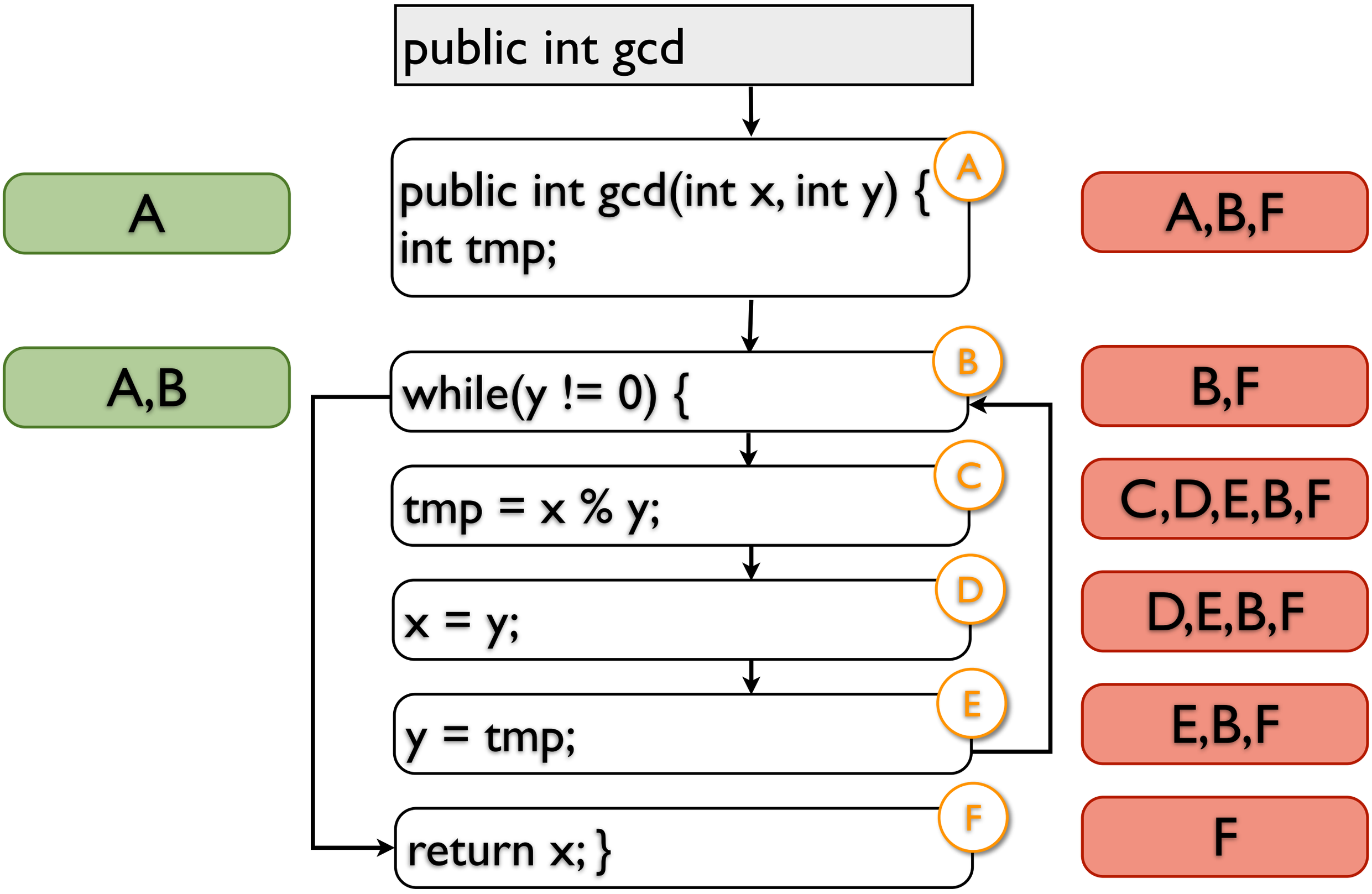

public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
D,E,B,F
E,B,F
F

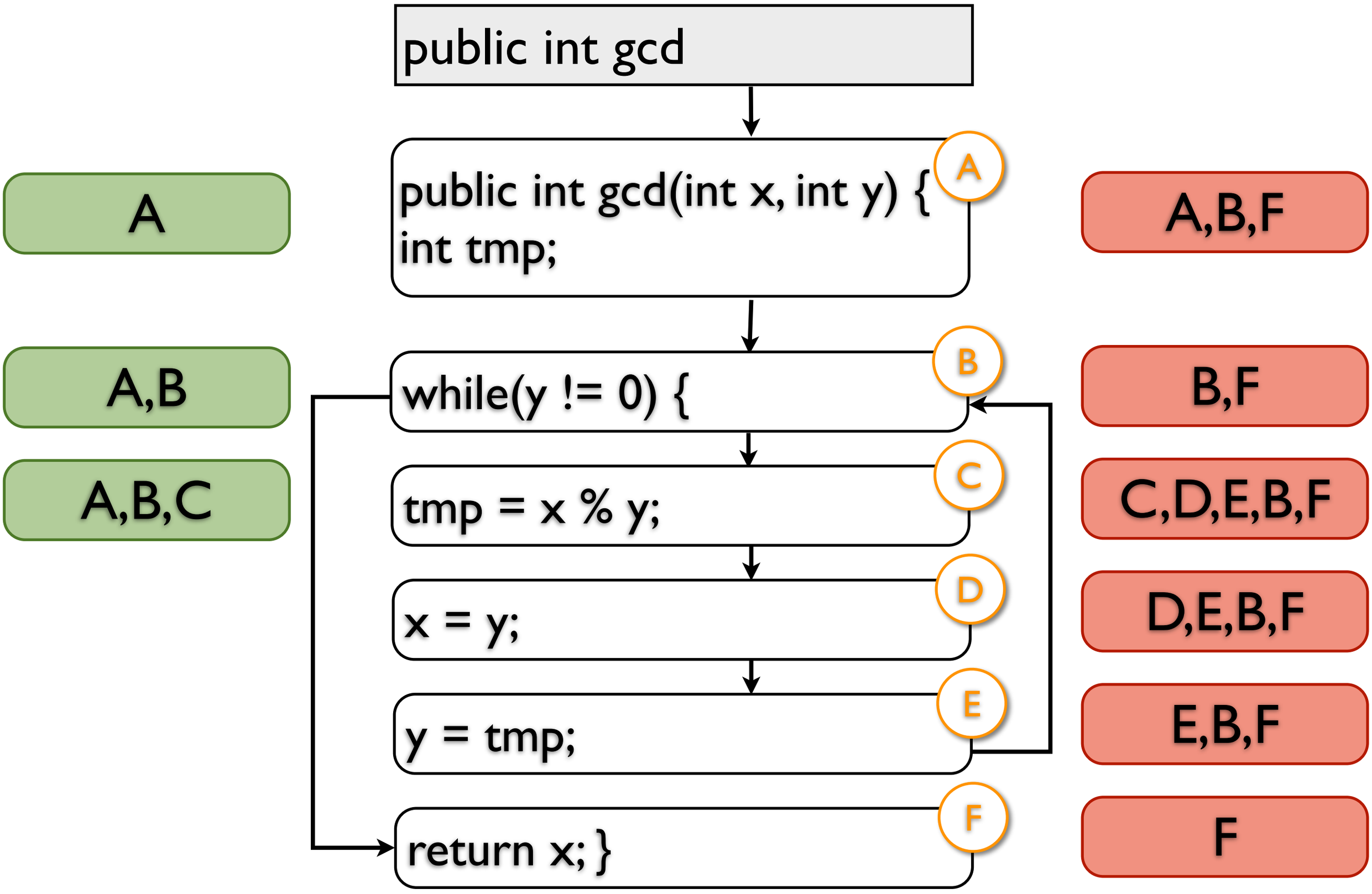

public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
C,D,E,B,F
D,E,B,F
E,B,F
F

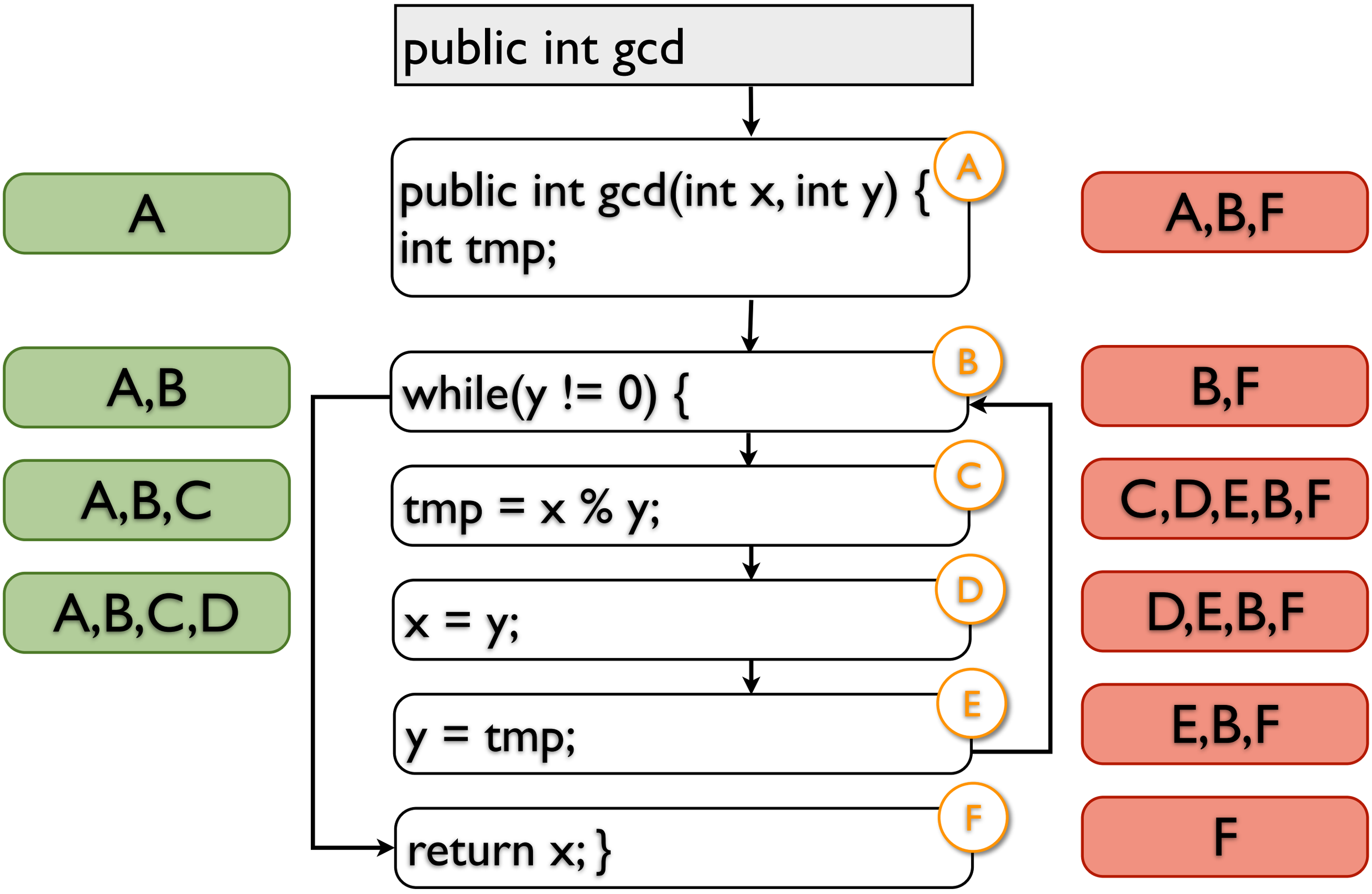

public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
B,F
C,D,E,B,F
D,E,B,F
E,B,F
F

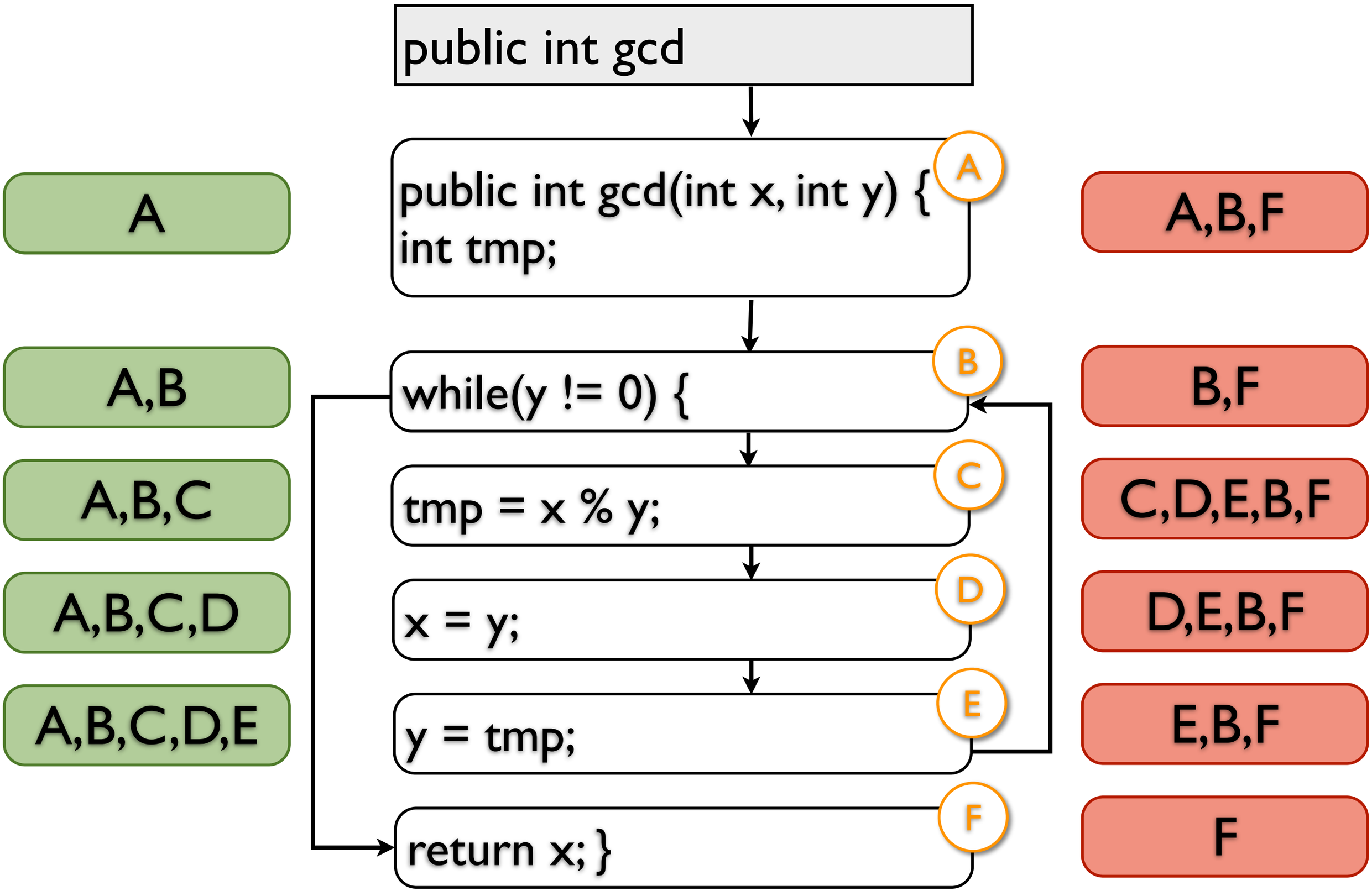

public int gcd
public int gcd(int x, int y) {
int tmp;
A
A,B,F
while(y != 0) {
B
B,F
tmp = x % y;
C
C,D,E,B,F
x = y;
D
D,E,B,F
y = tmp;
E
E,B,F
return x; }
F
F

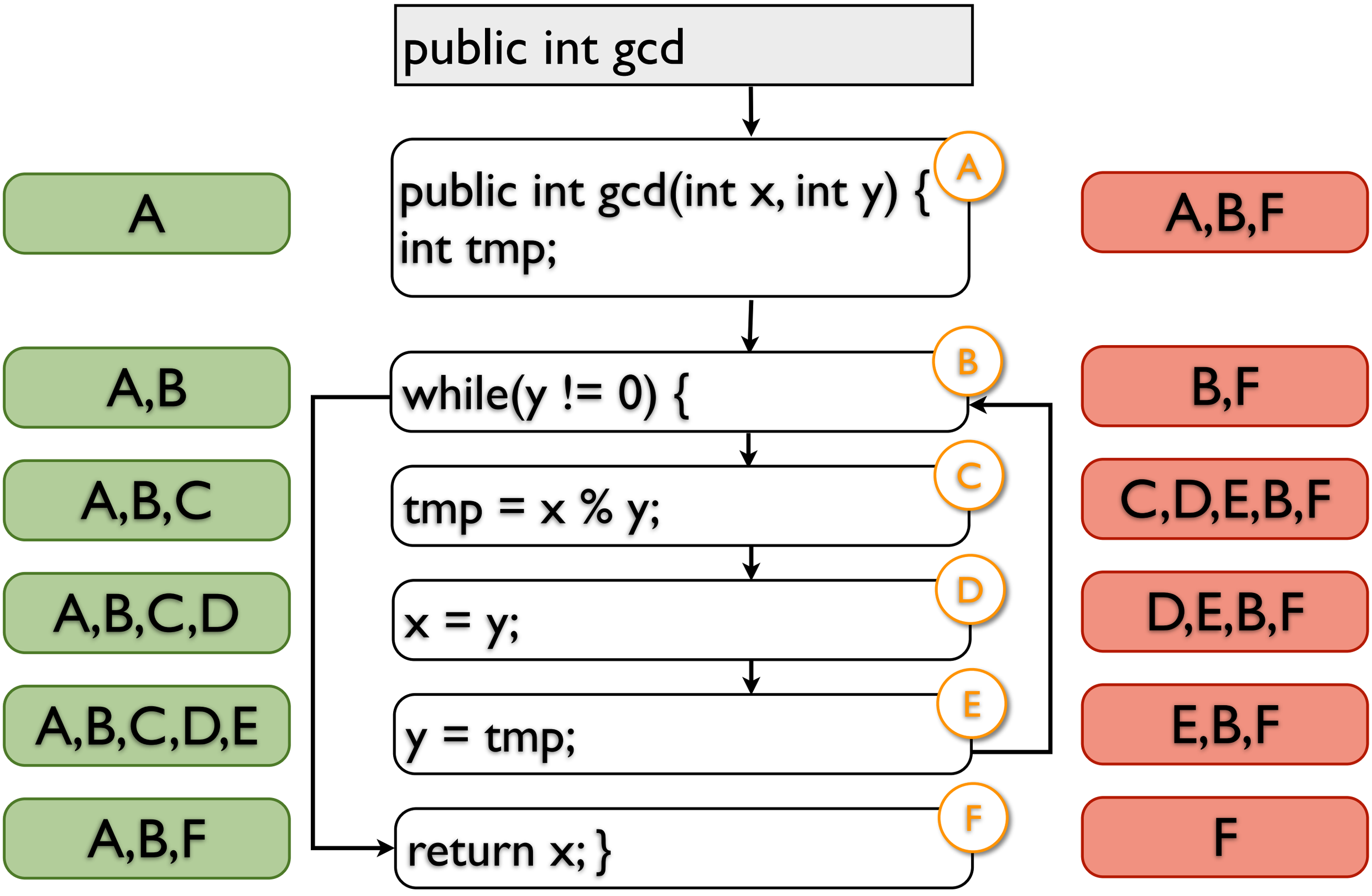

public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
A
A,B,F
B,F
C,D,E,B,F
D,E,B,F
E,B,F
F


public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
A
A,B
A,B,F
B,F
C,D,E,B,F
D,E,B,F
E,B,F
F


public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
A
A,B
A,B,C
A,B,F
B,F
C,D,E,B,F
D,E,B,F
E,B,F
F


public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
A
A,B
A,B,C
A,B,C,D
A,B,F
B,F
C,D,E,B,F
D,E,B,F
E,B,F
F


public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
A
A,B
A,B,C
A,B,C,D
A,B,C,D,E
A,B,F
B,F
C,D,E,B,F
D,E,B,F
E,B,F
F


public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
A
A,B
A,B,C
A,B,C,D
A,B,C,D,E
A,B,F
A,B,F
B,F
C,D,E,B,F
D,E,B,F
E,B,F
F

Calculating Post Dominance

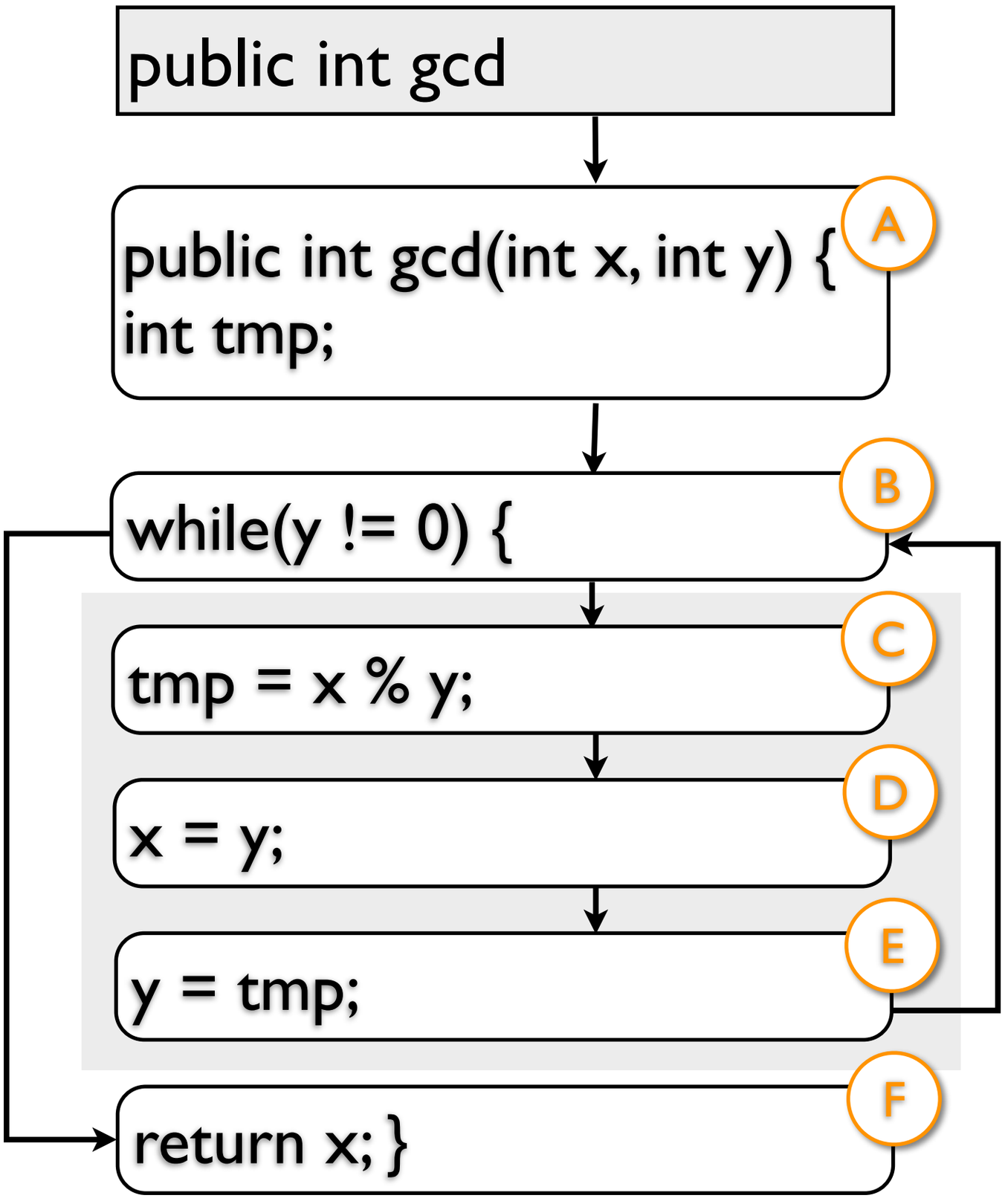
$$PDom(n_e) = \{n_e\}$$

$$PDom(n) = \left(\bigcap_{s \in succ(n)} PDom(s)\right) \cup \{n\}$$

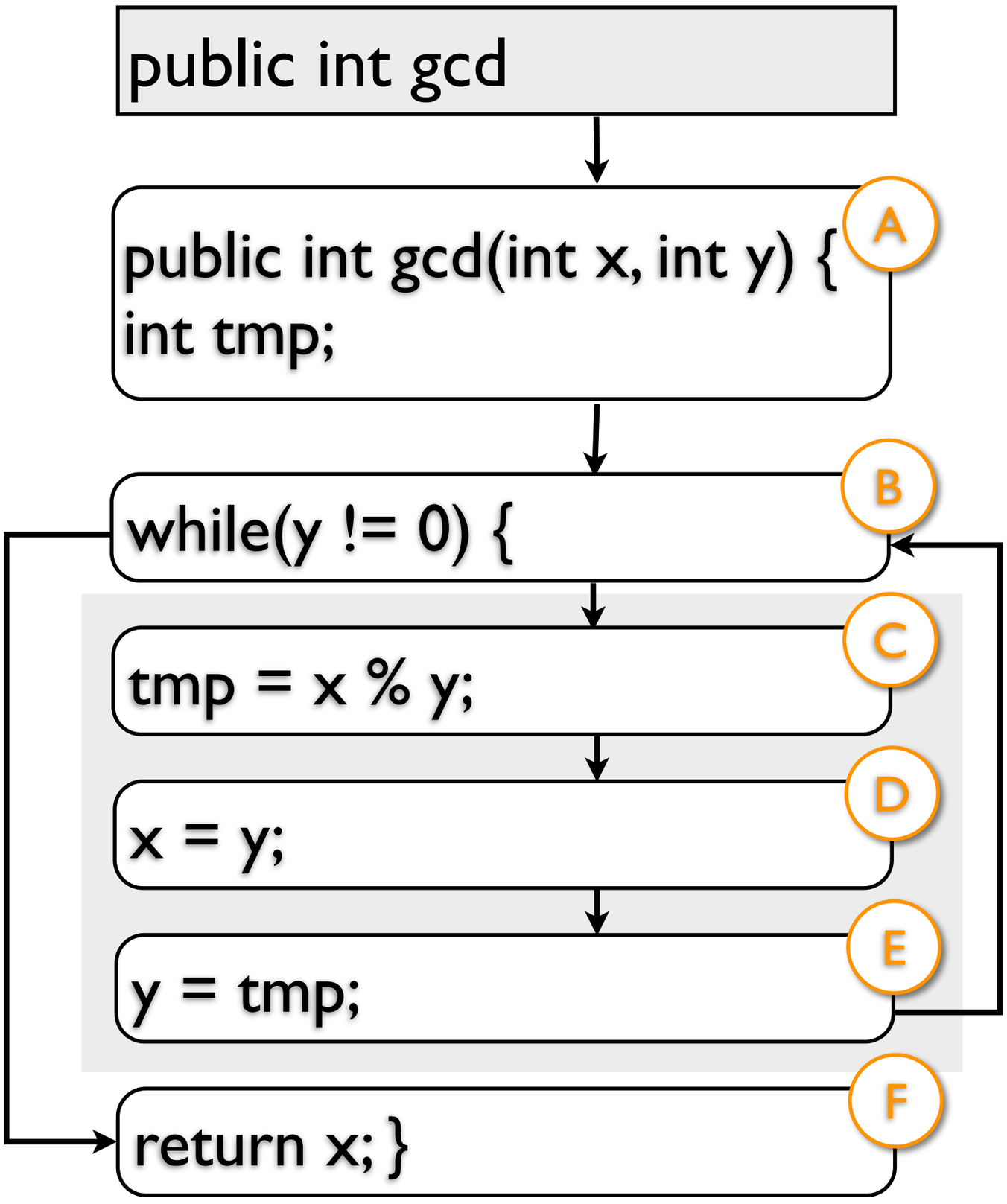
- Initialize PDom(n) with set of all nodes for all n except exit node n_e
- while some PDom(n) changed
  - recalculate PDom(n) for all n with above formula

Control Dependence

- A is control dependent on B if:
- B has at least two successors in the CFG
- B dominates A
- B is not post-dominated by A
- There is a successor of B that is post-dominated by A


public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F

A


public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F

A

A,B

```
public int gcd
public int gcd(int x, int y) {
int tmp;
while(y != 0) {
tmp = x % y;
x = y;
y = tmp;
return x; }
```

A

B

C

D

E

F

A

A,B

A,B,C

```
public int gcd
```

```
public int gcd(int x, int y) {
int tmp;
```

A

```
while(y != 0) {
```

B

```
tmp = x % y;
```

C

```
x = y;
```

D

```
y = tmp;
```

E

```
return x; }
```

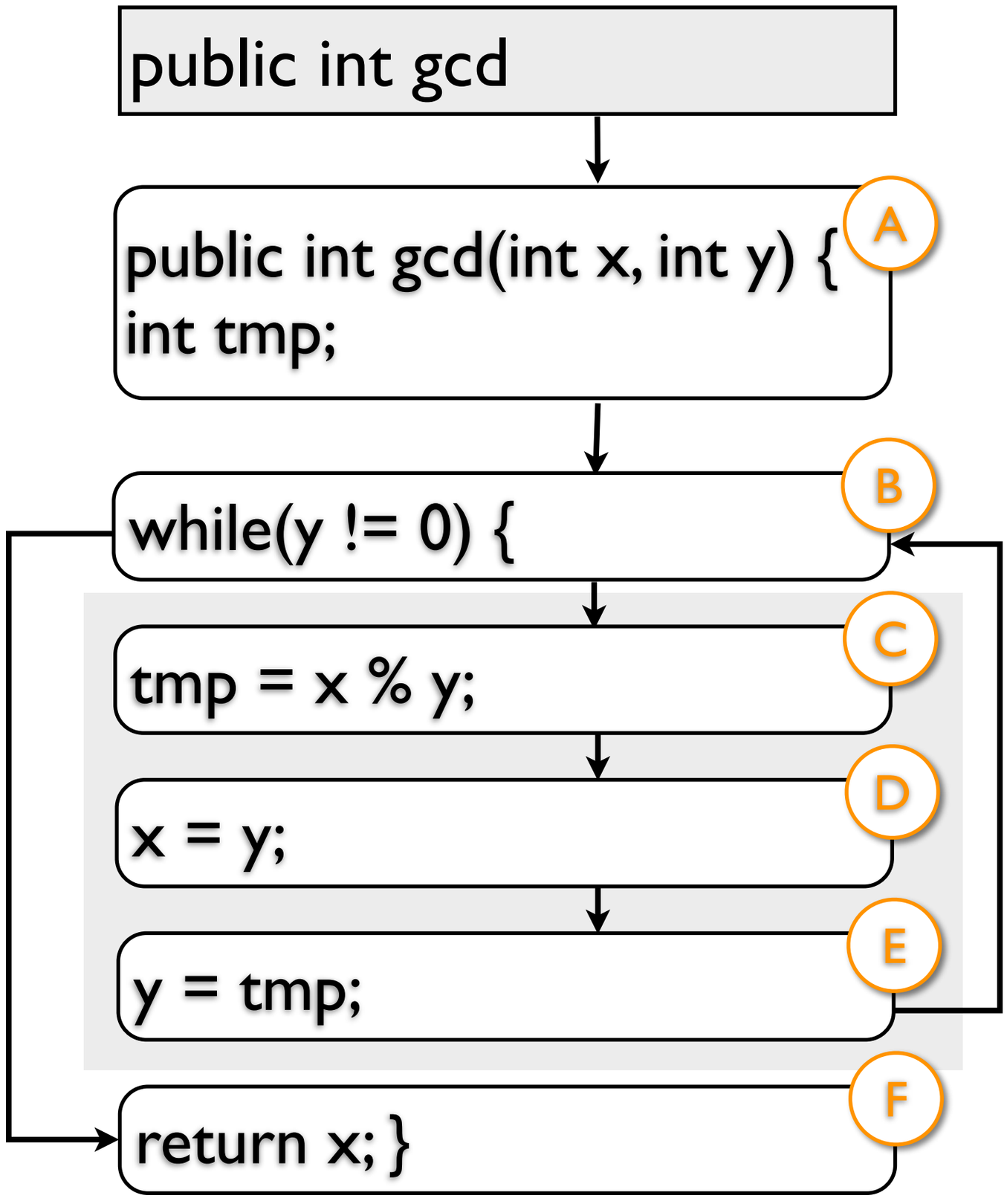
F

A

A,B

A,B,C

A,B,C,D


public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F

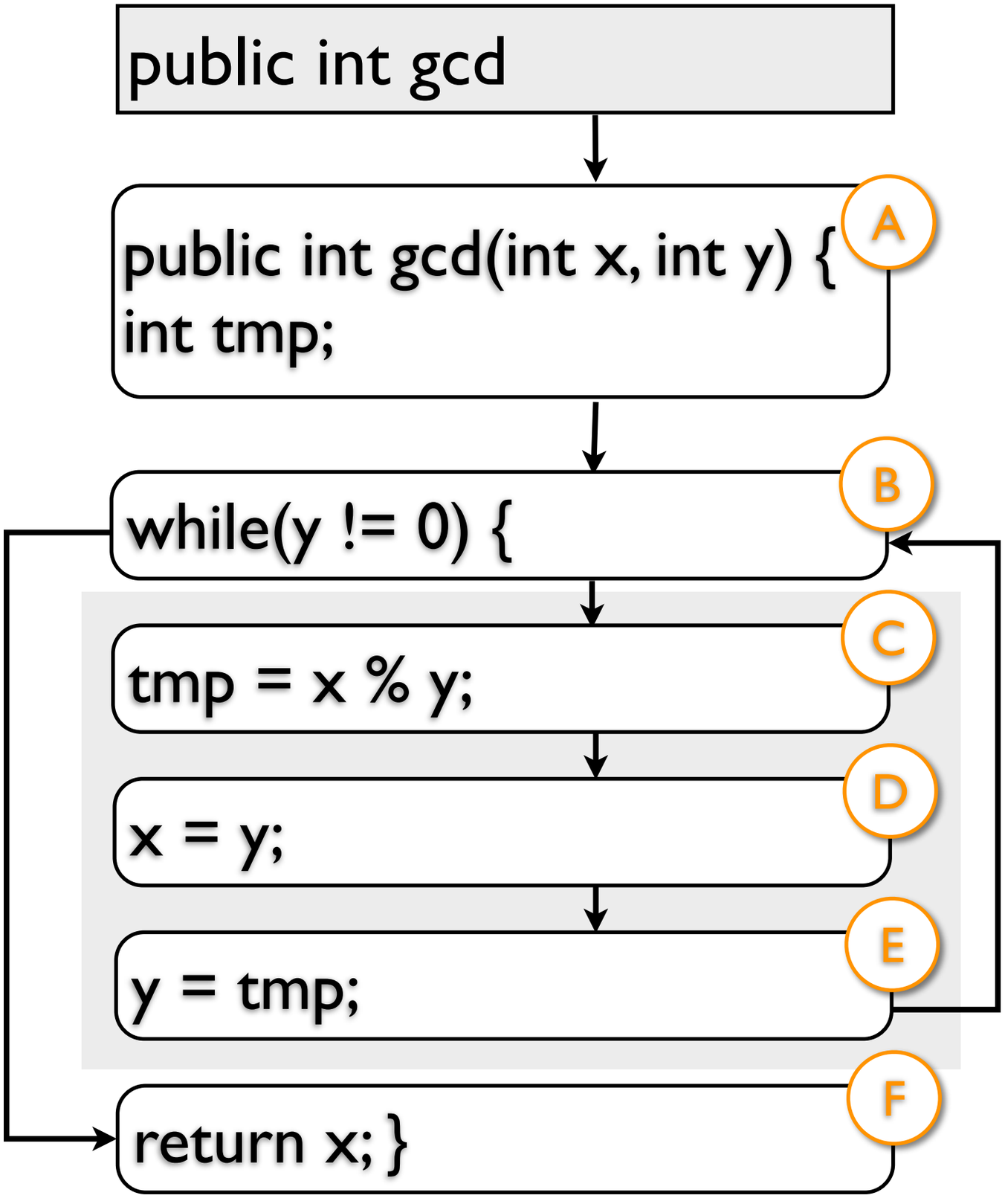
A

A,B

A,B,C

A,B,C,D

A,B,C,D,E



A

A,B

A,B,C

A,B,C,D

A,B,C,D,E

A,B,F

```
public int gcd
```

```
public int gcd(int x, int y) {
int tmp;
```

A

```
while(y != 0) {
```

B

```
tmp = x % y;
```

C

```
x = y;
```

D

```
y = tmp;
```

E

```
return x; }
```

F

A

A,B

A,B,C

A,B,C,D

A,B,C,D,E

A,B,F

```
public int gcd
```

```
public int gcd(int x, int y) {
int tmp;
```

A

```
while(y != 0) {
```

B

```
tmp = x % y;
```

C

```
x = y;
```

D

```
y = tmp;
```

E

```
return x; }
```

F

F

A

A,B

A,B,C

A,B,C,D

A,B,C,D,E

A,B,F

```
public int gcd
```

```
public int gcd(int x, int y) {
int tmp;
```

A

```
while(y != 0) {
```

B

```
tmp = x % y;
```

C

```
x = y;
```

D

```
y = tmp;
```

E

```
return x; }
```

F

E,B,F

F

A

A,B

A,B,C

A,B,C,D

A,B,C,D,E

A,B,F

```
public int gcd
```

```
public int gcd(int x, int y) {
int tmp;
```

A

```
while(y != 0) {
```

B

```
tmp = x % y;
```

C

```
x = y;
```

D

```
y = tmp;
```

E

```
return x; }
```

F

D,E,B,F

E,B,F

F

A

A,B

A,B,C

A,B,C,D

A,B,C,D,E

A,B,F

```
public int gcd
```

```
public int gcd(int x, int y) {
int tmp;
```

A

```
while(y != 0) {
```

B

```
tmp = x % y;
```

C

```
x = y;
```

D

```
y = tmp;
```

E

```
return x; }
```

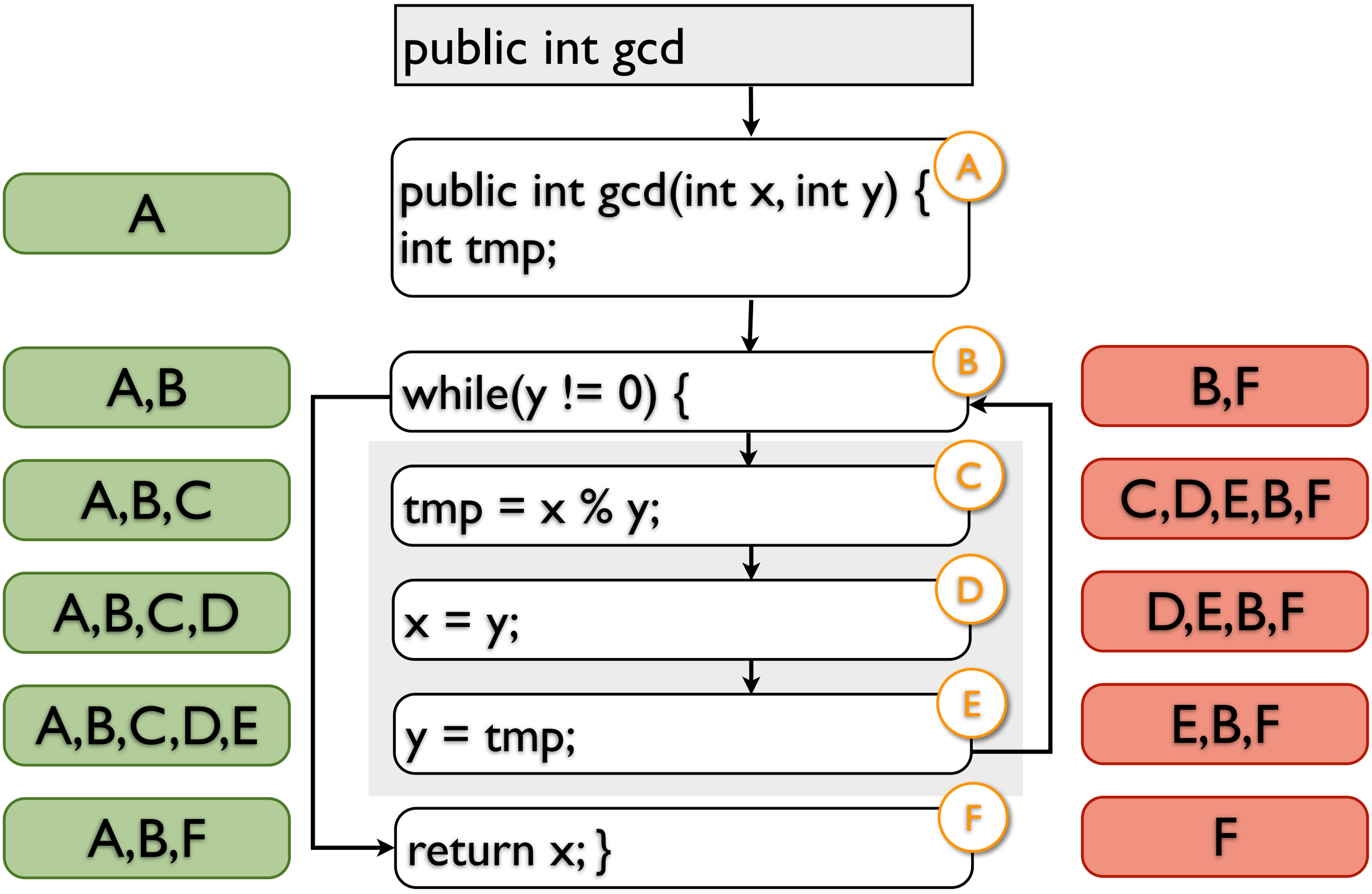
F

C,D,E,B,F

D,E,B,F

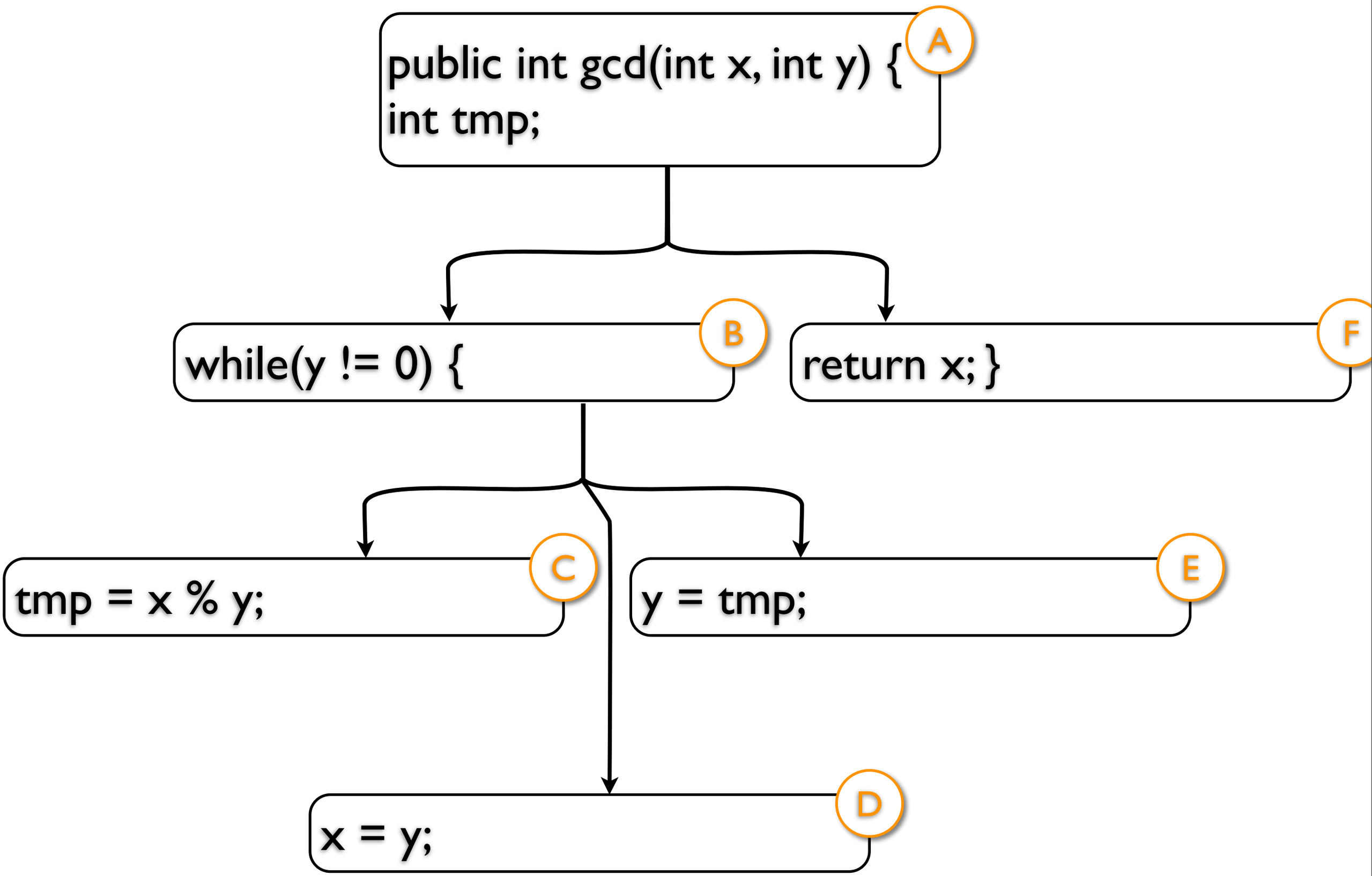
E,B,F

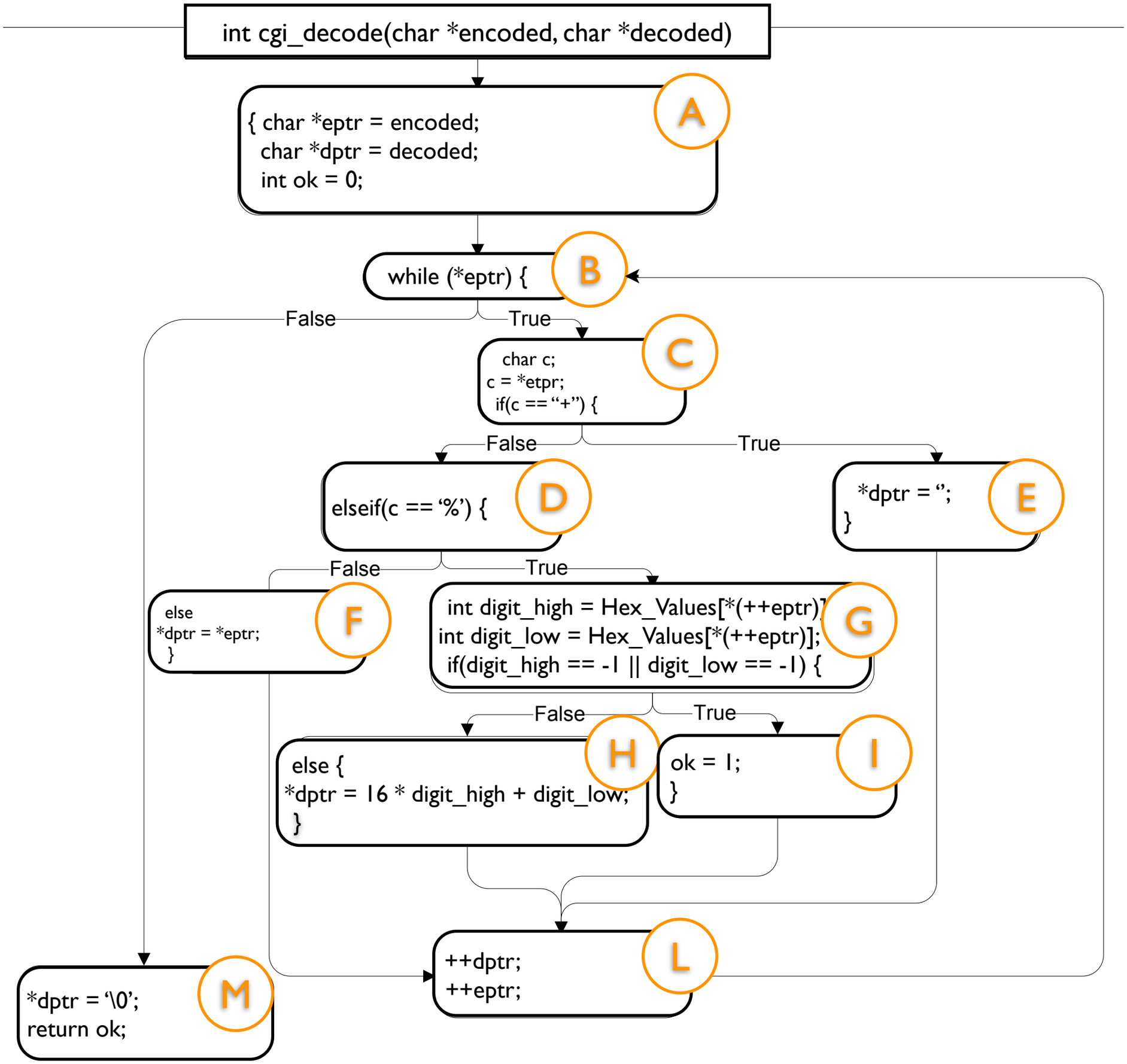
F

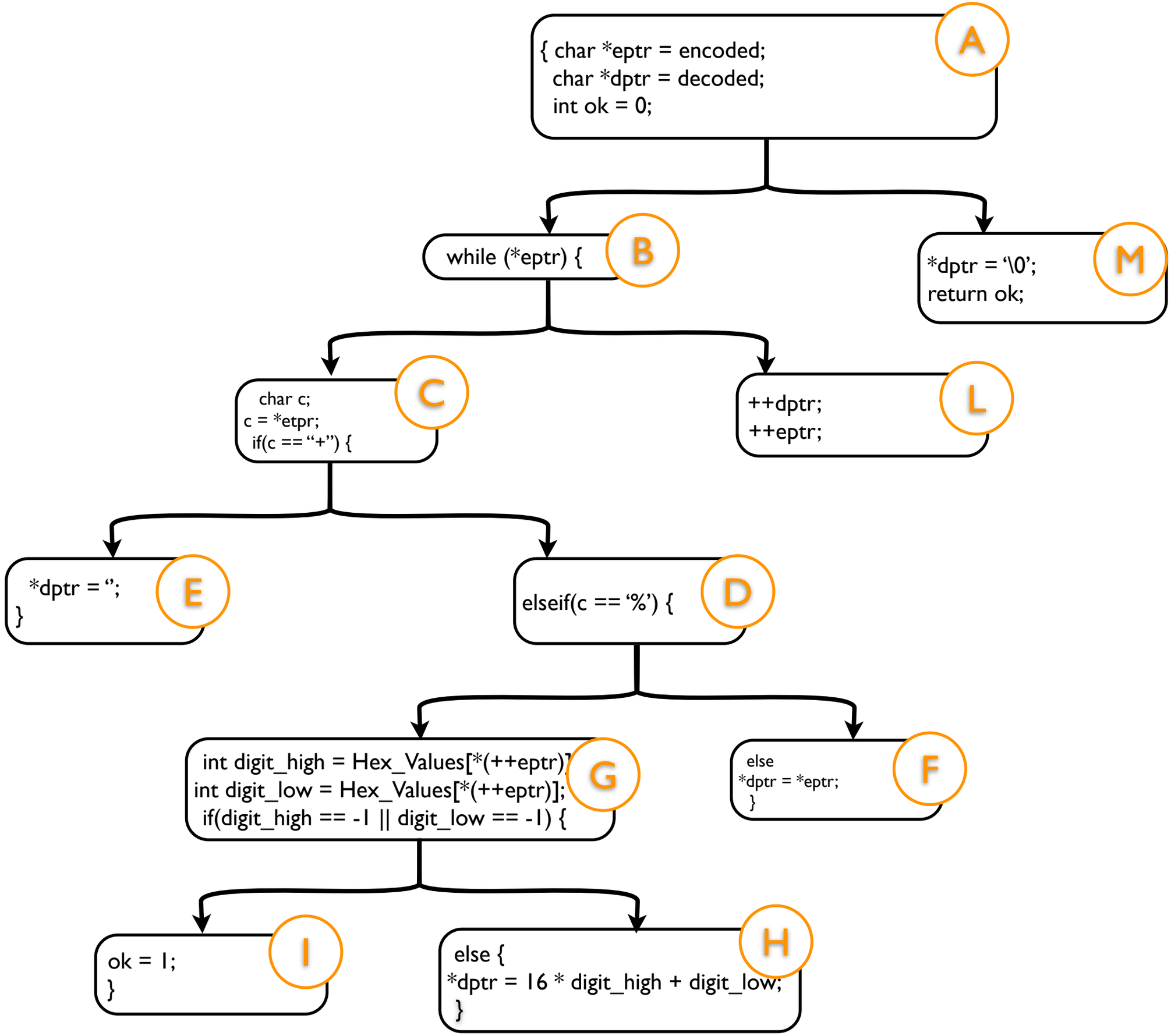

public int gcd
public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
tmp = x % y;
C
x = y;
D
y = tmp;
E
return x; }
F
A
A,B
A,B,C
A,B,C,D
A,B,C,D,E
A,B,F
B,F
C,D,E,B,F
D,E,B,F
E,B,F
F

