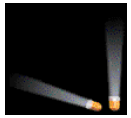
Applications of Stack

- Parentheses Matching
- Towers Of Hanoi/Brahma
- System Stack
- Rat In A Maze

Parentheses Matching by Stack

position: 0 1 2 3 4 5 6 7

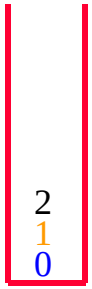
- ((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)
  - Output pairs (u,v) such that the left parenthesis at position u is matched with the right parenthesis at v.
    - (2,6) (1,13) (15,19) (21,25) (27,31) (0,32) (34,38)
- (a+b))*((c+d)
  - (0,4)
  - right parenthesis at 5 has no matching left parenthesis
  - (8,12)
  - left parenthesis at 7 has no matching right parenthesis

Parentheses Matching

- scan expression from left to right
- when a left parenthesis is encountered, add its position to the stack
- when a right parenthesis is encountered, remove matching position from stack

Example

- ((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)

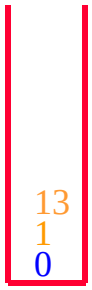


Example

- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)

Example

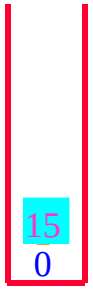
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6)

Example

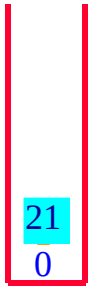
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13)

Example

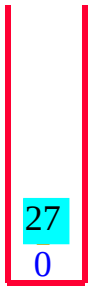
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19)

Example

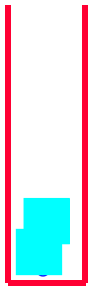
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19) (21,25)

Example

- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19) (21,25)(27,31) (0,32)

- and so on

Recursion

- *Recursion* is a fundamental programming technique that can provide an elegant solution for certain kinds of problems.
- A *recursive definition* is one which uses the word or concept being defined in the definition itself.

Example: Recursive Definition of N!

- N!, for any positive integer N, is defined to be the product of all integers between 1 and N inclusive.
- This definition can be expressed recursively as:

```
0!  =  1
1!  =  1 * 0!
N!  =  N * (N-1)!
```

- The concept of the factorial is defined in terms of another factorial.
- Eventually, the base case of 0! is reached.

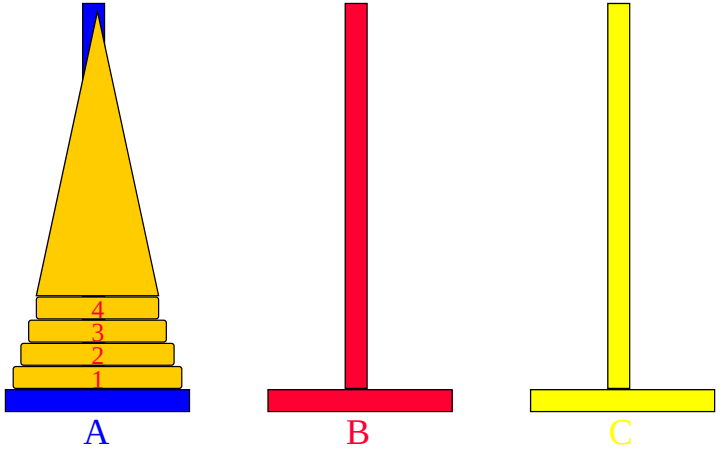
In class Exercise: Fibonacci Numbers

- 1, 1, 2, 3, 5, 8, 13, 21, …
- Using a recursive formula to define Fibonacci Numbers