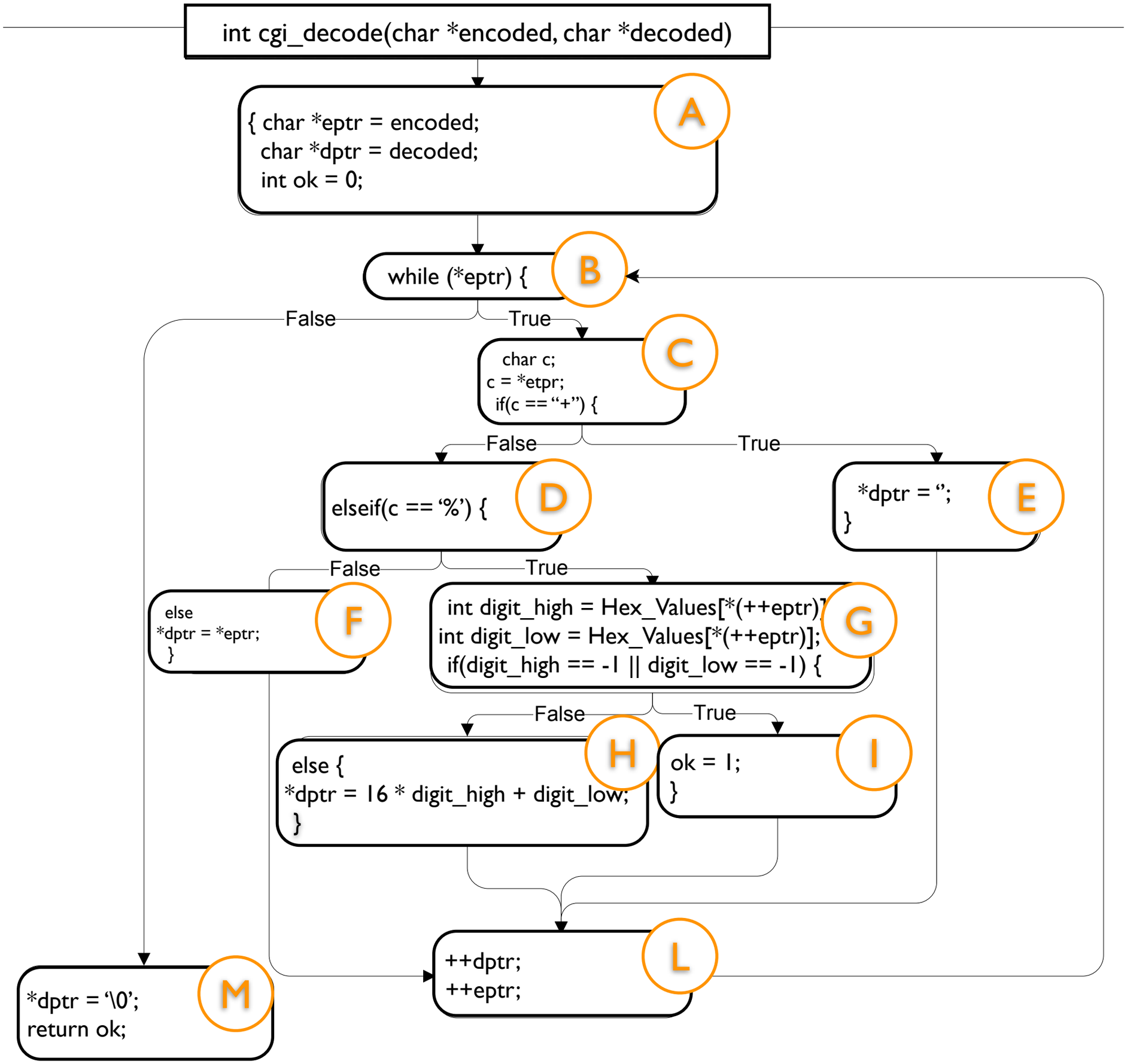
```
public int gcd
public int gcd(int x, int y) {          A
int tmp;
while(y != 0) {                         B
tmp = x % y;                            C
x = y;                                  D
y = tmp;                                E
return x; }                             F
```

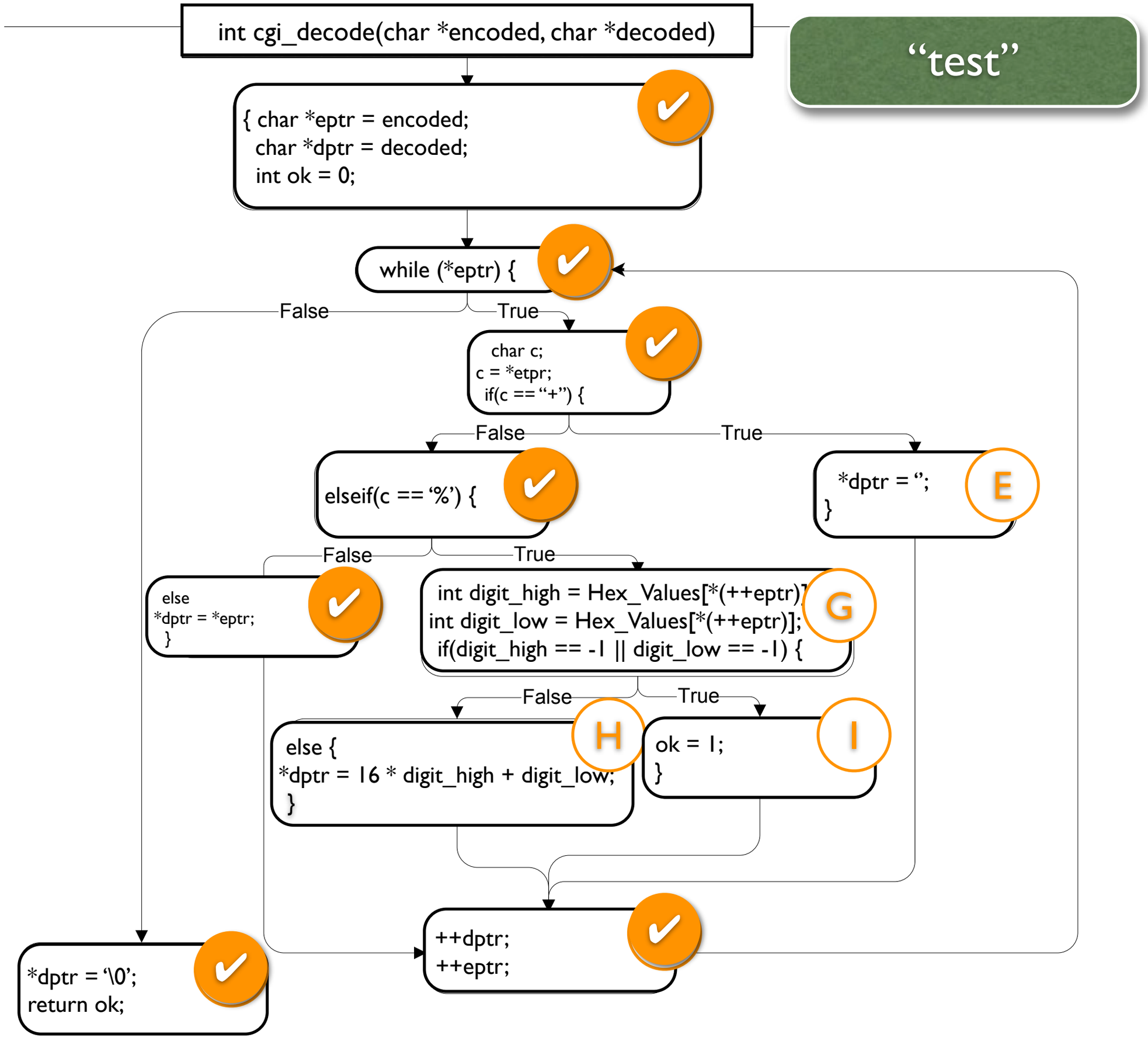
A	A,B,F
A,B	B,F
A,B,C	C,D,E,B,F
A,B,C,D	D,E,B,F
A,B,C,D,E	E,B,F
A,B,F	F

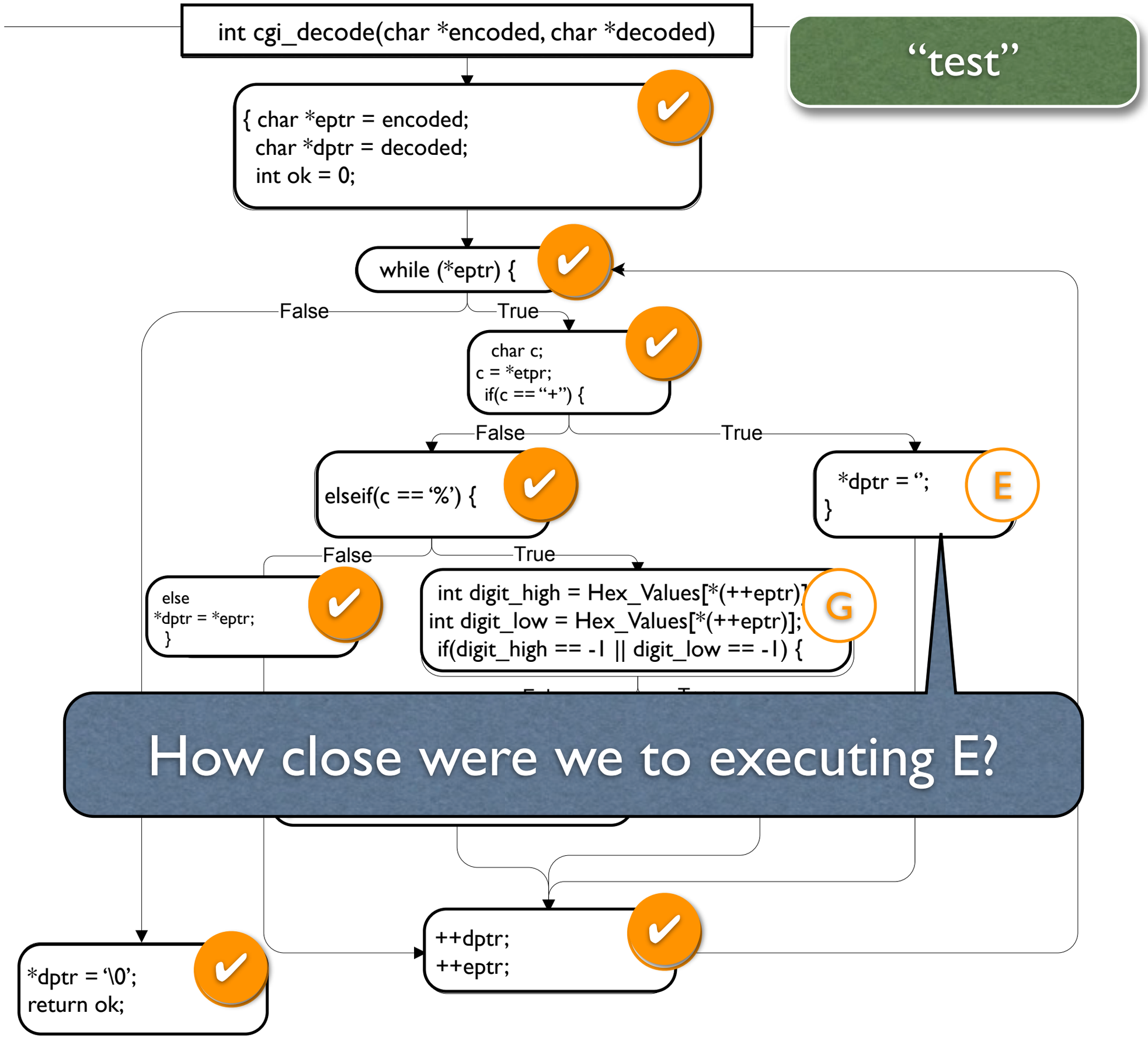

public int gcd(int x, int y) {
int tmp;
A
while(y != 0) {
B
return x; }
F
tmp = x % y;
C
y = tmp;
E
x = y;
D

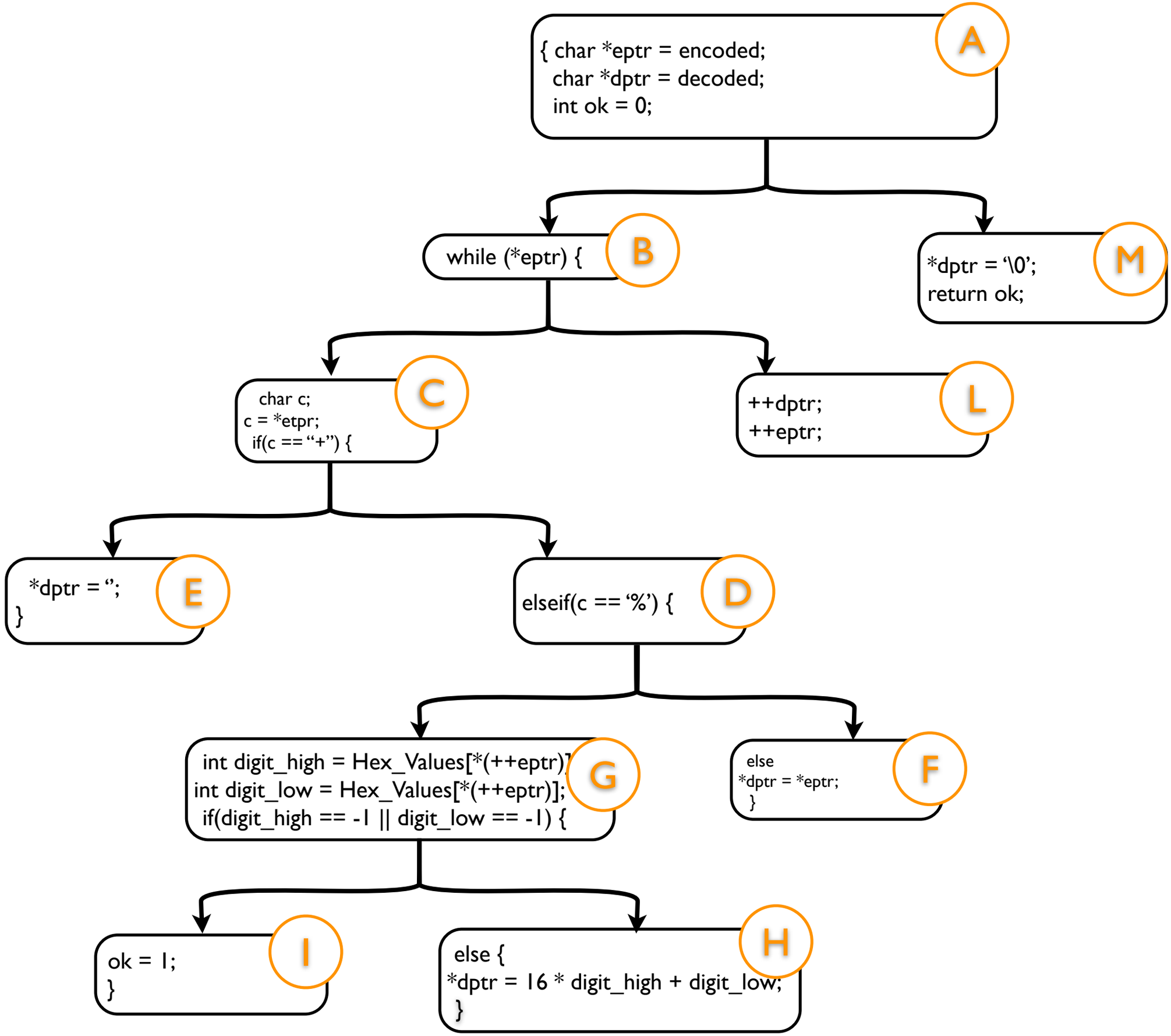

int cgi_decode(char *encoded, char *decoded)
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
A
while (*eptr) {
B
False
True
char c;
c = *etpr;
if(c == "+") {
C
False
True
elseif(c == '%') {
D
*dptr = '';
}
E
False
True
else
*dptr = *eptr;
}
F
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if(digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
L
*dptr = '\0';
return ok;
M


{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
A
while (*eptr) {
B
*dptr = '\0';
return ok;
M
char c;
c = *etpr;
if(c == "+") {
C
++dptr;
++eptr;
L
*dptr = '';
}
E
elseif(c == '%') {
D
int digit_high = Hex_Values[*(++eptr)]
int digit_low = Hex_Values[*(++eptr)];
if(digit_high == -1 || digit_low == -1) {
G
else
*dptr = *eptr;
}
F
ok = 1;
}
I
else {
*dptr = 16 * digit_high + digit_low;
}
H

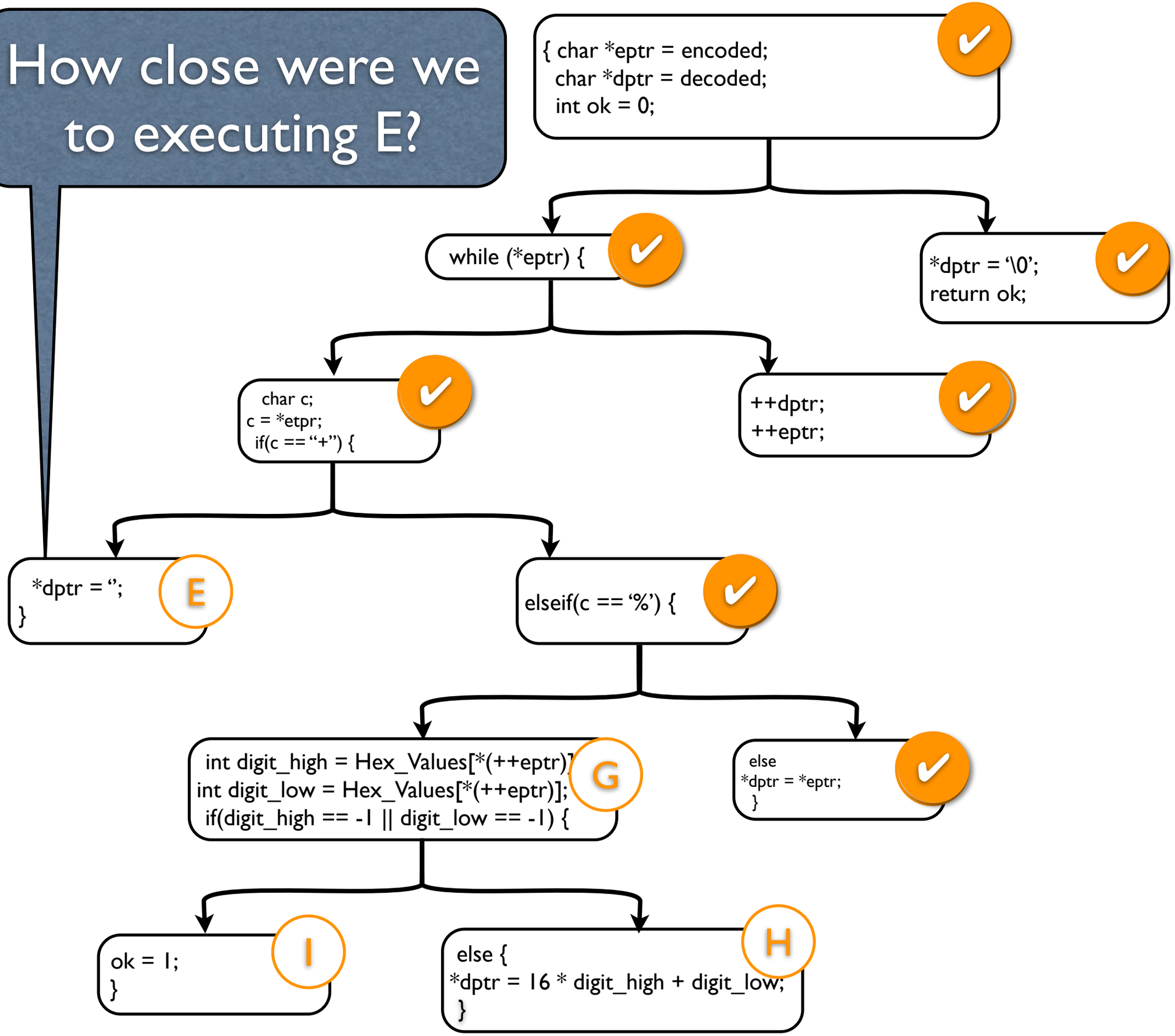

int cgi_decode(char *encoded, char *decoded)
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
A
while (*eptr) {
B
False
True
char c;
c = *etpr;
if(c == "+") {
C
False
True
elseif(c == '%') {
D
*dptr = '';
}
E
False
True
else
*dptr = *eptr;
}
F
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if(digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
L
*dptr = '\0';
return ok;
M


int cgi_decode(char *encoded, char *decoded)
"test"
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
while (*eptr) {
False
True
char c;
c = *etpr;
if(c == "+") {
False
True
elseif(c == '%') {
*dptr = '';
}
E
False
True
else
*dptr = *eptr;
}
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if(digit_high == -1 || digit_low == -1) {
G
False
True
else {
*dptr = 16 * digit_high + digit_low;
}
H
ok = 1;
}
I
++dptr;
++eptr;
*dptr = '\0';
return ok;

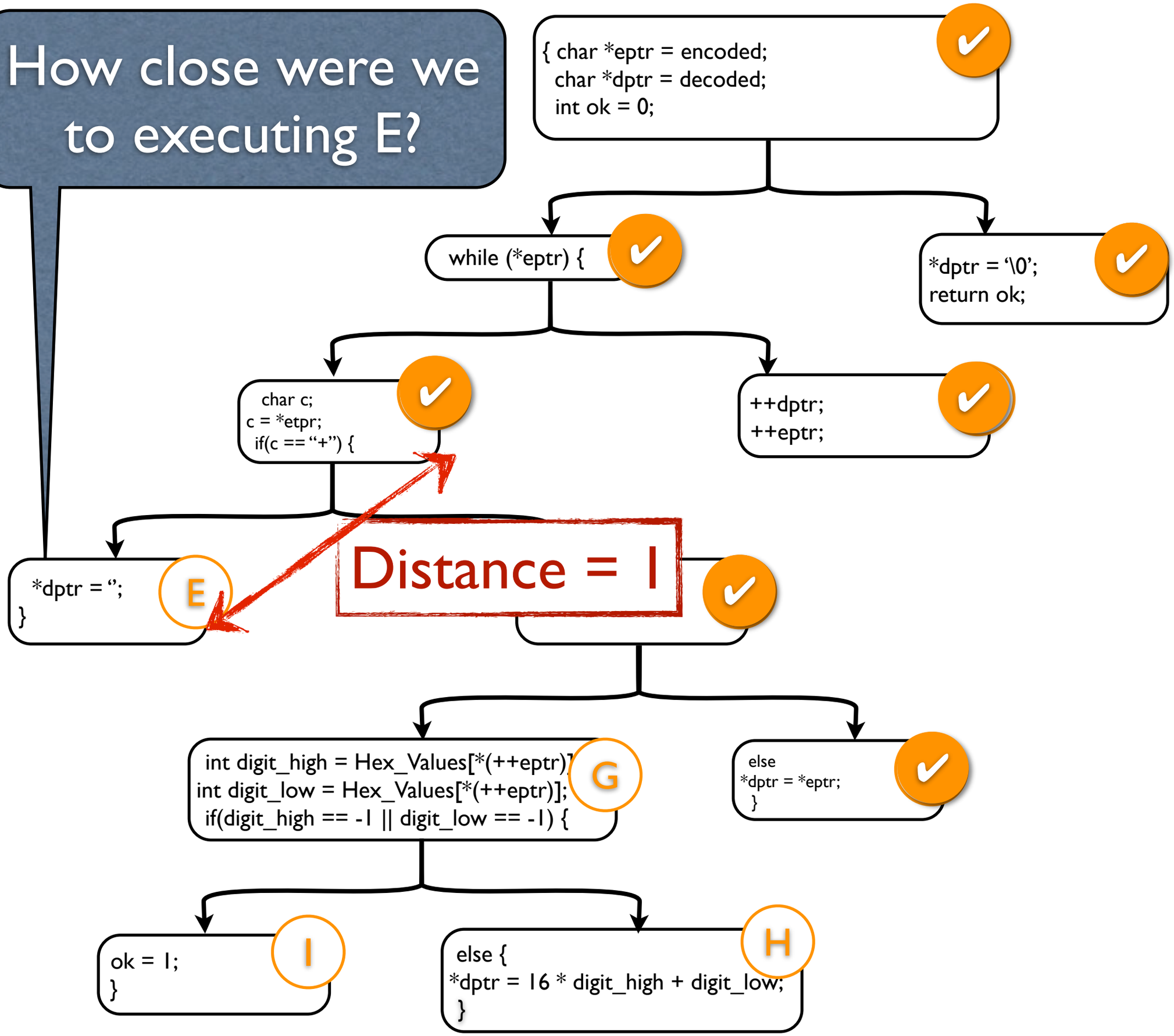

int cgi_decode(char *encoded, char *decoded)
"test"
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
while (*eptr) {
False
True
char c;
c = *etpr;
if(c == "+") {
False
True
elseif(c == '%') {
*dptr = '';
}
E
False
True
else
*dptr = *eptr;
}
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if(digit_high == -1 || digit_low == -1) {
G
How close were we to executing E?
++dptr;
++eptr;
*dptr = '\0';
return ok;


{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;
A
while (*eptr) {
B
*dptr = '\0';
return ok;
M
char c;
c = *etpr;
if(c == "+") {
C
++dptr;
++eptr;
L
*dptr = '';
}
E
elseif(c == '%') {
D
int digit_high = Hex_Values[*(++eptr)]
int digit_low = Hex_Values[*(++eptr)];
if(digit_high == -1 || digit_low == -1) {
G
else
*dptr = *eptr;
}
F
ok = 1;
}
I
else {
*dptr = 16 * digit_high + digit_low;
}
H

How close were we to executing E?



How close were we to executing E?

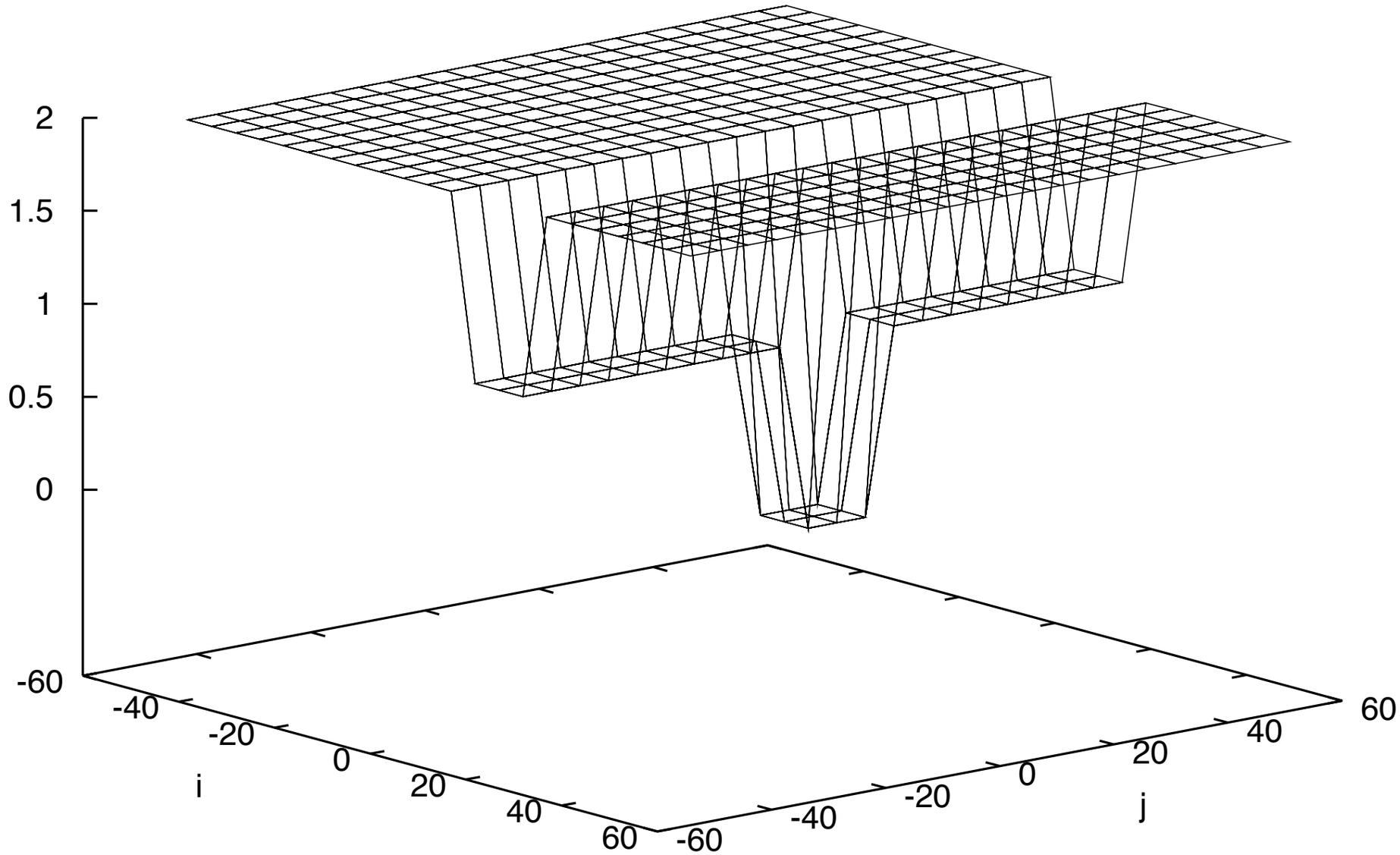


Approach Level

- Number of control dependent edges between goal and chosen path
- Approach = Number of dependent nodes - number of executed nodes

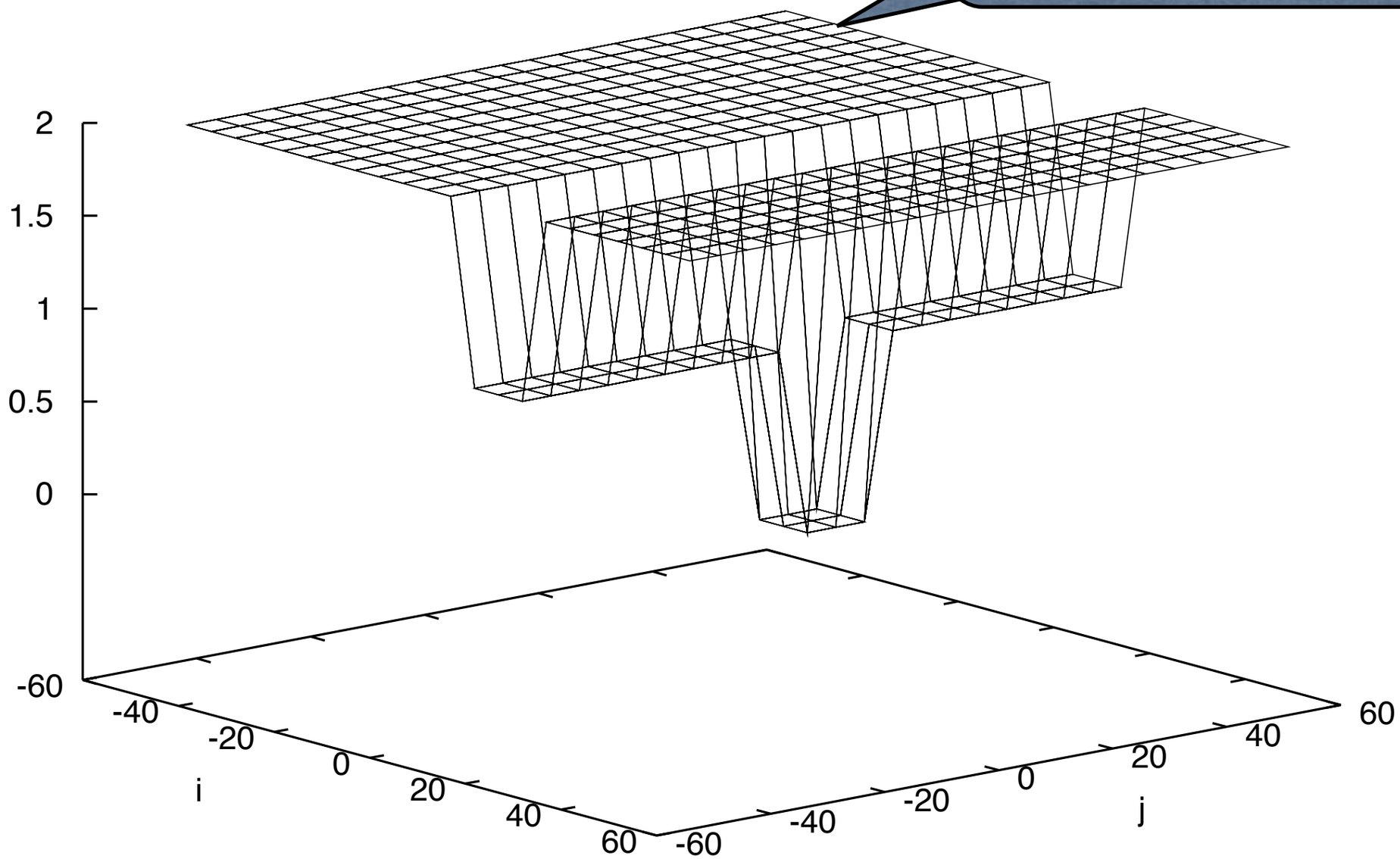
```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      //  ...
    }
  }
}
```

P. McMinn, Search-based Software Testing: A Survey



P. McMinn, Search-based Software Testing: A Survey

Plateaux are problematic for search



P. McMinn, Search-based Software Testing: A Survey

```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      //  ...
    }
  }
}
```

P. McMinn, Search-based Software Testing: A Survey

```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      // ...
    }
  }
}
```

How close is this predicate to being true?

P. McMinn, Search-based Software Testing: A Survey

Branch Distance

- Critical branch = branch where control flow diverged from reaching target
- Distance to branch = distance to predicate being true / false
- Distance metric for logical formulas
- E.g. distance from true - false = 1

Branch distance

a = b	abs(a-b) = 0 ? 0 : abs(a-b) + K
a != b	abs(a-b) != 0 ? 0 else K
a < b	a - b < 0 ? 0 : (a - b) + K
a <= b	a - b <= 0 ? 0 : (a - b) + K
a > b	b - a > 0 ? 0 : (b - a) + K
a >= b	b - a >= 0 ? 0 : (b - a) + K

Branch distance

Boolean	true ? 0 : K
A && B	distance(A) + distance(B)
A || B	min(distance(A), distance(B))
!a	Move inward and propagate

Example

j = 5

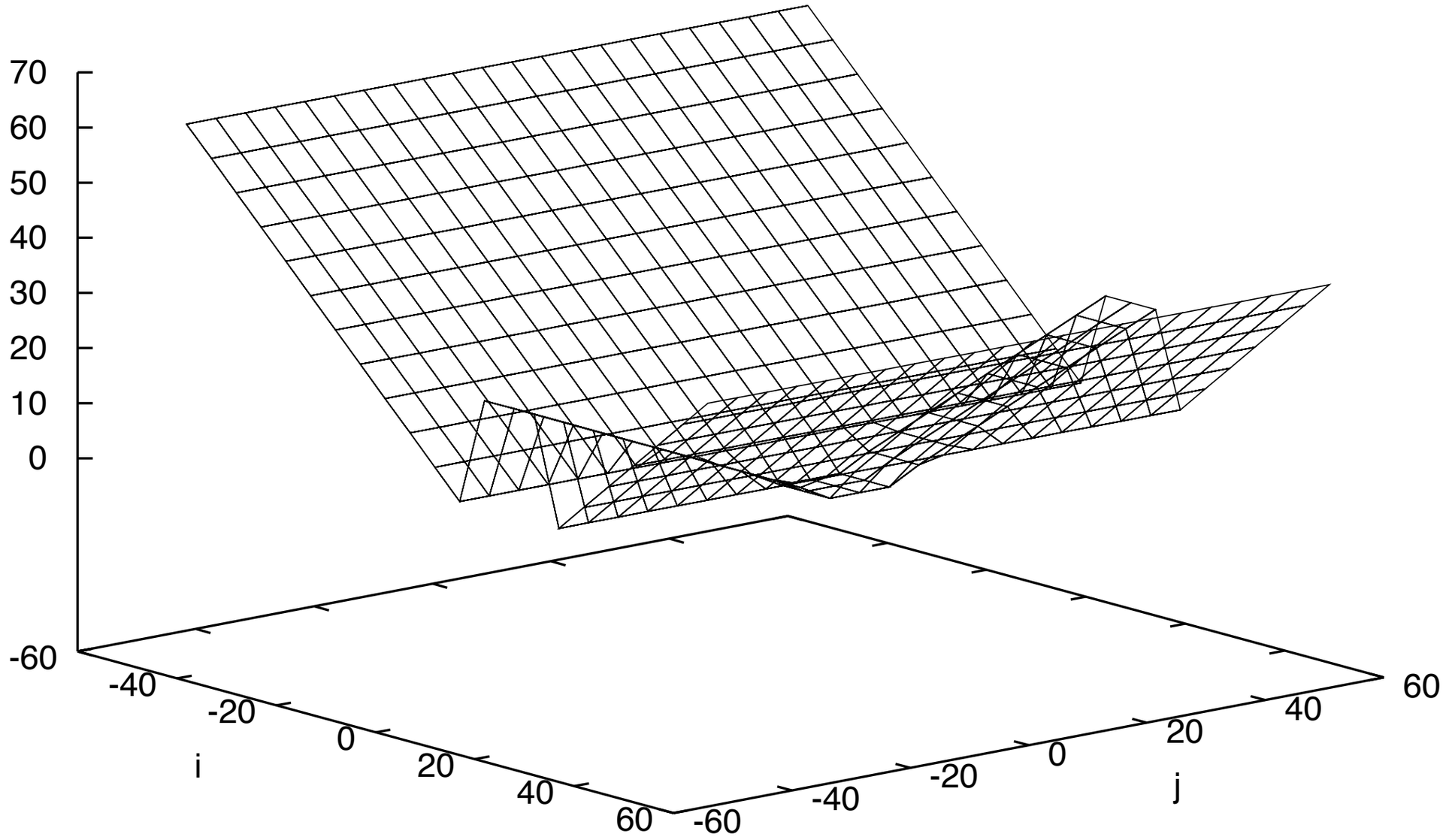
- `j >= 0`
- `j >= 10`
- `j >= 10 && j <= 20`

Branch Distance

- How close is the last branch that would lead closer to the goal to being taken?
- How do we make use of the branch distance predicates?

```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      //  ...
    }
  }
}
```

P. McMinn, Search-based Software Testing: A Survey



P. McMinn, Search-based Software Testing: A Survey

Control+Branch Distance

- Control distance results in plateaux
- Branch distance results in local optima
- Combination!
- (dependent - executed) + branch_distance

Control+Branch Distance

- Control distance results in plateaux
- Branch distance results in local optima
- Combination!
- (dependent - executed) + branch_distance

Approach level

Control+Branch Distance

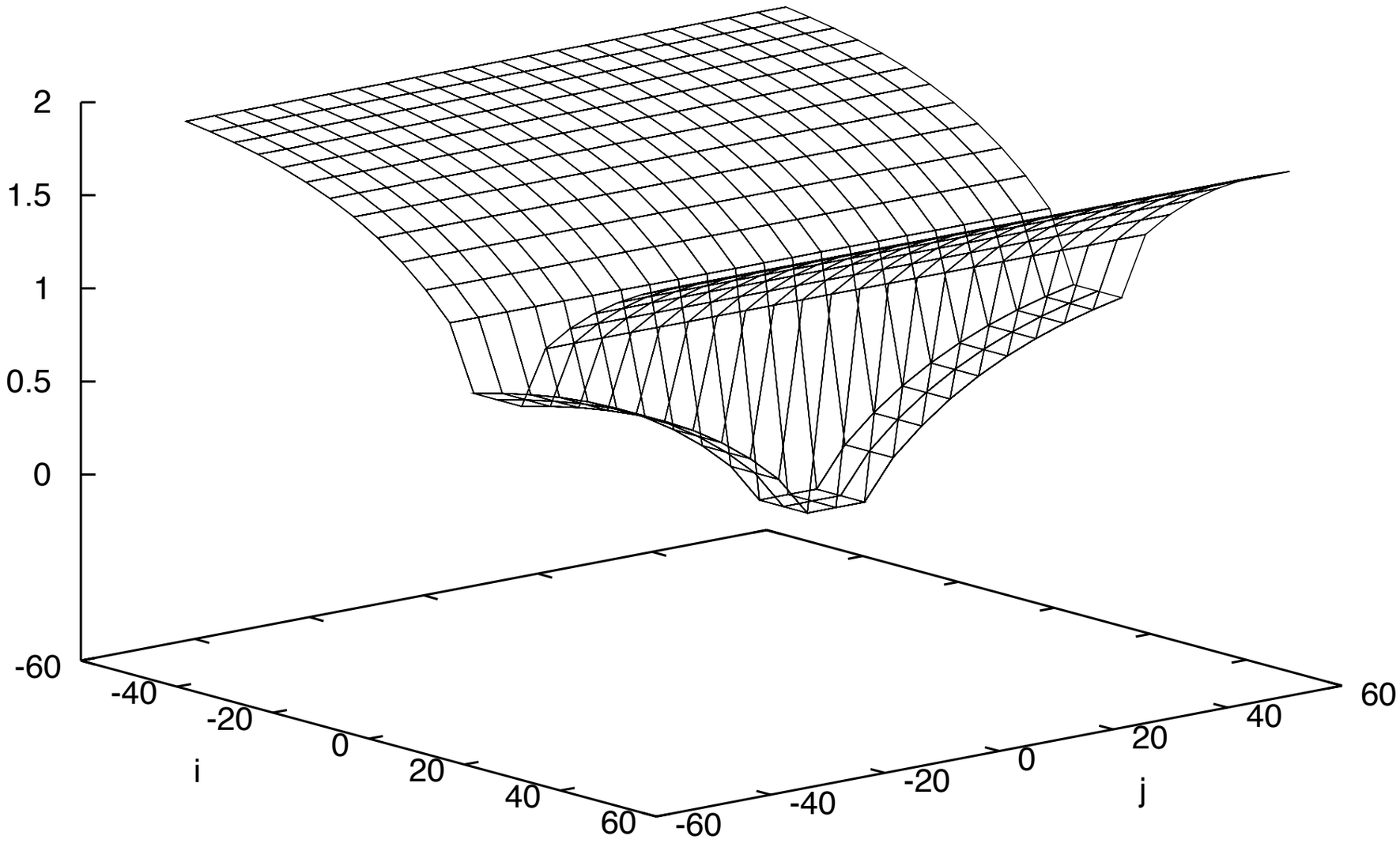
- Control distance results in plateaux
- Branch distance results in local optima
- Combination!
- (dependent - executed) + branch_distance

Approach level

Branch distance at node of diversion

```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      //  ...
      }
    }
  }
```

P. McMinn, Search-based Software Testing: A Survey



P. McMinn, Search-based Software Testing: A Survey

```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      // ...
    }
  }
}
```

Which one is closer?

```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      // ...
    }
  }
}
```

- i = 0, j = 0

Which one is closer?

```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      // ...
    }
  }
}
```

- i = 0, j = 0
- Approach level: 2

Which one is closer?

```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      // ...
    }
  }
}
```

- i = 0, j = 0
- Approach level: 2
- Branch distance: 10

Which one is closer?

```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      // ...
    }
  }
}
```

- i = 0, j = 0
- Approach level: 2
- Branch distance: 10

- i = 10, j = 25

Which one is closer?

```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      // ...
    }
  }
}
```

- i = 0, j = 0
- Approach level: 2
- Branch distance: 10

- i = 10, j = 25
- Approach level: 1

Which one is closer?

```
void landscape_example(int i, int j) {
  if (10 <= i && i <= 20) {
    if (0 <= j && j <= 10) {
      // target statement
      // ...
    }
  }
}
```

- i = 0, j = 0
- Approach level: 2
- Branch distance: 10

- i = 10, j = 25
- Approach level: 1
- Branch distance: 15

Which one is closer?

Normalization Functions

- Approach level should be dominant in fitness
- The branch distance has to be normalized

Normalization Functions

- Approach level should be dominant in fitness
- The branch distance has to be normalized

$$1 - \alpha^{-x}$$

Normalization Functions

- Approach level should be dominant in fitness
- The branch distance has to be normalized

$$1 - \alpha^{-x}$$

$$\frac{x}{x + 1}$$

Normalization Functions

- Approach level should be dominant in fitness
- The branch distance has to be normalized

$$1 - \alpha^{-x}$$

$$\frac{x}{x + 1}$$

Use this!

Measuring Fitness

To measure the fitness of an individual, we need to

- Execute the test case
- Record the control flow path taken
- Record the branch distance at every branch
- Instrumentation is necessary for this

```
void landscape_example(int i, int j) {
  if (i >= 10 && i <= 20) {
    if (j >= 0 && j <= 10) {
      // target statement
      //  ...
    }
  }
}
```

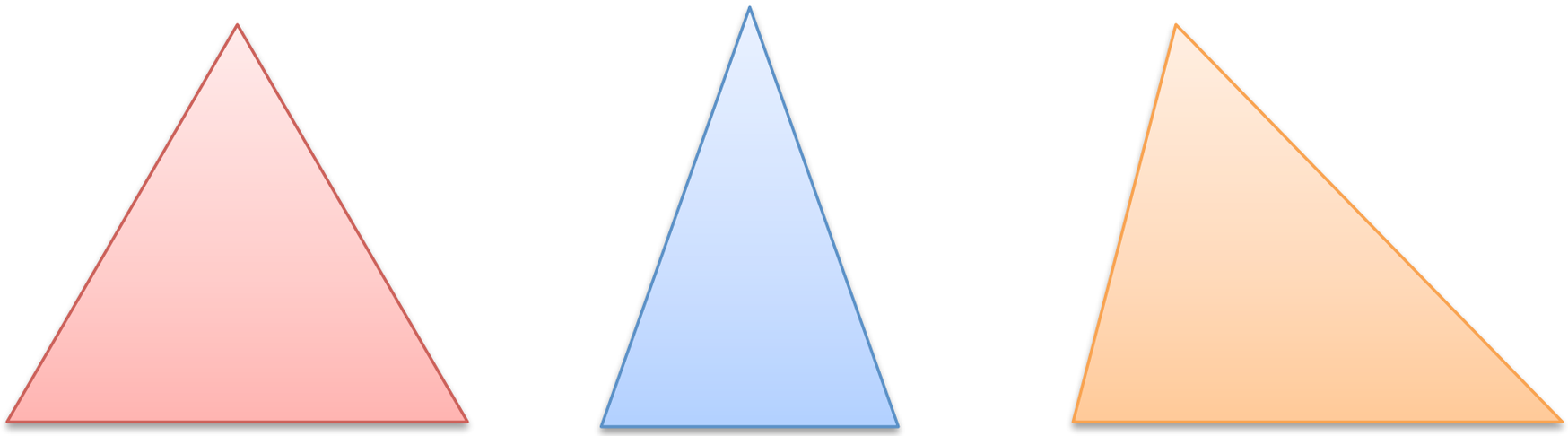
```
void landscape_example(int i, int j) {
  Trace(Entry);
  if (i >= 10 && i <= 20) {
    Trace(Branch1);
    if (j >= 0 && j <= 10) {
      Trace(Branch2);
      // target statement
      // ...
    }
  }
}
```

```
void landscape_example(int i, int j) {
  Trace(Entry);
  if (i >= 10 && i <= 20) {
    Trace(Branch1);
    if (j >= 0 && j <= 10) {
      Trace(Branch2);
      // target statement
      // ...
    }
  }
}
```

```
void landscape_example(int i, int j) {
  Trace(Entry);
  Distance(Branch1, (10 - i <= 0 ? 0 : 10 - i)
                  + (i - 20 <= 0 ? 0 : i - 20));
  if (i >= 10 && i <= 20) {
    Trace(Branch1);
    Distance(Branch2, (0-j <= 0 ? 0 : 0 - j)
                  + (j - 10 <= 0 ? 0 : j - 10))
    if (j >= 0 && j <= 10) {
      Trace(Branch2);
      // target statement
      // ...
    }
  }
}
```

Example

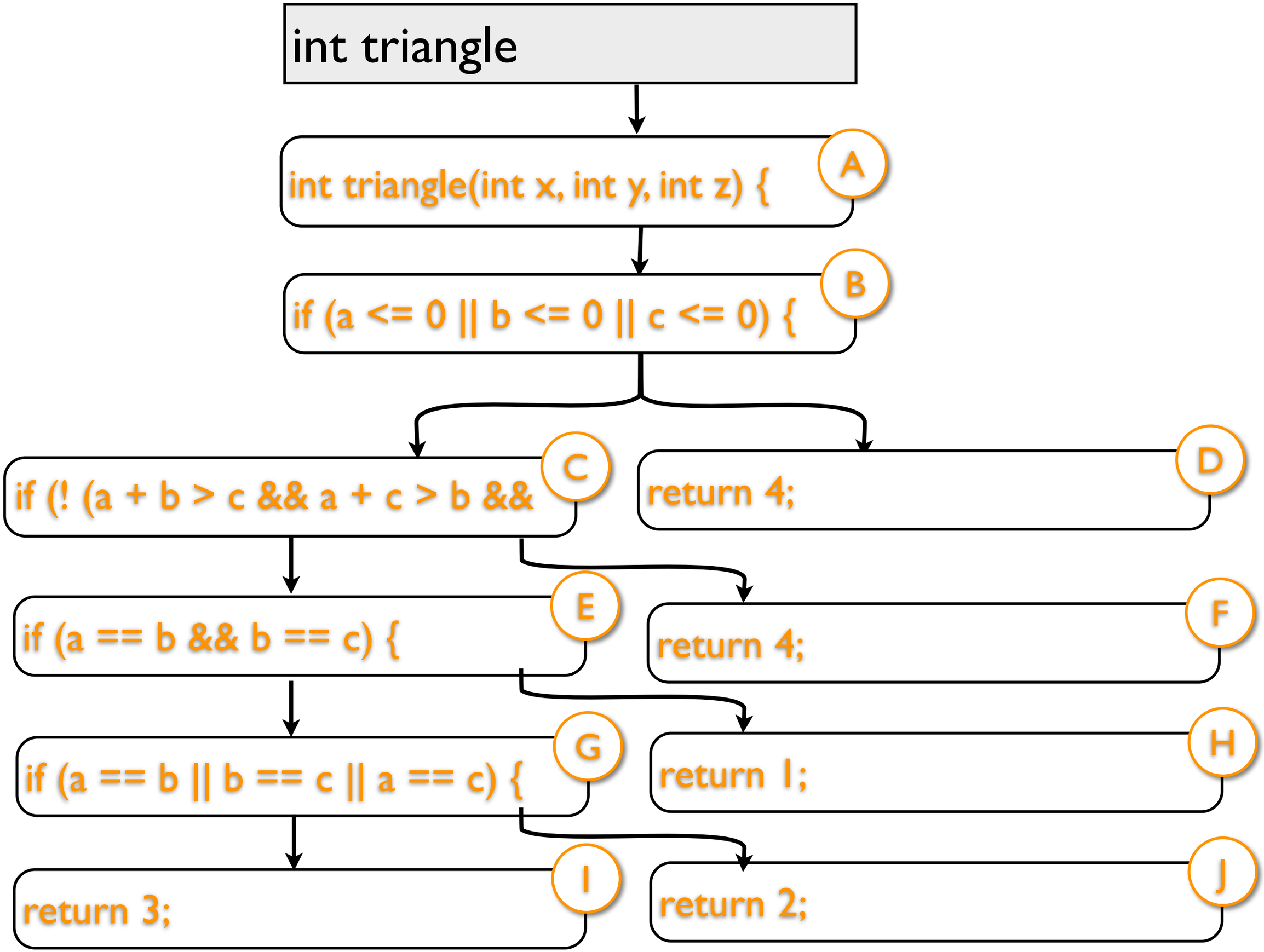
Classify triangle by the length of the sides

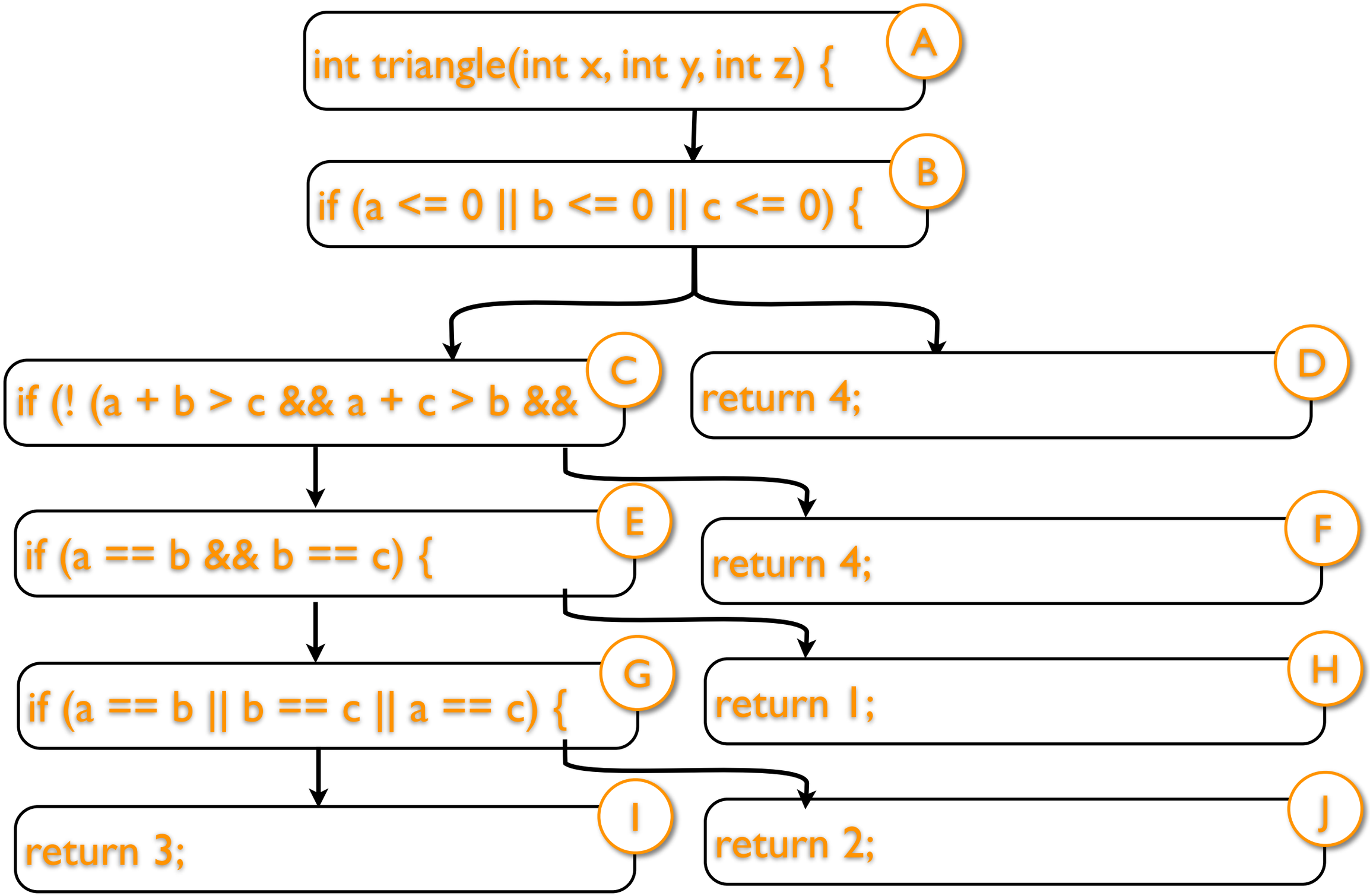


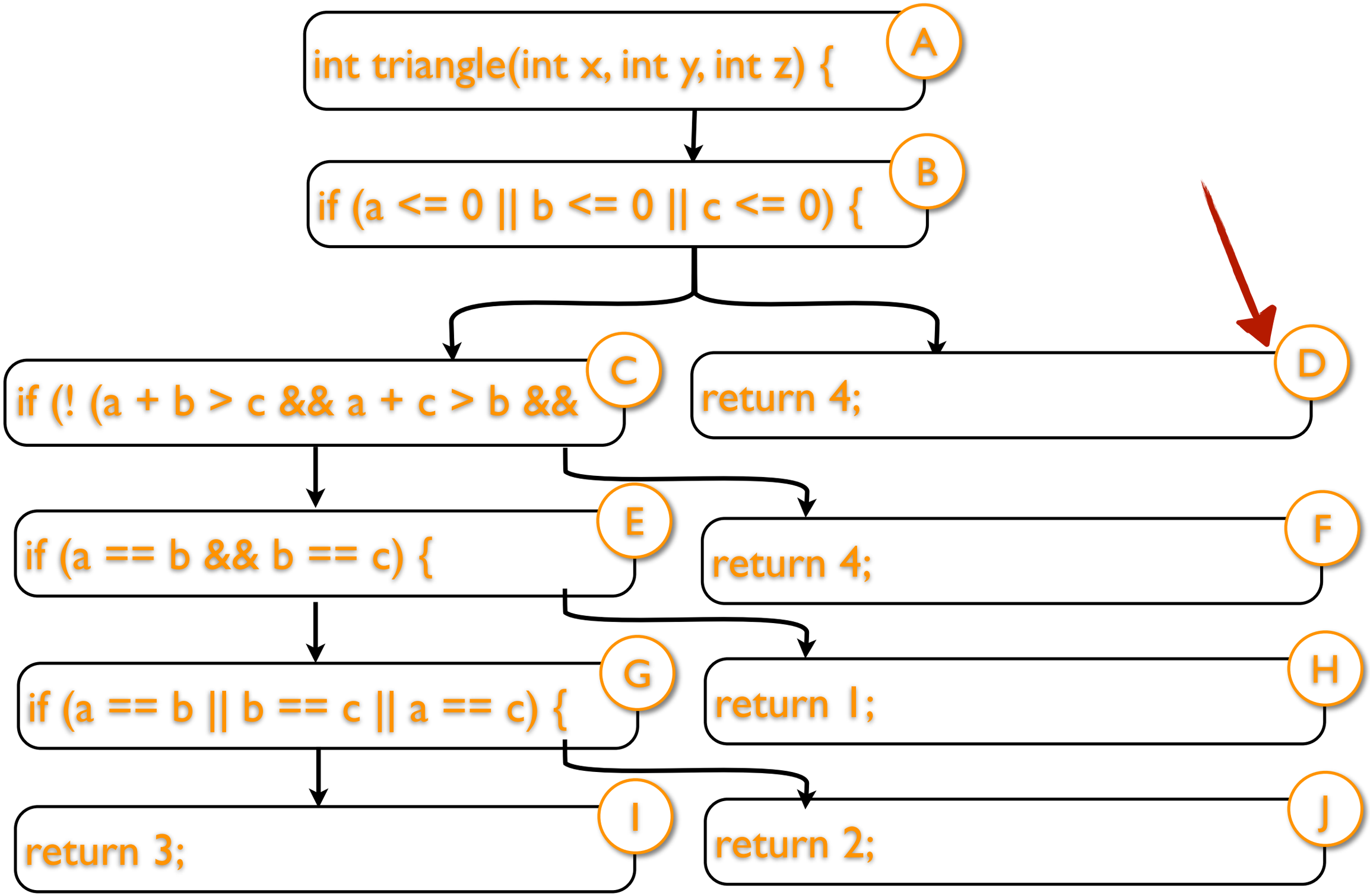
Equilateral Isosceles Scalene

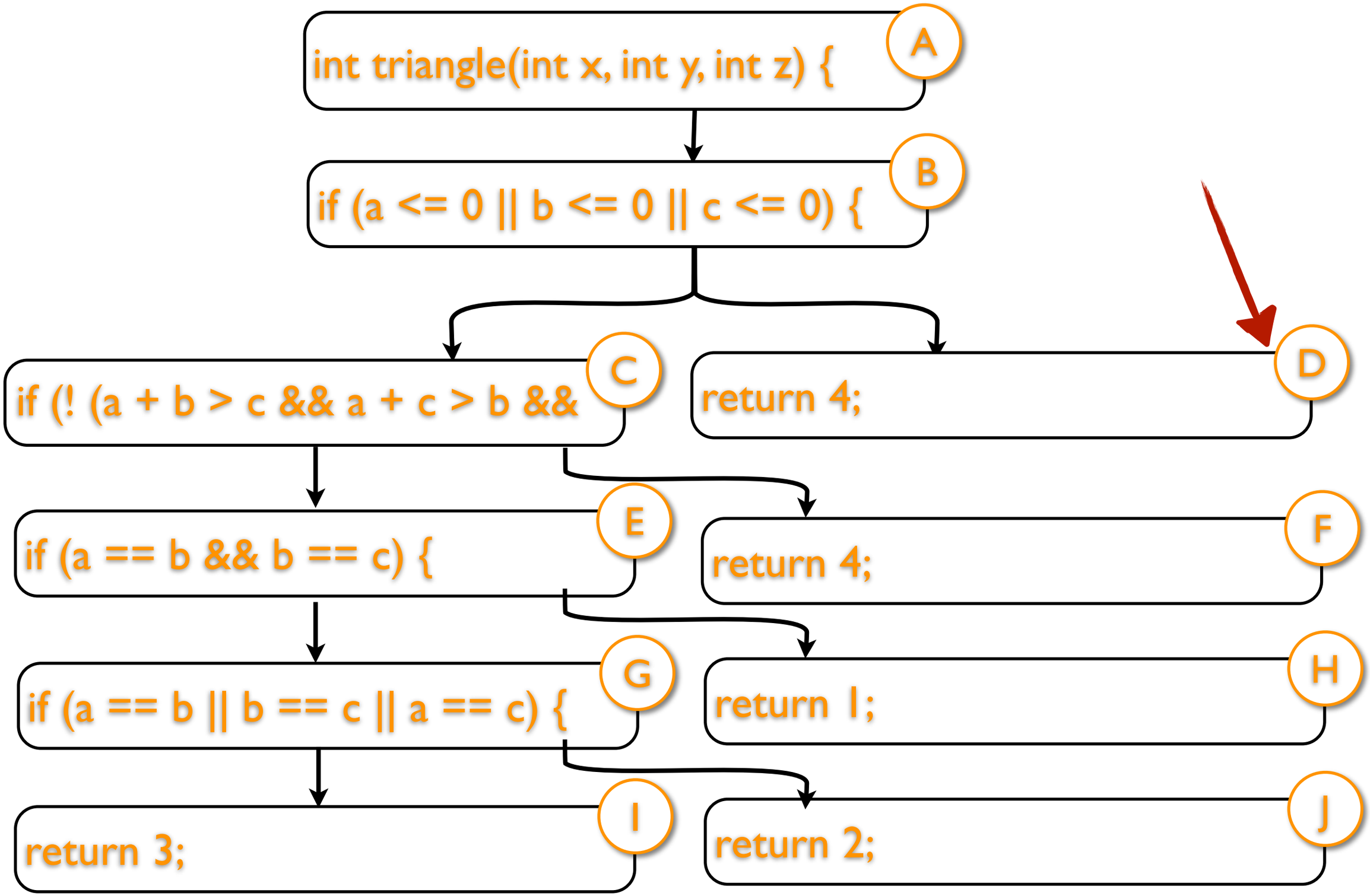
```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```


int triangle
int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J

(21, 53, 38)

```
int triangle(int x, int y, int z) {                    A
if (a <= 0 || b <= 0 || c <= 0) {                      B
if (! (a + b > c && a + c > b &&                       C
return 4;                                              D
if (a == b && b == c) {                                E
return 4;                                              F
if (a == b || b == c || a == c) {                      G
return 1;                                              H
return 3;                                              I
return 2;                                              J
```

(21, 53, 38)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
if (a == b && b == c) {
if (a == b || b == c || a == c) {
return 3;
return 4;
return 4;
return 1;
return 2;
```

D

F

H

J

(21, 53, 38)

int triangle(int x, int y, int z) {

if (a <= 0 || b <= 0 || c <= 0) {

if (! (a + b > c && a + c > b &&

return 4;

D

if (a == b && b == c) {

return 4;

F

if (a == b || b == c || a == c) {

return 1;

H

return 3;

return 2;

J

(21, 53, 38)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
    return 4;
if (! (a + b > c && a + c > b &&
    return 4;
if (a == b && b == c) {
    return 1;
if (a == b || b == c || a == c) {
    return 2;
return 3;
```

D

F

H

J

(21, 53, 38)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
if (a == b && b == c) {
if (a == b || b == c || a == c) {
return 3;
return 4;
return 4;
return 1;
return 2;
```

Approach: 1

D

F

H

J

Branch Distance

Branch Distance

- Input (21, 53, 38)

Branch Distance

- Input (21, 53, 38)
- Distance to (a <= 0 || b <= 0 || c <= 0)?

Branch Distance

- Input (21, 53, 38)
- Distance to (a <= 0 || b <= 0 || c <= 0)?
- Minimum of distance to a<=0, b<=0, c<=0

Branch Distance

- Input (21, 53, 38)
- Distance to (a <= 0 || b <= 0 || c <= 0)?
- Minimum of distance to a<=0, b<=0, c<=0
- a <= 0

Branch Distance

- Input (21, 53, 38)
- Distance to (a <= 0 || b <= 0 || c <= 0)?
- Minimum of distance to a<=0, b<=0, c<=0
- a <= 0
- a - b <= 0 ? 0 : (a - b) + K (K = 1)

Branch Distance

- Input (21, 53, 38)
- Distance to (a <= 0 || b <= 0 || c <= 0)?
- Minimum of distance to a<=0, b<=0, c<=0
- a <= 0
- a - b <= 0 ? 0 : (a - b) + K (K = 1)
- Minimum of (21, 53, 38) = 22

Fitness

- Approach level: 1
- Branch distance: 22
- Fitness = 1 + 22/(22+1) = 1.96

(21, 53, 38)

```
int triangle(int x, int y, int z) {                    A
if (a <= 0 || b <= 0 || c <= 0) {                      B
if (! (a + b > c && a + c > b &&                       C
return 4;                                              D
if (a == b && b == c) {                                E
return 4;                                              F
if (a == b || b == c || a == c) {                      G
return 1;                                              H
return 3;                                              I
return 2;                                              J
```

(20, 53, 38)

```
int triangle(int x, int y, int z) {                    A
```

```
if (a <= 0 || b <= 0 || c <= 0) {                      B
```

```
if (! (a + b > c && a + c > b &&                       C
```

```
return 4;                                              D
```

```
if (a == b && b == c) {                                E
```

```
return 4;                                              F
```

```
if (a == b || b == c || a == c) {                      G
```

```
return 1;                                              H
```

```
return 3;                                              I
```

```
return 2;                                              J
```

(20, 53, 38)

int triangle(int x, int y, int z) {

if (a <= 0 || b <= 0 || c <= 0) {

D

return 4;

if (! (a + b > c && a + c > b &&

if (a == b && b == c) {

F

return 4;

if (a == b || b == c || a == c) {

H

return 1;

return 3;

J

return 2;

Search

Search

- Next best solution:

Search

- Next best solution:
- (20, 53, 38) - Approach 1, Branch 21

Search

- Next best solution:
- (20, 53, 38) - Approach 1, Branch 21
- (19, 53, 38) - Approach 1, Branch 20

Search

- Next best solution:
- (20, 53, 38) - Approach 1, Branch 21
- (19, 53, 38) - Approach 1, Branch 20
- (18, 53, 38) - Approach 1, Branch 19

Search

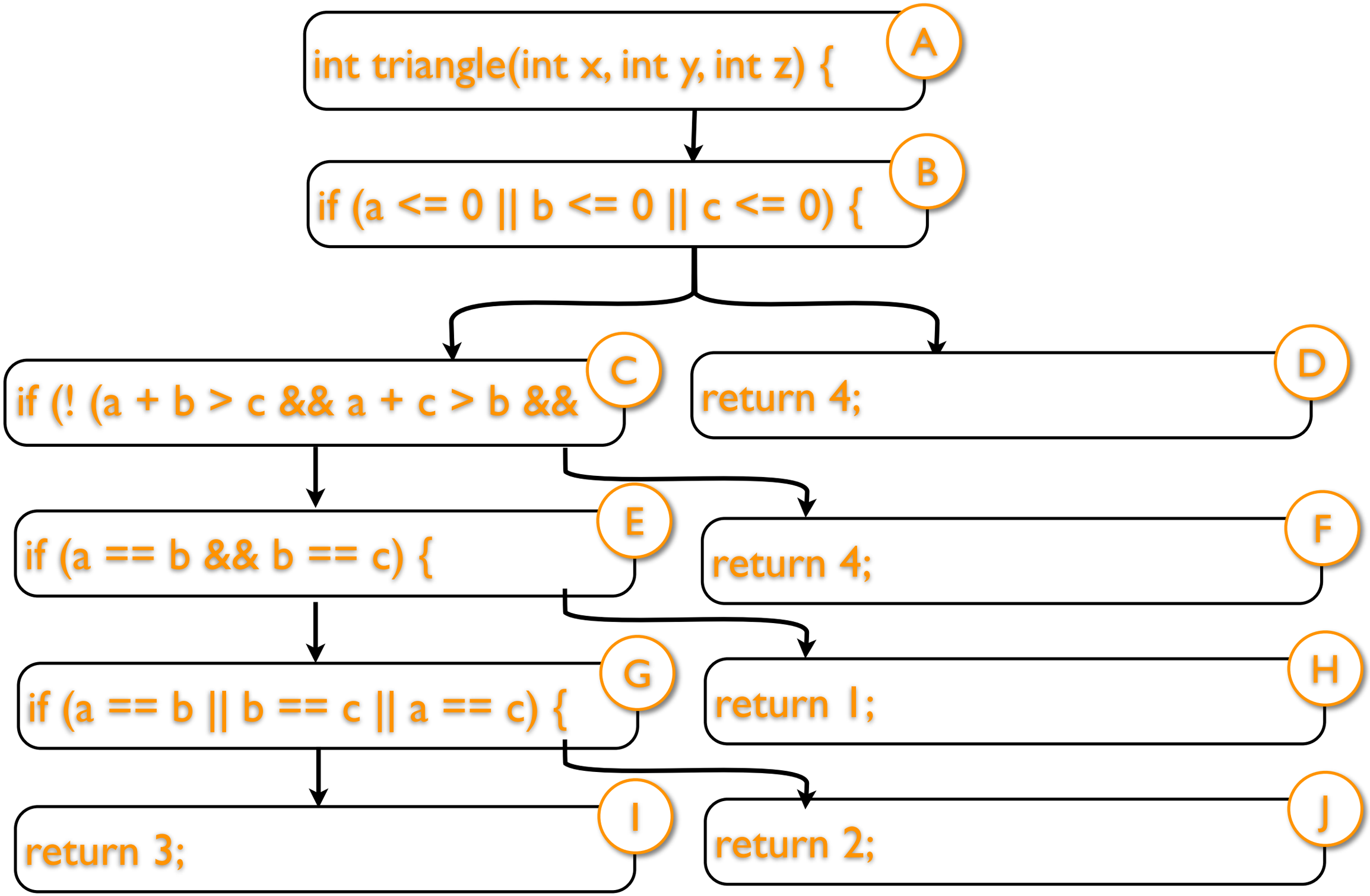
- Next best solution:
- (20, 53, 38) - Approach 1, Branch 21
- (19, 53, 38) - Approach 1, Branch 20
- (18, 53, 38) - Approach 1, Branch 19
- (17, 53, 38) - Approach 1, Branch 18

Search

- Next best solution:
- (20, 53, 38) - Approach 1, Branch 21
- (19, 53, 38) - Approach 1, Branch 20
- (18, 53, 38) - Approach 1, Branch 19
- (17, 53, 38) - Approach 1, Branch 18
- (16, 53, 38) - Approach 1, Branch 17

Search

- Next best solution:
- (20, 53, 38) - Approach 1, Branch 21
- (19, 53, 38) - Approach 1, Branch 20
- (18, 53, 38) - Approach 1, Branch 19
- (17, 53, 38) - Approach 1, Branch 18
- (16, 53, 38) - Approach 1, Branch 17
- (15, 53, 38)


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J

(15, 53, 38)

A
```
int triangle(int x, int y, int z) {
```

B
```
if (a <= 0 || b <= 0 || c <= 0) {
```

C
```
if (! (a + b > c && a + c > b &&
```

D
```
return 4;
```

E
```
if (a == b && b == c) {
```

F
```
return 4;
```

G
```
if (a == b || b == c || a == c) {
```

H
```
return 1;
```

I
```
return 3;
```

J
```
return 2;
```

(15, 53, 38)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
return 4;
if (a == b && b == c) {
return 4;
if (a == b || b == c || a == c) {
return 1;
return 3;
return 2;
```

D E G H I J

(15, 53, 38)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
if (a == b && b == c) {
if (a == b || b == c || a == c) {
return 3;
return 4;
return 4;
return 1;
return 2;
```

Approach: 1

D

E

G

I

H

J

Search

Search

- (15, 53, 38) - Approach 1, Branch 16

Search

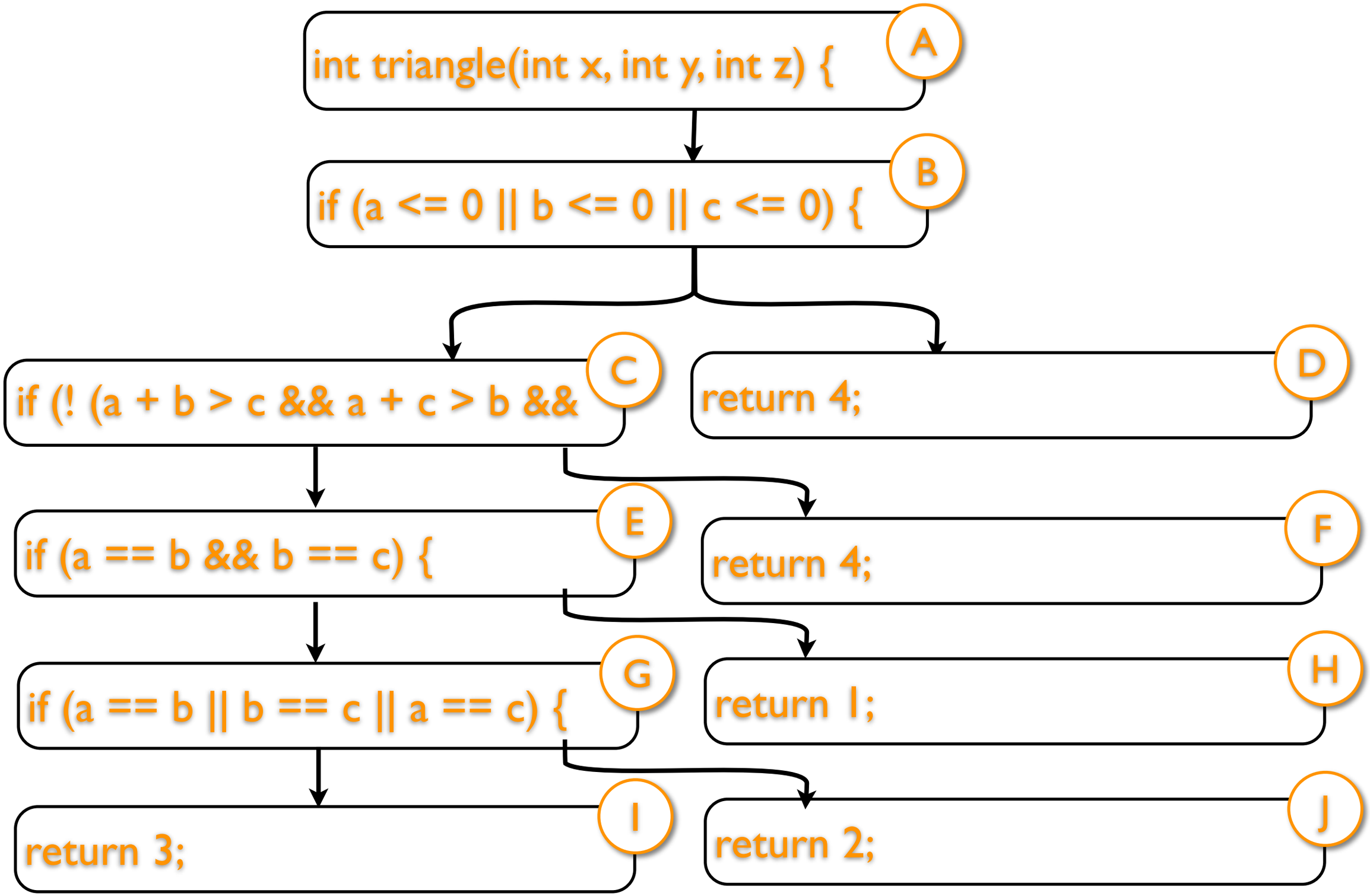
- (15, 53, 38) - Approach 1, Branch 16
- (14, 53, 38) - Approach 1, Branch 15

Search

- (15, 53, 38) - Approach 1, Branch 16
- (14, 53, 38) - Approach 1, Branch 15
- ...

Search

- (15, 53, 38) - Approach 1, Branch 16
- (14, 53, 38) - Approach 1, Branch 15
- ...
- (0, 53, 38) - Approach 0, Branch 0


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J

(0, 53, 38)

```
int triangle(int x, int y, int z) {                    A
```

```
if (a <= 0 || b <= 0 || c <= 0) {                      B
```

```
if (! (a + b > c && a + c > b &&                       C
```

```
return 4;                                              D
```

```
if (a == b && b == c) {                                E
```

```
return 4;                                              F
```

```
if (a == b || b == c || a == c) {                      G
```

```
return 1;                                              H
```

```
return 3;                                              I
```

```
return 2;                                              J
```

(0, 53, 38)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
return 4;
if (a == b && b == c) {
return 4;
if (a == b || b == c || a == c) {
return 1;
return 3;
return 2;
```

C E F G H I J

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

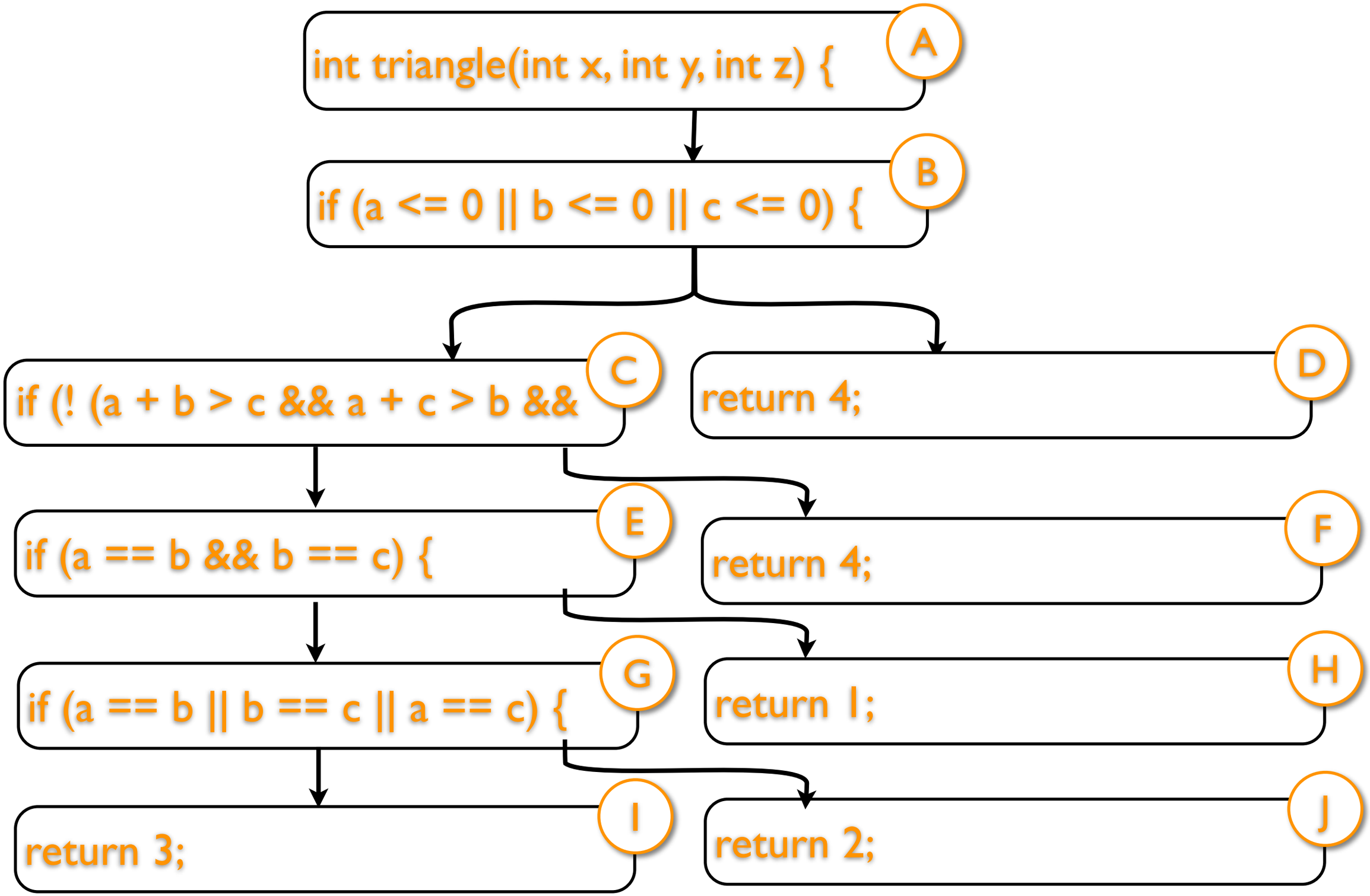
```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

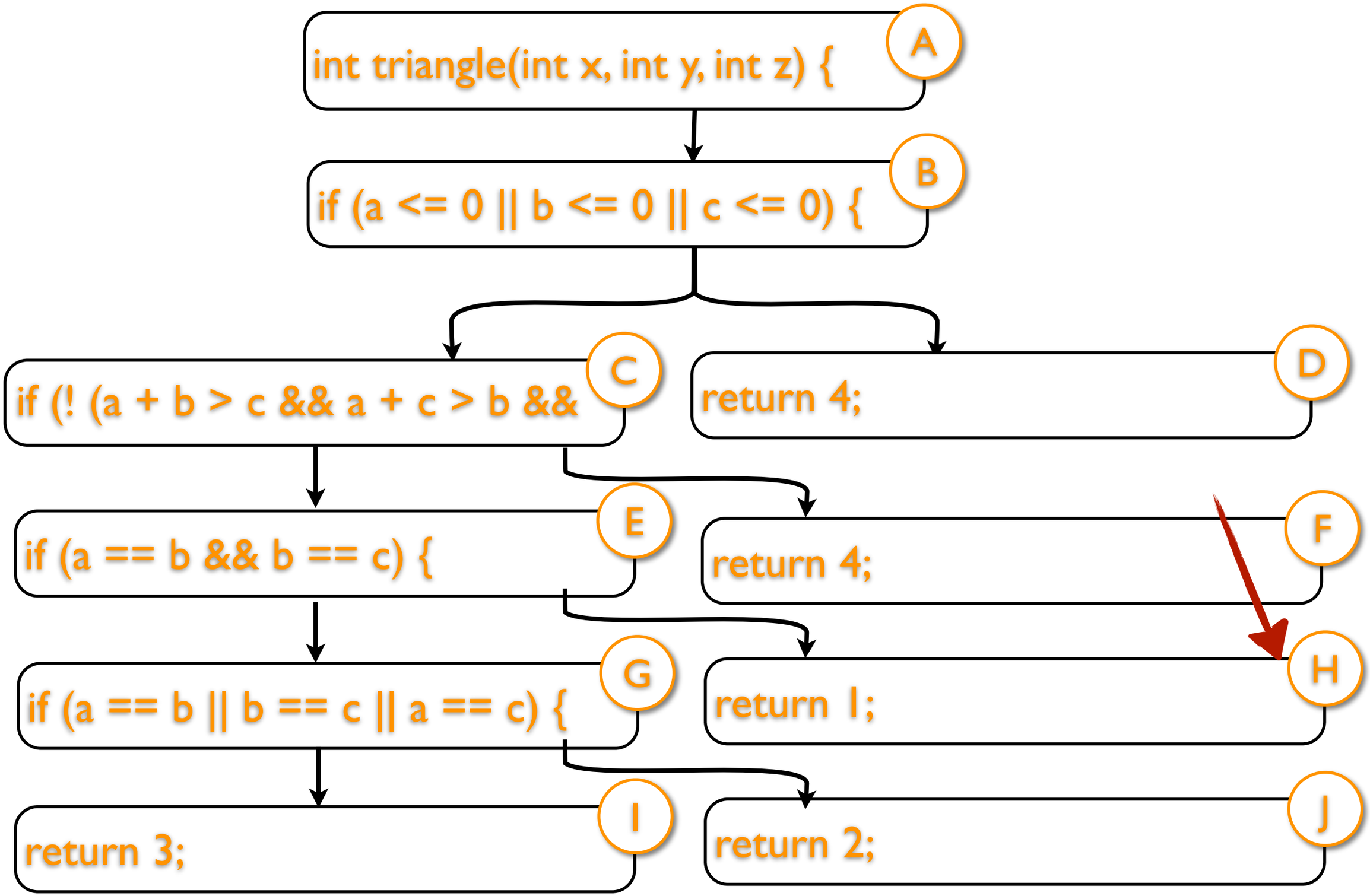
```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

(0, 53, 38)

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

(0, 53, 38)


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J

(5, 0, 9)

A

```
int triangle(int x, int y, int z) {
```

B

```
if (a <= 0 || b <= 0 || c <= 0) {
```

C

```
if (! (a + b > c && a + c > b &&
```

D

```
return 4;
```

E

```
if (a == b && b == c) {
```

F

```
return 4;
```

G

```
if (a == b || b == c || a == c) {
```

H

```
return 1;
```

I

```
return 3;
```

J

```
return 2;
```

(5, 0, 9)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
return 4;
if (a == b && b == c) {
return 4;
if (a == b || b == c || a == c) {
return 1;
return 3;
return 2;
```

C E F G H I J

(5, 0, 9)

int triangle(int x, int y, int z) {

if (a <= 0 || b <= 0 || c <= 0) {

C

if (! (a + b > c && a + c > b &&

return 4;

E

if (a == b && b == c) {

F

return 4;

G

if (a == b || b == c || a == c) {

H

return 1;

I

return 3;

J

return 2;

Approach: 3

Branch Distance

Branch Distance

- Input (5, 0, 9)

Branch Distance

- Input (5, 0, 9)
- Distance to !(a <= 0 || b <= 0 || c <= 0)?

Branch Distance

- Input (5, 0, 9)
- Distance to !(a <= 0 || b <= 0 || c <= 0)?
- Sum of distance to a>0, b>0, c>0

Branch Distance

- Input (5, 0, 9)
- Distance to !(a <= 0 || b <= 0 || c <= 0)?
- Sum of distance to a>0, b>0, c>0
- b - a < 0 ? 0 : (b - a) + K (K = 1)

Branch Distance

- Input (5, 0, 9)
- Distance to !(a <= 0 || b <= 0 || c <= 0)?
- Sum of distance to a>0, b>0, c>0
- b - a < 0 ? 0 : (b - a) + K (K = 1)
- 0 + 1 + 0

Fitness

- Input (5, 0, 9)
- Approach level: 3
- Branch distance: 1
- Fitness = 3 + 1/(1+1) = 3.5

(5, 0, 9)

A

```
int triangle(int x, int y, int z) {
```

B

```
if (a <= 0 || b <= 0 || c <= 0) {
```

C

```
if (! (a + b > c && a + c > b &&
```

D

```
return 4;
```

E

```
if (a == b && b == c) {
```

F

```
return 4;
```

G

```
if (a == b || b == c || a == c) {
```

H

```
return 1;
```

I

```
return 3;
```

J

```
return 2;
```

(5, 1, 9)

```
int triangle(int x, int y, int z) {                    A
if (a <= 0 || b <= 0 || c <= 0) {                      B
if (! (a + b > c && a + c > b &&                       C
return 4;                                              D
if (a == b && b == c) {                                E
return 4;                                              F
if (a == b || b == c || a == c) {                      G
return 1;                                              H
return 3;                                              I
return 2;                                              J
```

(5, 1, 9)

int triangle(int x, int y, int z) { ✔

if (a <= 0 || b <= 0 || c <= 0) { ✔

if (! (a + b > c && a + c > b && ✔

return 4; D

if (a == b && b == c) { E

return 4; ✔

if (a == b || b == c || a == c) { G

return 1; H

return 3; I

return 2; J

(5, 1, 9)

int triangle(int x, int y, int z) {

if (a <= 0 || b <= 0 || c <= 0) {

Approach: 2

if (! (a + b > c && a + c > b &&

D

return 4;

E

if (a == b && b == c) {

return 4;

G

if (a == b || b == c || a == c) {

H

return 1;

I

return 3;

J

return 2;

Branch Distance

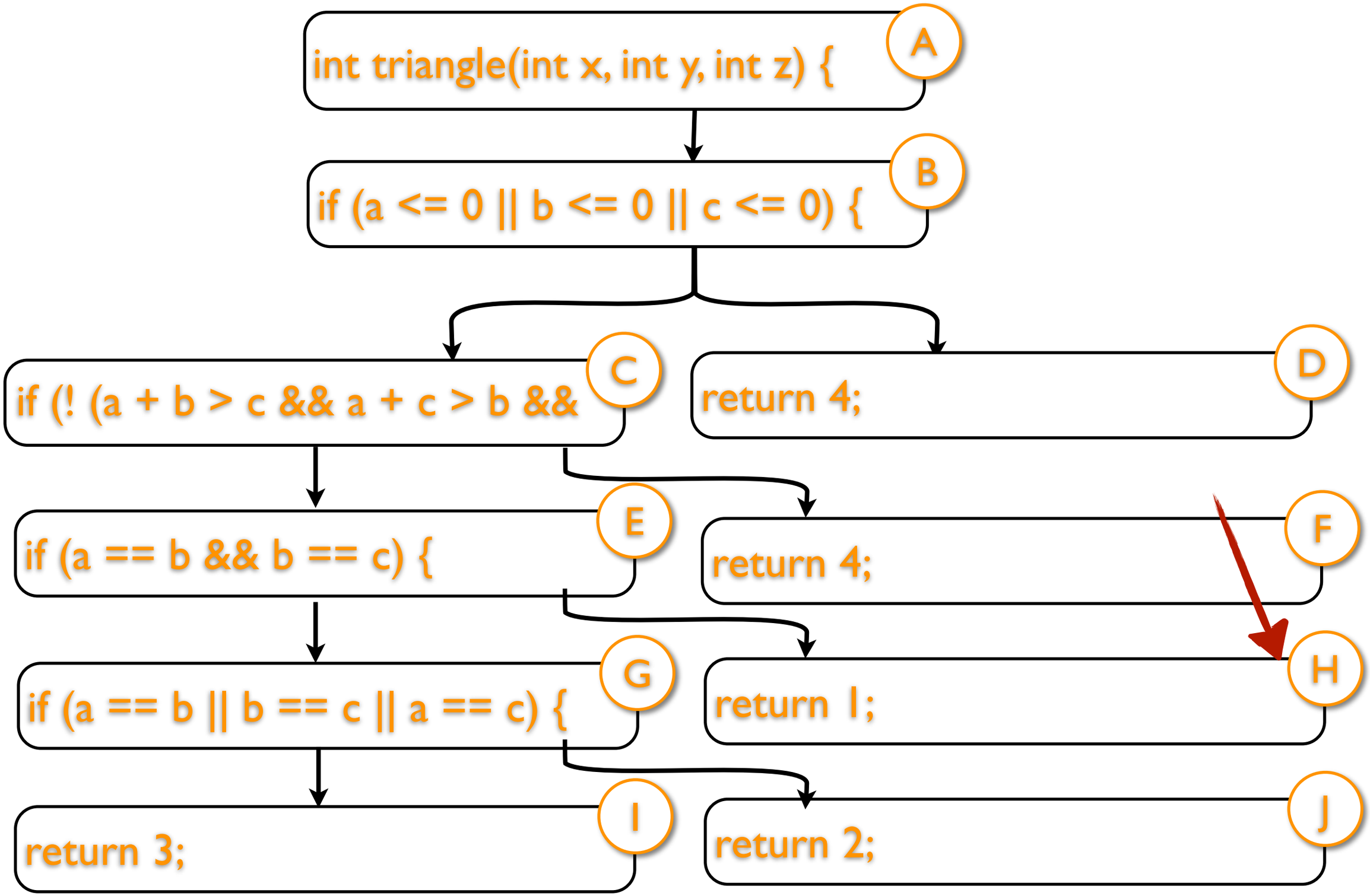
- Input (5, 1, 9)
- Distance to !(a == b && b == c)?
- Sum of distance to a==b + b==c
- abs(a-b) = 0 ? 0 : abs(a-b) + K (K = 1)
- 5 + 9

Fitness

- Input (5, 1, 9)
- Approach level: 2
- Branch distance: 14
- Fitness = 2 + 14/(14+1) = 2.9

Search

- (5, 2, 9) - Approach 2, Branch 12
- (5, 3, 9) - Approach 2, Branch 10
- (5, 4, 9) - Approach 2, Branch 8
- (5, 5, 9)


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J

(5, 5, 9)

int triangle(int x, int y, int z) { ✔

if (a <= 0 || b <= 0 || c <= 0) { ✔

if (! (a + b > c && a + c > b && ✔

return 4; D

if (a == b && b == c) { ✔

return 4; F

if (a == b || b == c || a == c) { ✔

return 1; H

return 3; I

return 2; ✔

(5, 5, 9)

int triangle(int x, int y, int z) {

if (a <= 0 || b <= 0 || c <= 0) {

Approach: 1

if (! (a + b > c && a + c > b &&

return 4;

D

if (a == b && b == c) {

return 4;

F

if (a == b || b == c || a == c) {

return 1;

H

return 3;

I

return 2;

Branch Distance

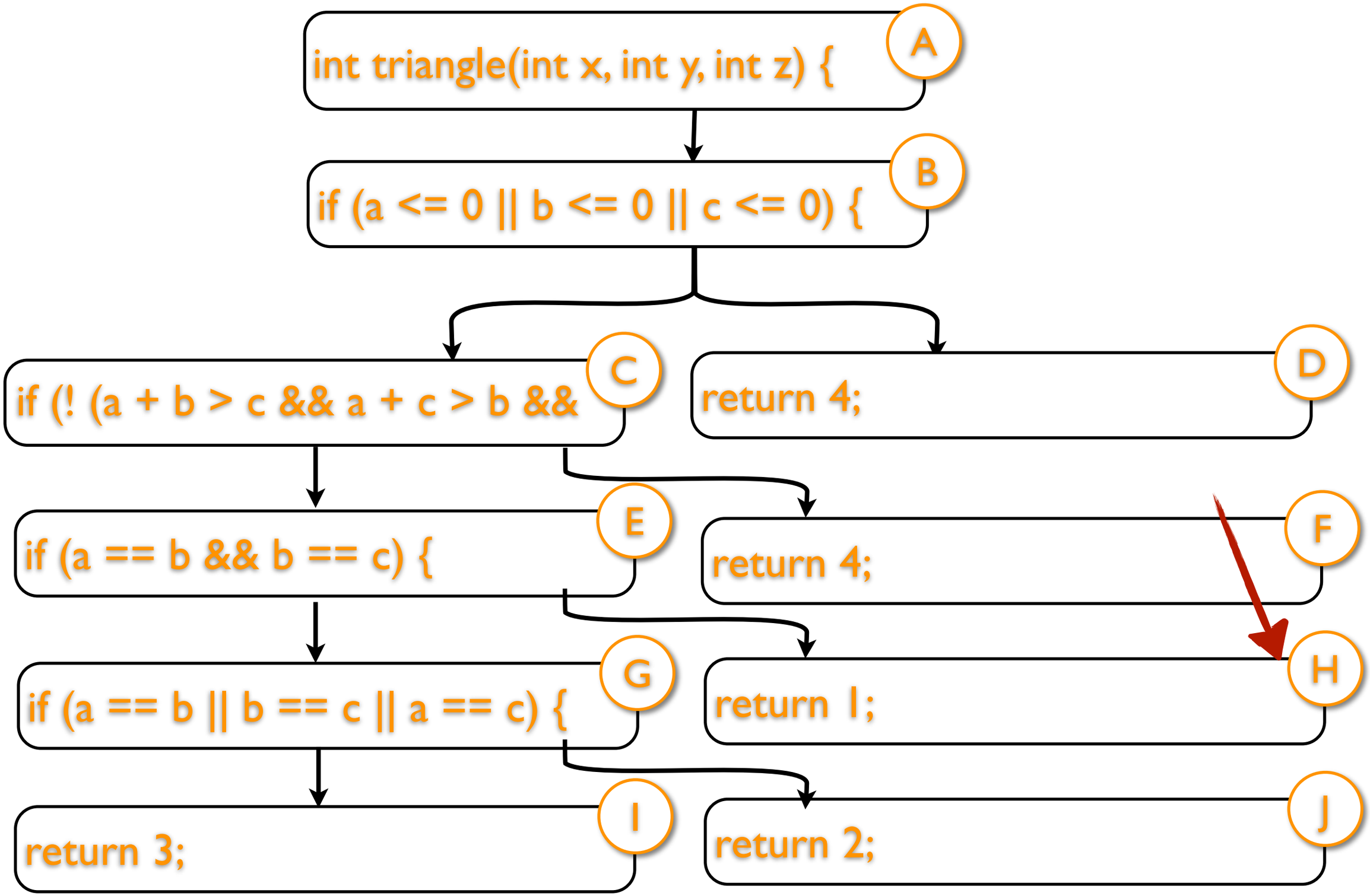
- Input (5, 5, 9)
- Distance to !(a == b && b == c)?
- Sum of distance to a==b + b==c
- abs(a-b) = 0 ? 0 : abs(a-b) + K (K = 1)
- 0 + 5

Fitness

- Input (5, 5, 9)
- Approach level: 1
- Branch distance: 5
- Fitness = 2 + 5/(5+1) = 1.8

Search

- (5, 5, 8) - Approach 1, Branch 4
- (5, 5, 7) - Approach 1, Branch 3
- (5, 5, 6) - Approach 1, Branch 2
- (5, 5, 5) - Approach 0, Branch 0


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J

(5, 5, 5)

```
int triangle(int x, int y, int z) {                    A
if (a <= 0 || b <= 0 || c <= 0) {                      B
if (! (a + b > c && a + c > b &&                       C
return 4;                                              D
if (a == b && b == c) {                                E
return 4;                                              F
if (a == b || b == c || a == c) {                      G
return 1;                                              H
return 3;                                              I
return 2;                                              J
```

(5, 5, 5)

int triangle(int x, int y, int z) {

if (a <= 0 || b <= 0 || c <= 0) {

if (! (a + b > c && a + c > b &&

return 4;

D

if (a == b && b == c) {

return 4;

F

if (a == b || b == c || a == c) {

G

return 1;

return 3;

I

return 2;

J

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

(0, 53, 38)

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

(0, 53, 38)

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

(0, 53, 38)

(5, 5, 5)

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

(0, 53, 38)