Homework

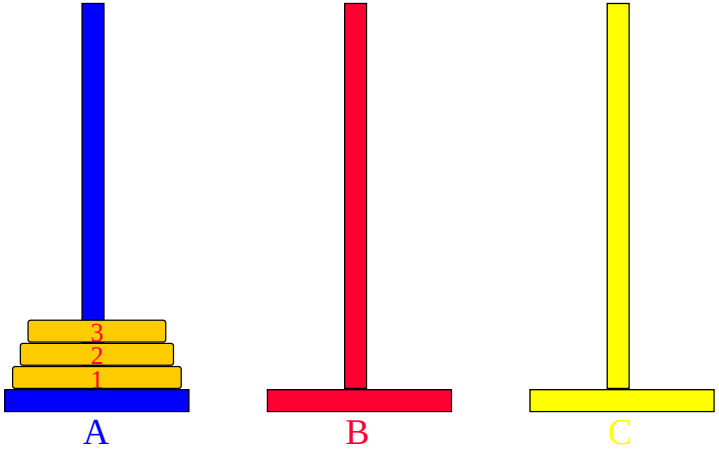
- 撰寫一個可執行的 iterative program
- input:：n
  - 需判斷 n必須爲一個大於等於零的整數
- Output：fib(0) ~fib(n)
  -

Towers Of Hanoi/Brahma



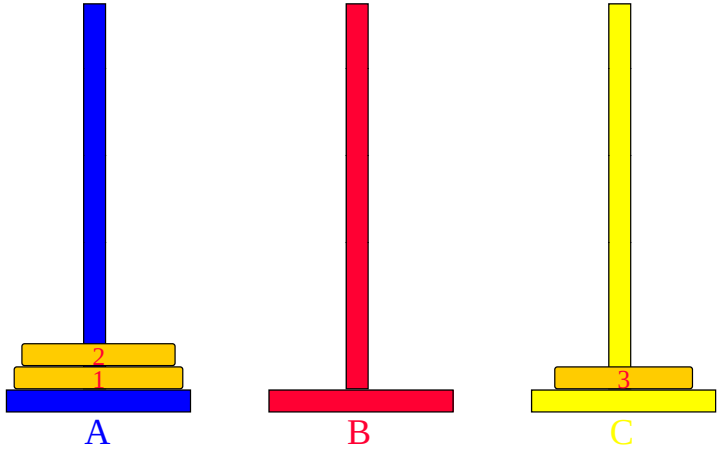
- 64 gold disks to be moved from tower A to tower C
- each tower operates as a stack
- cannot place big disk on top of a smaller one

Towers Of Hanoi/Brahma



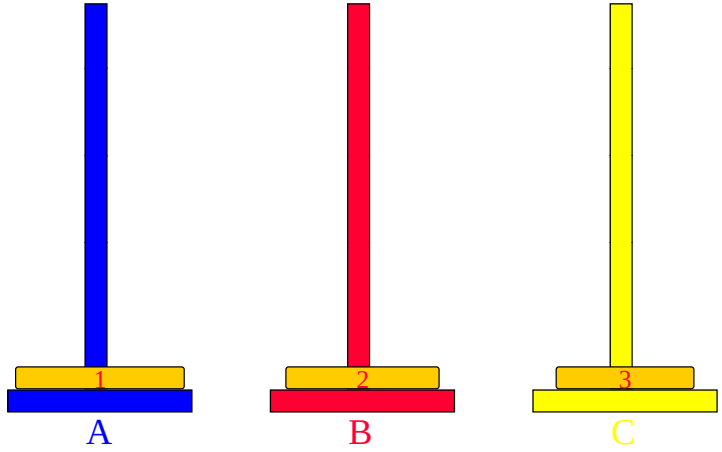
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



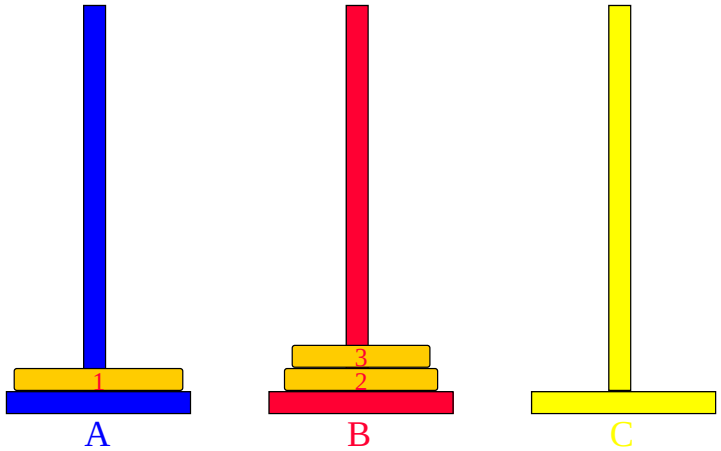
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