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

(0, 53, 38)

```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

(0, 53, 38)

(5, 5, 5)

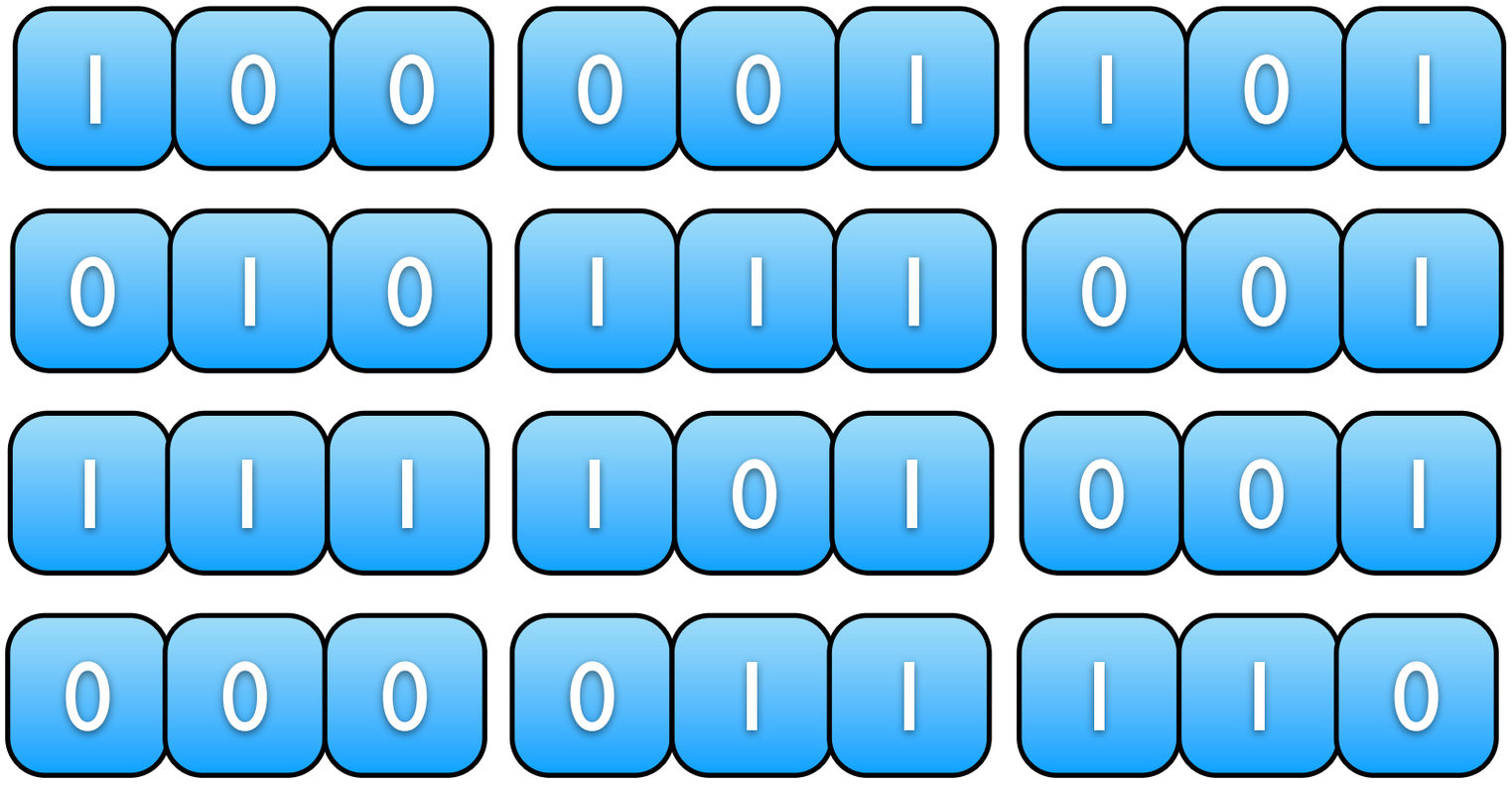
```
int triangle(int a, int b, int c) {

    if (a <= 0 || b <= 0 || c <= 0) {
        return 4; // invalid
    }
    if (! (a + b > c && a + c > b && b + c > a)) {
        return 4; // invalid
    }
    if (a == b && b == c) {
        return 1; // equilateral
    }
    if (a == b || b == c || a == c) {
        return 2; // isosceles
    }

    return 3;  // scalene
}
```

(0, 53, 38)

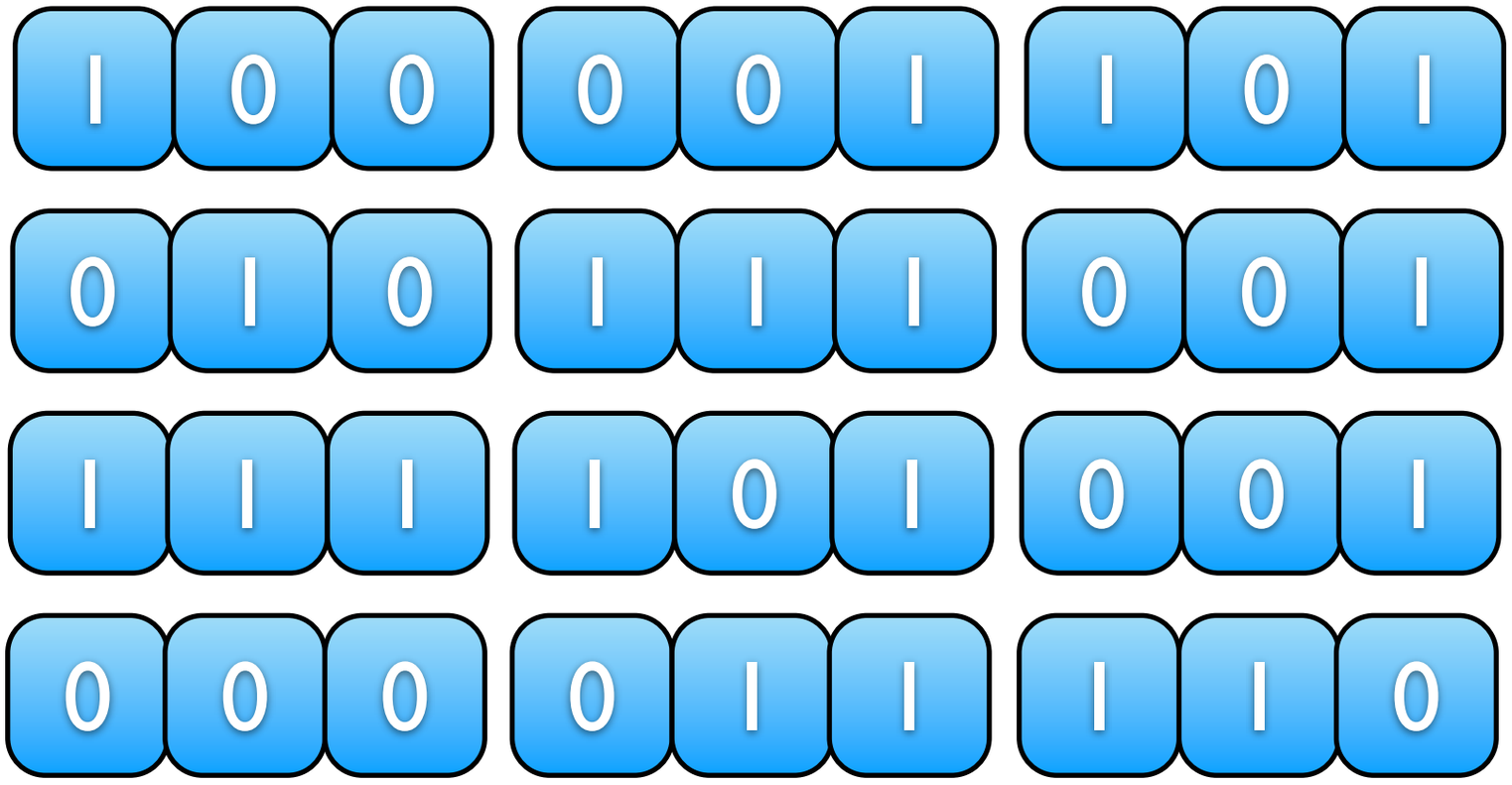
(5, 5, 5)

GA Example

GA Example

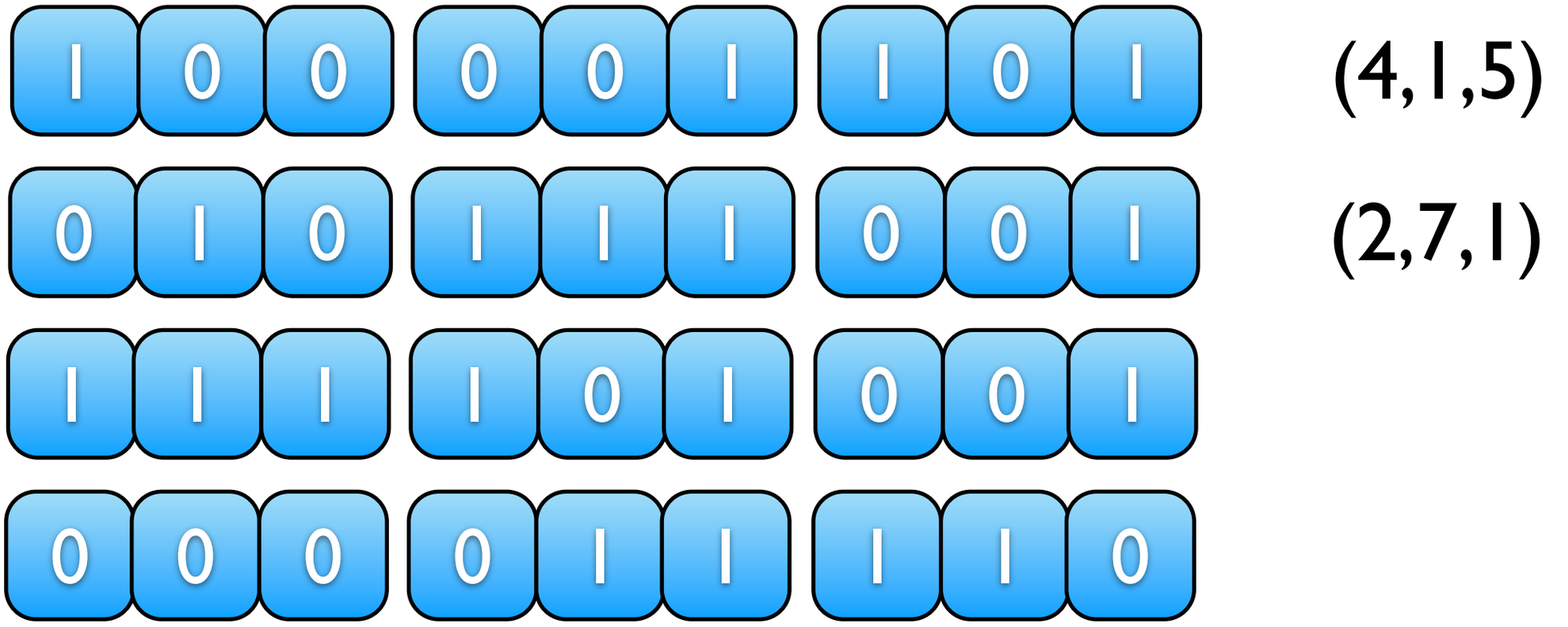


GA Example

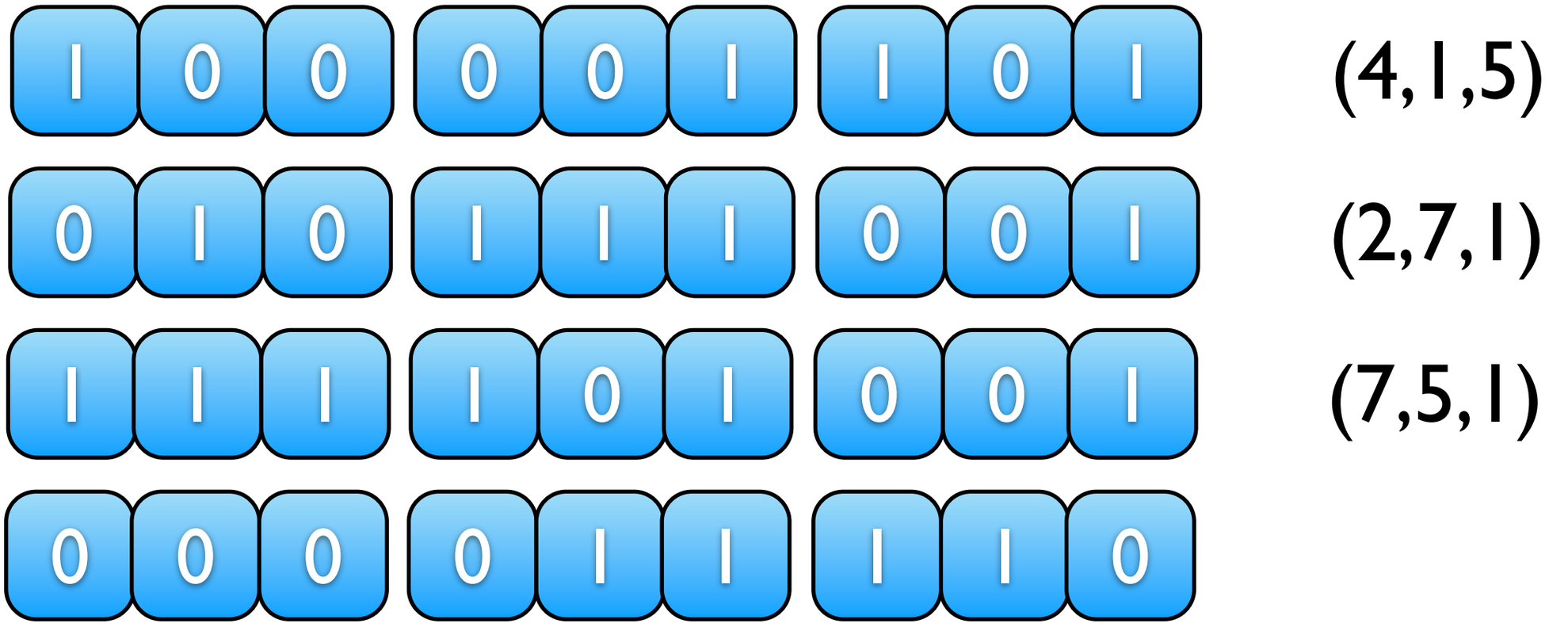


(4,1,5)

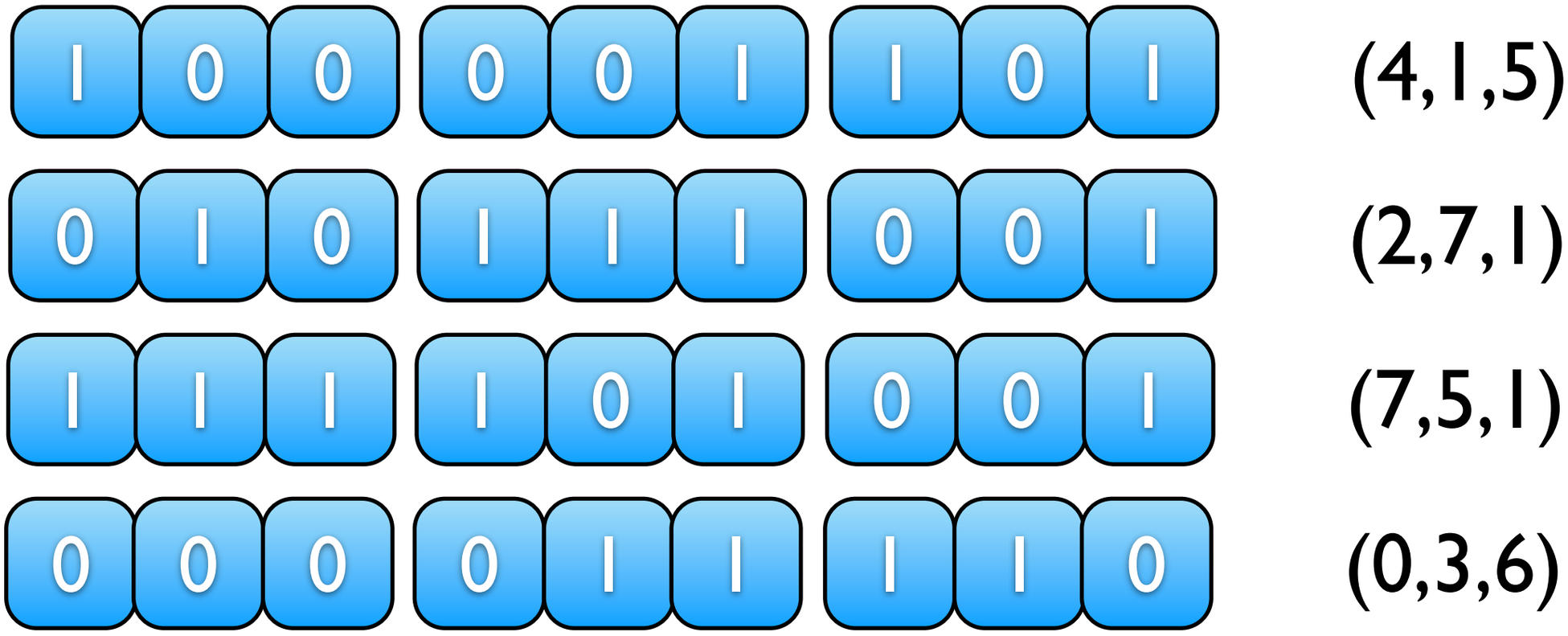
GA Example

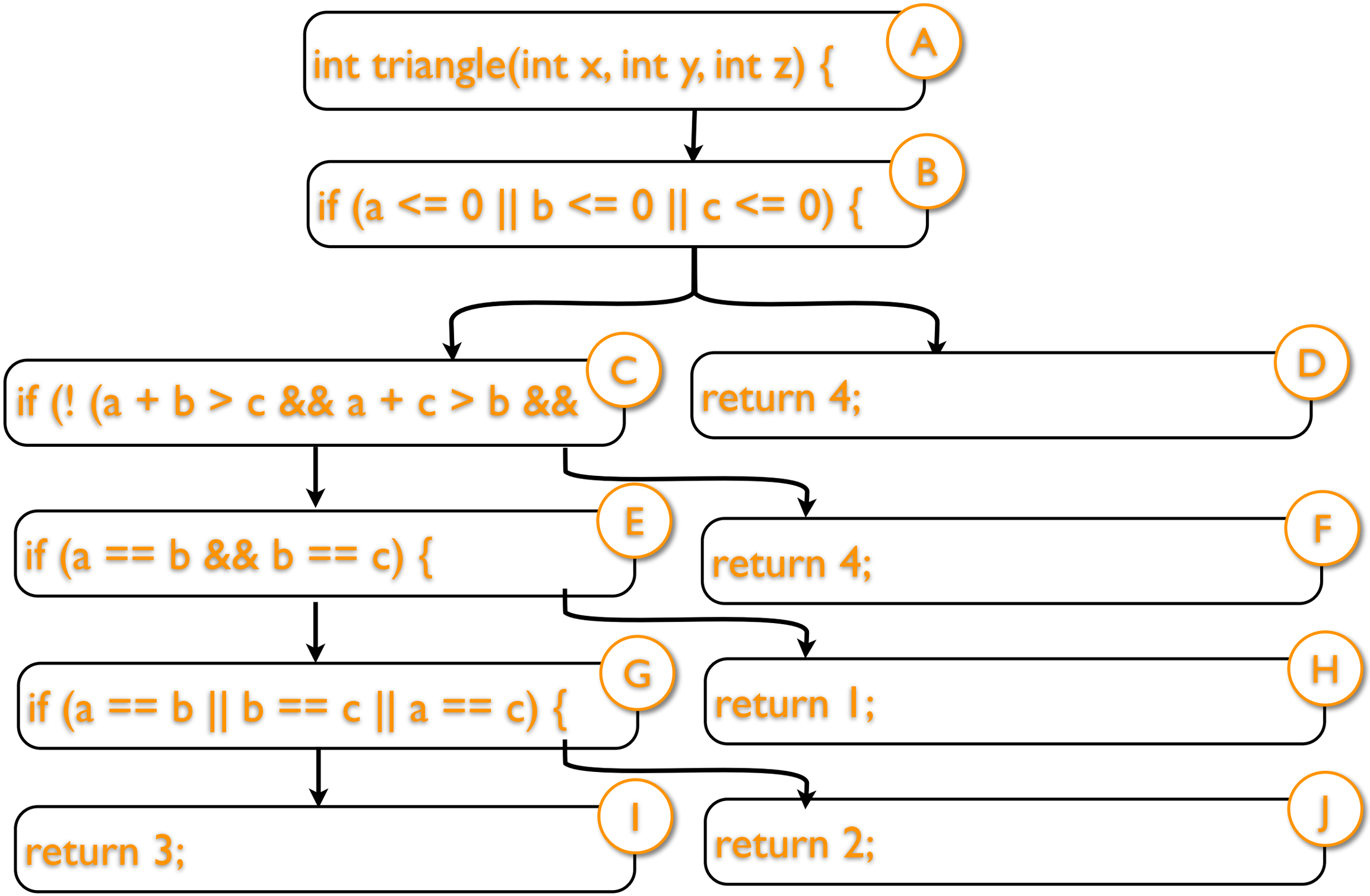


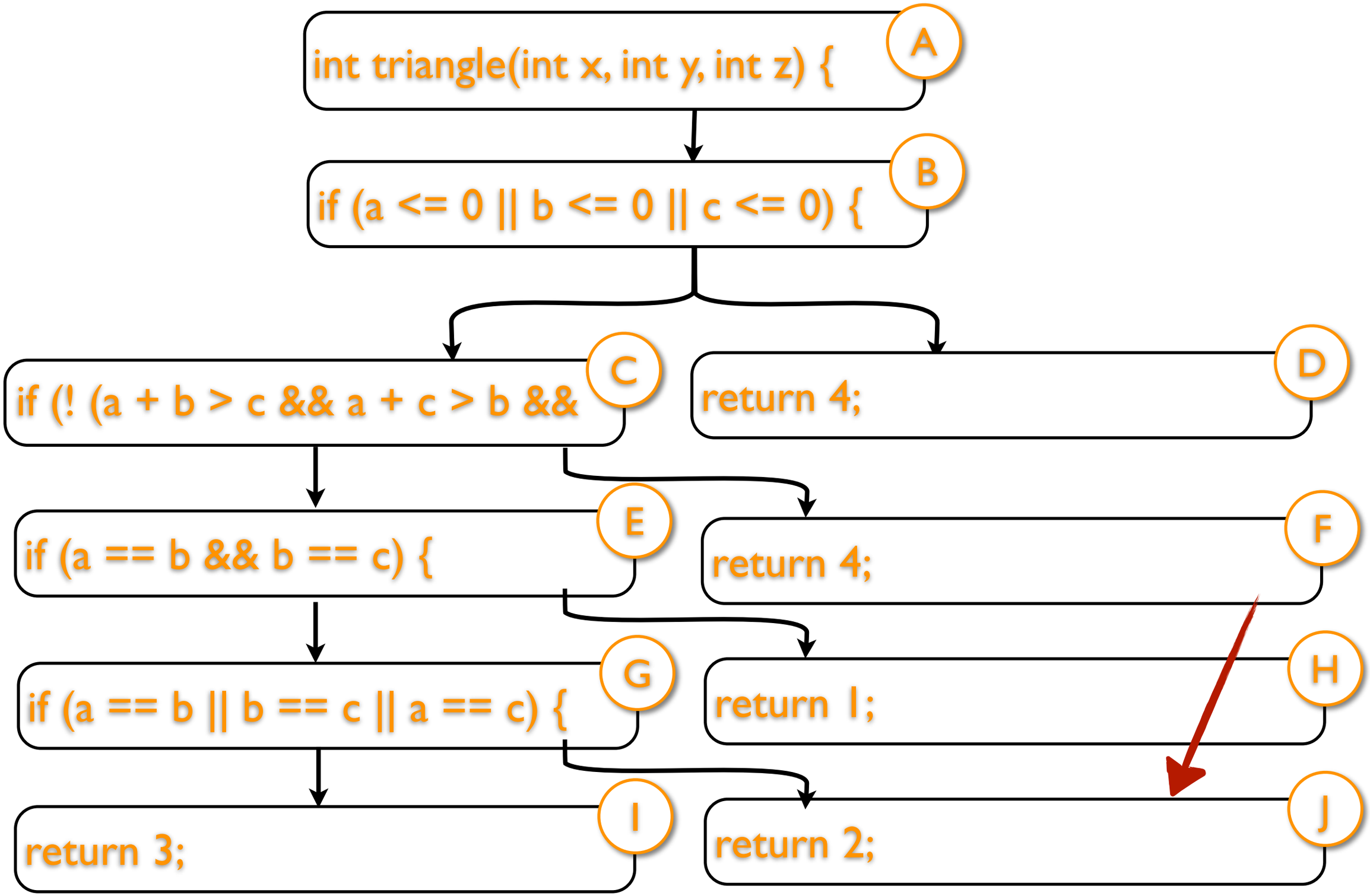
GA Example



GA Example




int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J

(4, 1, 5)

int triangle(int x, int y, int z) {

if (a <= 0 || b <= 0 || c <= 0) {

if (! (a + b > c && a + c > b &&

return 4;

D

if (a == b && b == c) {

E

return 4;

if (a == b || b == c || a == c) {

G

return 1;

H

return 3;

I

return 2;

J

(4, 1, 5)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
if (a == b && b == c) {
if (a == b || b == c || a == c) {
return 3;
return 4;
return 1;
return 2;
return 4;
```

D E G I H J

(4, 1, 5)

int triangle(int x, int y, int z) {

if (a <= 0 || b <= 0 || c <= 0) {

if (! (a + b > c && a + c > b &&

return 4;

D

if (a == b && b == c) {

E

return 4;

if (a == b || b == c || a == c) {

G

return 1;

H

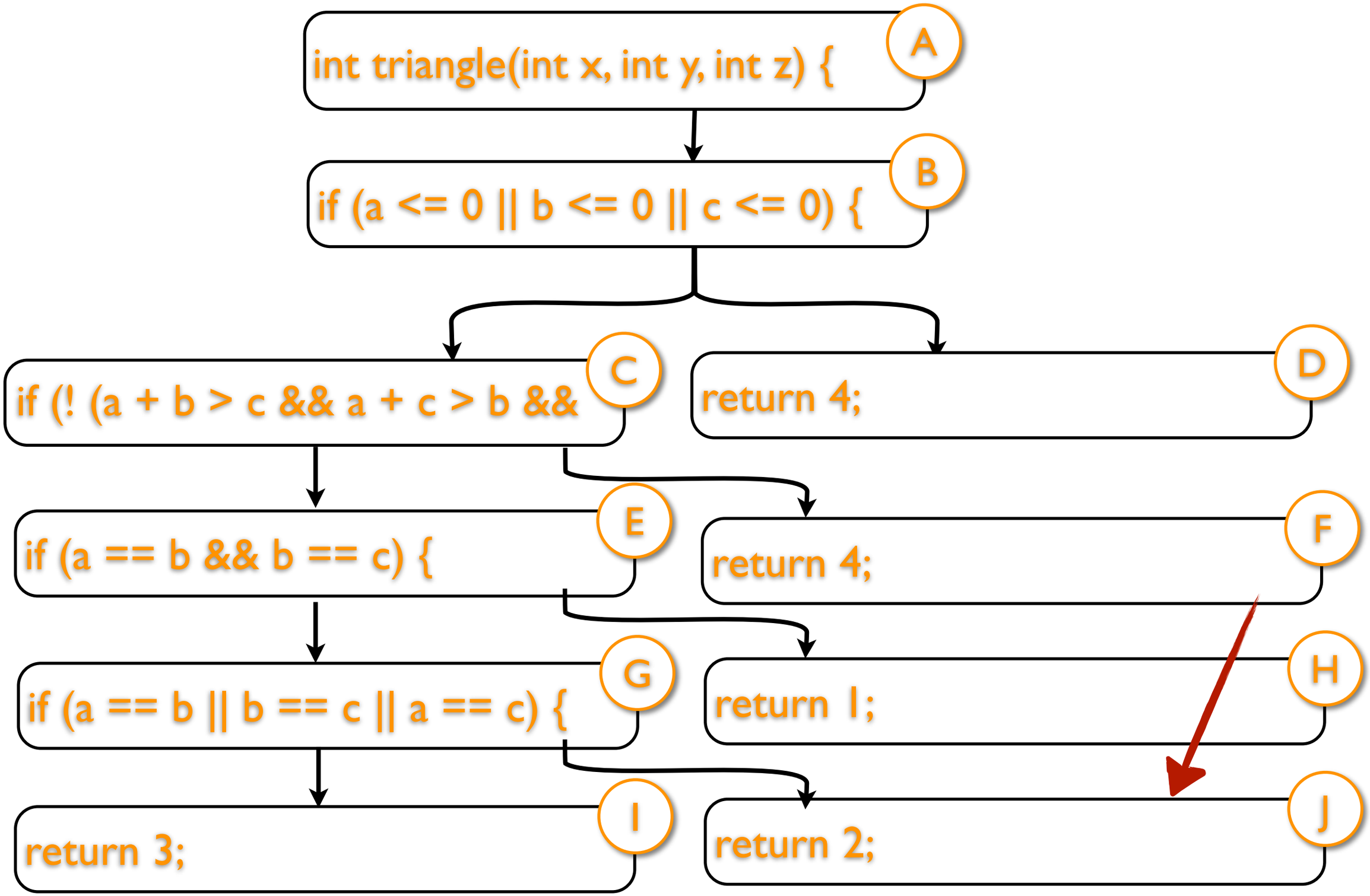
return 3;

I

return 2;

J

Approach: 3
Branch: 1


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J

(2, 7, 1)

int triangle(int x, int y, int z) {

if (a <= 0 || b <= 0 || c <= 0) {

if (! (a + b > c && a + c > b &&

return 4;

D

if (a == b && b == c) {

E

return 4;

if (a == b || b == c || a == c) {

G

return 1;

H

return 3;

I

return 2;

J

(2, 7, 1)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
return 4;
if (a == b && b == c) {
return 4;
if (a == b || b == c || a == c) {
return 1;
return 3;
return 2;
```

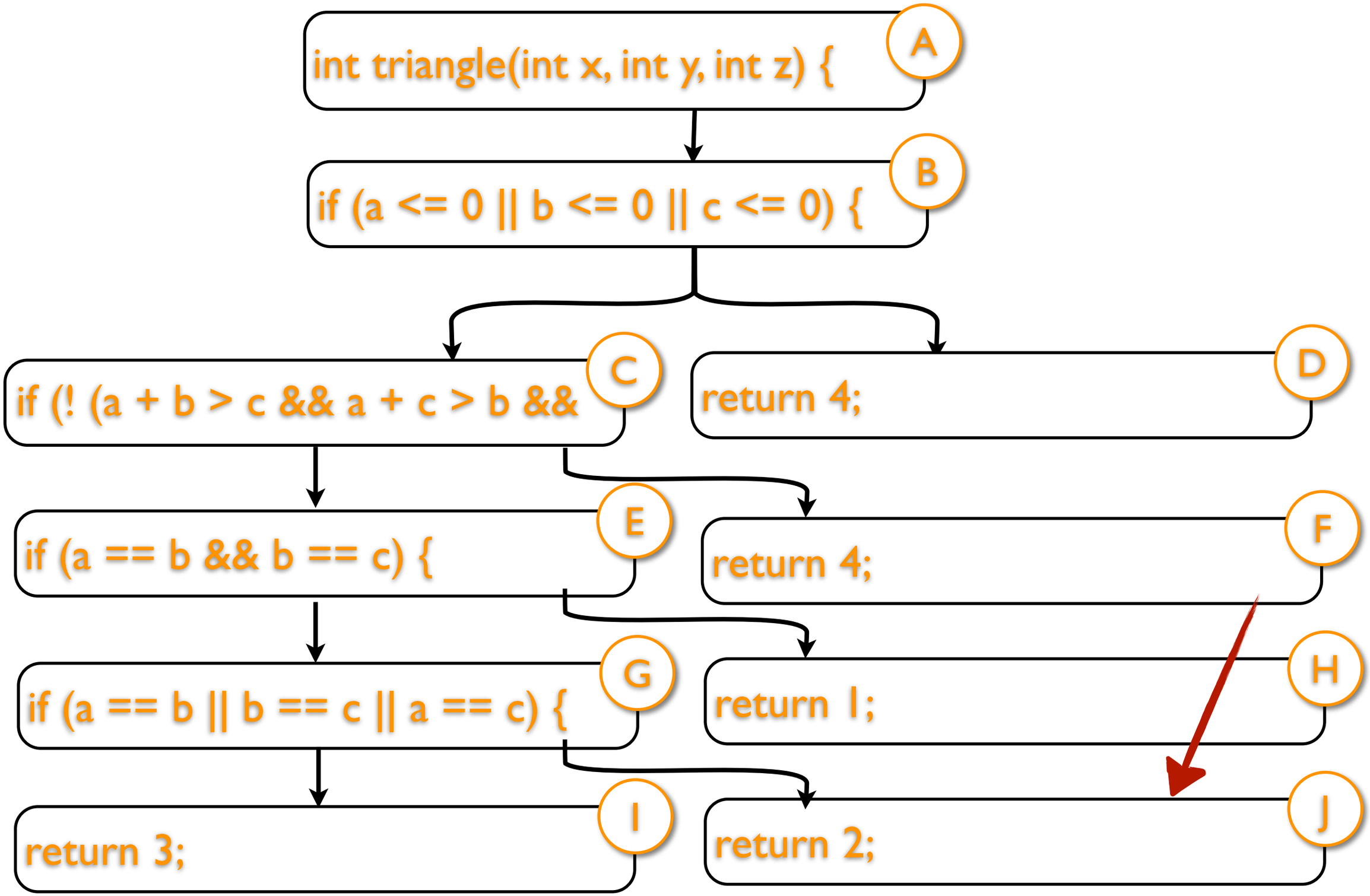
D E G H I J

(2, 7, 1)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
if (a == b && b == c) {
if (a == b || b == c || a == c) {
return 3;
return 4;
return 4;
return 1;
return 2;
```

D E G I H J

Approach: 3
Branch: 5


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J

(7, 5, 1)

int triangle(int x, int y, int z) {

if (a <= 0 || b <= 0 || c <= 0) {

if (! (a + b > c && a + c > b &&

return 4;

D

if (a == b && b == c) {

E

return 4;

if (a == b || b == c || a == c) {

G

return 1;

H

return 3;

I

return 2;

J

(7, 5, 1)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
return 4;
if (a == b && b == c) {
return 4;
if (a == b || b == c || a == c) {
return 1;
return 3;
return 2;
```

D E G H I J

(7, 5, 1)

```
int triangle(int x, int y, int z) {
```

```
if (a <= 0 || b <= 0 || c <= 0) {
```

Approach: 3
Branch: 2

```
if (! (a + b > c && a + c > b &&
```

```
return 4;
```

D

```
if (a == b && b == c) {
```

E

```
return 4;
```

```
if (a == b || b == c || a == c) {
```

G

```
return 1;
```

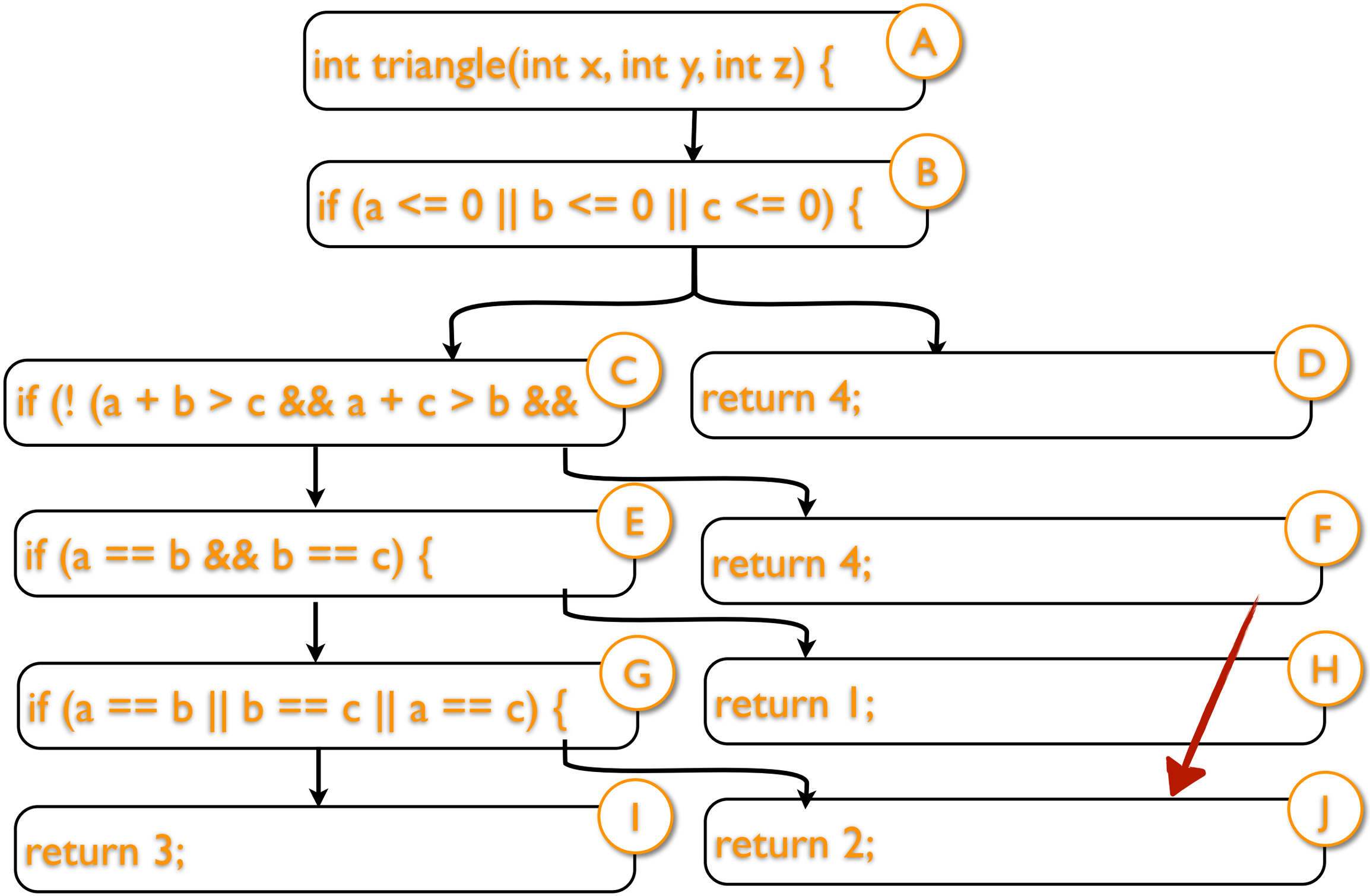
H

```
return 3;
```

I

```
return 2;
```

J


int triangle(int x, int y, int z) {
A
if (a <= 0 || b <= 0 || c <= 0) {
B
if (! (a + b > c && a + c > b &&
C
return 4;
D
if (a == b && b == c) {
E
return 4;
F
if (a == b || b == c || a == c) {
G
return 1;
H
return 3;
I
return 2;
J

(0, 3, 6)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
if (a == b && b == c) {
if (a == b || b == c || a == c) {
return 3;
return 4;
return 4;
return 1;
return 2;
```

C E F G H I J

(0, 3, 6)

```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
if (a == b && b == c) {
if (a == b || b == c || a == c) {
return 3;
return 4;
return 4;
return 1;
return 2;
```

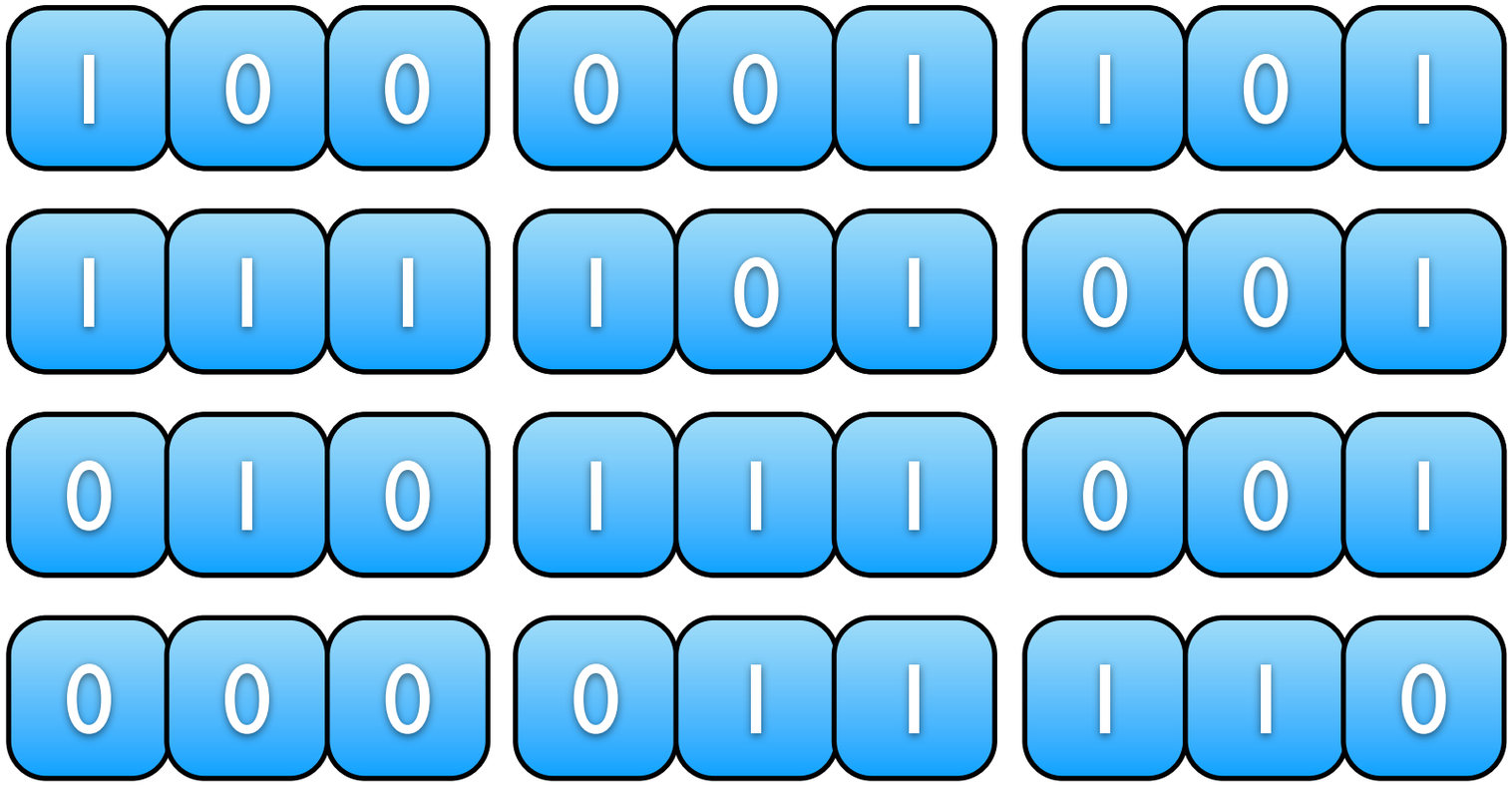
C E F G H I J

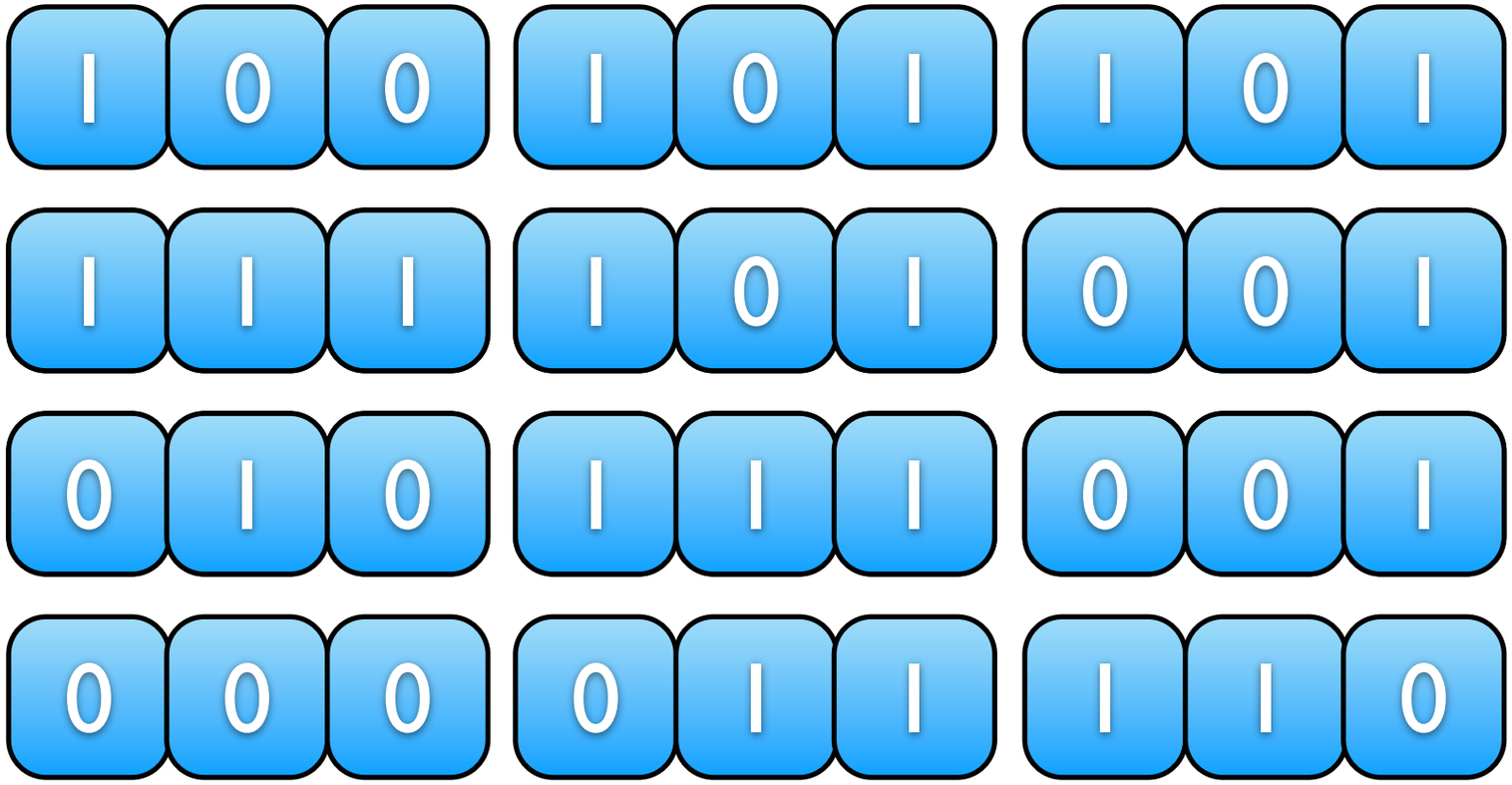
(0, 3, 6)

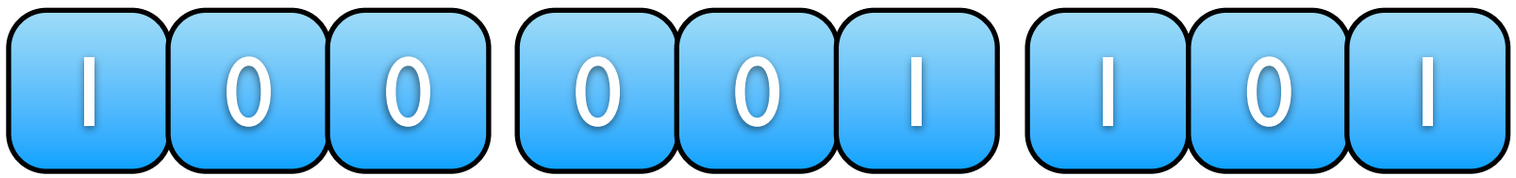
```
int triangle(int x, int y, int z) {
if (a <= 0 || b <= 0 || c <= 0) {
if (! (a + b > c && a + c > b &&
if (a == b && b == c) {
if (a == b || b == c || a == c) {
return 3;
return 4;
return 4;
return 1;
return 2;
```

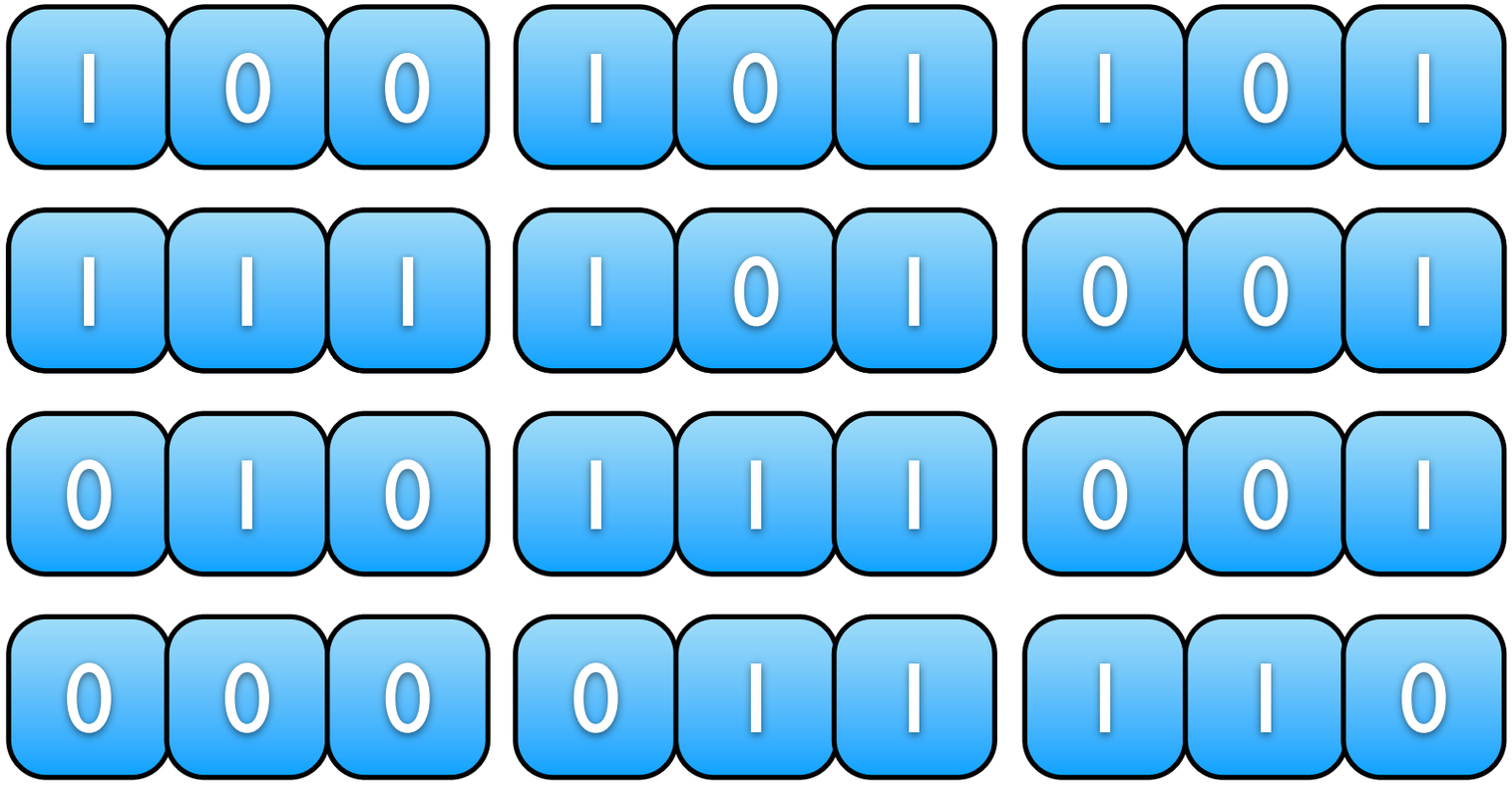
C E F G H I J

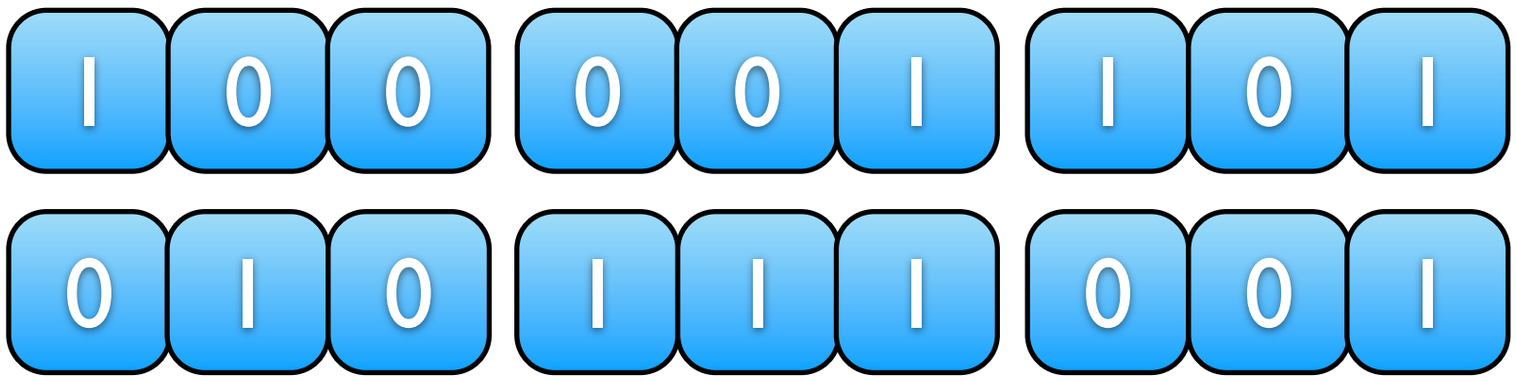
Approach: 4
Branch: 1

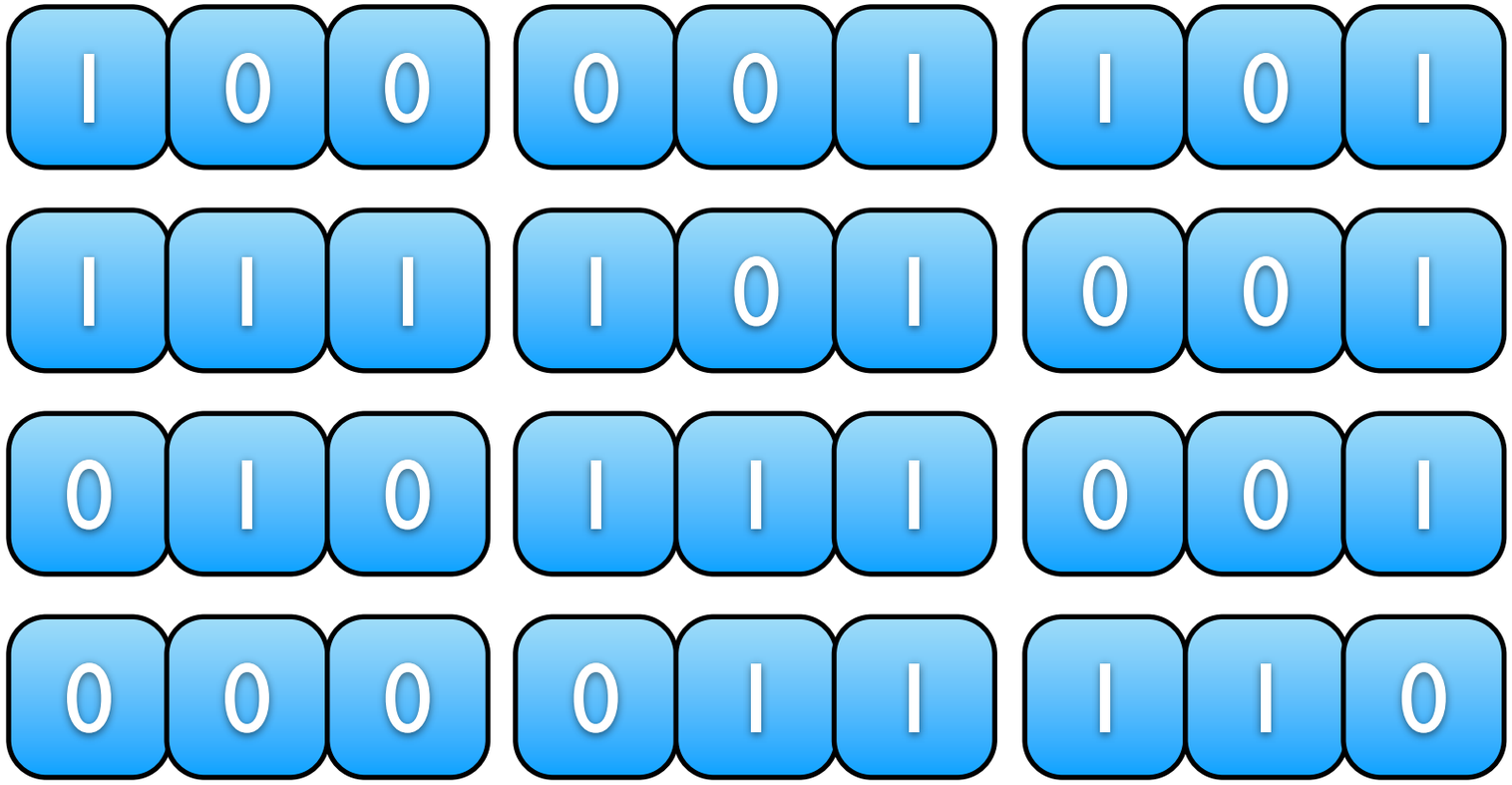

1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2
0 1 0 1 1 1 0 0 1
(2,7,1) A: 3, B: 5
0 0 0 0 1 1 1 1 0
(0,3,6) A: 4, B: 1


1 0 0 1 0 1 1 0 1
(4,1,5)
A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1)
A: 3, B: 2
0 1 0 1 1 1 0 0 1
(2,7,1)
A: 3, B: 5
0 0 0 0 1 1 1 1 0
(0,3,6)
A: 4, B: 1


1 0 0 0 0 1 1 0 1


1 0 0 1 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2
0 1 0 1 1 1 0 0 1
(2,7,1) A: 3, B: 5
0 0 0 0 1 1 1 1 0
(0,3,6) A: 4, B: 1


1 0 0 0 0 1 1 0 1
0 1 0 1 1 1 0 0 1

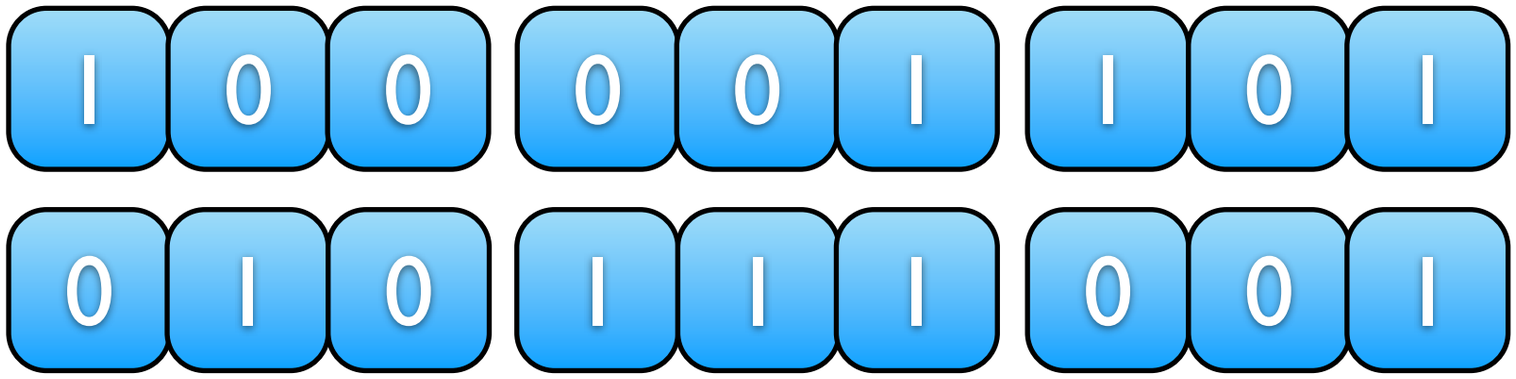


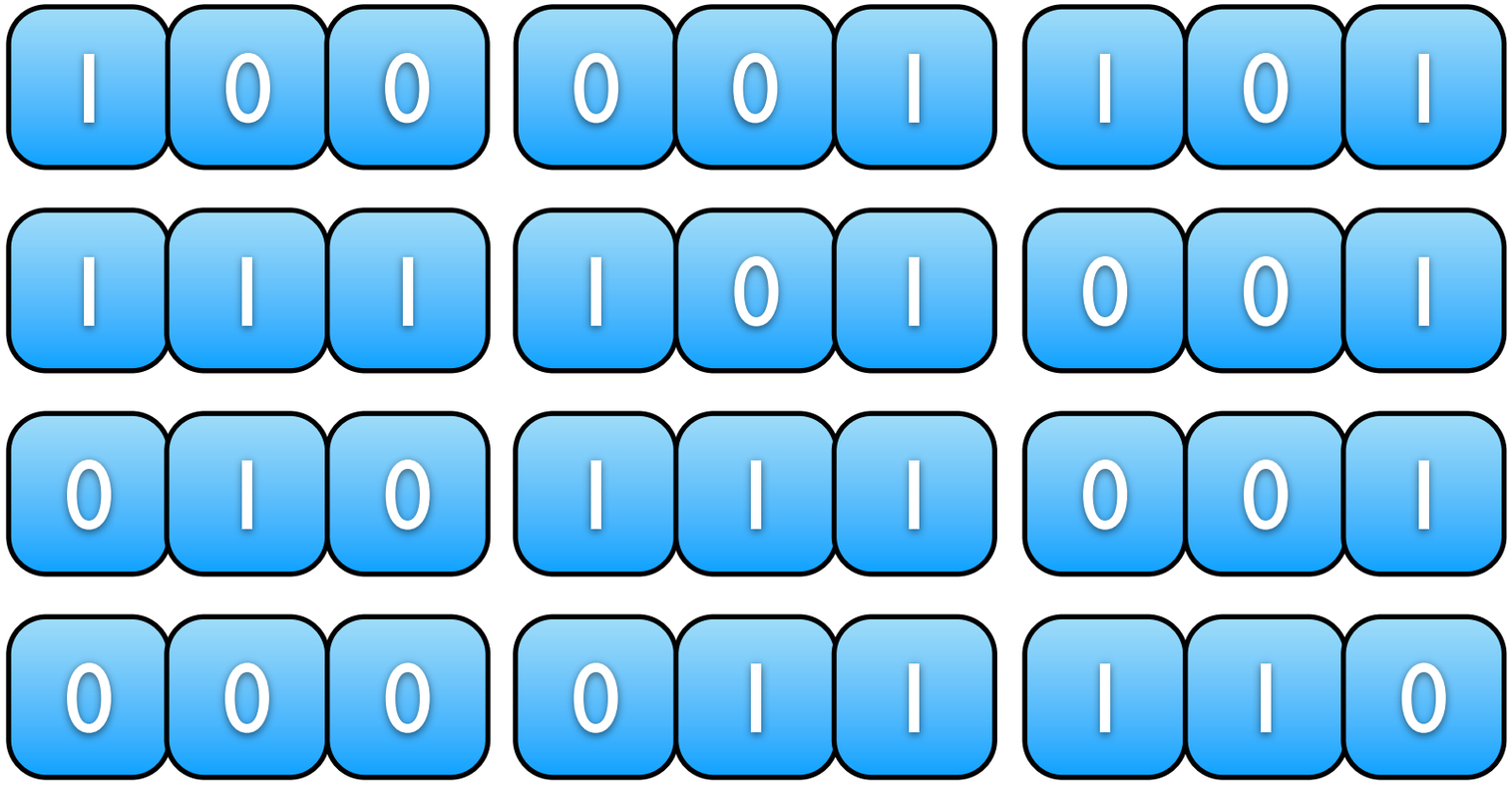
(4,1,5) A: 3, B: 1

(7,5,1) A: 3, B: 2

(2,7,1) A: 3, B: 5

(0,3,6) A: 4, B: 1



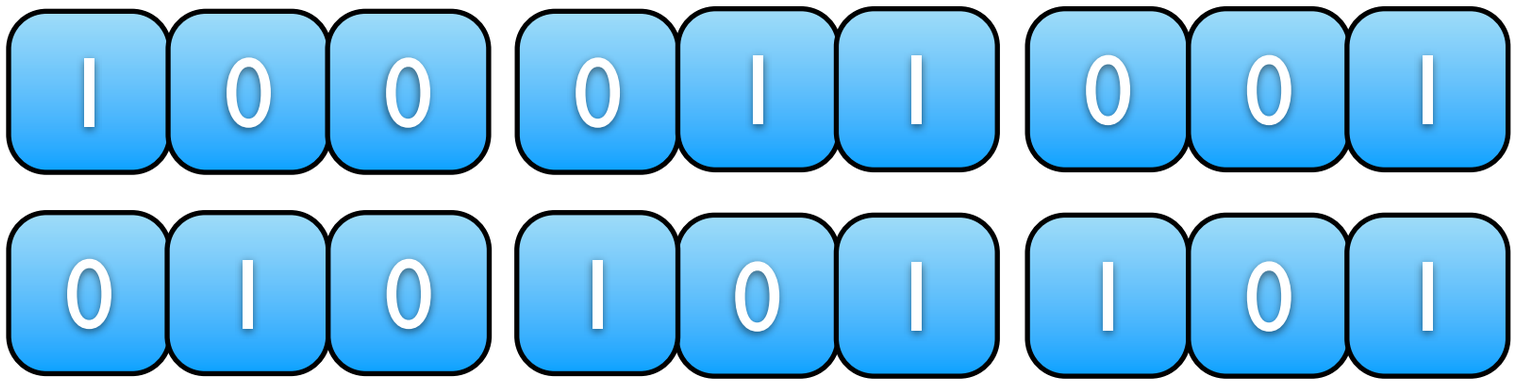


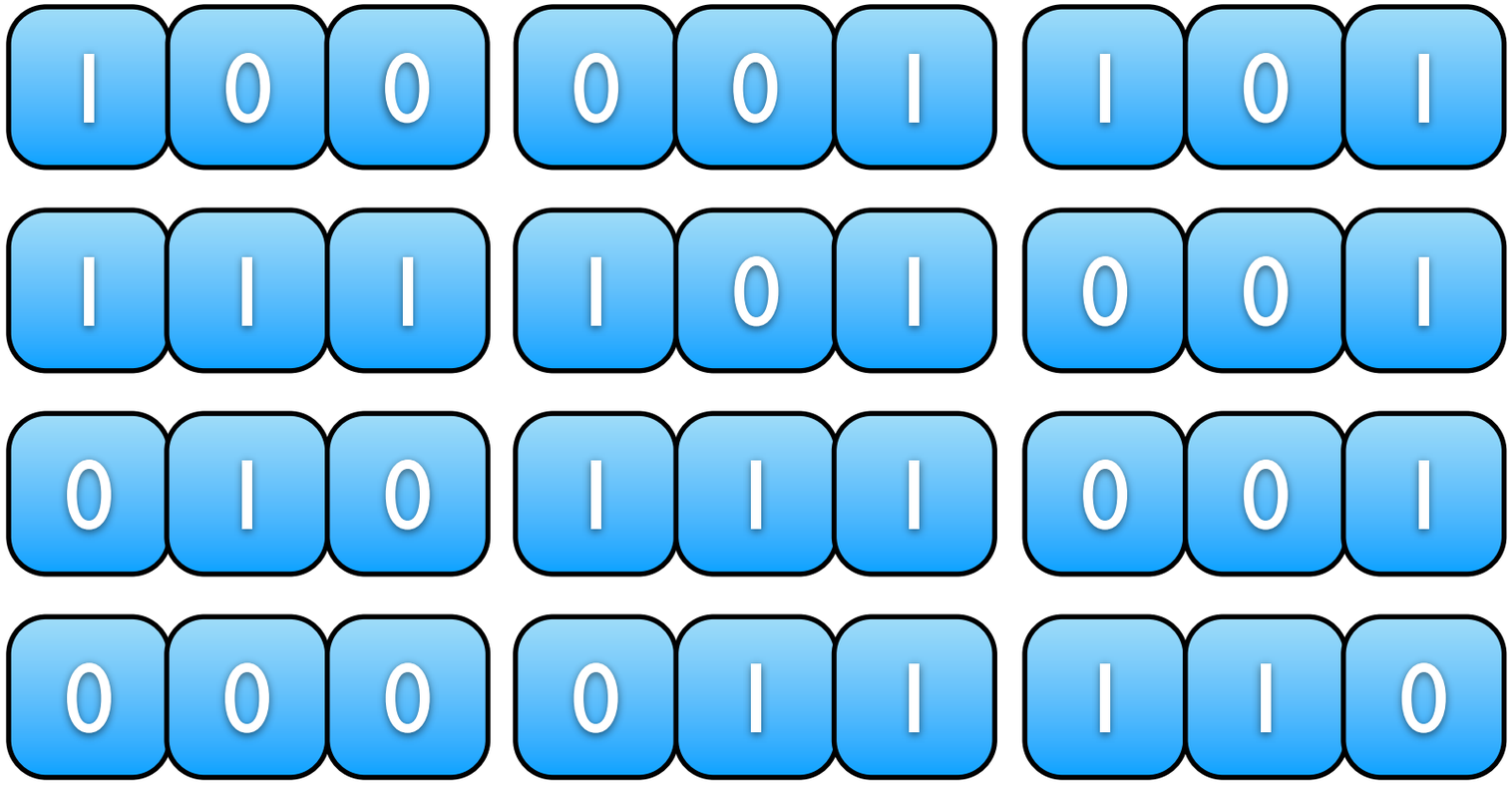
(4,1,5) A: 3, B: 1

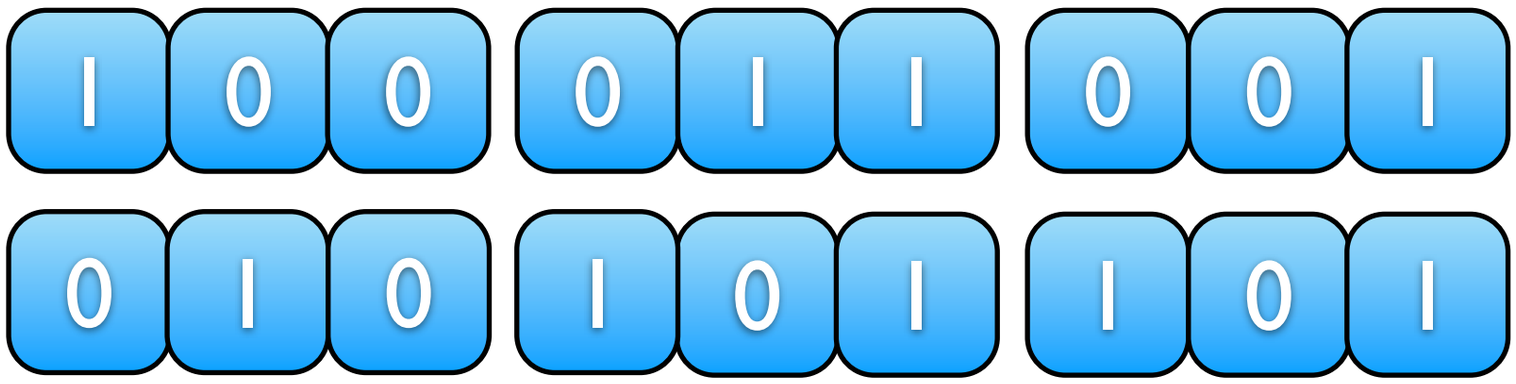
(7,5,1) A: 3, B: 2

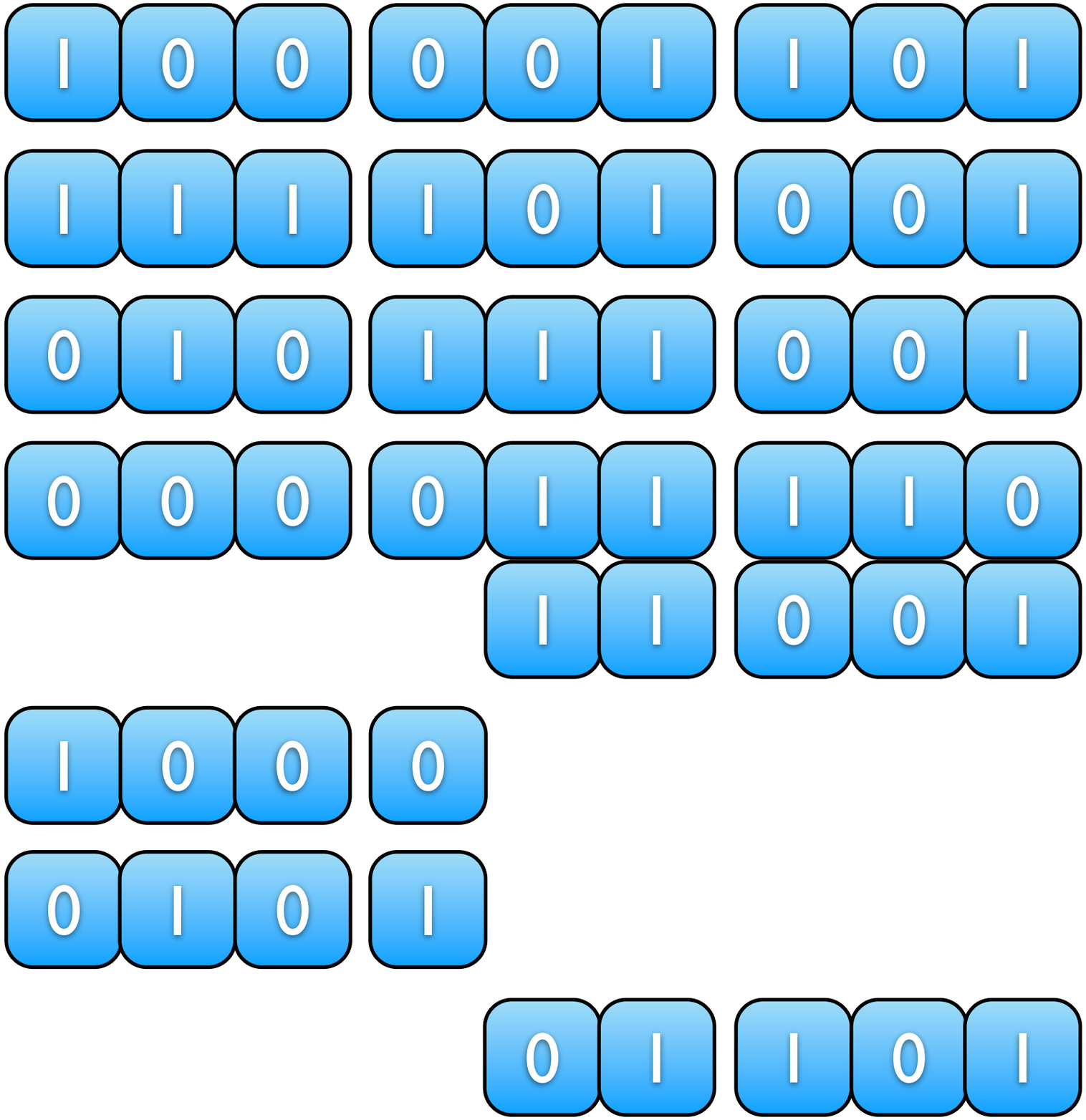
(2,7,1) A: 3, B: 5

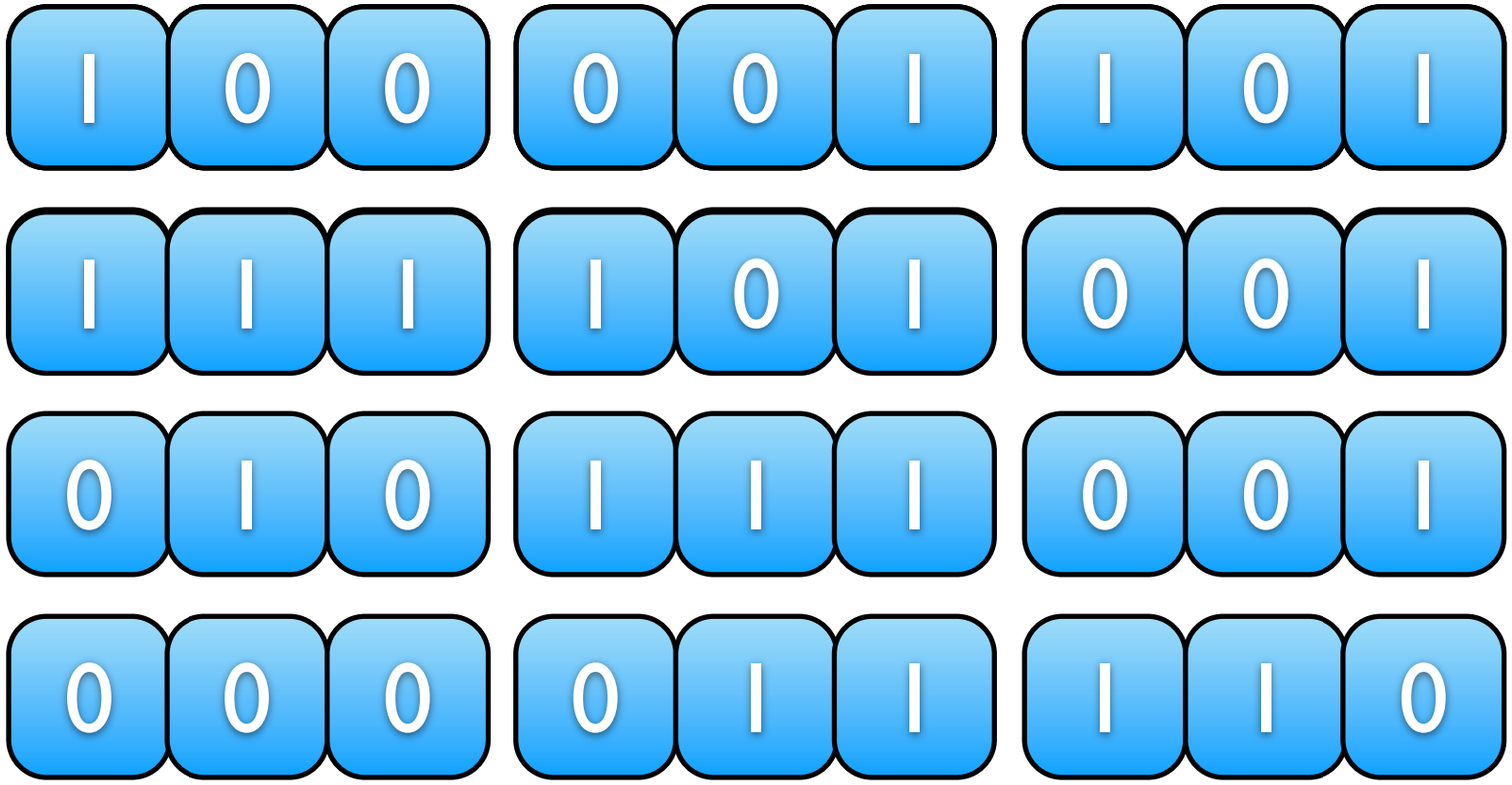
(0,3,6) A: 4, B: 1

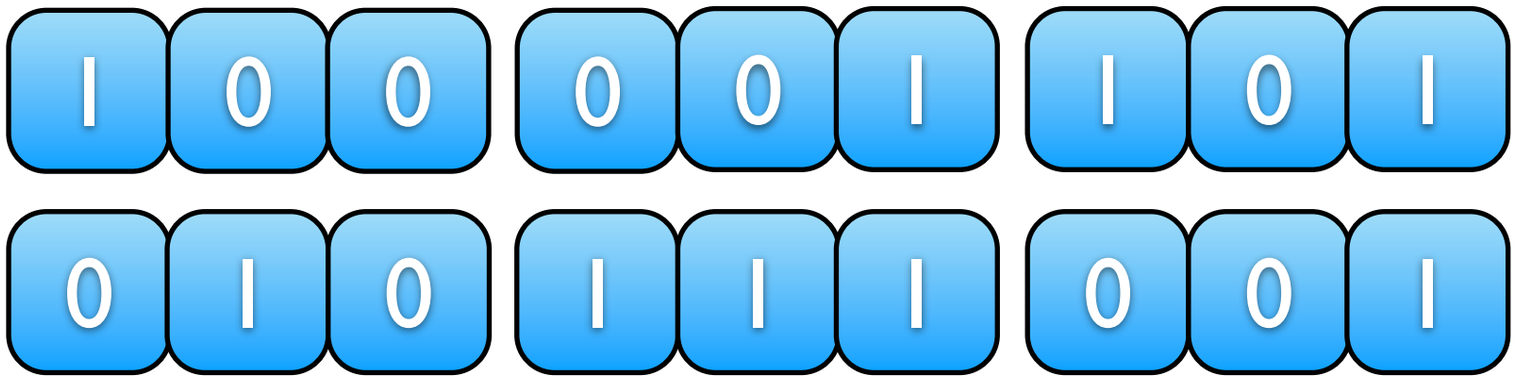


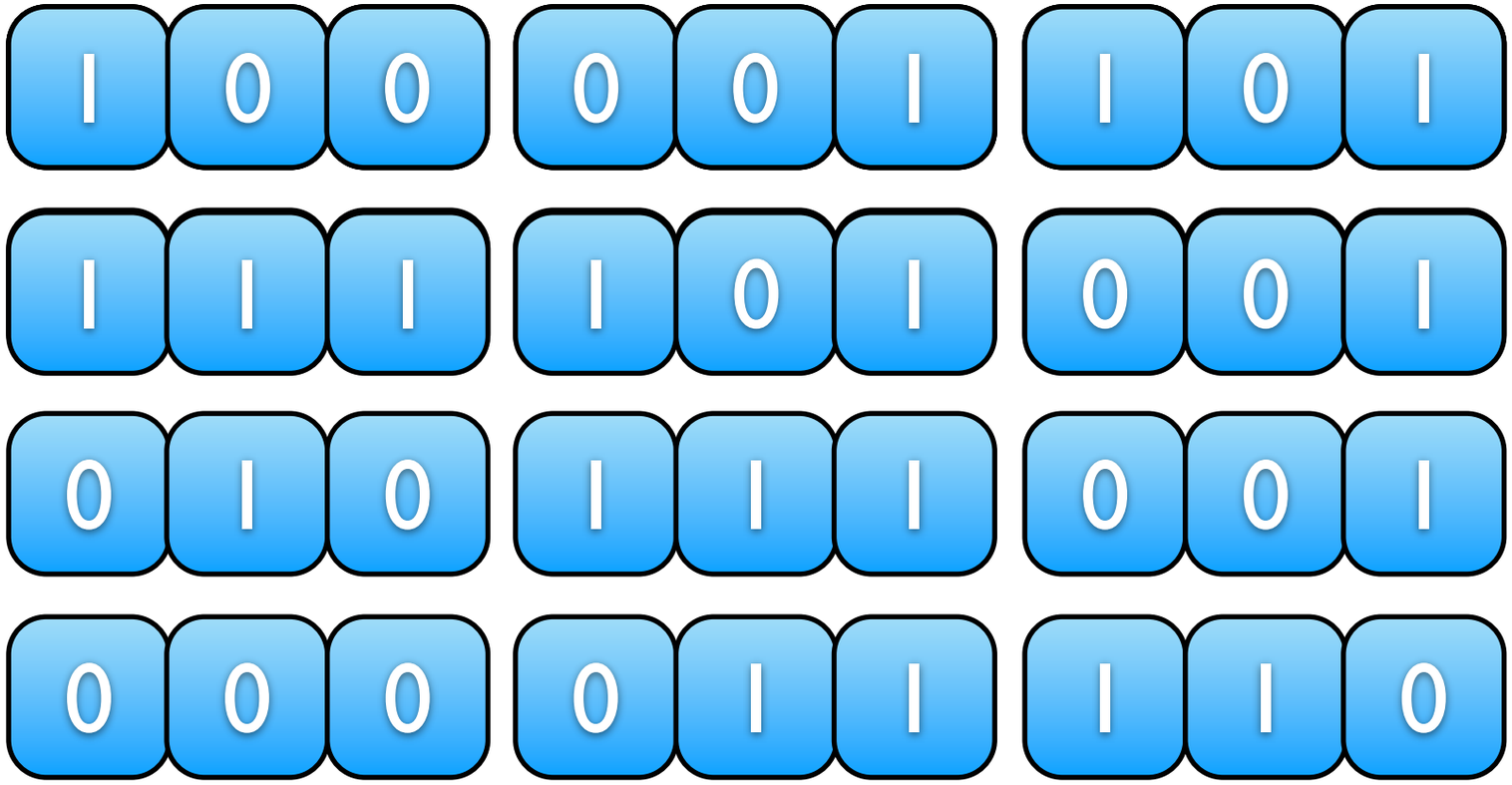

1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2
0 1 0 1 1 1 0 0 1
(2,7,1) A: 3, B: 5
0 0 0 0 1 1 1 1 0
(0,3,6) A: 4, B: 1

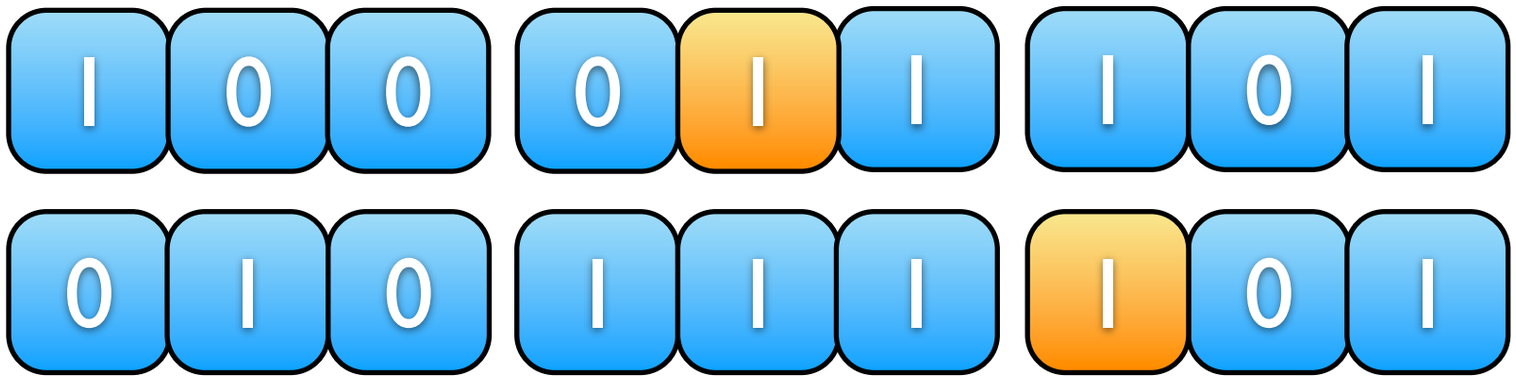

1 0 0 0 1 1 0 0 1
0 1 0 1 0 1 1 0 1

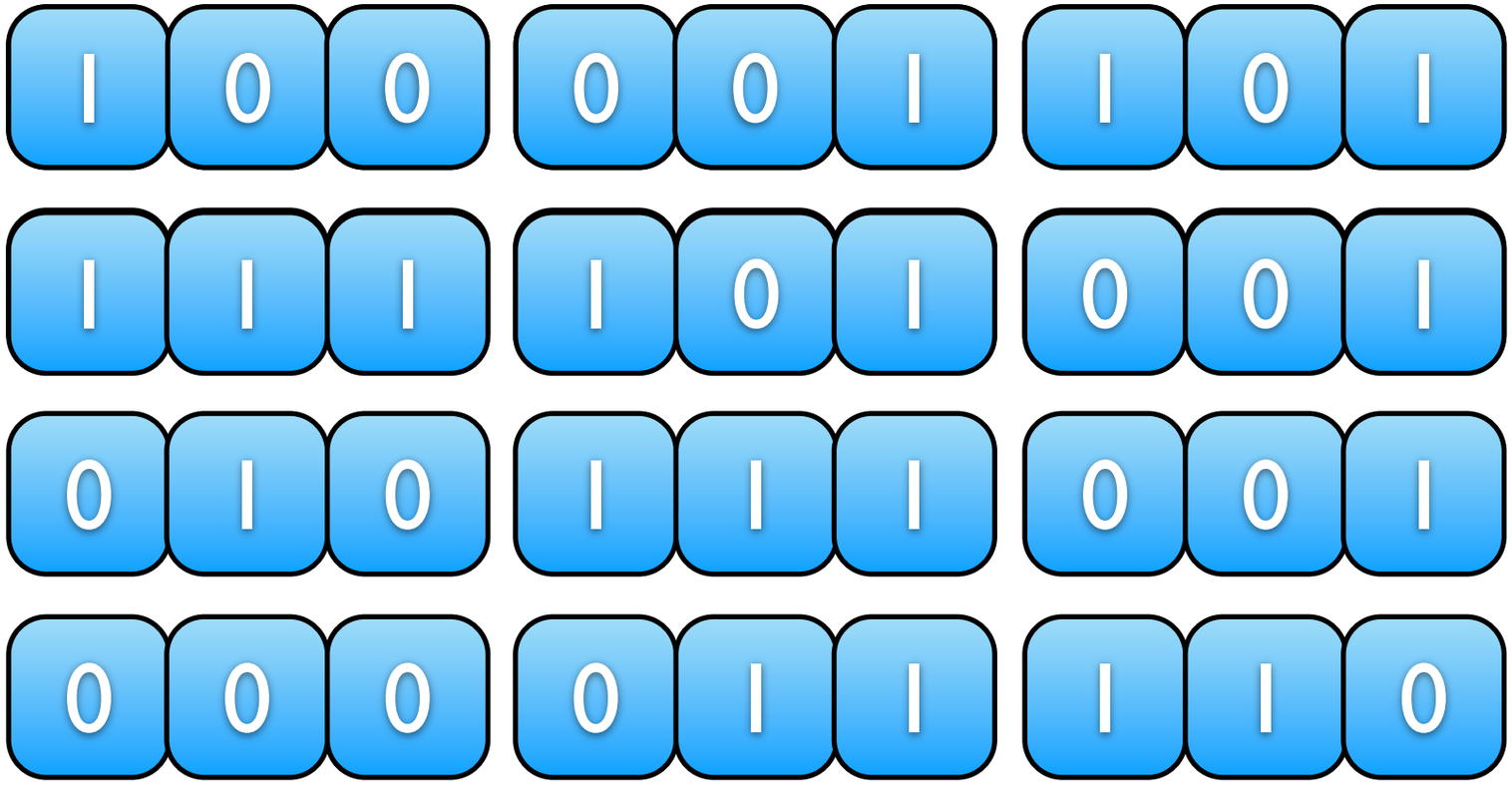

1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2
0 1 0 1 1 1 0 0 1
(2,7,1) A: 3, B: 5
0 0 0 0 1 1 1 1 0
1 1 0 0 1
(0,3,6) A: 4, B: 1
1 0 0 0
0 1 0 1
0 1 1 0 1

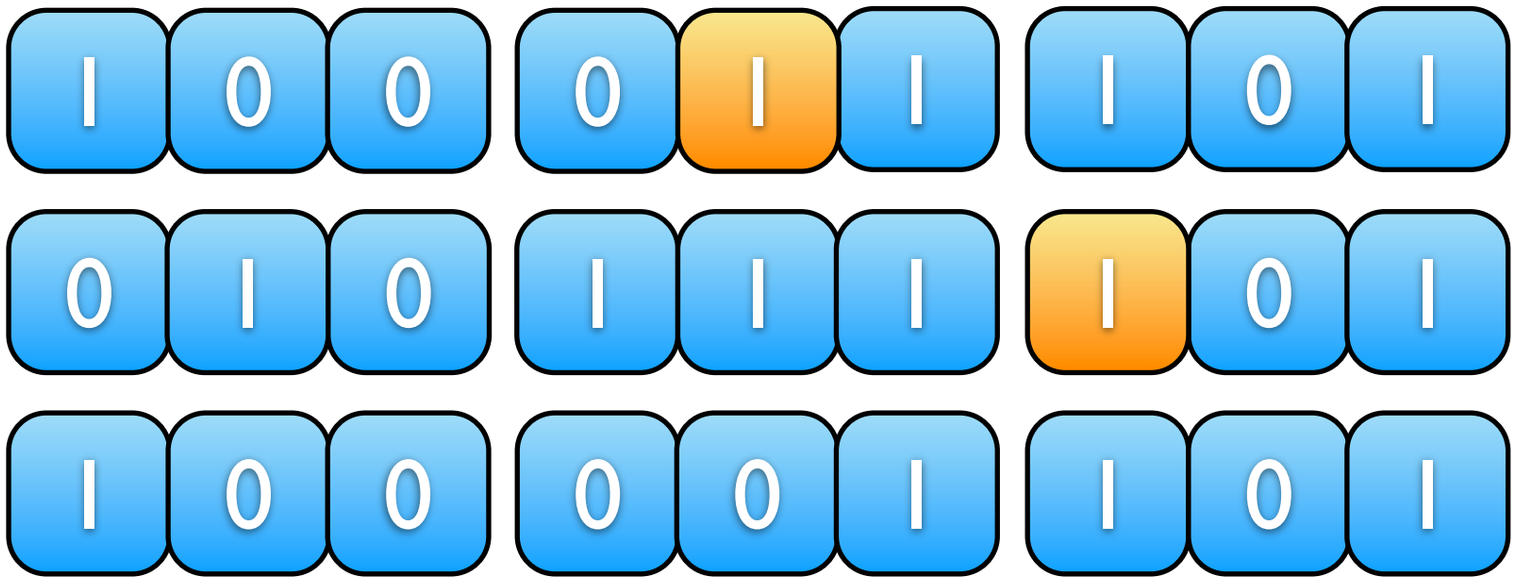

1 0 0 0 0 1 1 0 1
(4,1,5)
A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1)
A: 3, B: 2
0 1 0 1 1 1 0 0 1
(2,7,1)
A: 3, B: 5
0 0 0 0 1 1 1 1 0
(0,3,6)
A: 4, B: 1

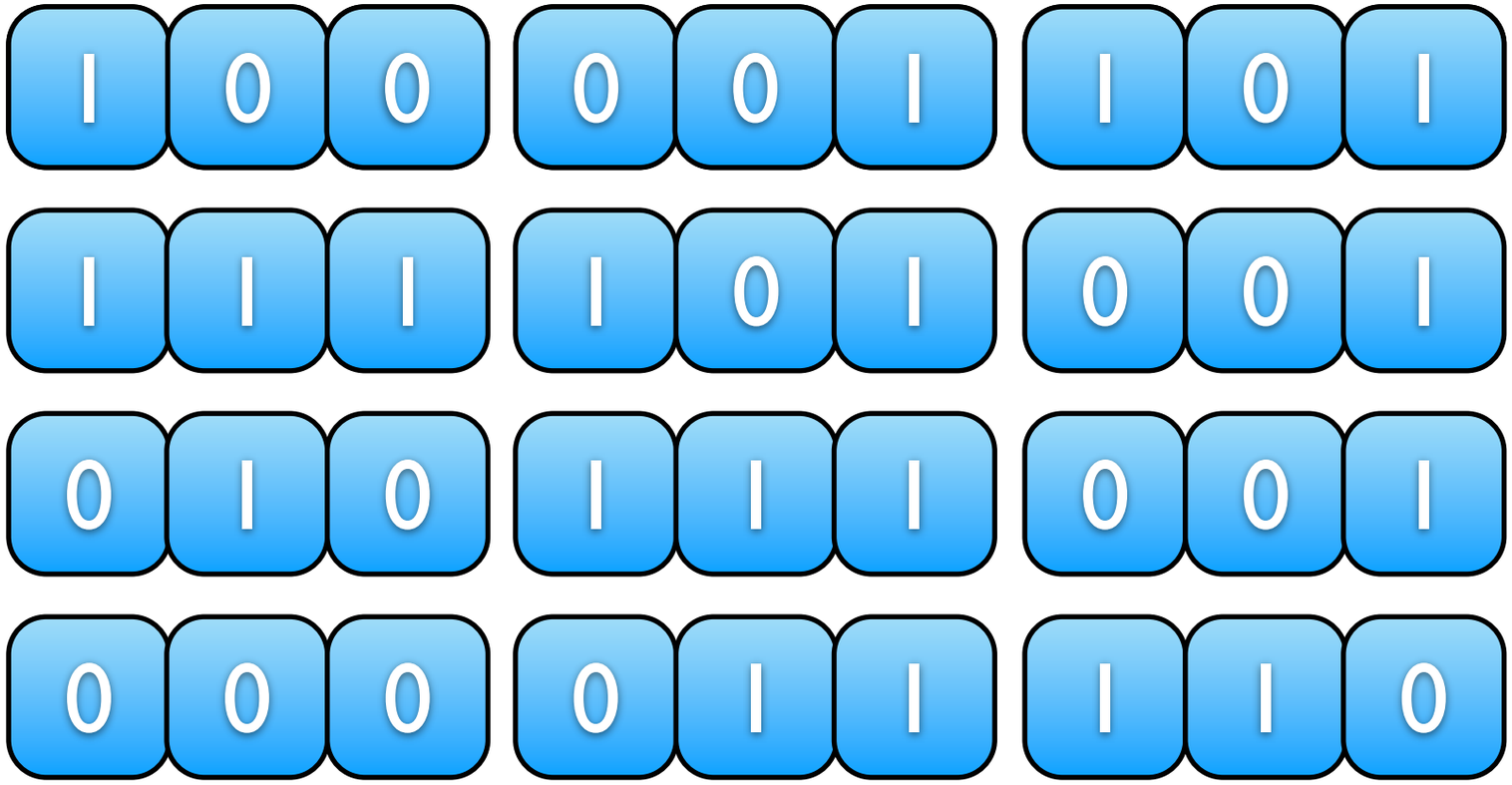

1 0 0 0 0 1 1 0 1
0 1 0 1 1 1 0 0 1

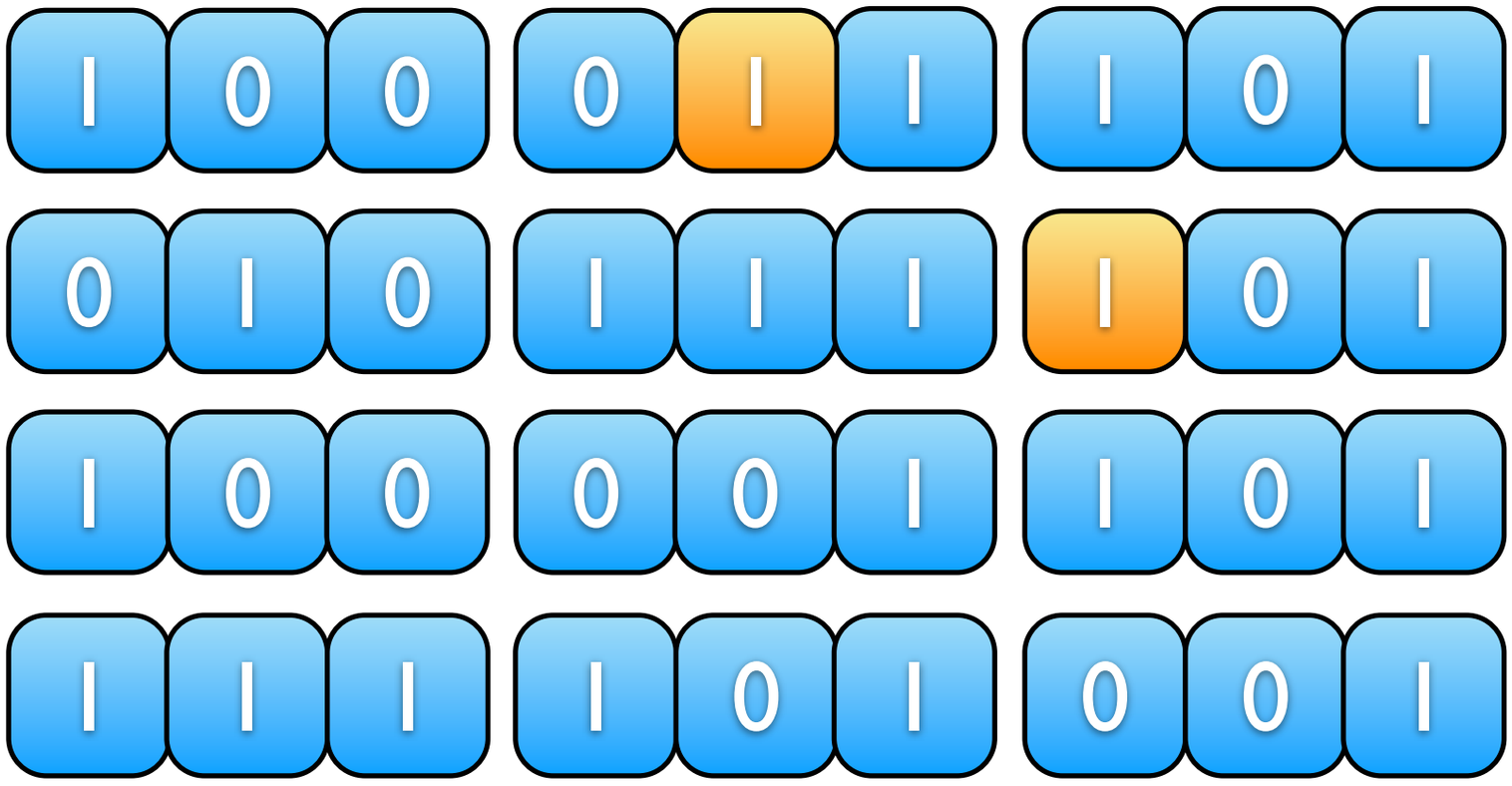

1 0 0 0 0 1 1 0 1
(4,1,5)
A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1)
A: 3, B: 2
0 1 0 1 1 1 0 0 1
(2,7,1)
A: 3, B: 5
0 0 0 0 1 1 1 1 0
(0,3,6)
A: 4, B: 1

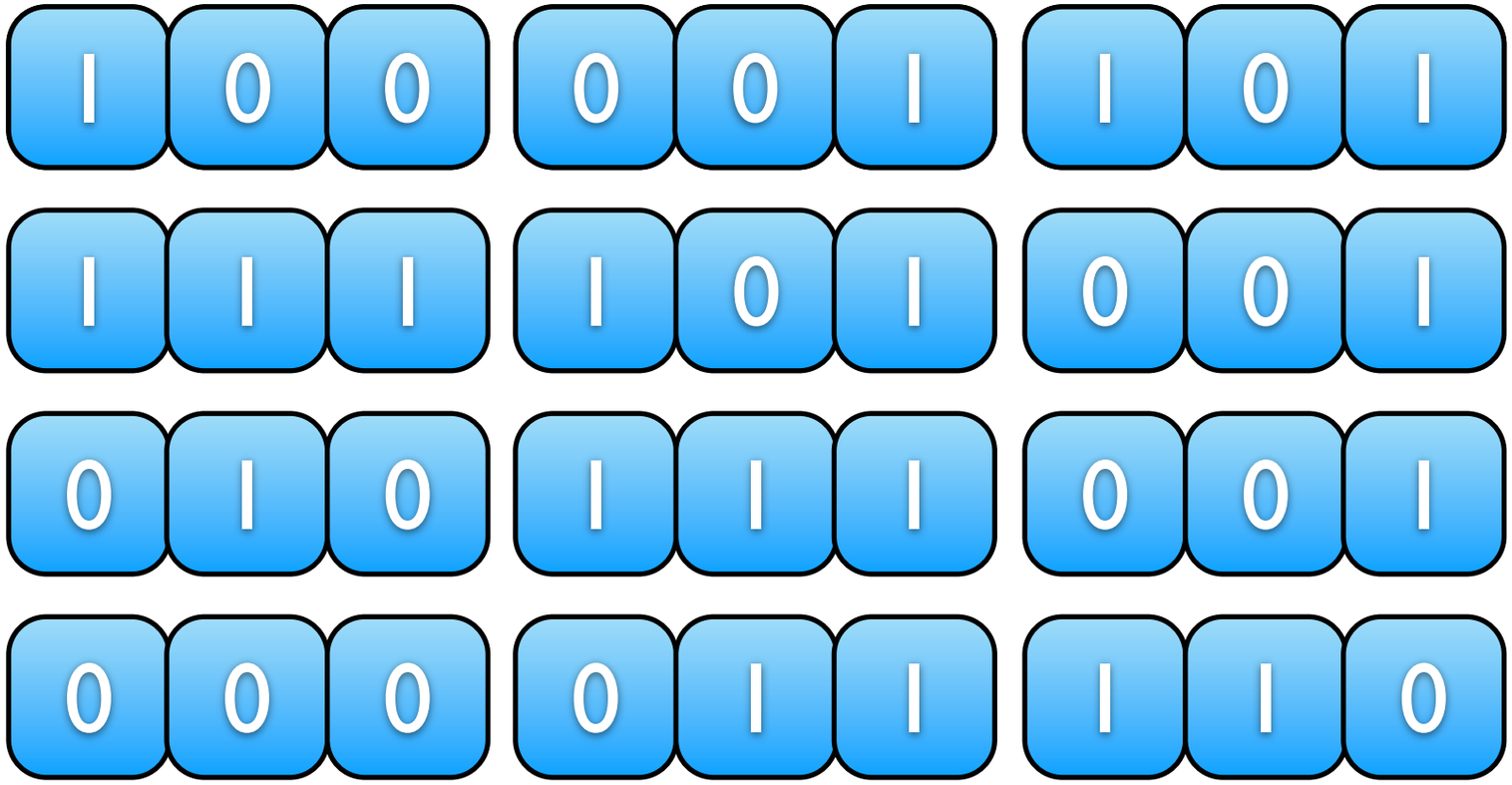

1 0 0 0 1 1 1 0 1
0 1 0 1 1 1 1 0 1


1 0 0 0 0 1 1 0 1
(4,1,5)
A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1)
A: 3, B: 2
0 1 0 1 1 1 0 0 1
(2,7,1)
A: 3, B: 5
0 0 0 0 1 1 1 1 0
(0,3,6)
A: 4, B: 1


1 0 0 0 1 1 1 0 1
0 1 0 1 1 1 1 0 1
1 0 0 0 0 1 1 0 1


1 0 0 0 0 1 1 0 1
(4,1,5)
A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1)
A: 3, B: 2
0 1 0 1 1 1 0 0 1
(2,7,1)
A: 3, B: 5
0 0 0 0 1 1 1 1 0
(0,3,6)
A: 4, B: 1