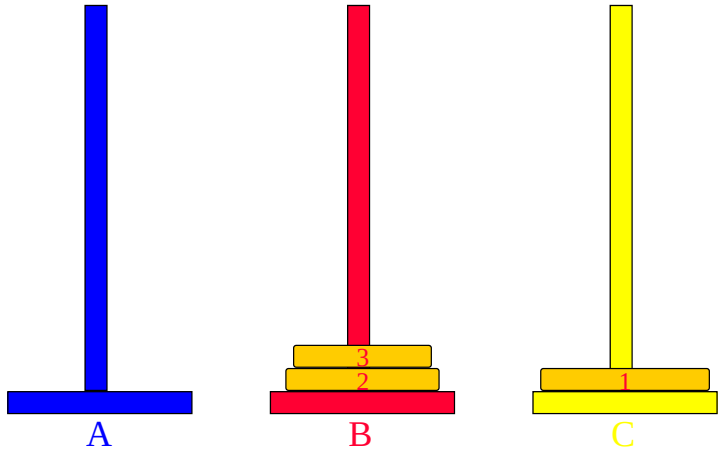
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



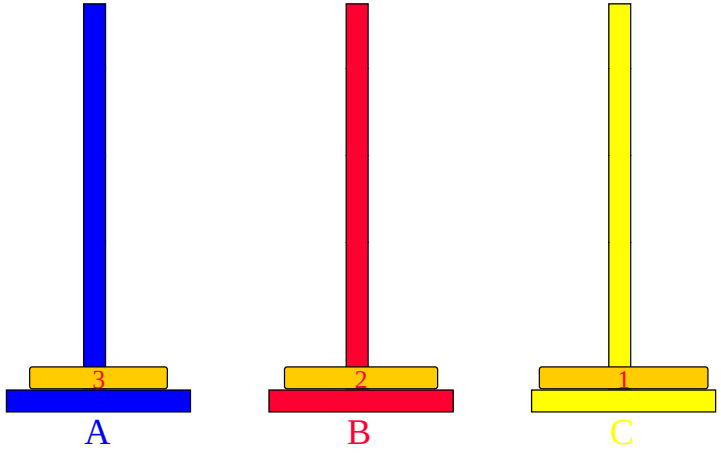
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



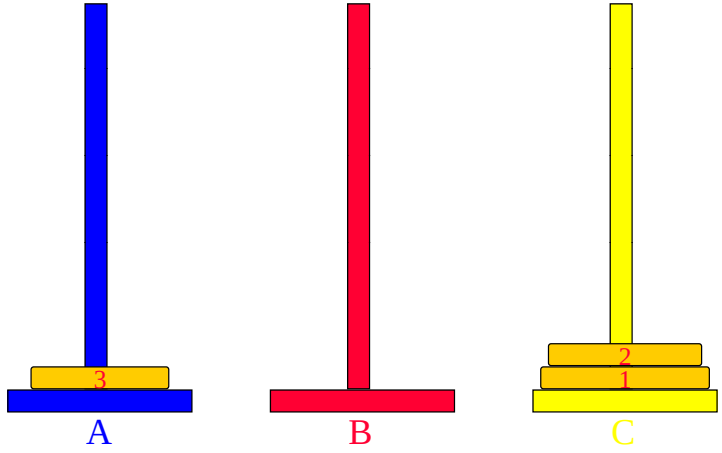
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



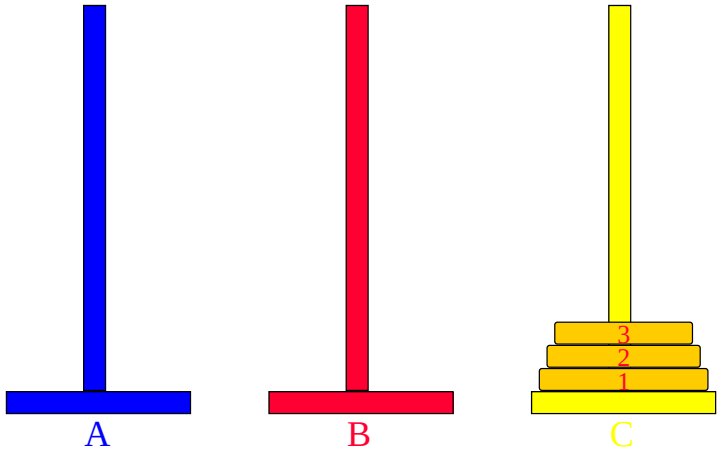
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