1 0 0 0 1 1 1 0 1
0 1 0 1 1 1 1 0 1
1 0 0 0 0 1 1 0 1
1 1 1 1 0 1 0 0 1

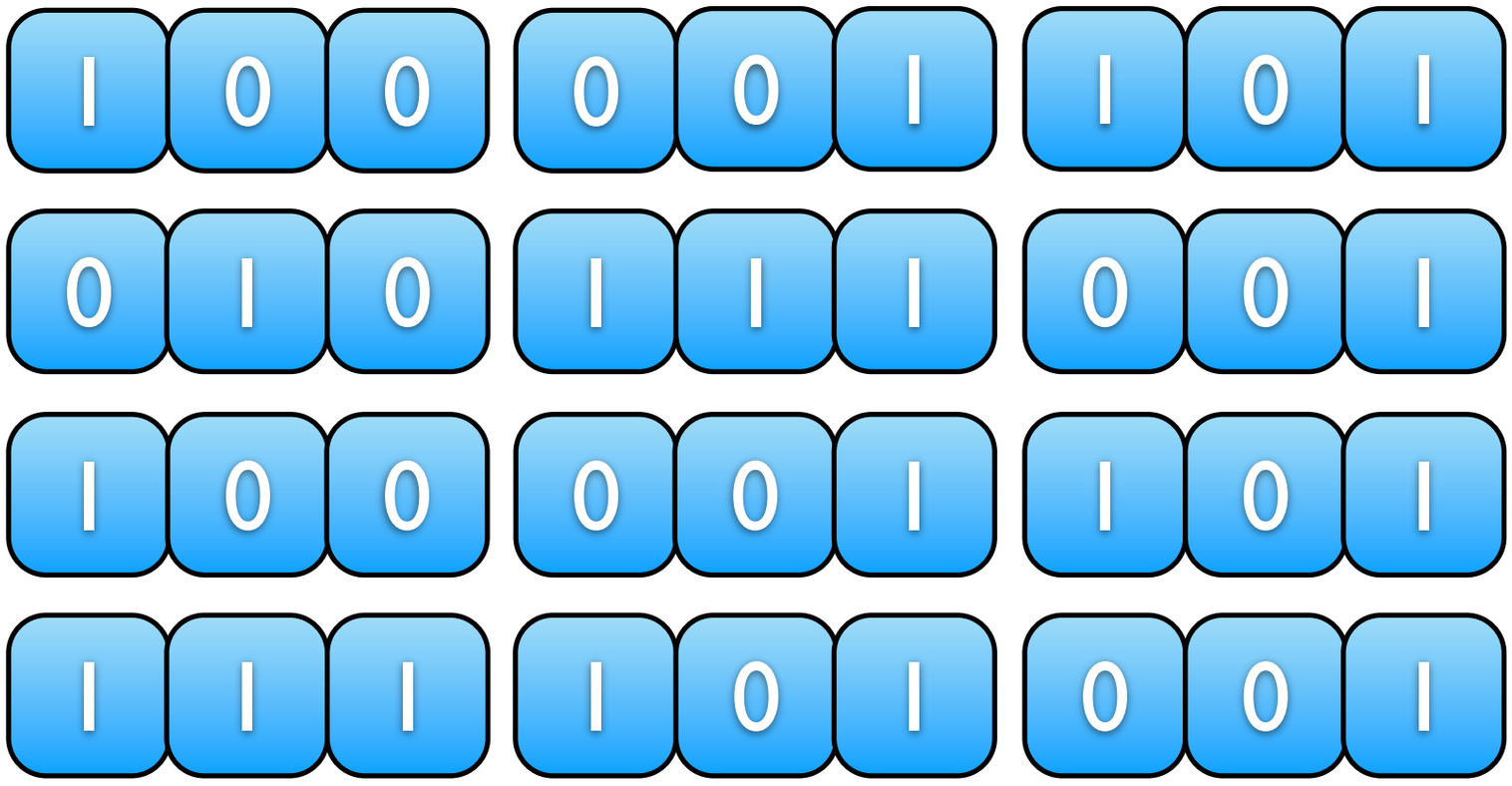


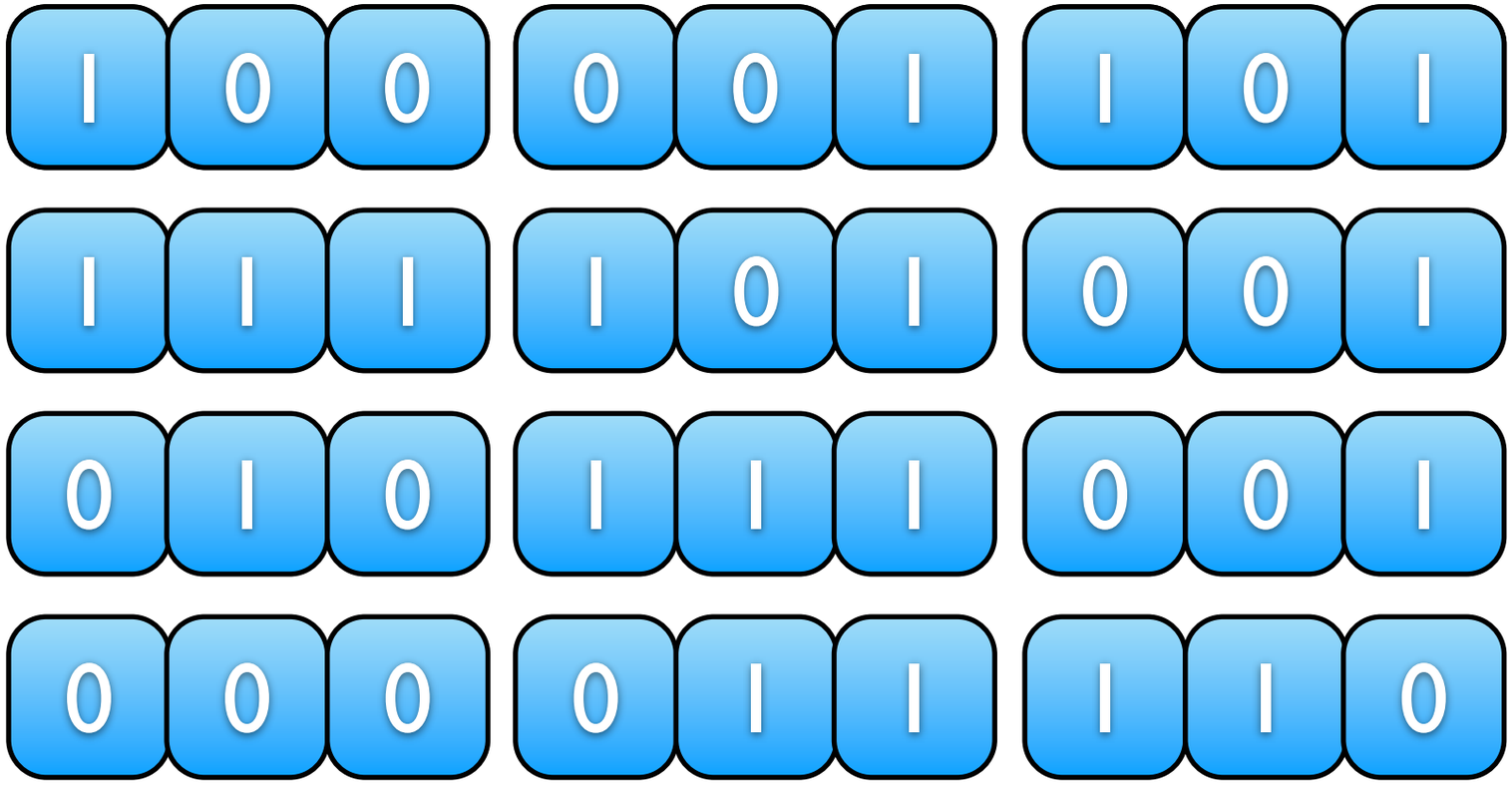
(4,1,5) A: 3, B: 1

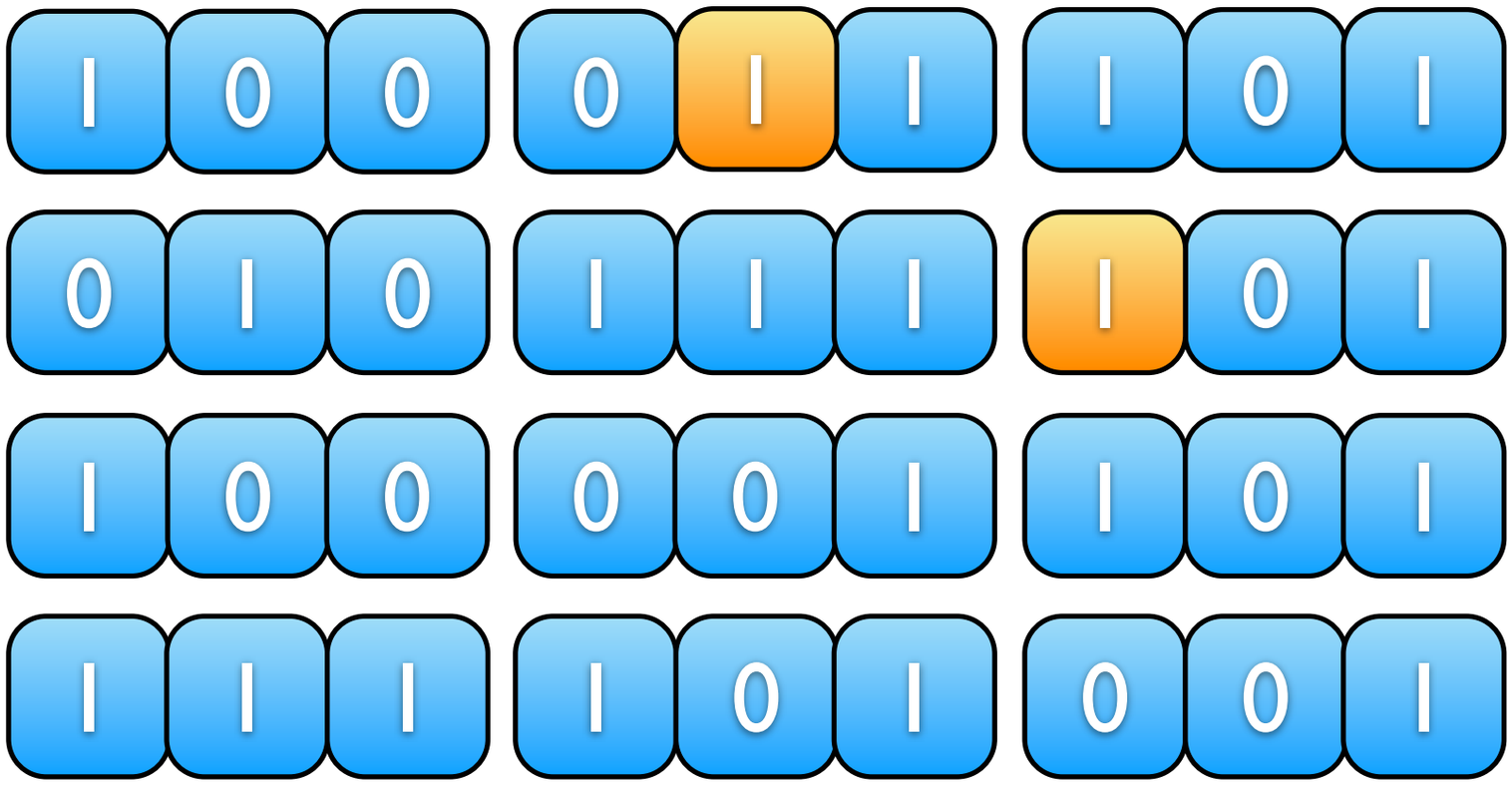
(7,5,1) A: 3, B: 2

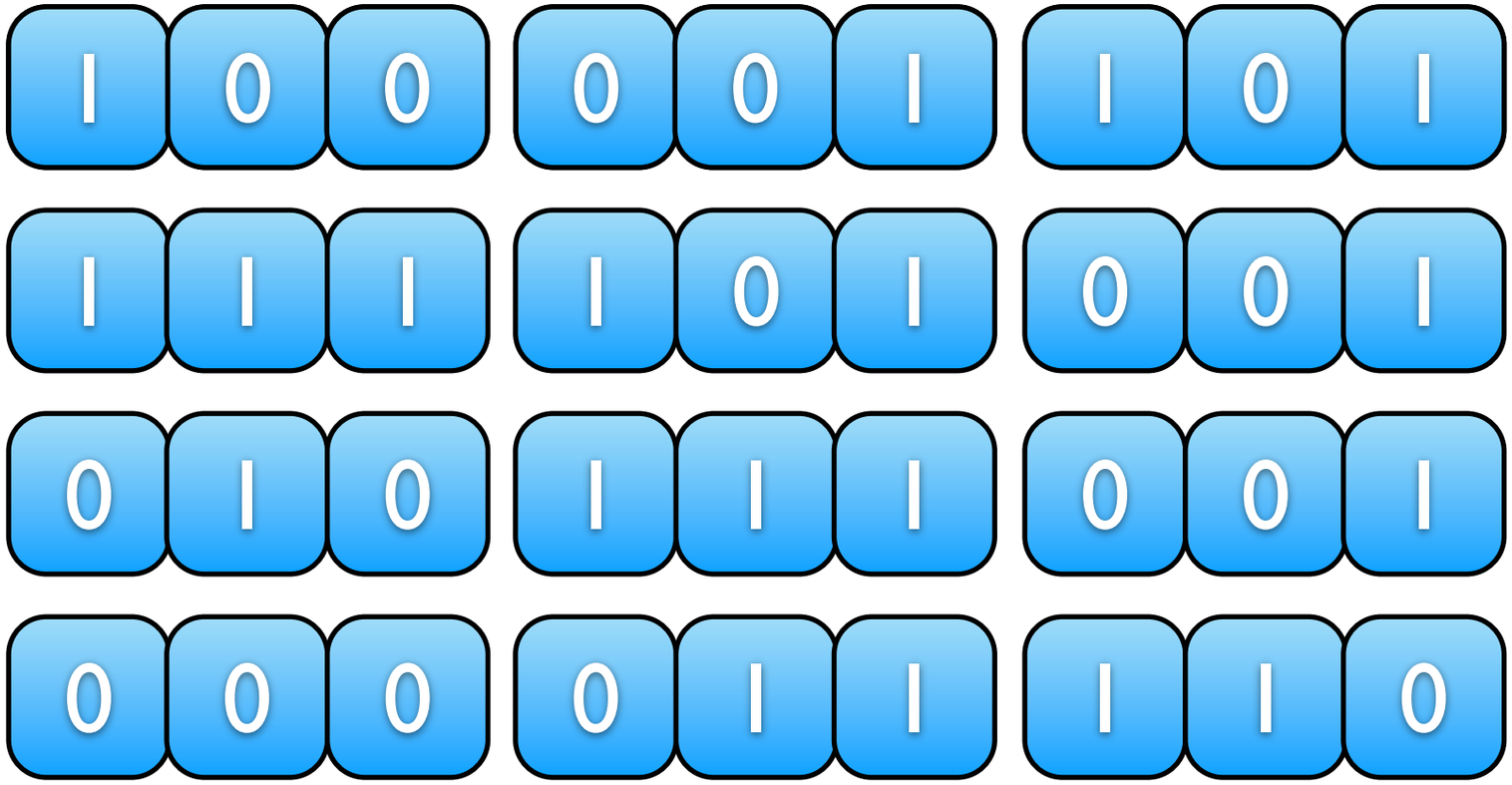
(2,7,1) A: 3, B: 5

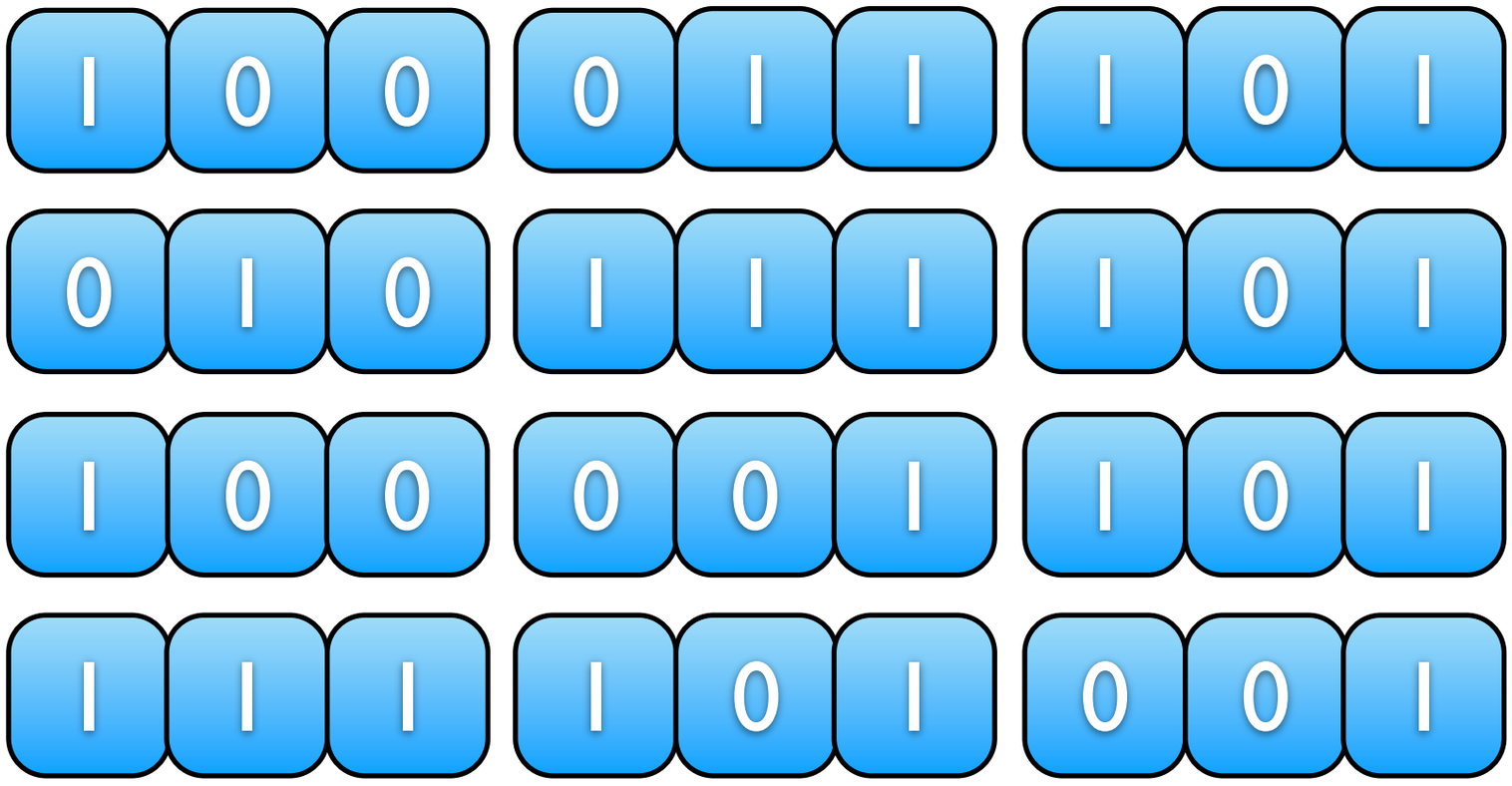
(0,3,6) A: 4, B: 1

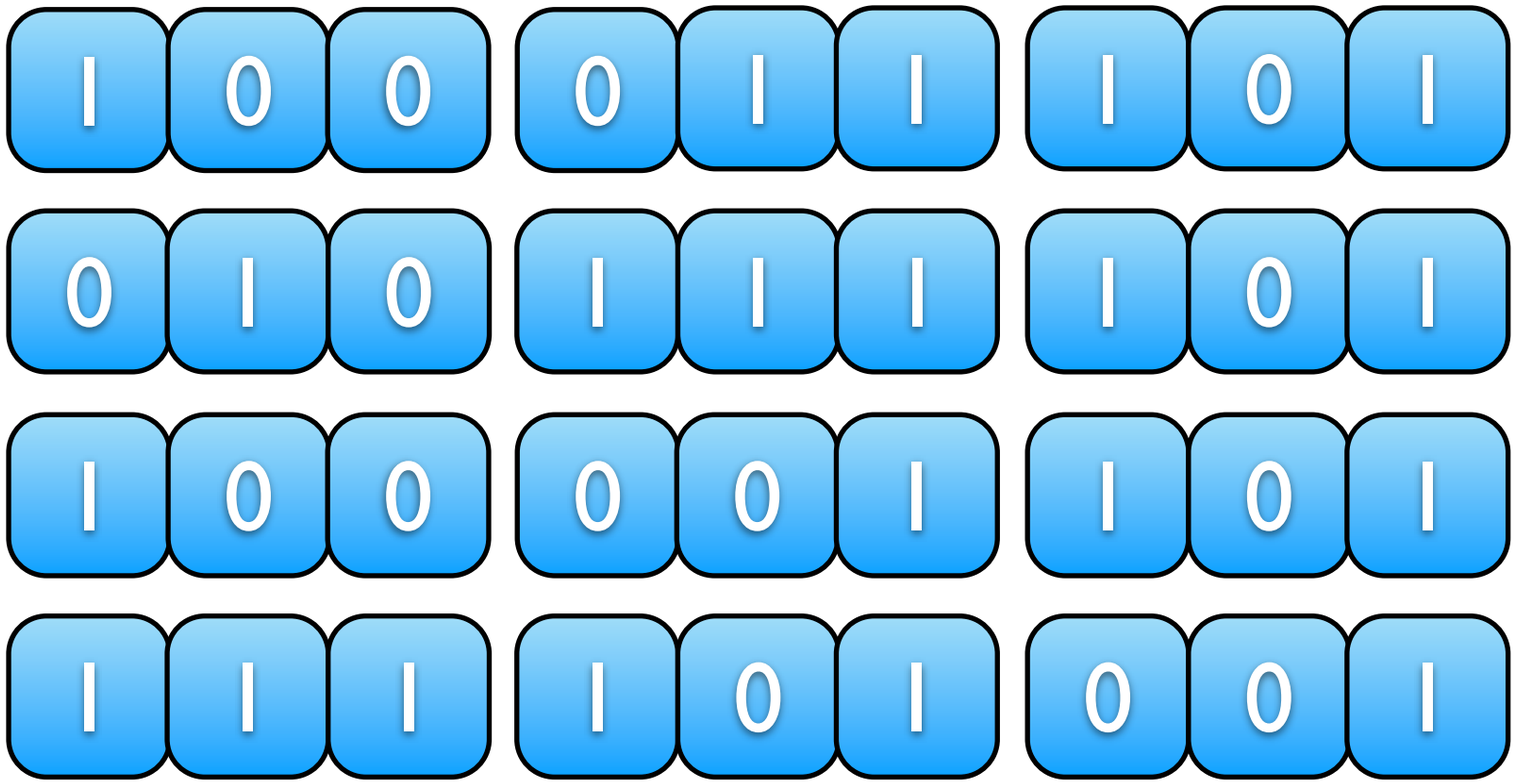


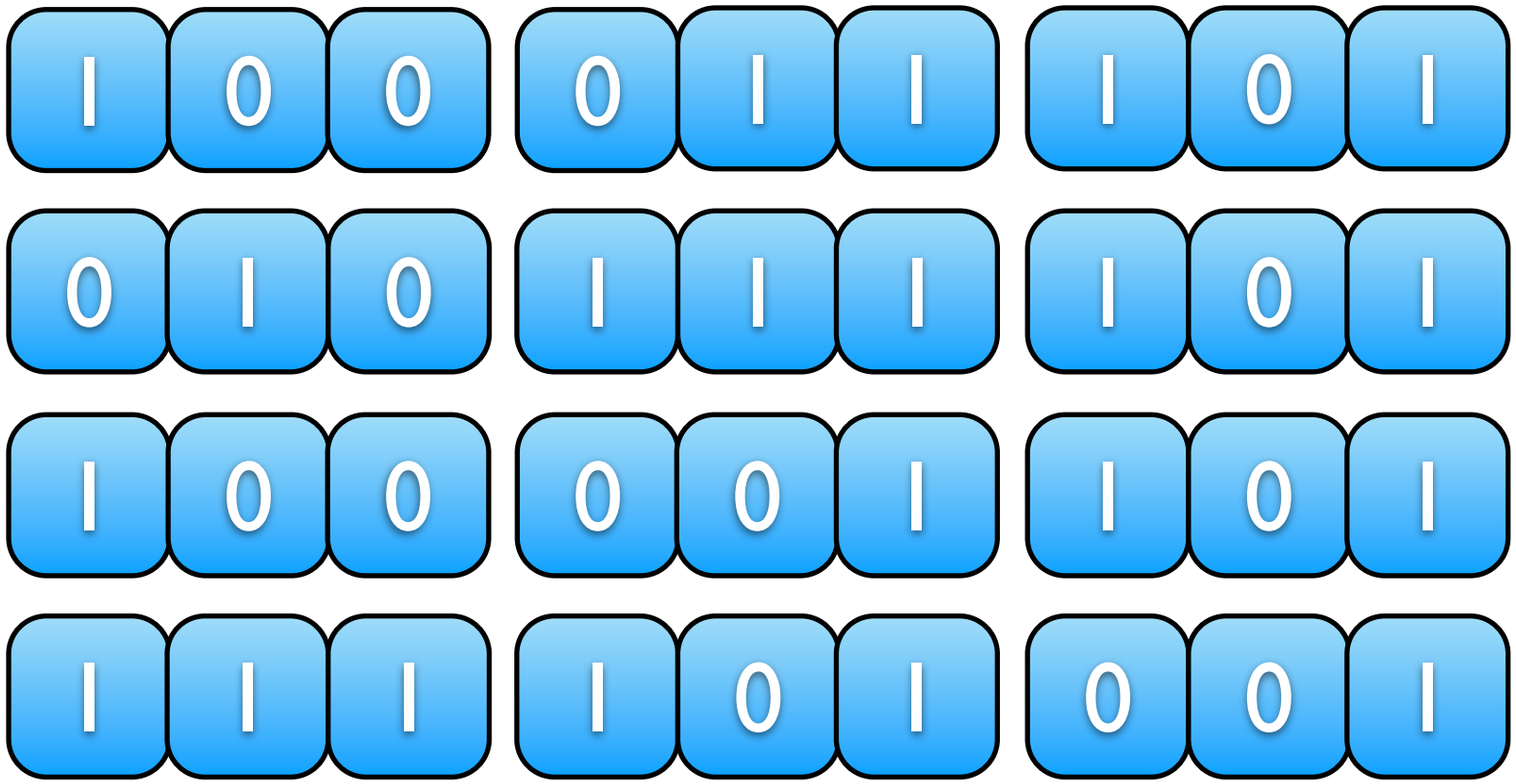

1 0 0 0 0 1 1 0 1
(4,1,5)
A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1)
A: 3, B: 2
0 1 0 1 1 1 0 0 1
(2,7,1)
A: 3, B: 5
0 0 0 0 1 1 1 1 0
(0,3,6)
A: 4, B: 1


1 0 0 0 1 1 1 0 1
0 1 0 1 1 1 1 0 1
1 0 0 0 0 1 1 0 1
1 1 1 1 0 1 0 0 1

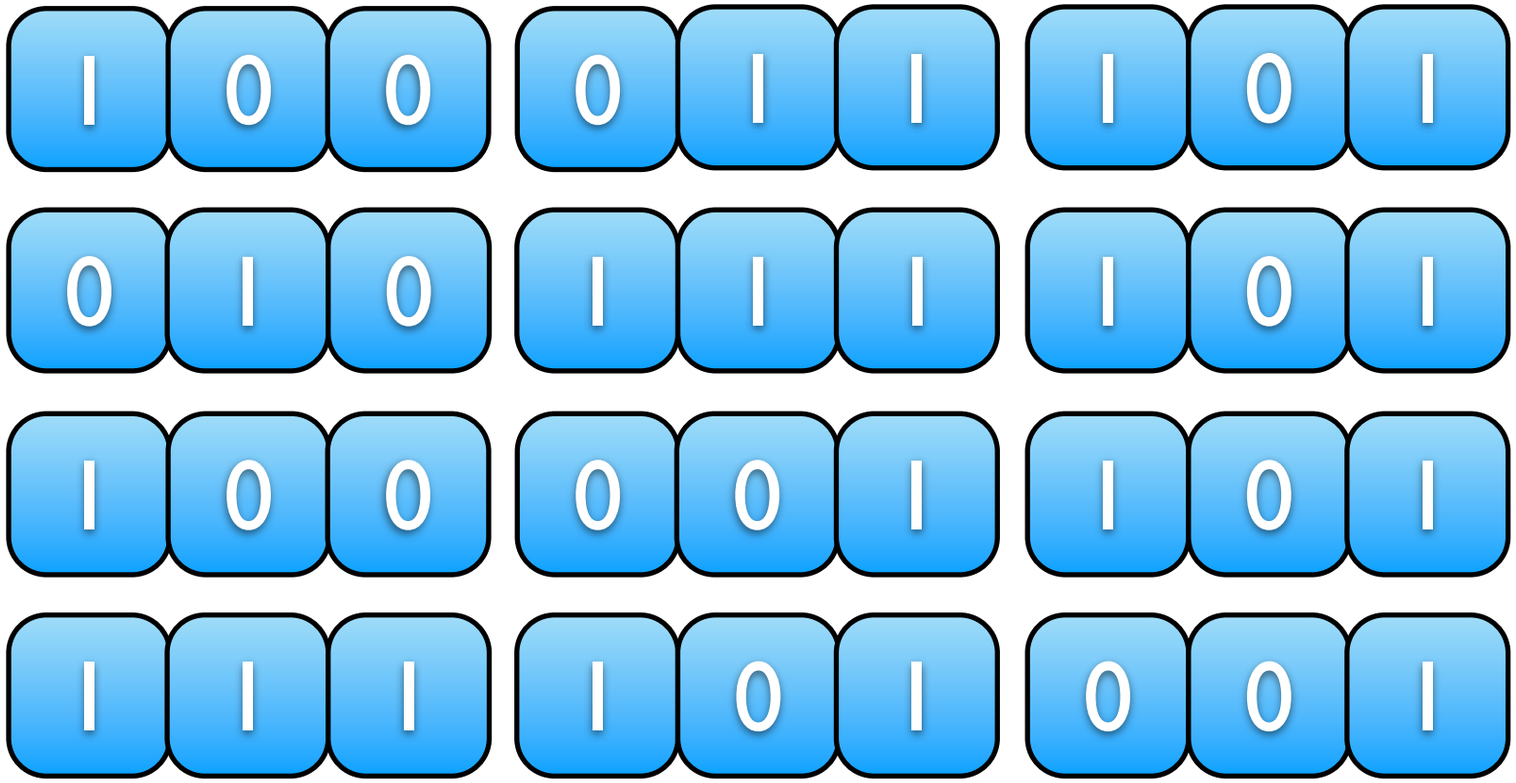

1 0 0 0 0 1 1 0 1
(4,1,5)
A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1)
A: 3, B: 2
0 1 0 1 1 1 0 0 1
(2,7,1)
A: 3, B: 5
0 0 0 0 1 1 1 1 0
(0,3,6)
A: 4, B: 1


1 0 0 0 1 1 1 0 1
0 1 0 1 1 1 1 0 1
1 0 0 0 0 1 1 0 1
1 1 1 1 0 1 0 0 1

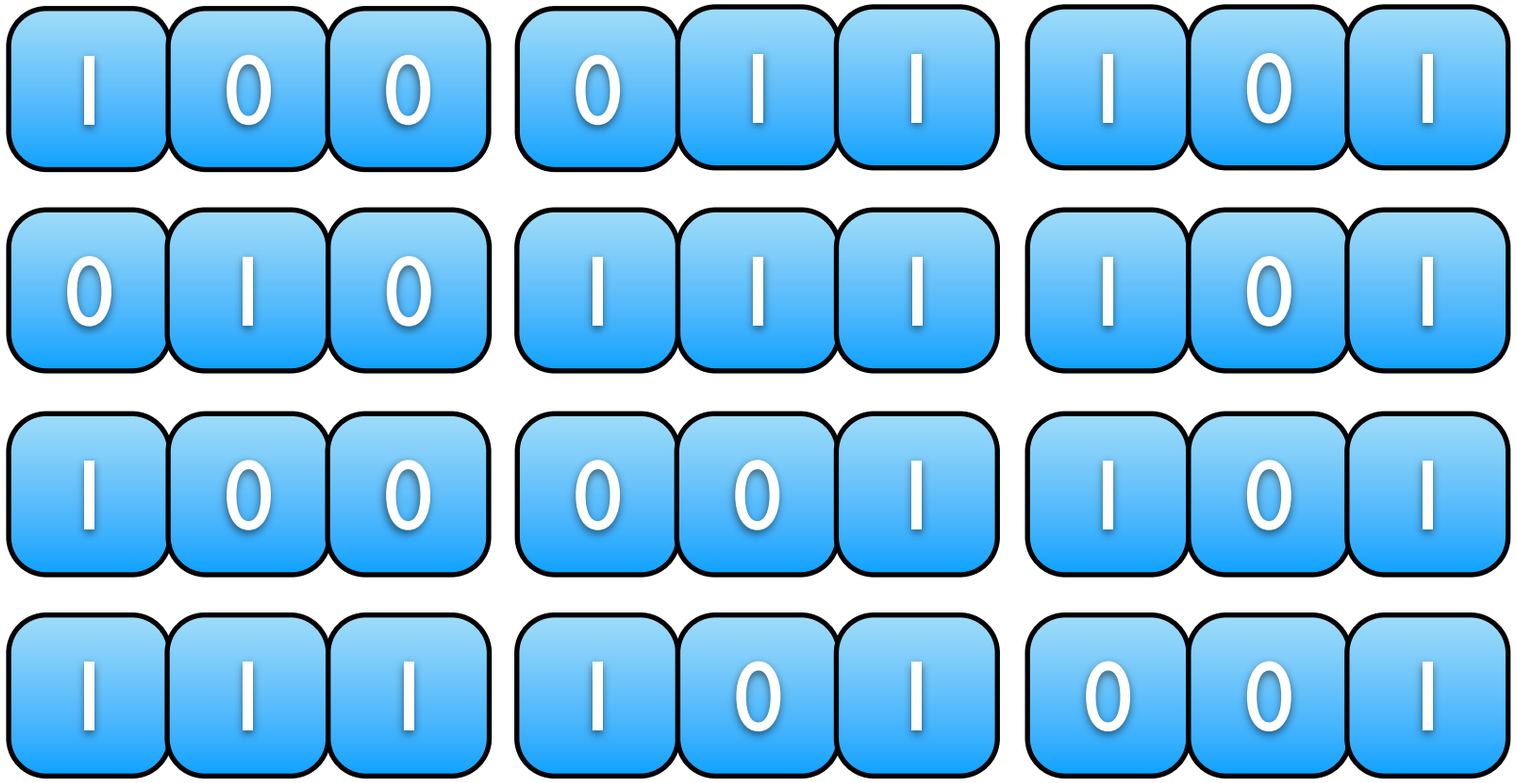

1 0 0 0 1 1 1 0 1
0 1 0 1 1 1 1 0 1
1 0 0 0 0 1 1 0 1
1 1 1 1 0 1 0 0 1



(4,3,5)

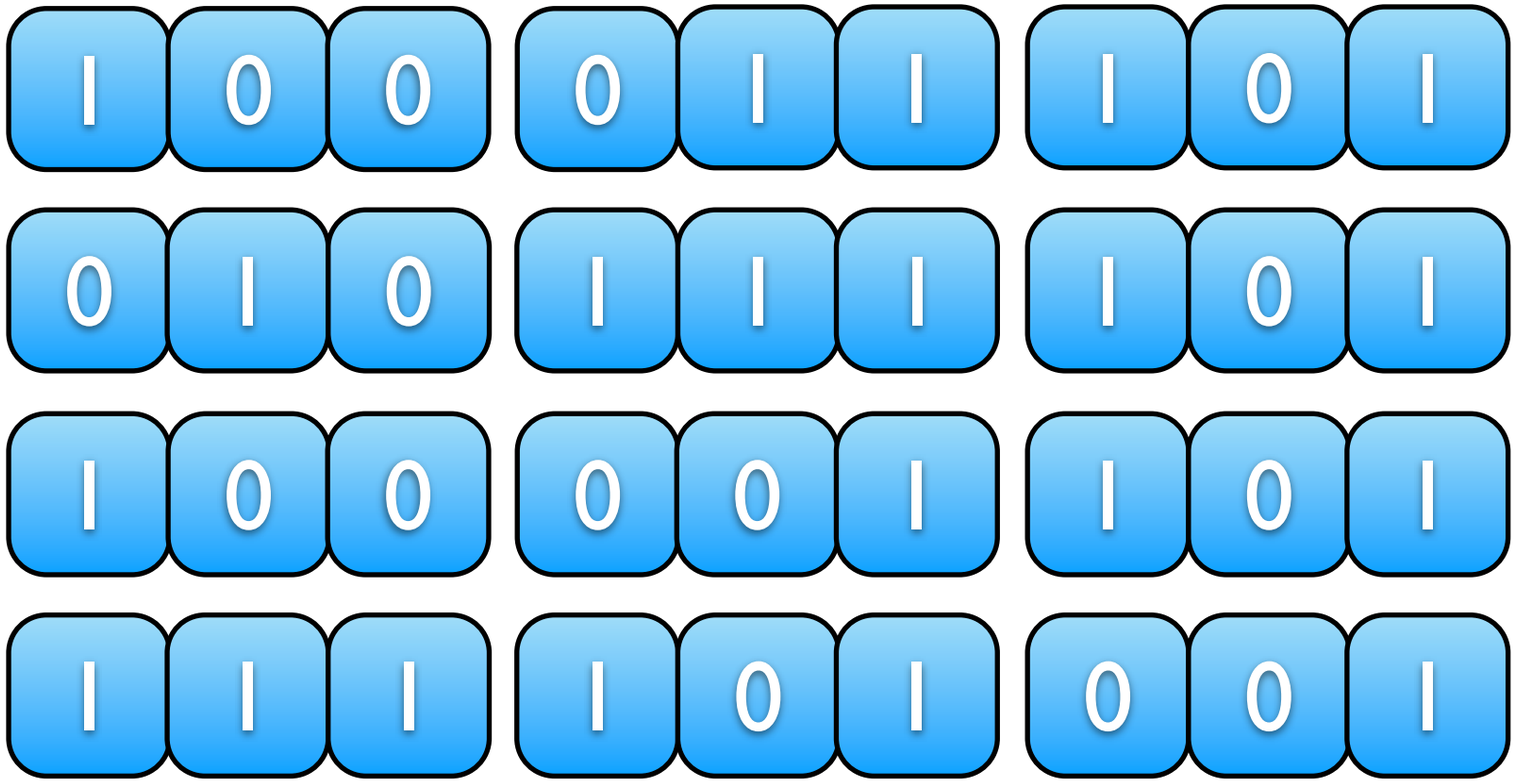


(4,3,5) A: 2, B: 2



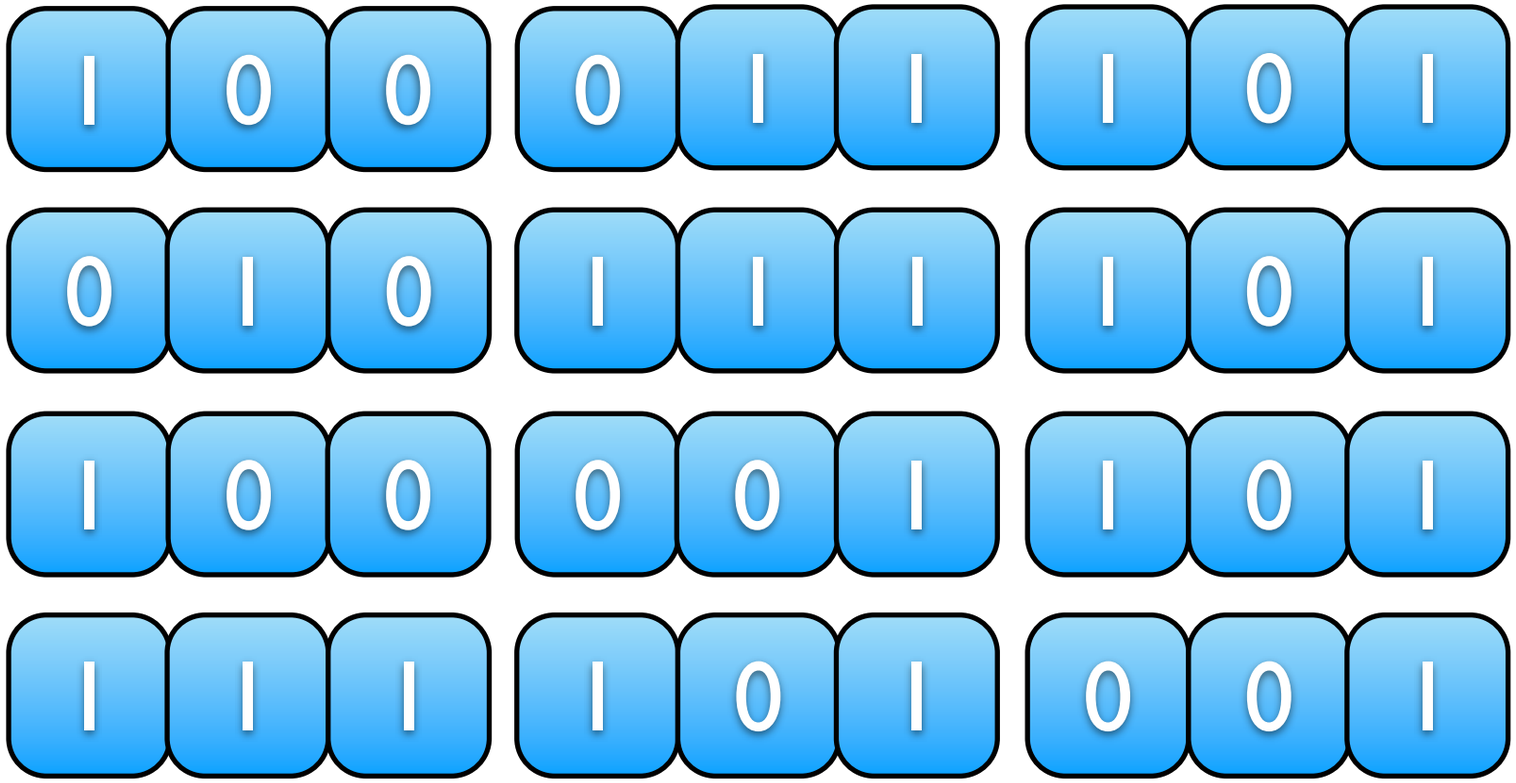
(4,3,5) A: 2, B: 2

(2,7,5)



(4,3,5) A: 2, B: 2

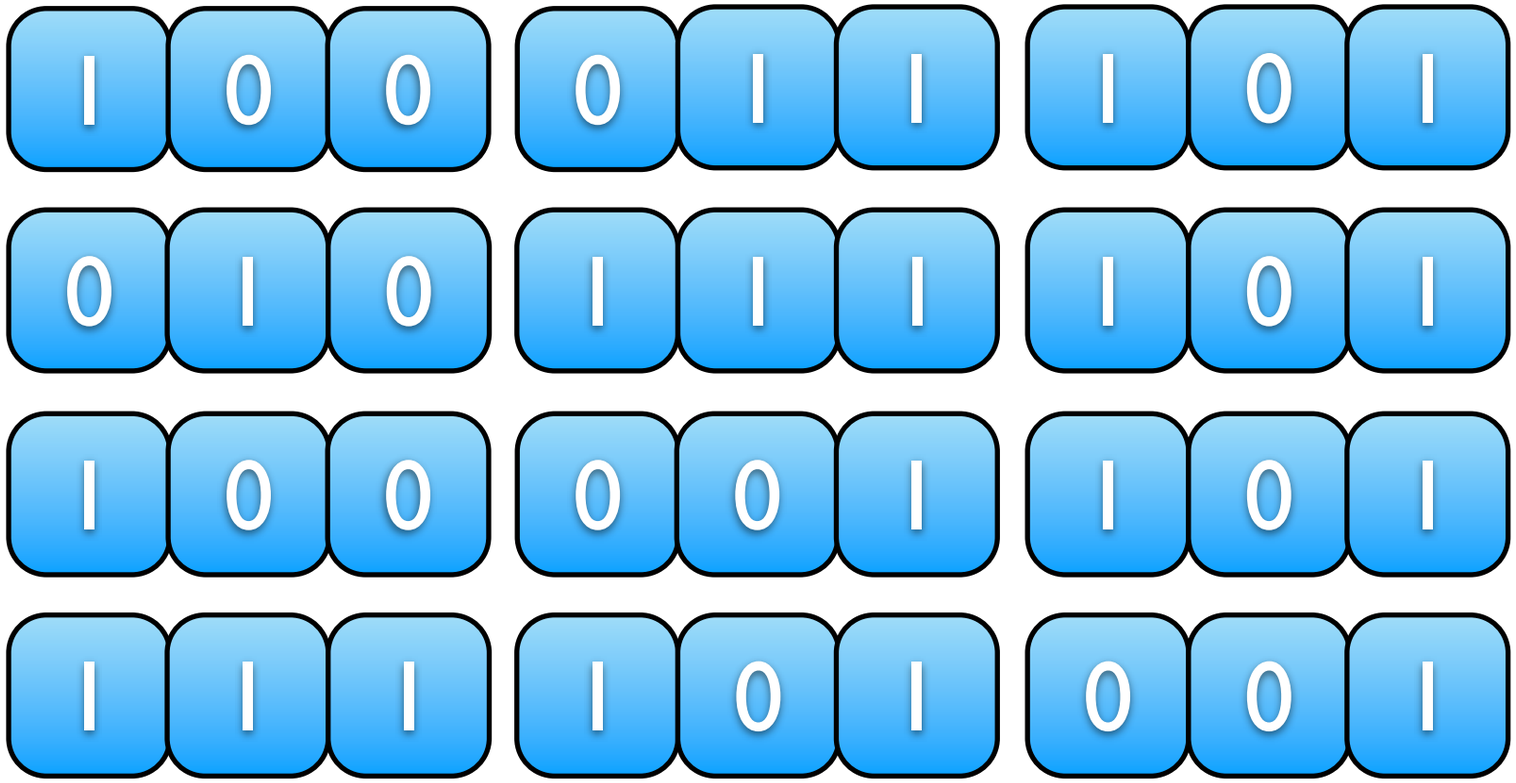
(2,7,5) A: 3, B: 1



(4,3,5) A: 2, B: 2

(2,7,5) A: 3, B: 1

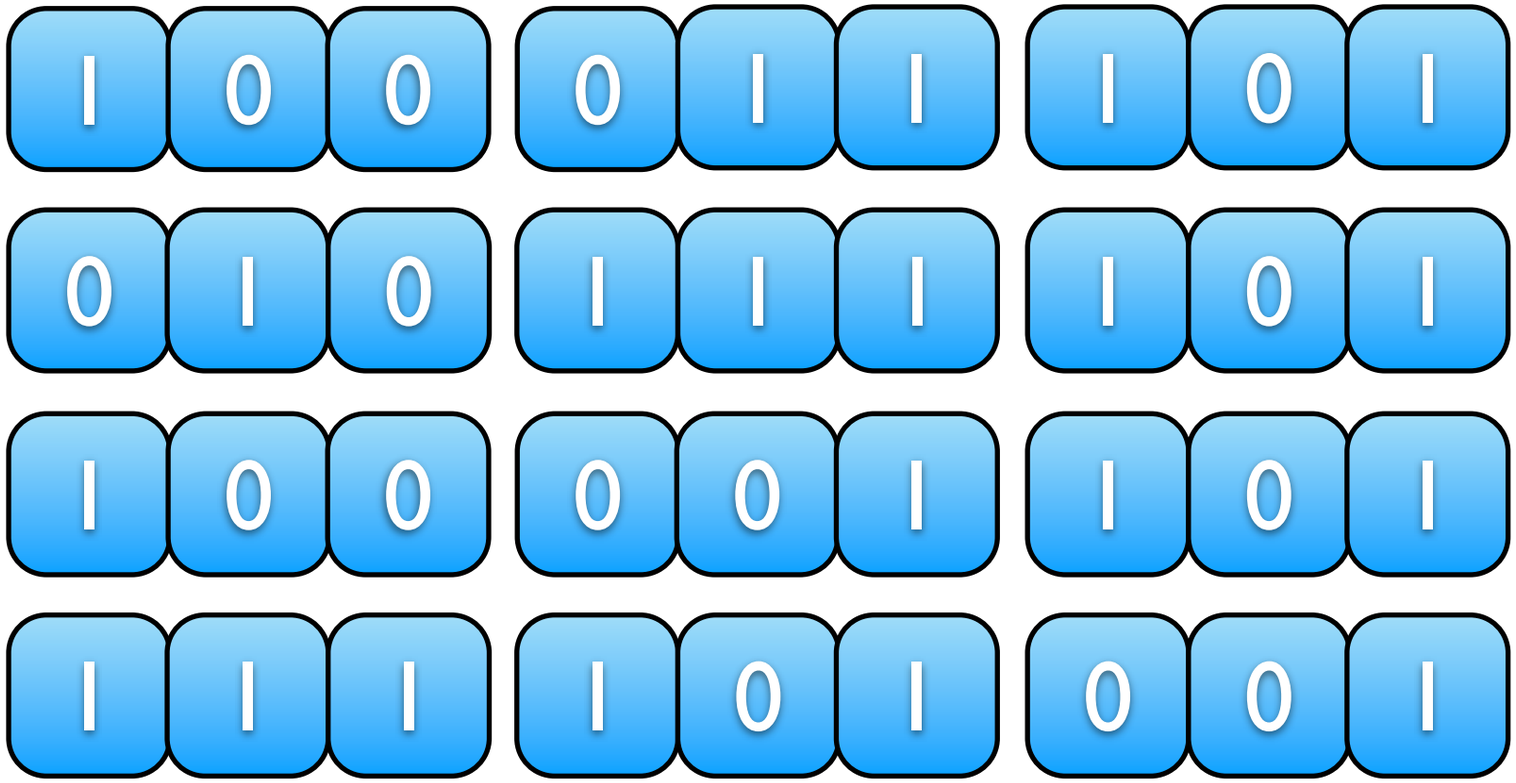
(4,1,5)

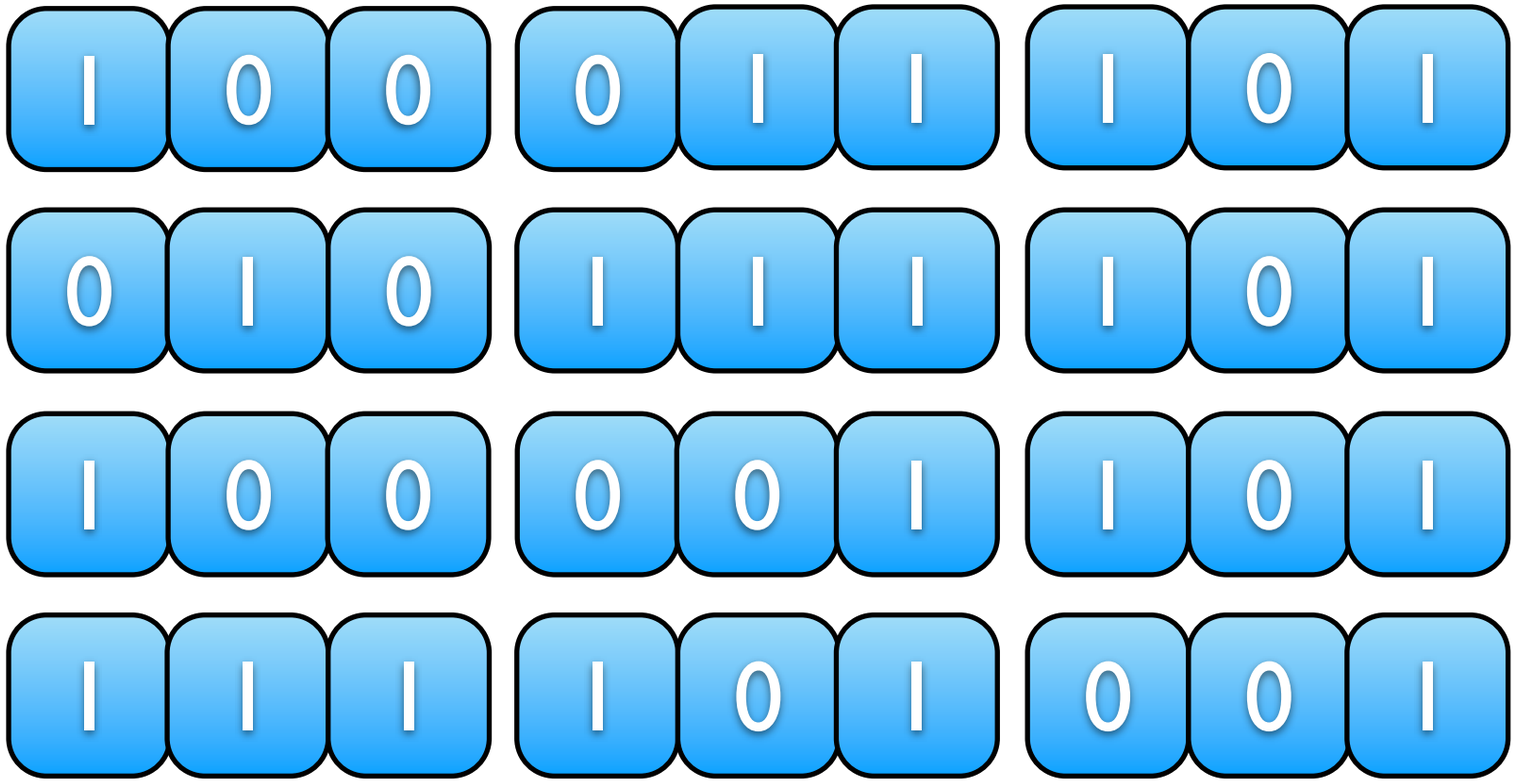


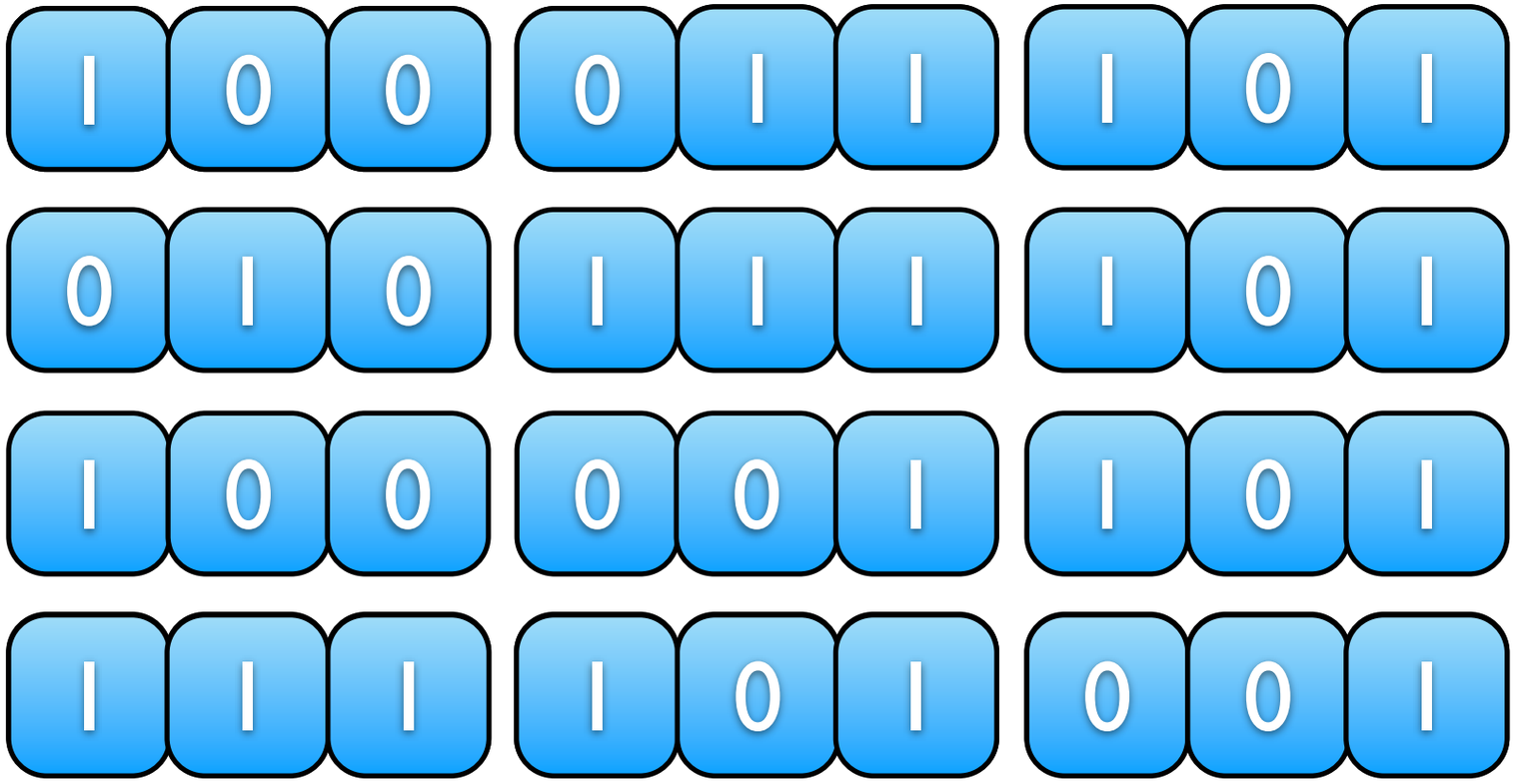
(4,3,5) A: 2, B: 2

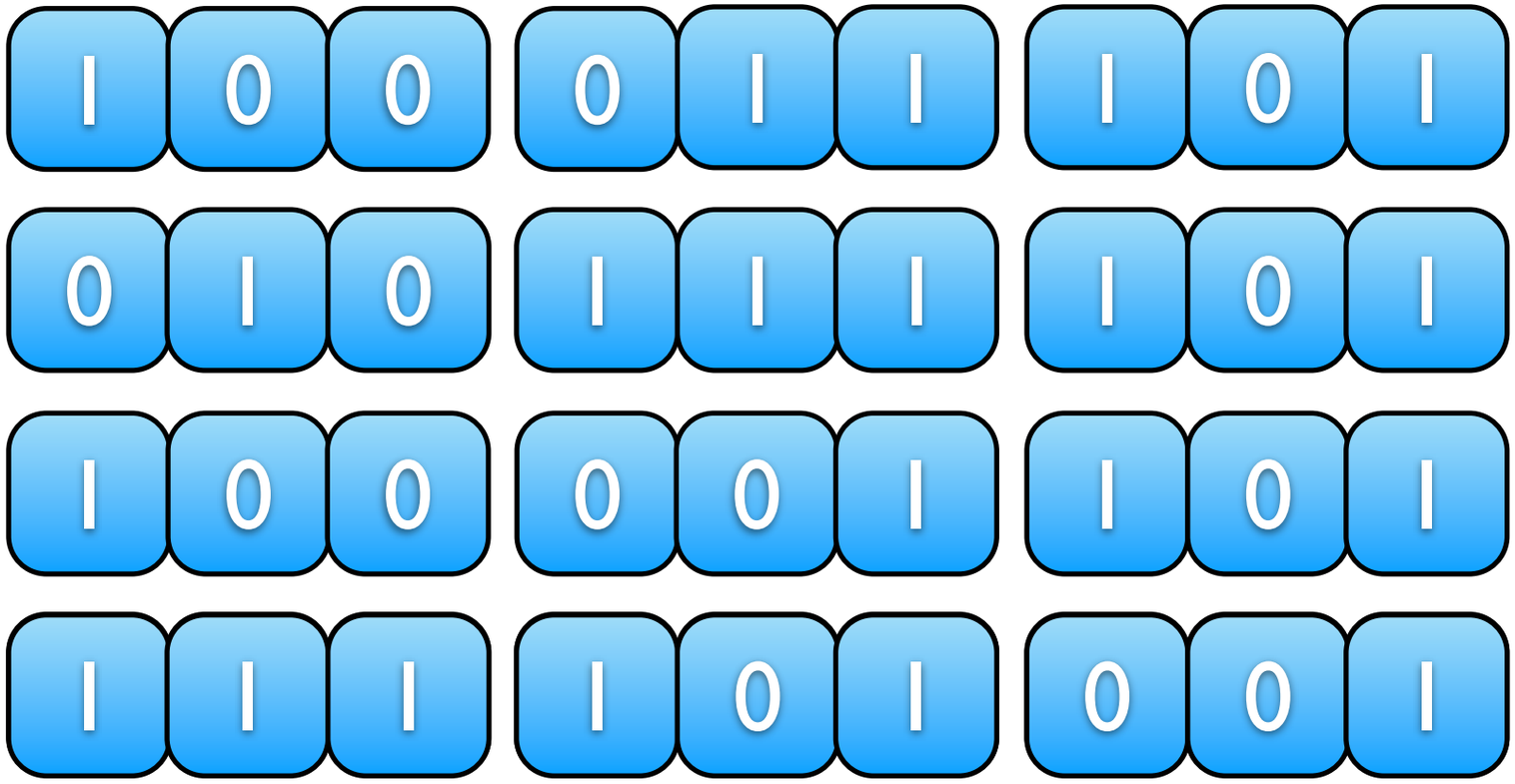
(2,7,5) A: 3, B: 1

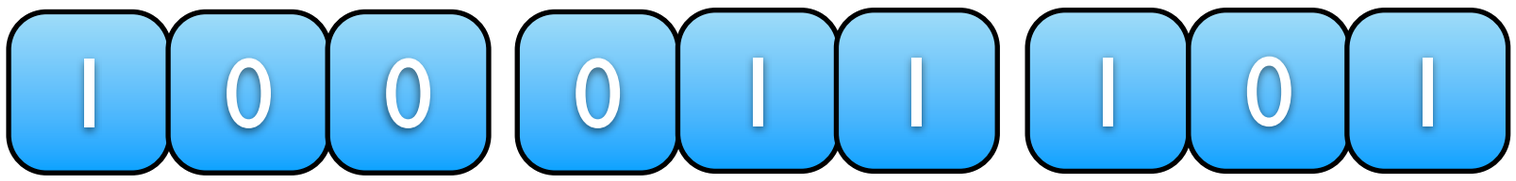
(4,1,5) A: 3, B: 1

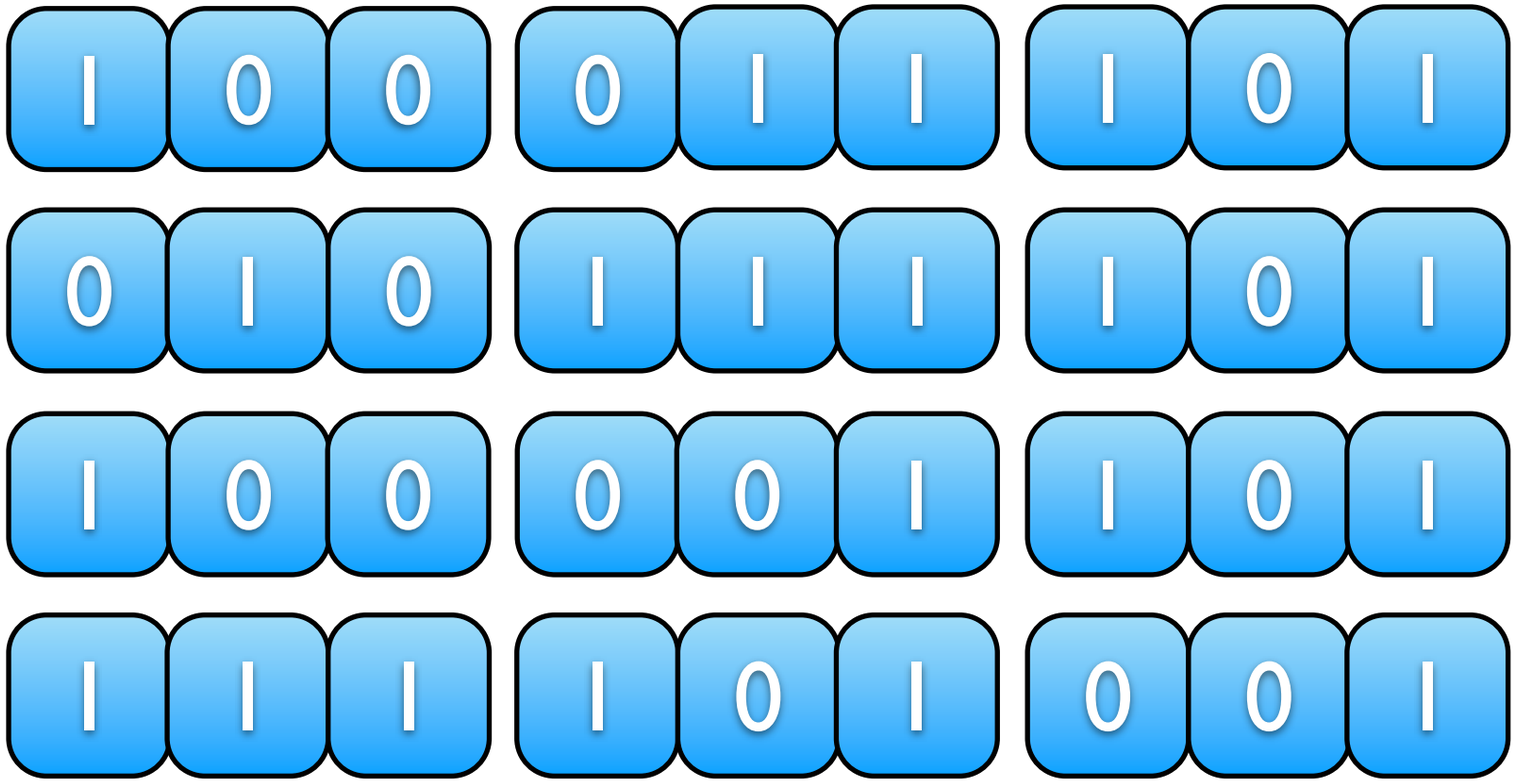

1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1)

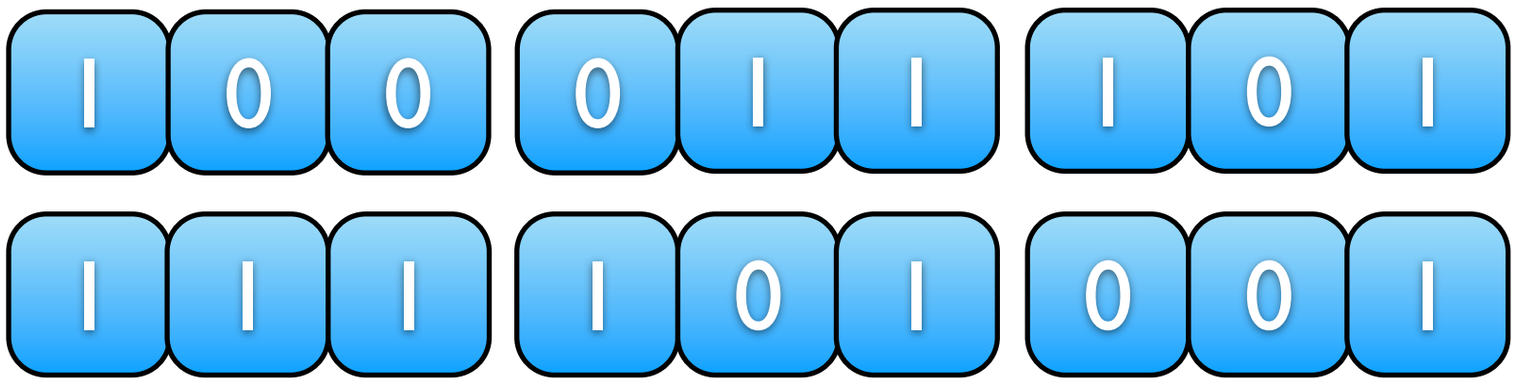

1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2

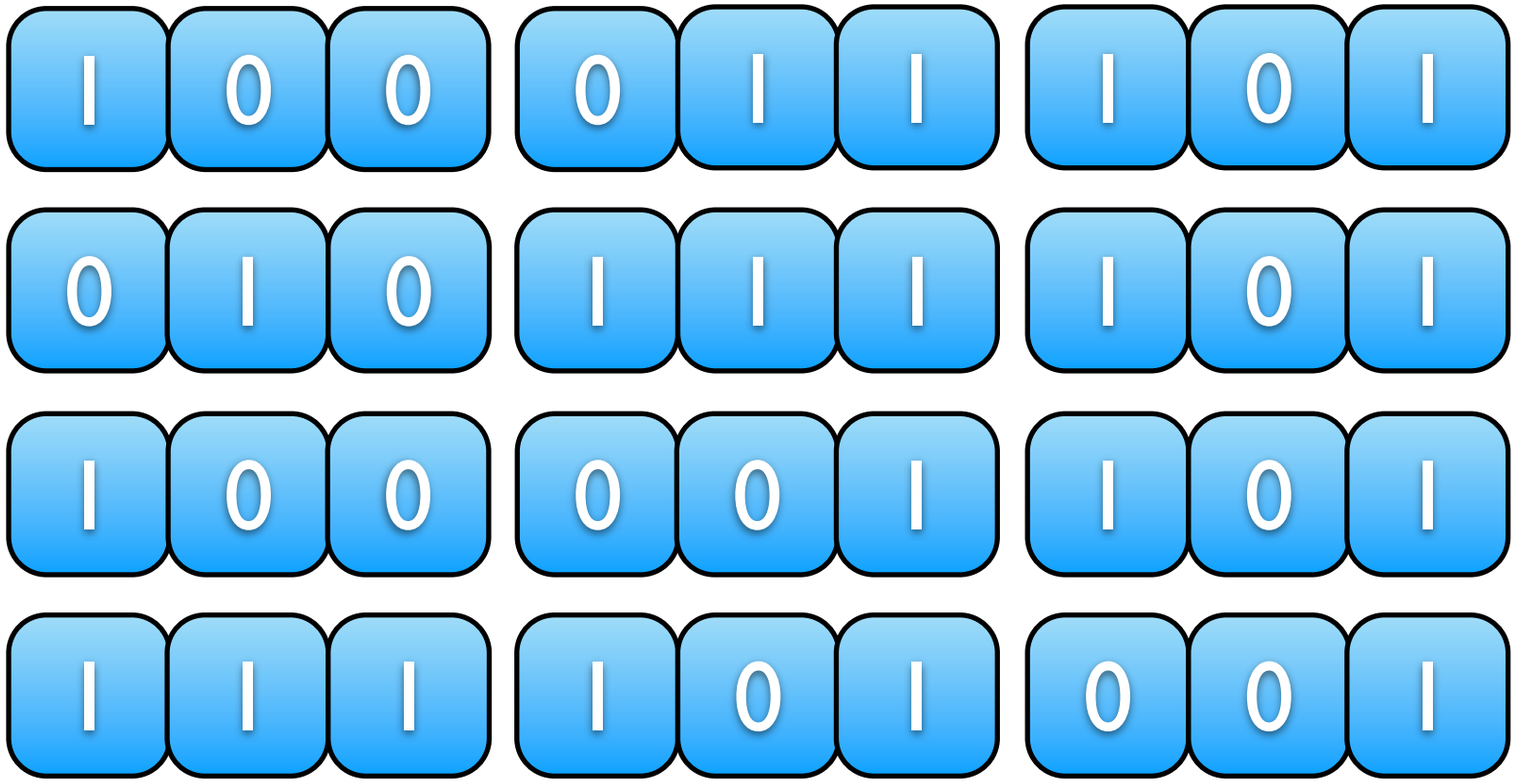

1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2

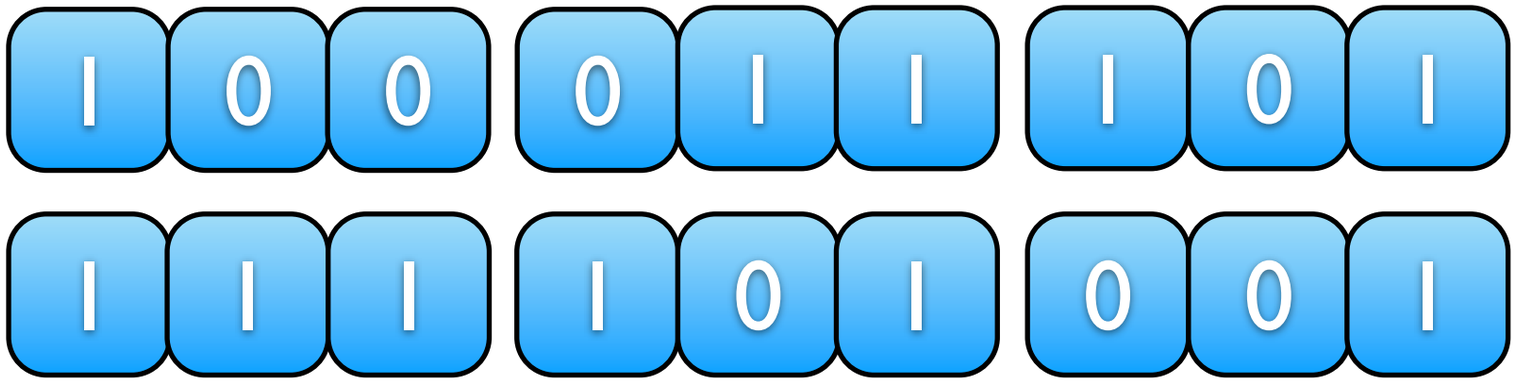

1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2

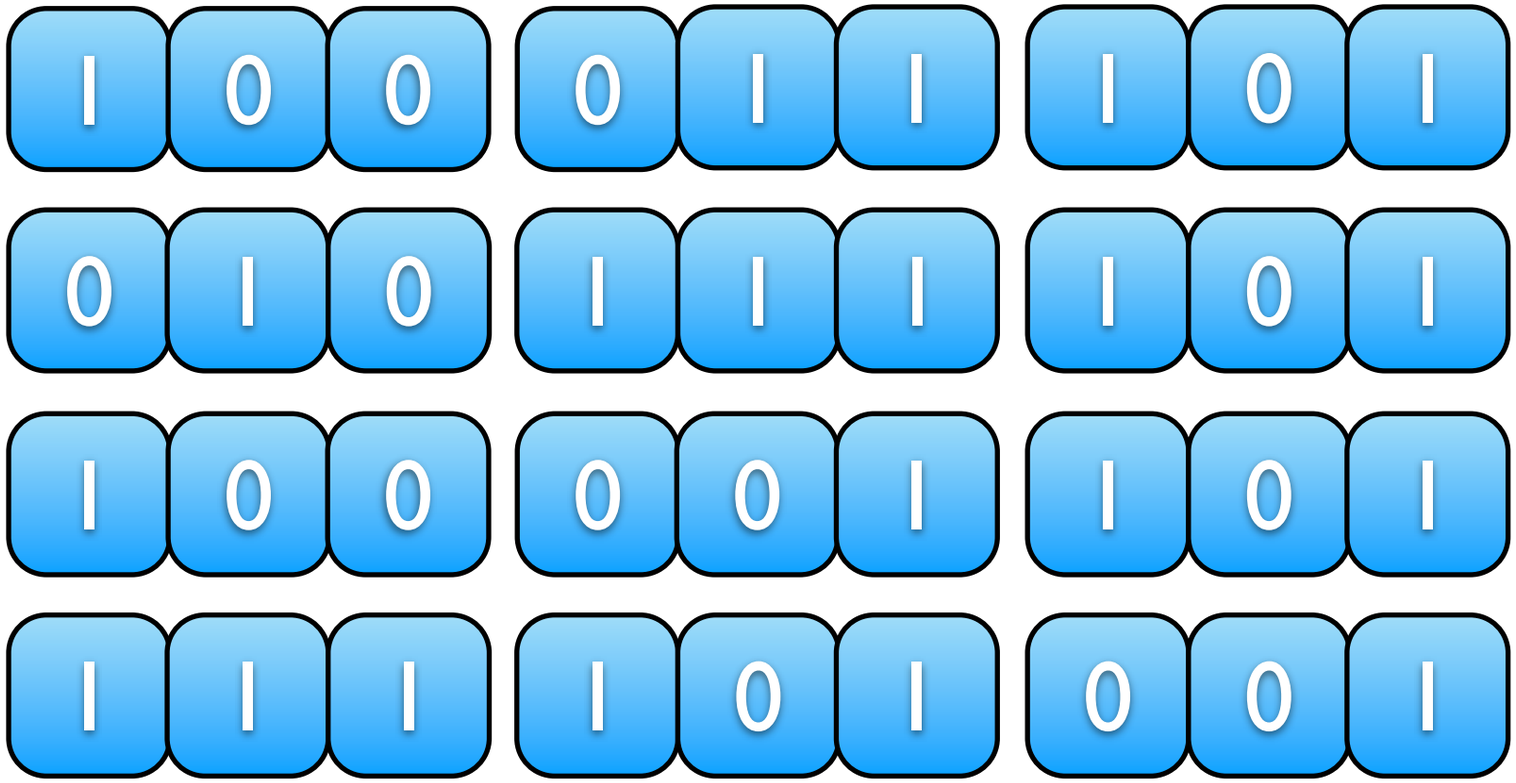

1 0 0 0 1 1 1 0 1


1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2


1 0 0 0 1 1 1 0 1
1 1 1 1 0 1 0 0 1


1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2


1 0 0 0 1 1 1 0 1
1 1 1 1 0 1 0 0 1



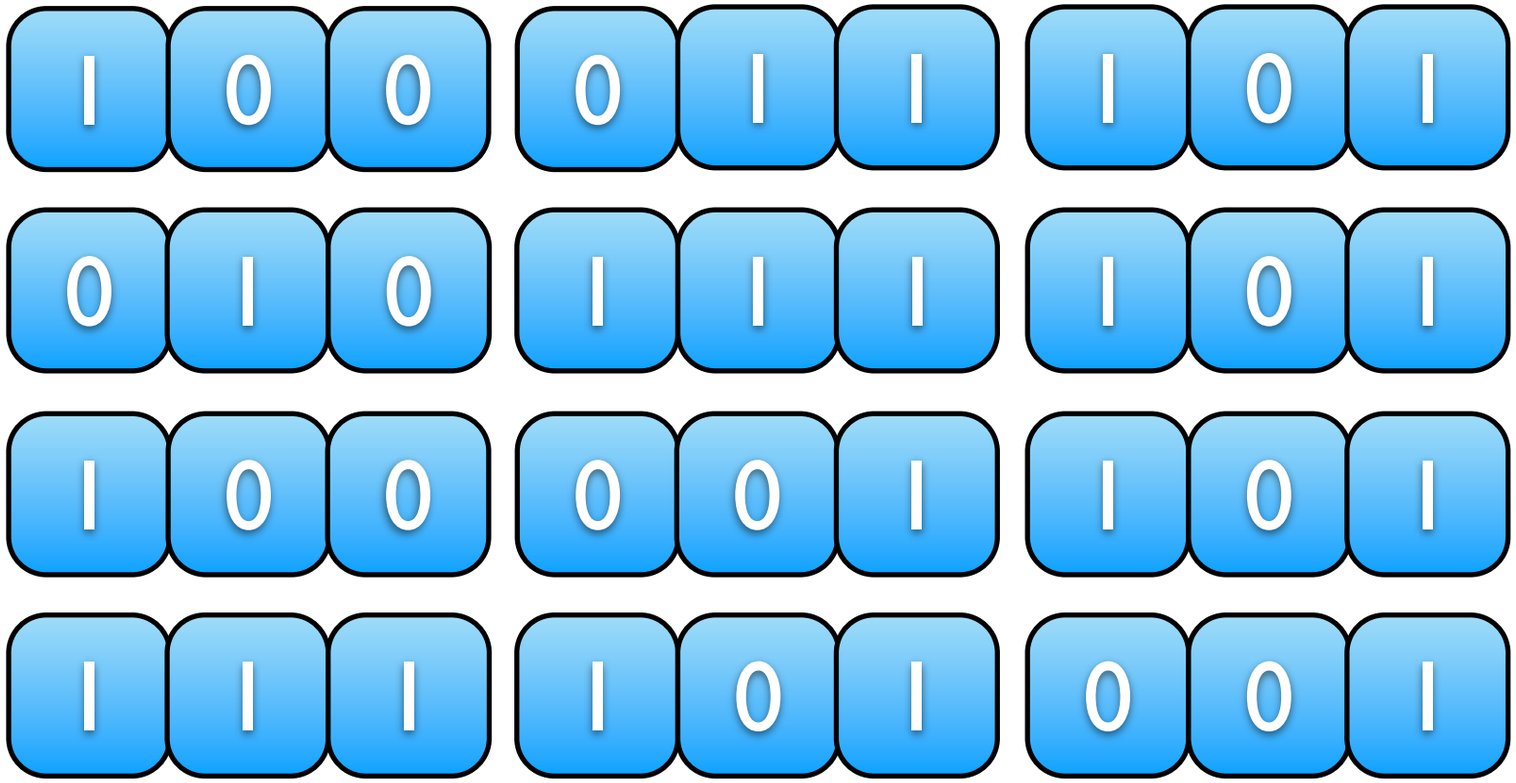
(4,3,5) A: 2, B: 2

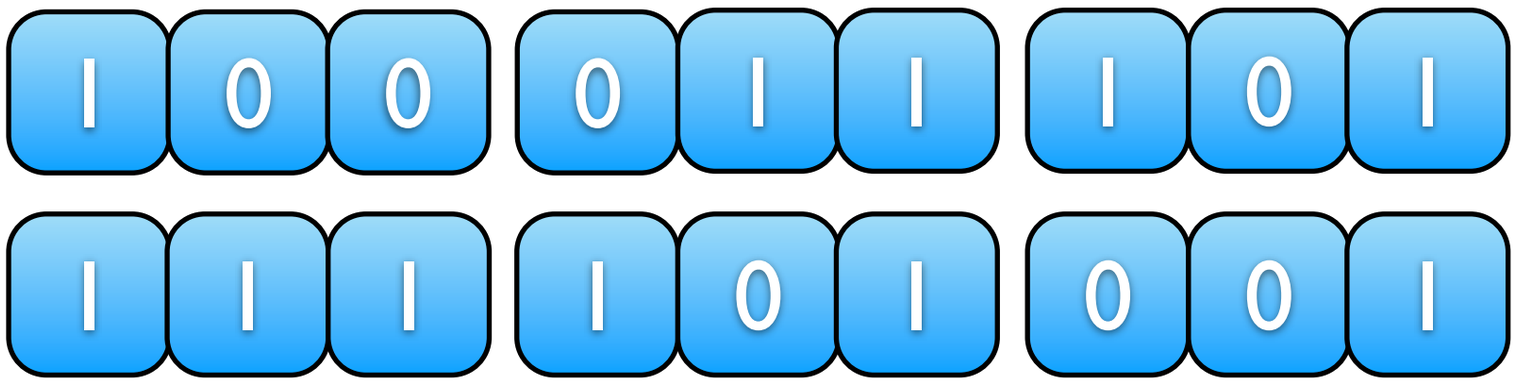
(2,7,5) A: 3, B: 1

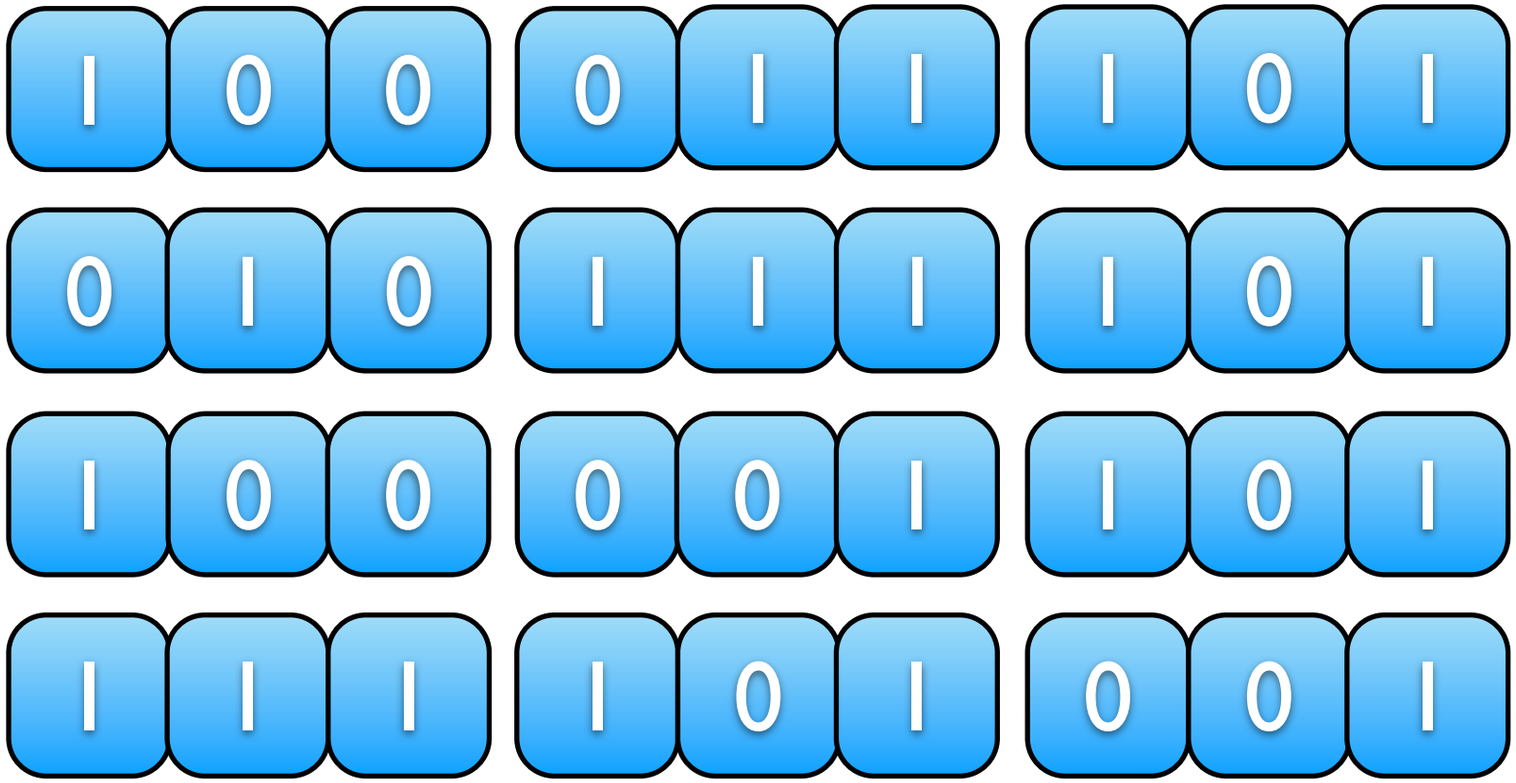
(4,1,5) A: 3, B: 1

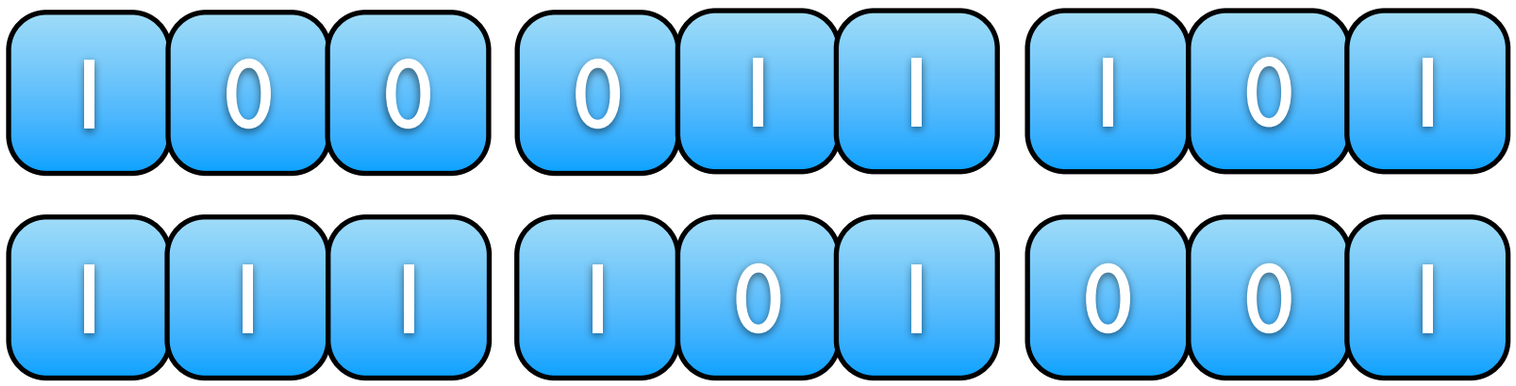
(7,5,1) A: 3, B: 2

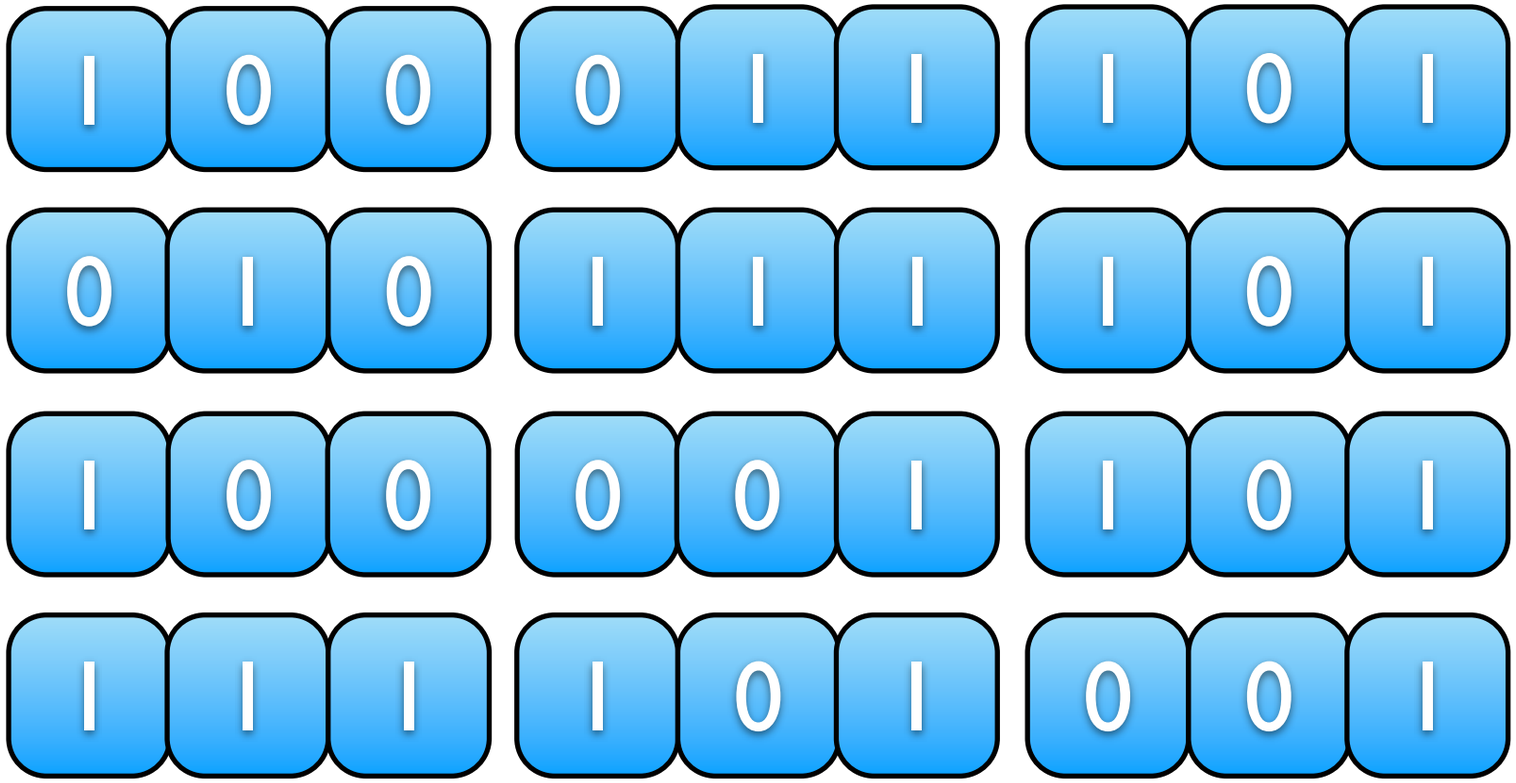
1 0 0 0 1 1 1 0 1
1 1 1 1 0 1 0 0 1

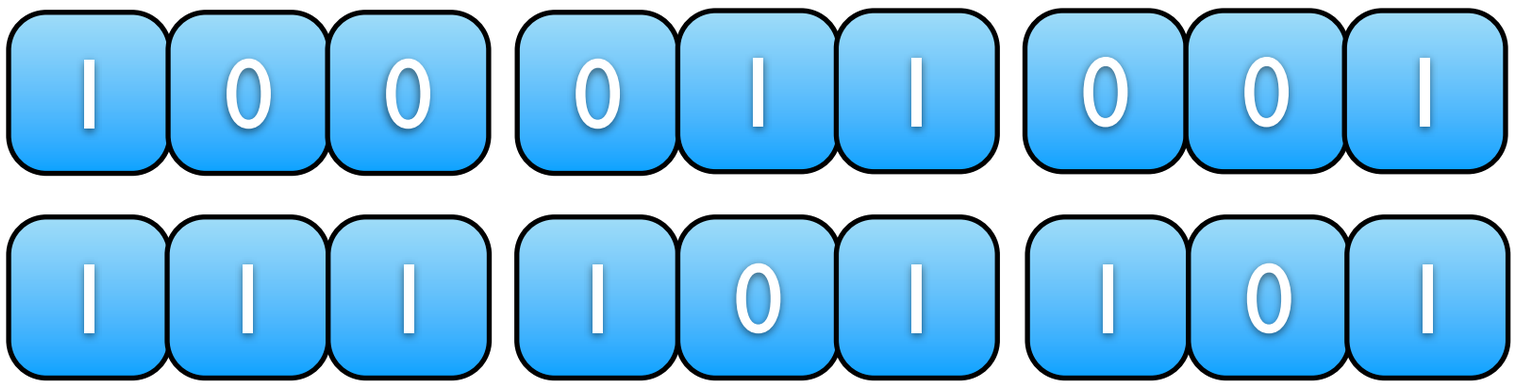

1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2

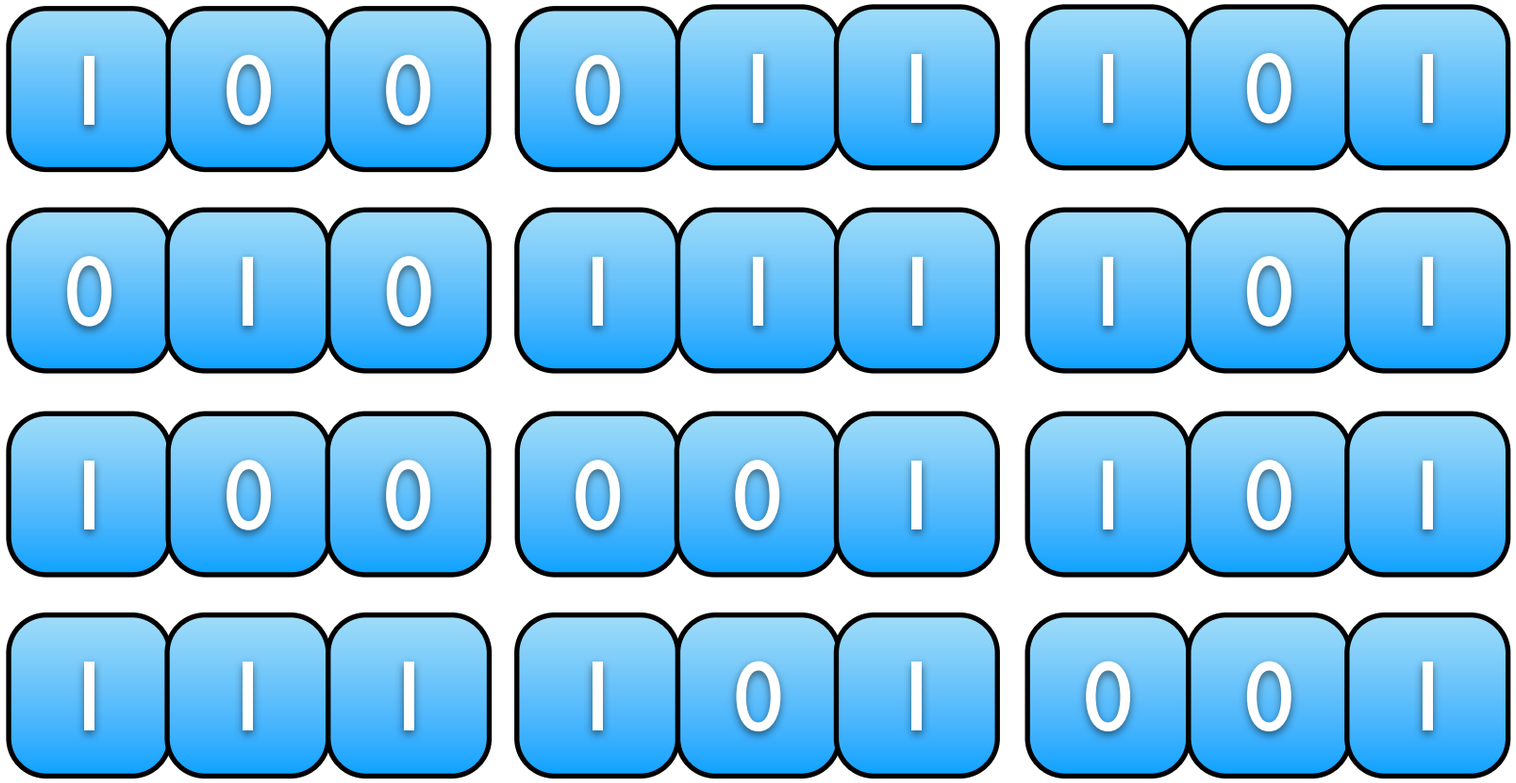

1 0 0 0 1 1 1 0 1
1 1 1 1 0 1 0 0 1


1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2


1 0 0 0 1 1 1 0 1
1 1 1 1 0 1 0 0 1


1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2


1 0 0 0 1 1 0 0 1
1 1 1 1 0 1 1 0 1

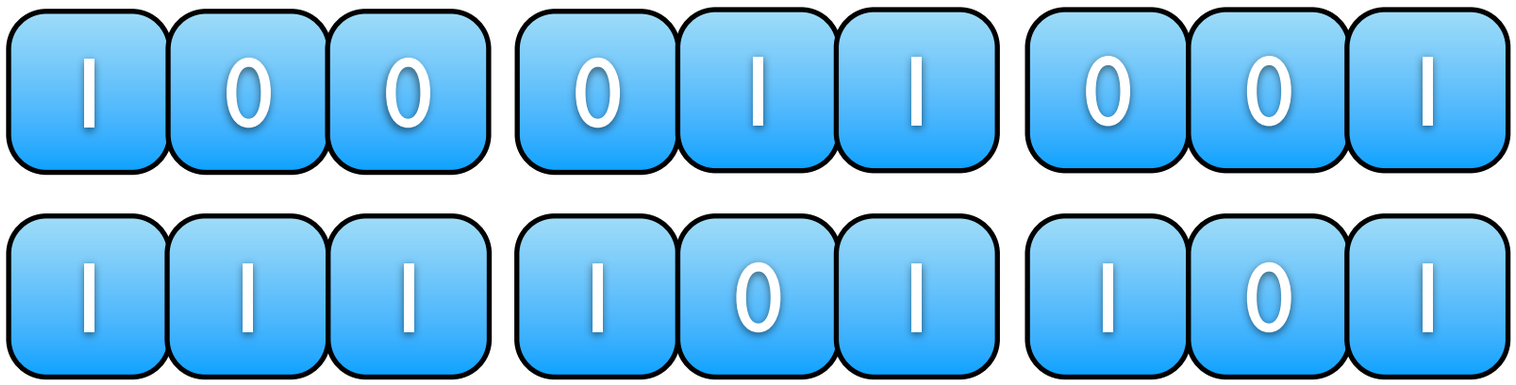


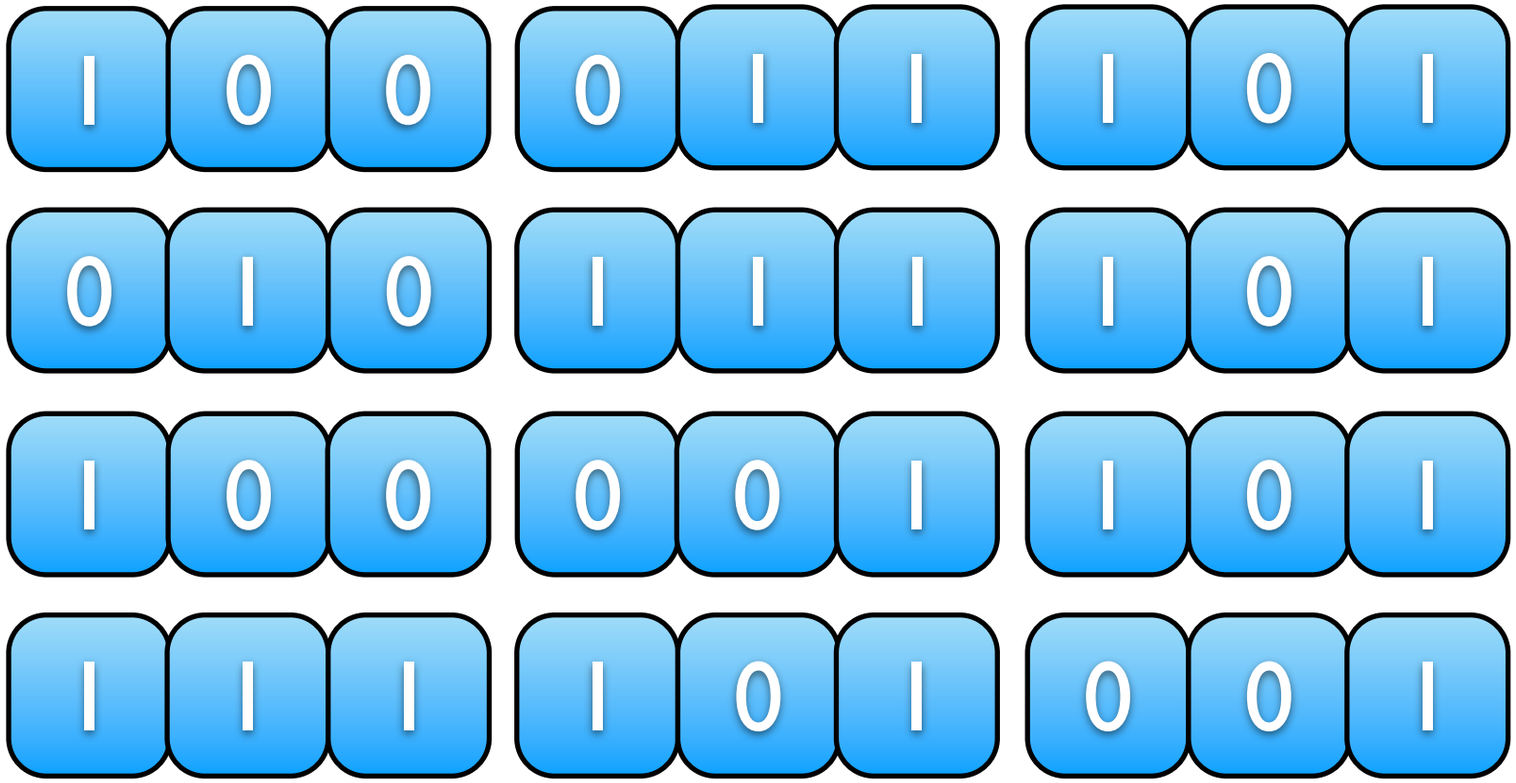
(4,3,5) A: 2, B: 2

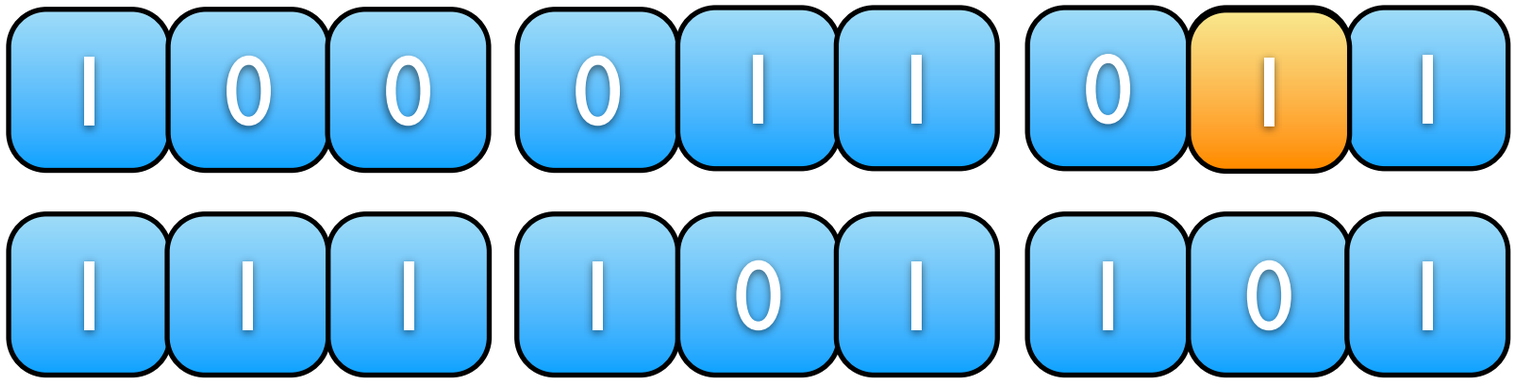
(2,7,5) A: 3, B: 1

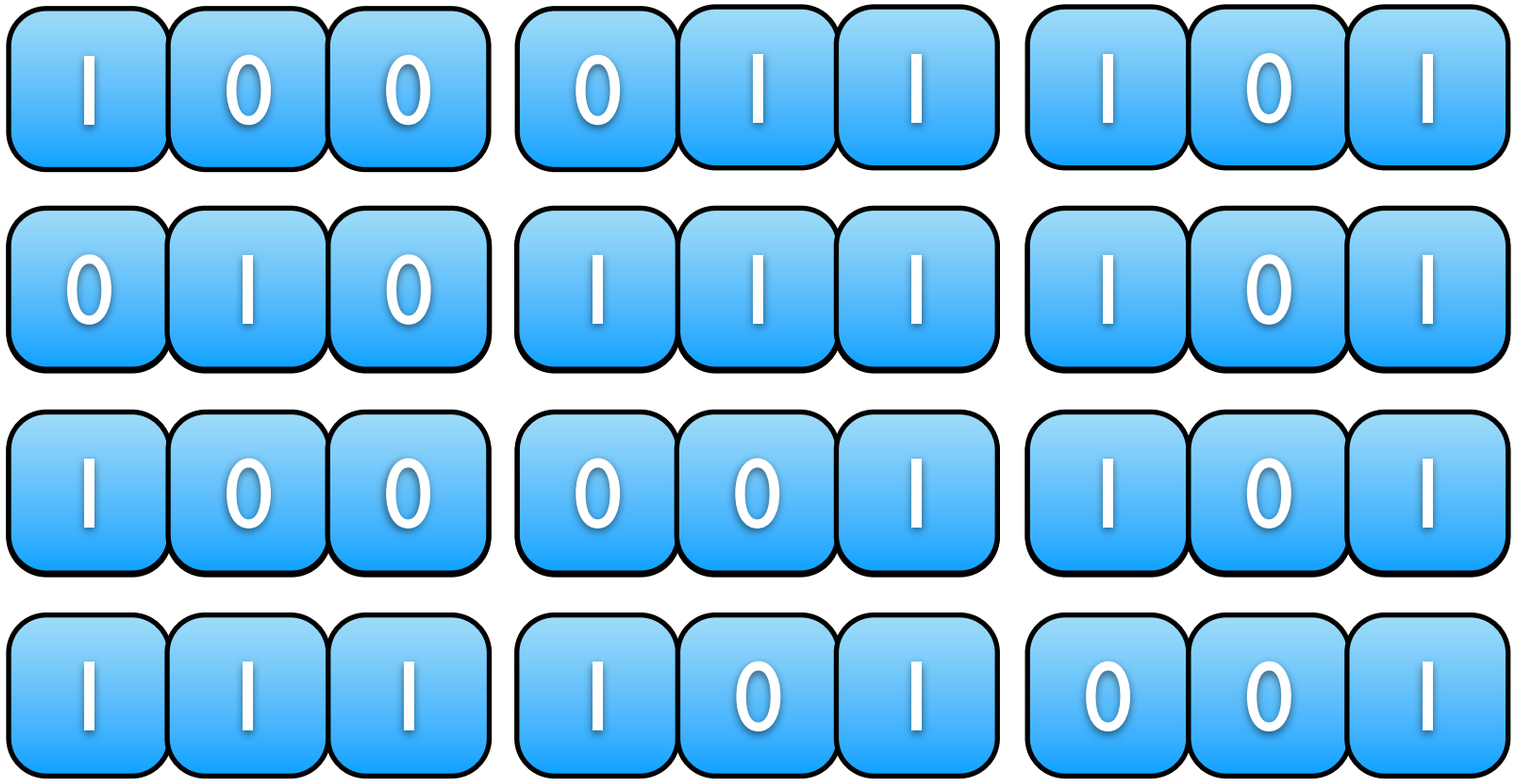
(4,1,5) A: 3, B: 1

(7,5,1) A: 3, B: 2




1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2


1 0 0 0 1 1 0 1 1
1 1 1 1 0 1 1 0 1

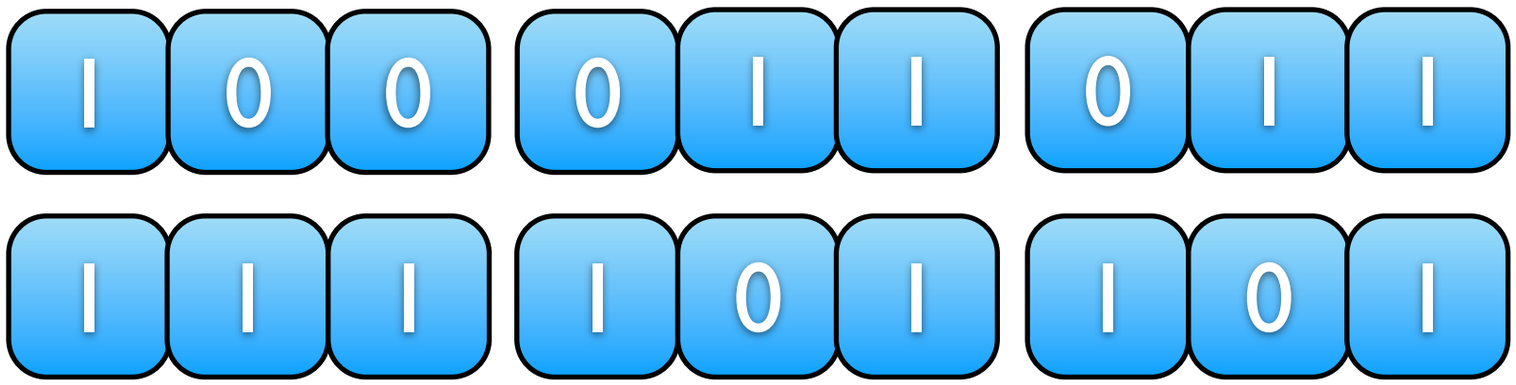


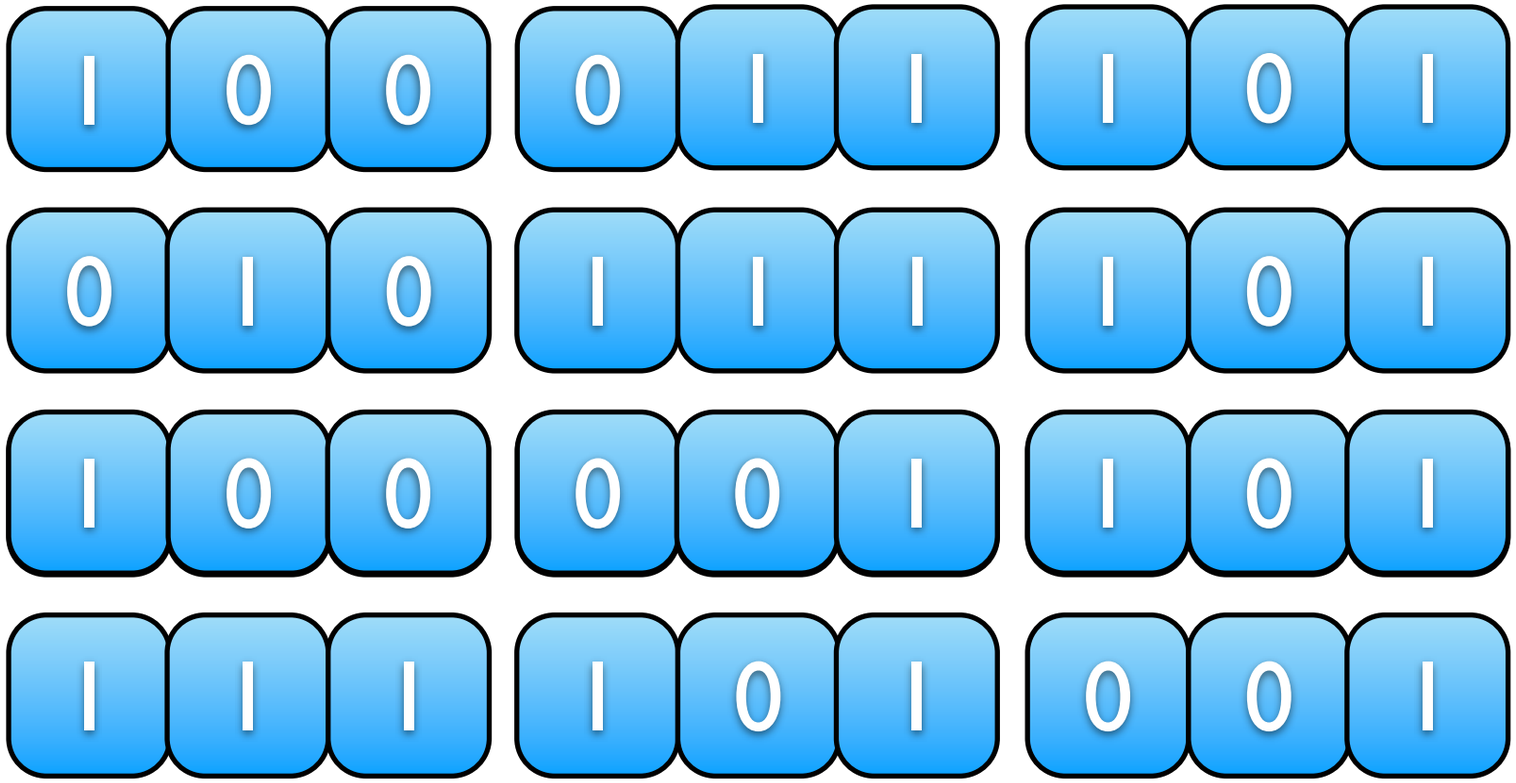
(4,3,5) A: 2, B: 2

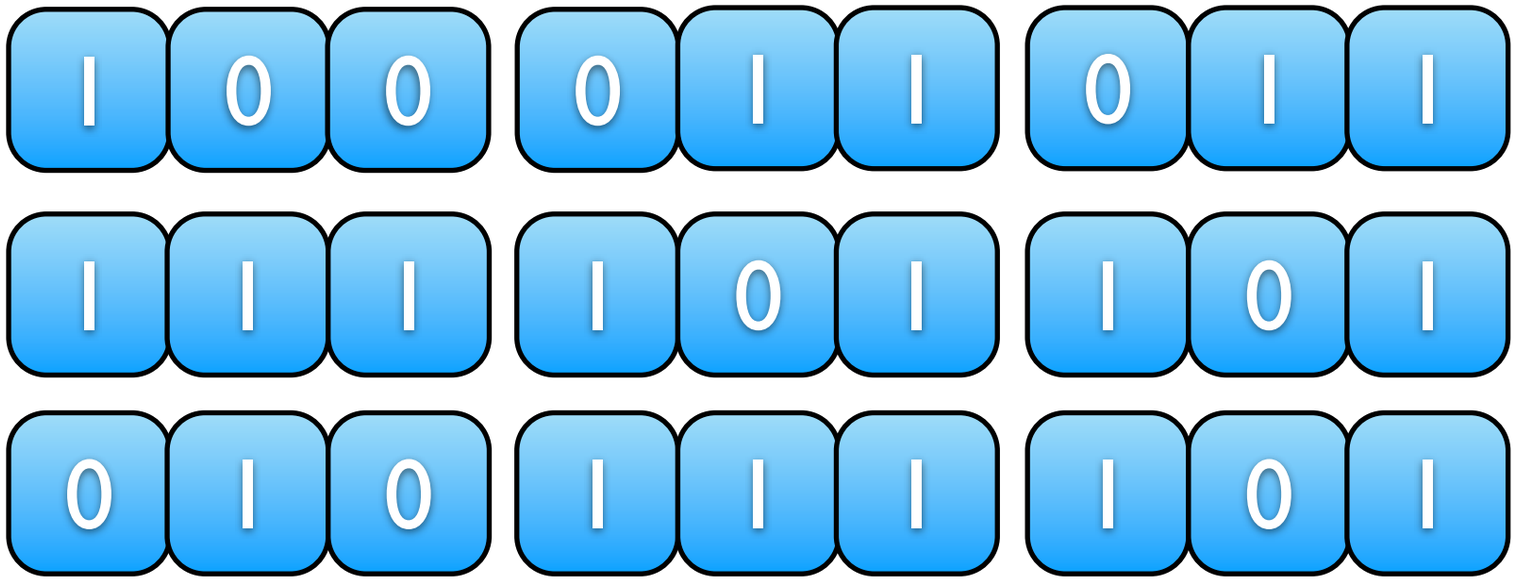
(2,7,5) A: 3, B: 1

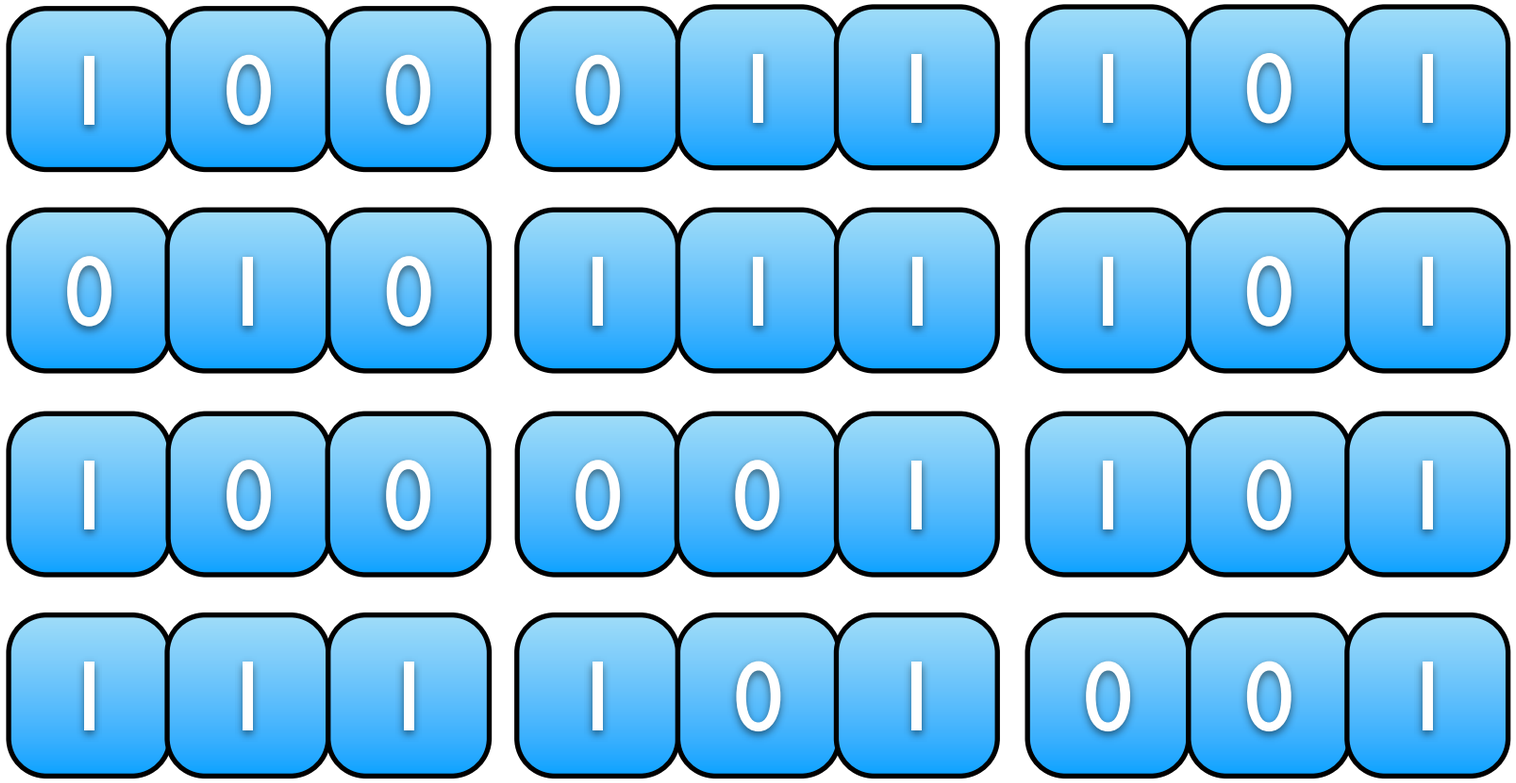
(4,1,5) A: 3, B: 1

(7,5,1) A: 3, B: 2