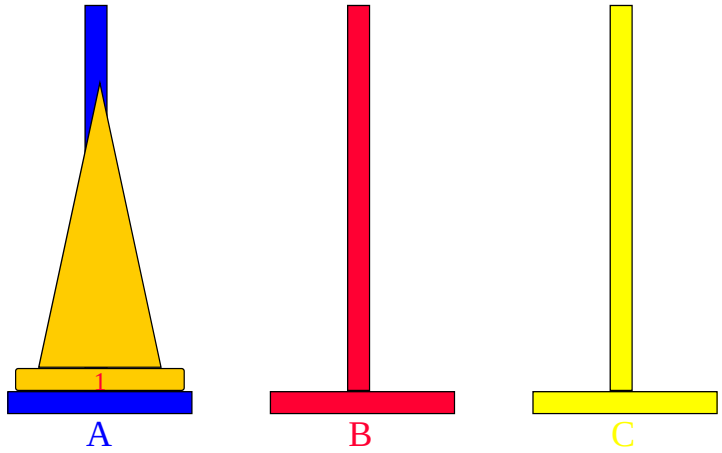
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



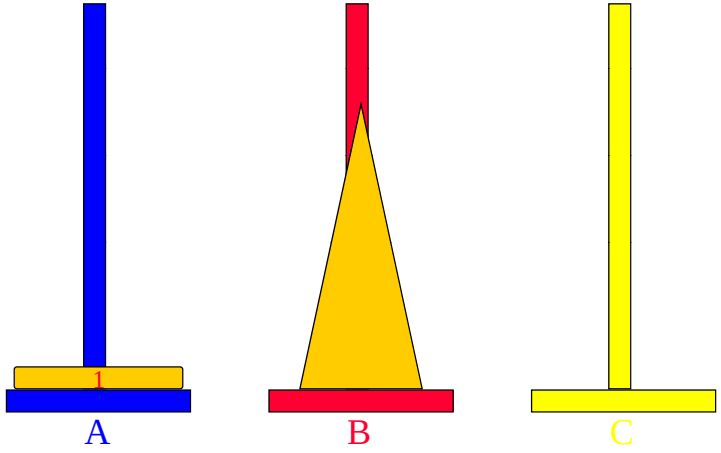
- 3-disk Towers Of Hanoi/Brahma
- 7 disk moves

Recursive Solution



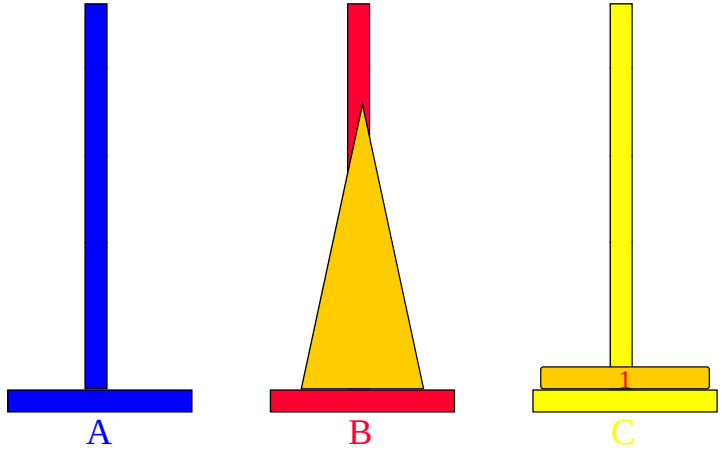
- n > 0 gold disks to be moved from A to C using B
- move top n-1 disks from A to B using C