1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2


1 0 0 0 1 1 0 1 1
1 1 1 1 0 1 1 0 1
0 1 0 1 1 1 1 0 1

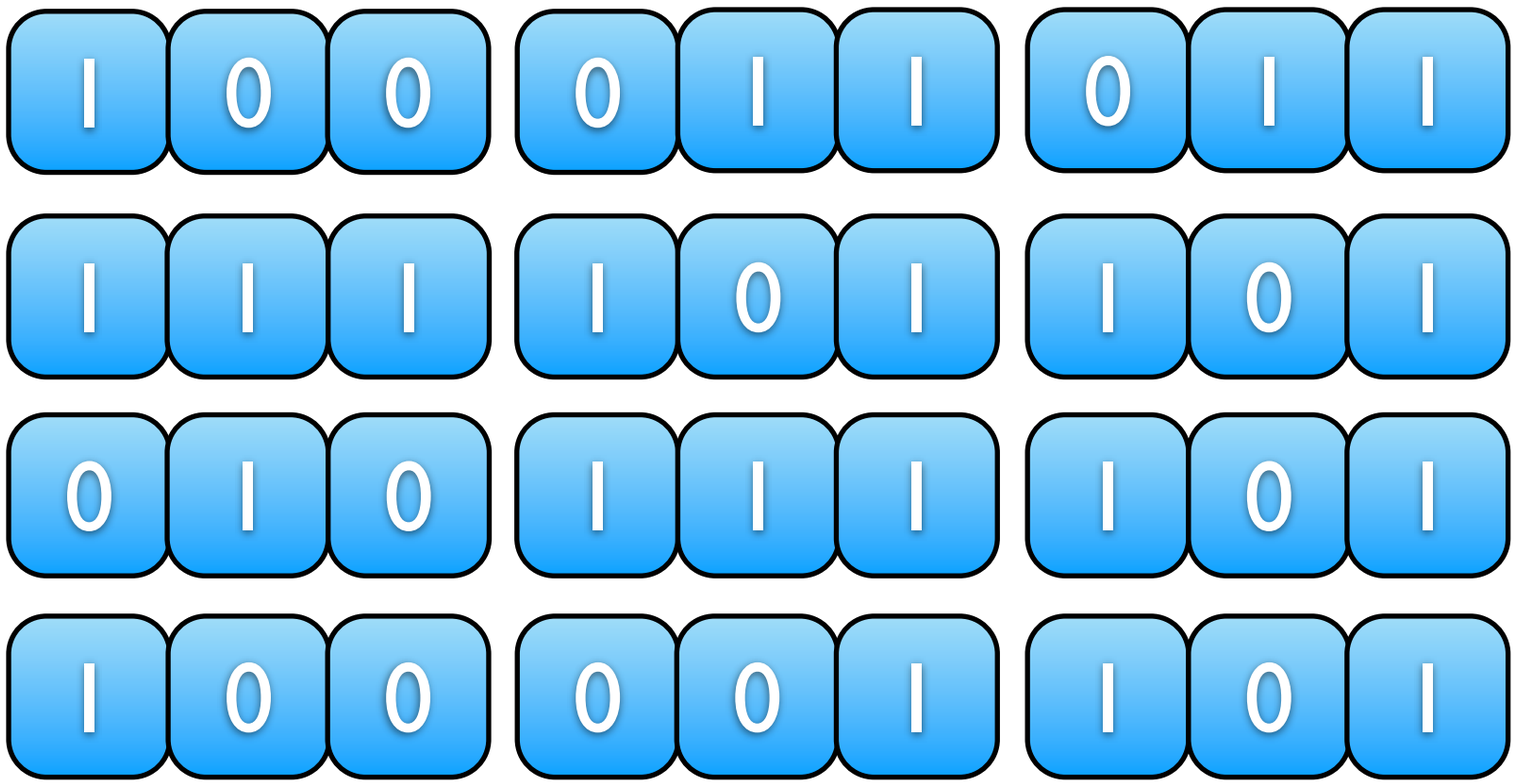


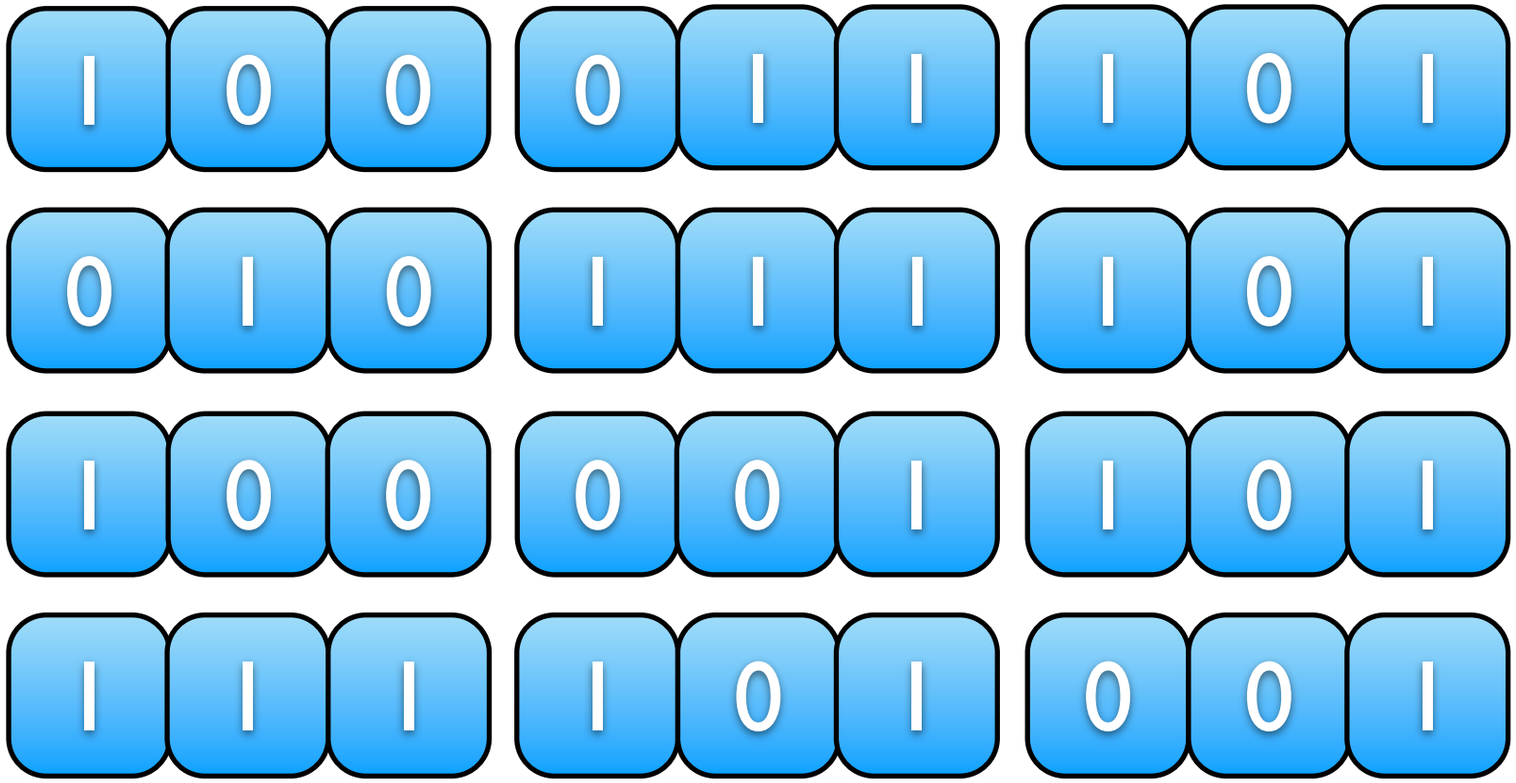
(4,3,5) A: 2, B: 2

(2,7,5) A: 3, B: 1

(4,1,5) A: 3, B: 1

(7,5,1) A: 3, B: 2



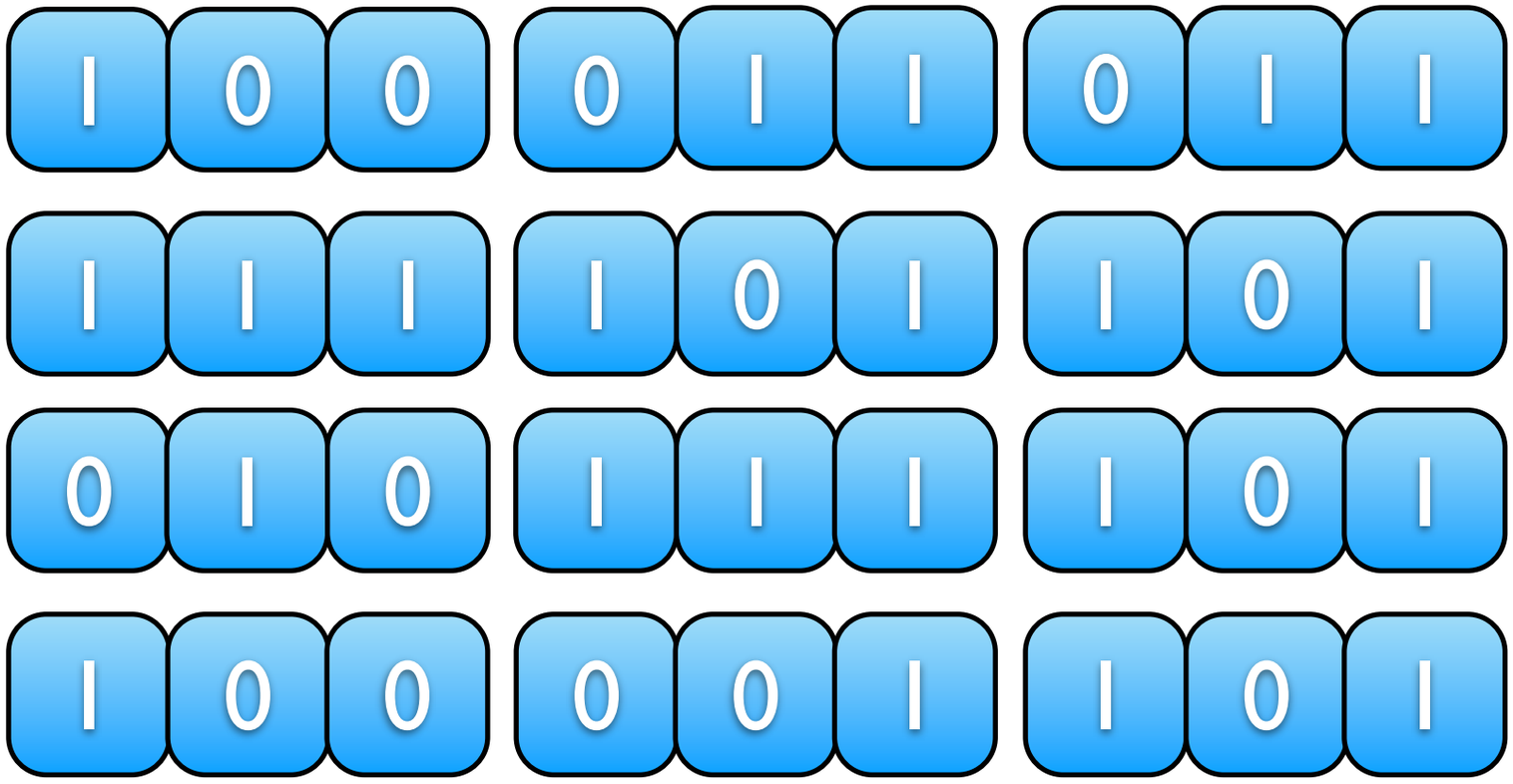


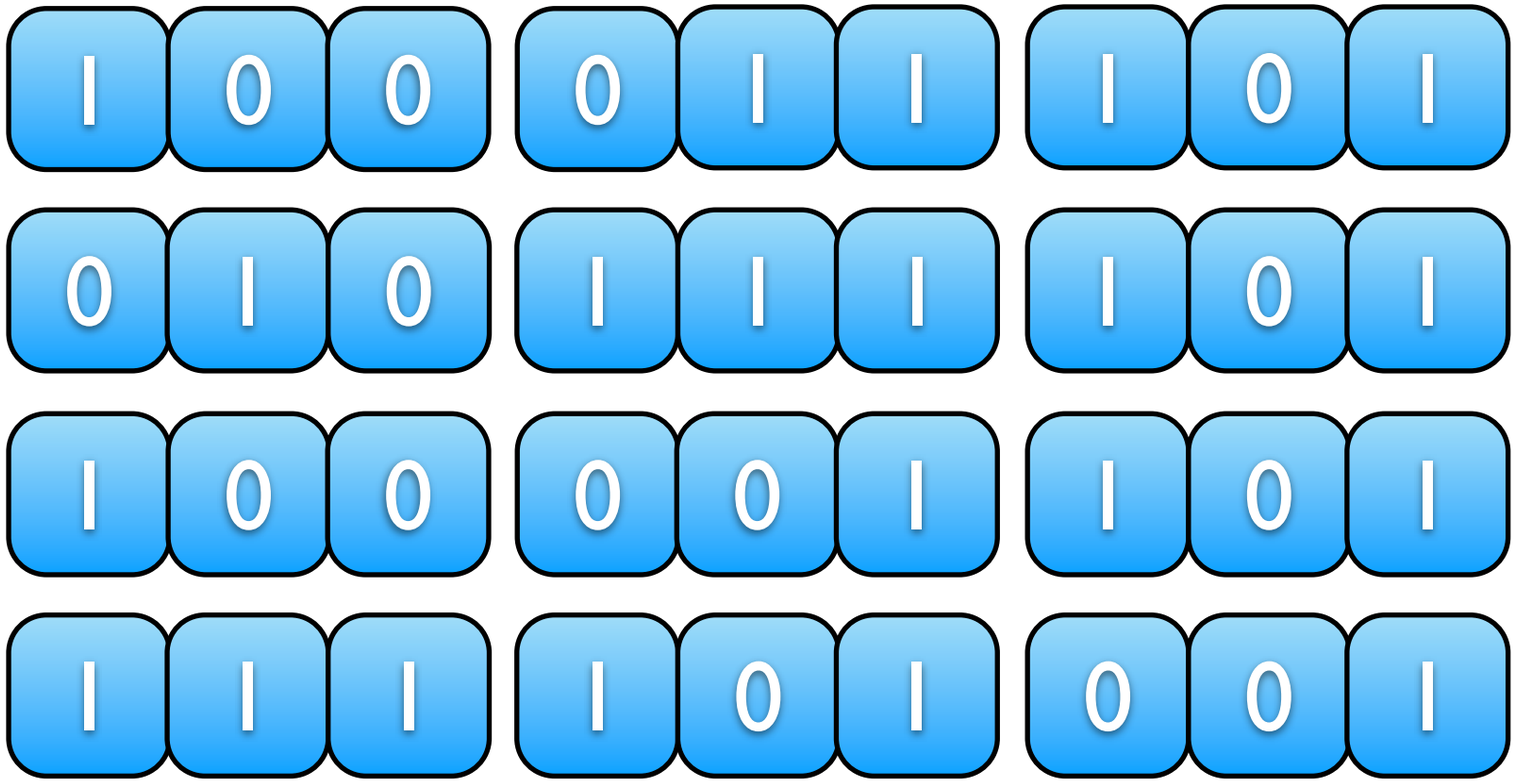
(4,3,5) A: 2, B: 2

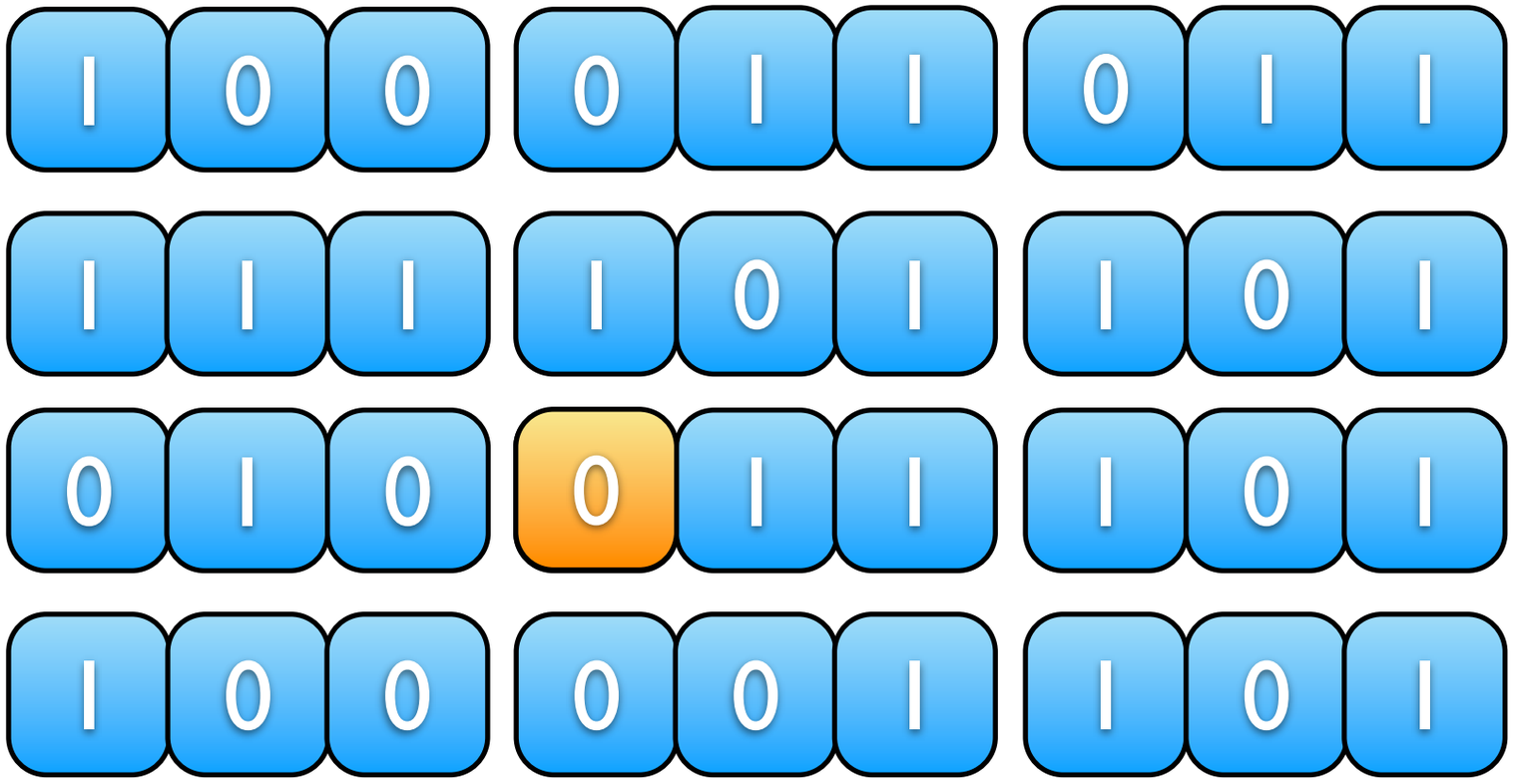
(2,7,5) A: 3, B: 1

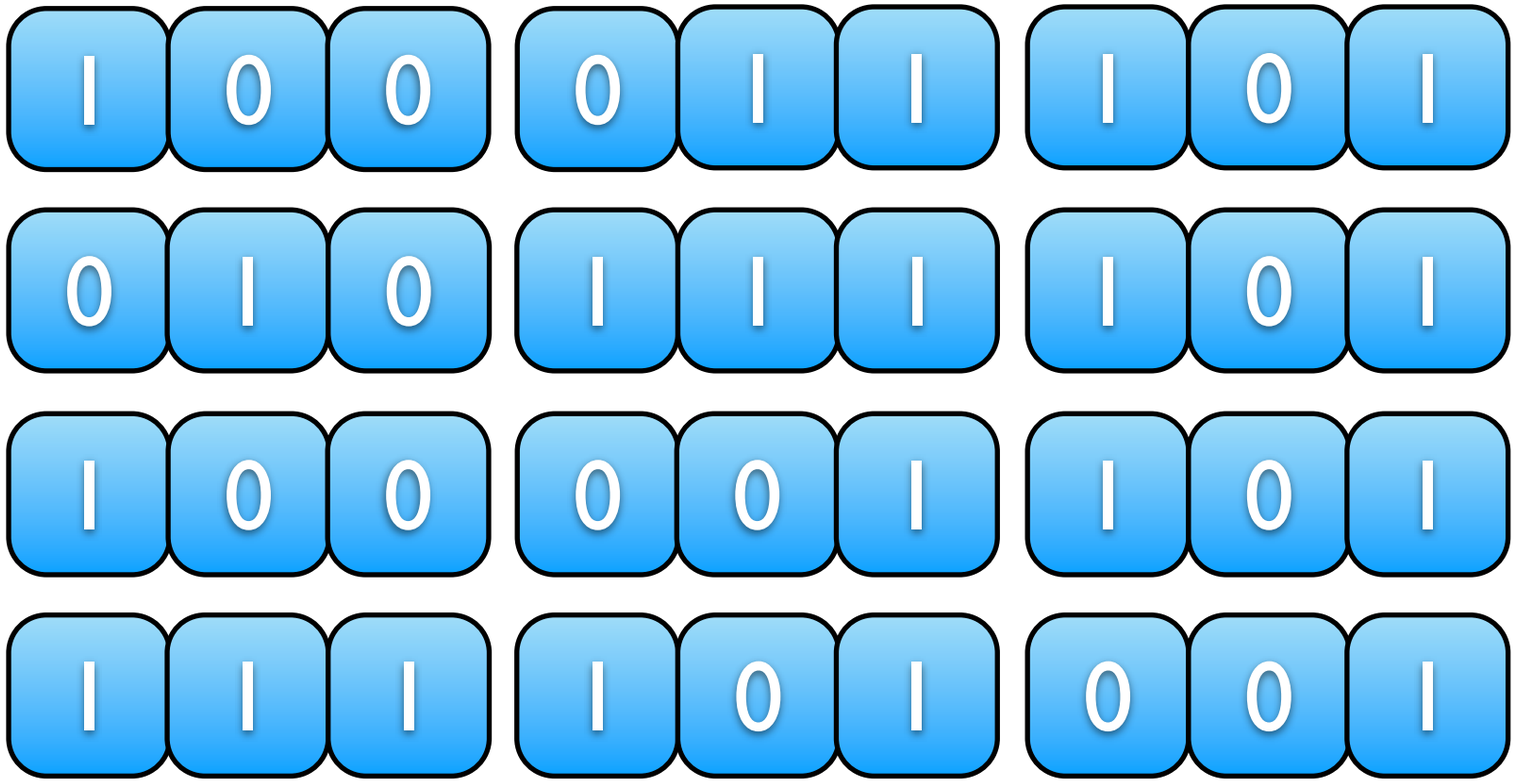
(4,1,5) A: 3, B: 1

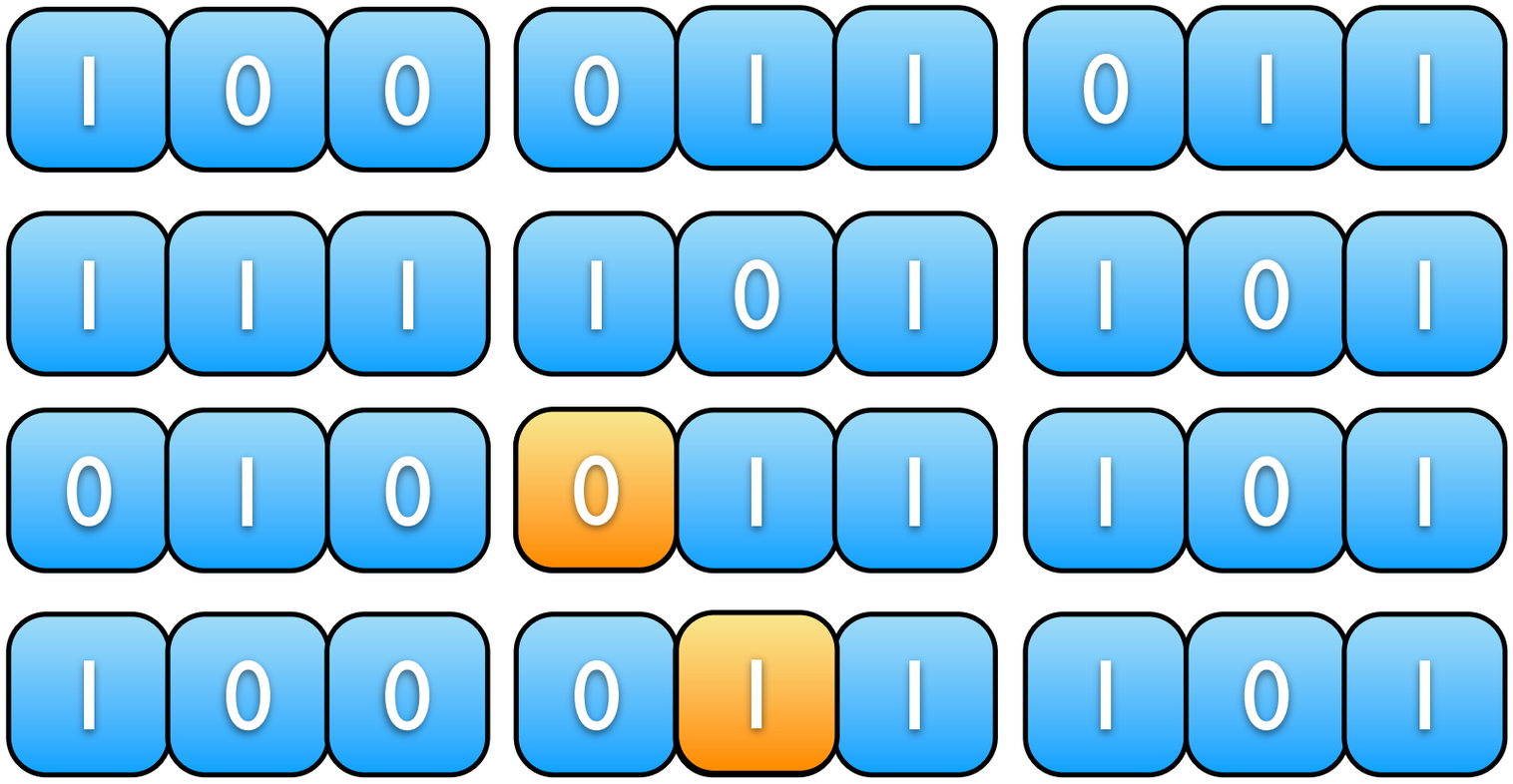
(7,5,1) A: 3, B: 2

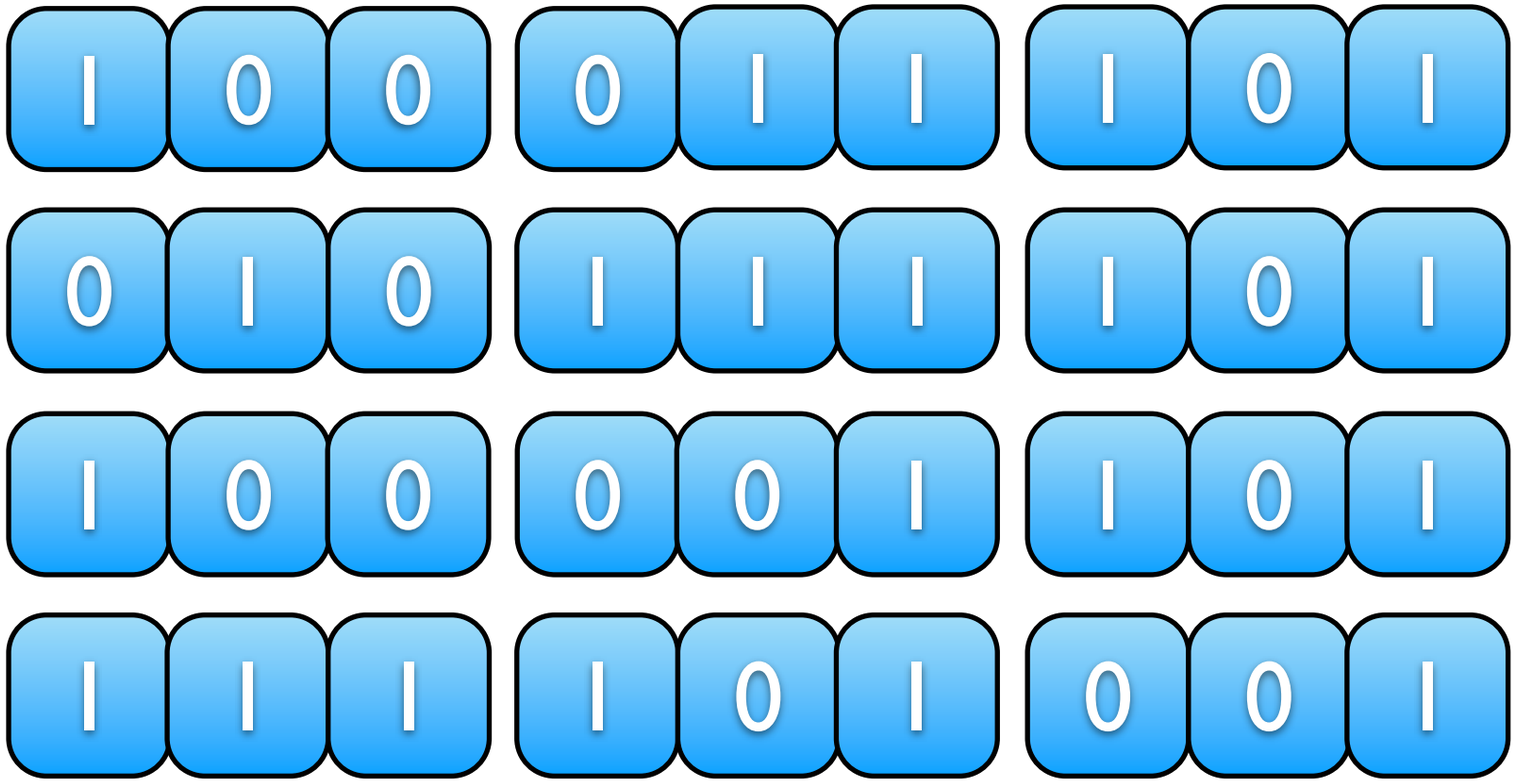


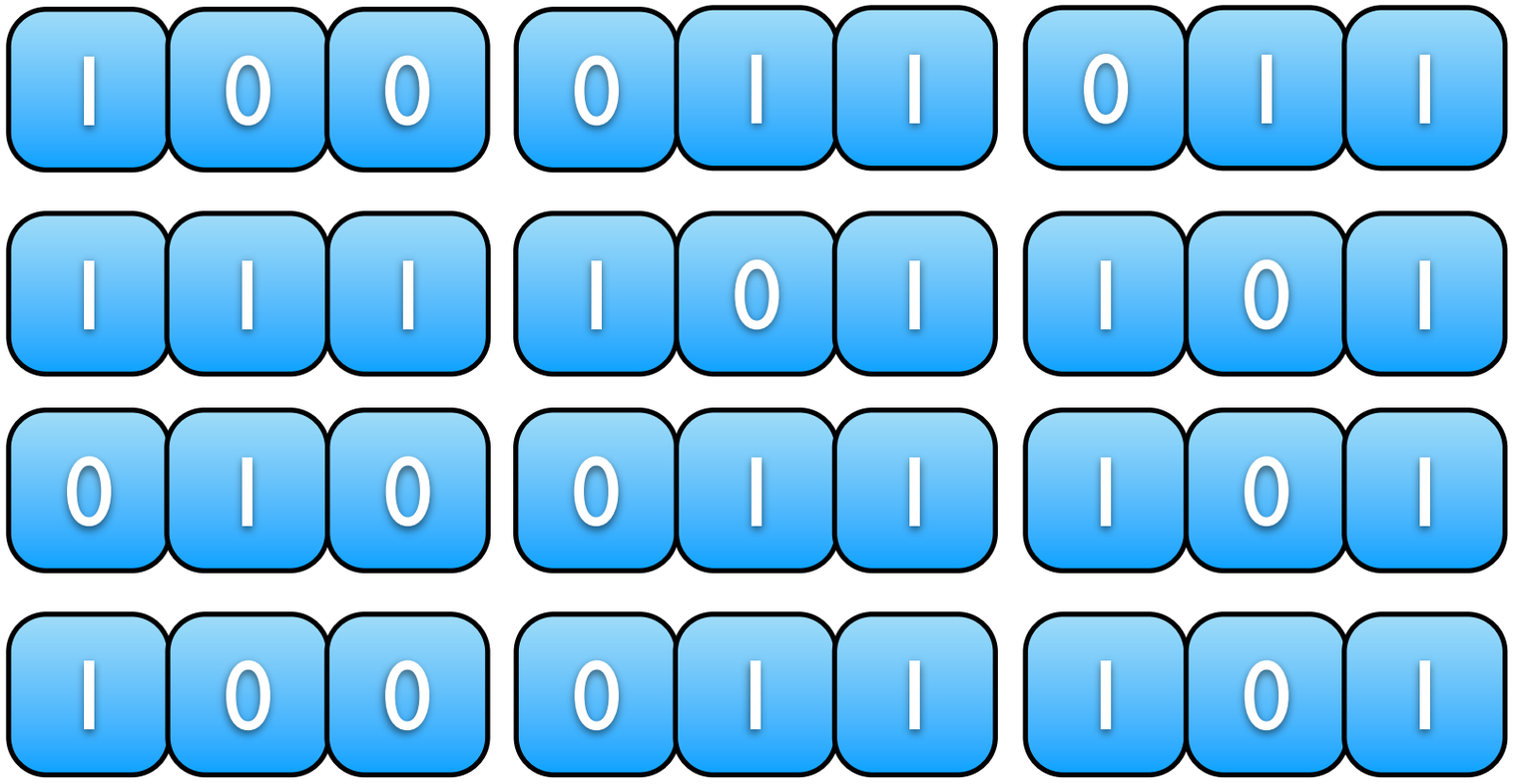

1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2

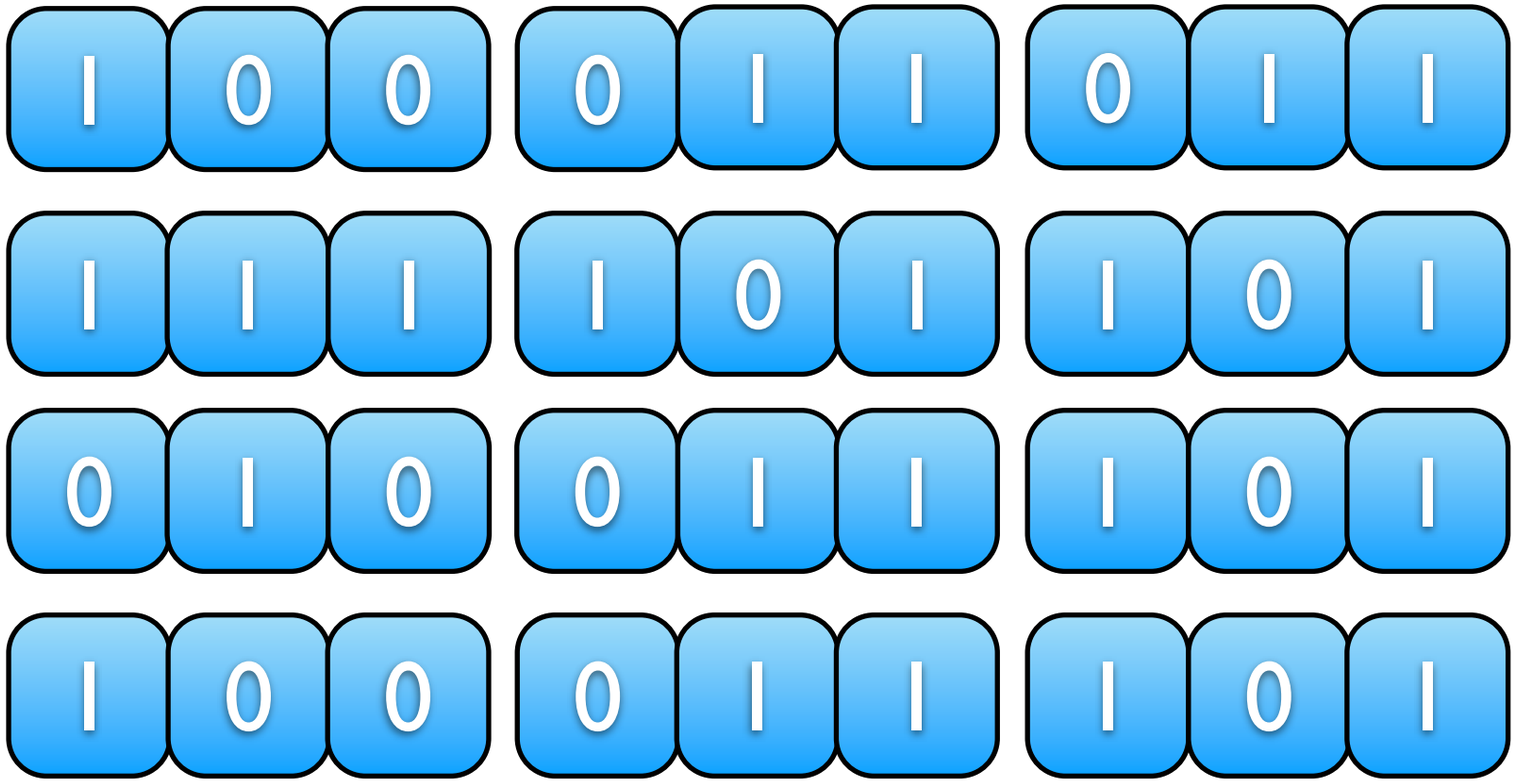

1 0 0 0 1 1 0 1 1
1 1 1 1 0 1 1 0 1
0 1 0 0 1 1 1 0 1
1 0 0 0 0 1 1 0 1

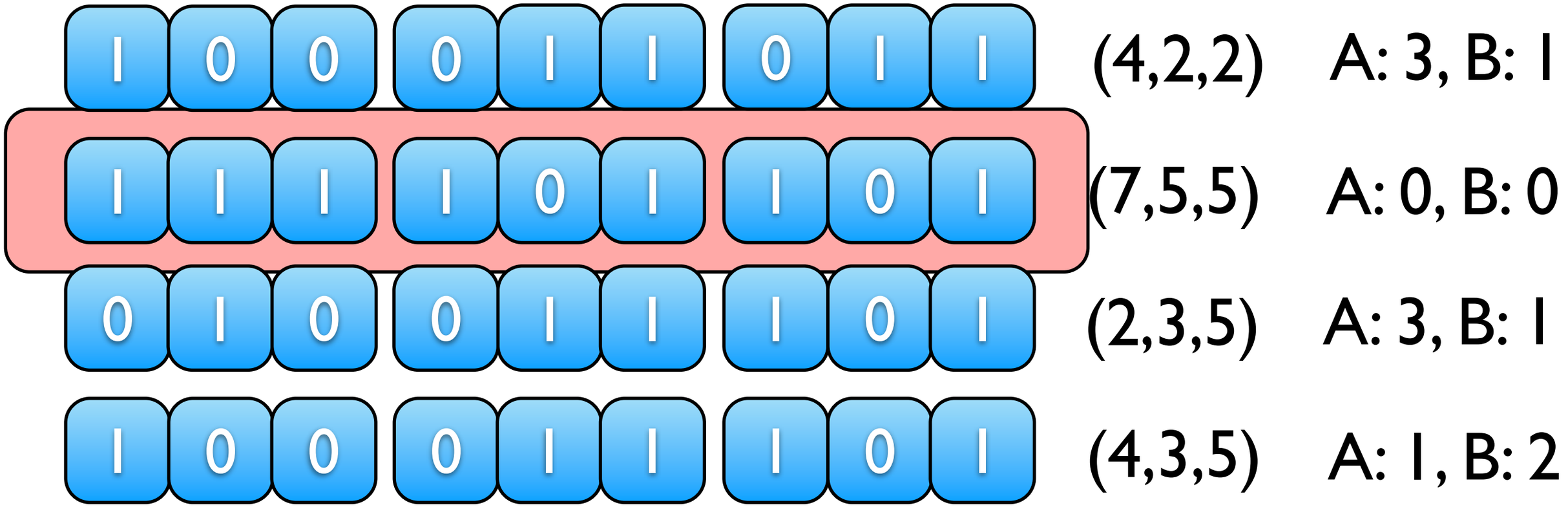

1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2


1 0 0 0 1 1 0 1 1
1 1 1 1 0 1 1 0 1
0 1 0 0 1 1 1 0 1
1 0 0 0 1 1 1 0 1


1 0 0 0 1 1 1 0 1
(4,3,5) A: 2, B: 2
0 1 0 1 1 1 1 0 1
(2,7,5) A: 3, B: 1
1 0 0 0 0 1 1 0 1
(4,1,5) A: 3, B: 1
1 1 1 1 0 1 0 0 1
(7,5,1) A: 3, B: 2

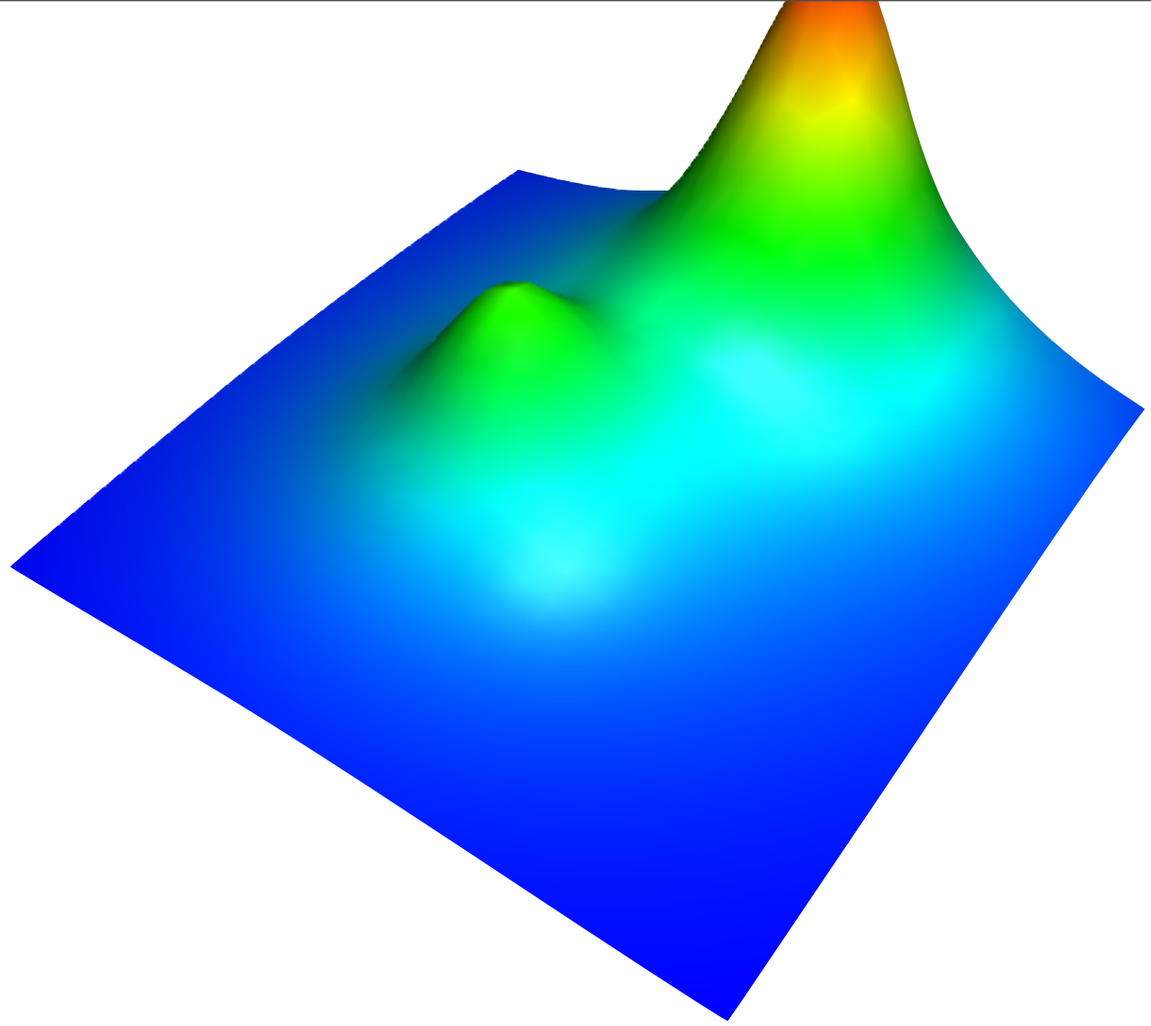

1 0 0 0 1 1 0 1 1
1 1 1 1 0 1 1 0 1
0 1 0 0 1 1 1 0 1
1 0 0 0 1 1 1 0 1


1 0 0 0 1 1 0 1 1
(4,2,2) A: 3, B: 1
1 1 1 1 0 1 1 0 1
(7,5,5) A: 0, B: 0
0 1 0 0 1 1 1 0 1
(2,3,5) A: 3, B: 1
1 0 0 0 1 1 1 0 1
(4,3,5) A: 1, B: 2


1 0 0 0 1 1 0 1 1
(4,2,2)
A: 3, B: 1
1 1 1 1 0 1 1 0 1
(7,5,5)
A: 0, B: 0
0 1 0 0 1 1 1 0 1
(2,3,5)
A: 3, B: 1
1 0 0 0 1 1 1 0 1
(4,3,5)
A: 1, B: 2

Test Data Generation

Given a function and a location we want to reach, how do we derive inputs to the function that lead the control flow to the desired statement?



Biological Evolution

- Gene
  Unit of information • passed from one generation to another
- Natural Selection
- Survival of The Fittest
- Origin of New Species