Recursive Solution



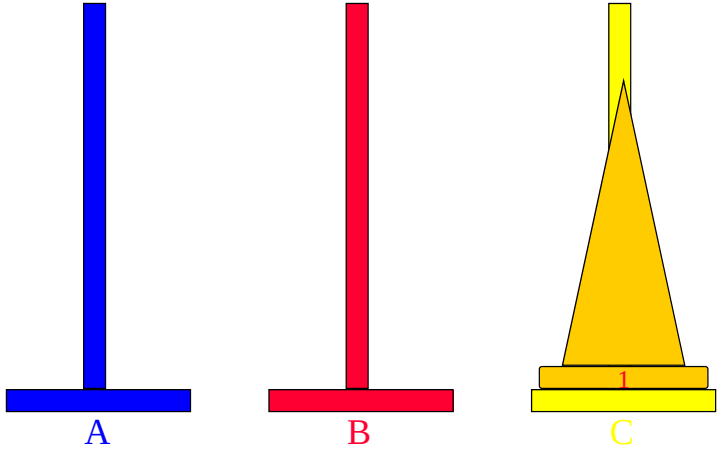
- move top disk from A to C

Recursive Solution



- move top n-1 disks from B to C using A

Recursive Solution



- moves(n) = 0 when n = 0
- moves(n) = 2*moves(n-1) + 1 = 2^n-1 when n > 0

In Class Exercise

- Show that

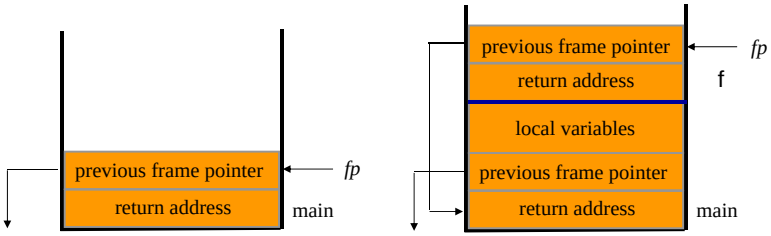
moves(n) = 2*moves(n-1) + 1 = 2^n-1 when n > 0

Towers Of Hanoi/Brahma

- moves(64) = $1.8 * 10^{19}$ (approximately)
- Performing 10^9 moves/second, a computer would take about 570 years to complete.
- At 1 disk move/min, the monks will take about $3.4 * 10^{13}$ years.

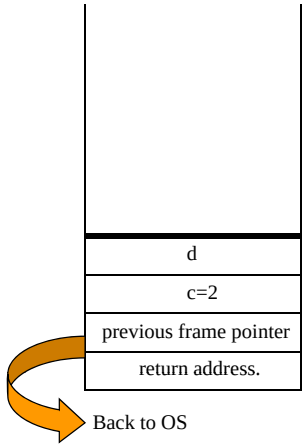
System Stack

- Whenever a function is invoked, the program creates a structure, referred to as an activation record or a stack frame, and places it on top of the system stack.



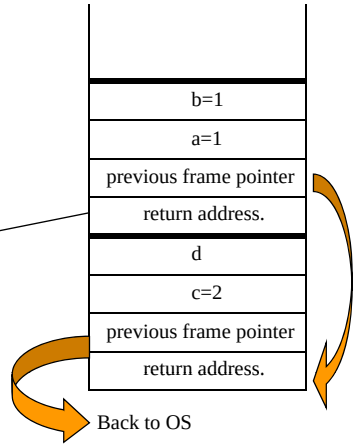
Trace System Stack

```
double f()
{
  int a=1,b=1;
  return a+b;
}
void main()
{
  int c=2, d;
  d=f();
  d=d+d;
}
```



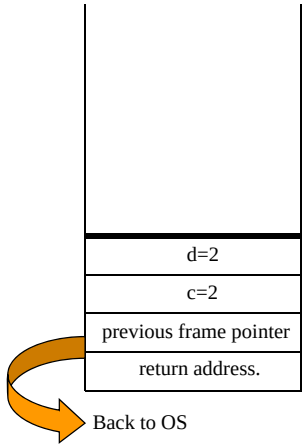
Trace System Stack

```
double f()
{
  int a=1,b=1;
  return a+b;
}
void main()
{
  int c=2, d;
  d=f();
  d=d+d;
}
```

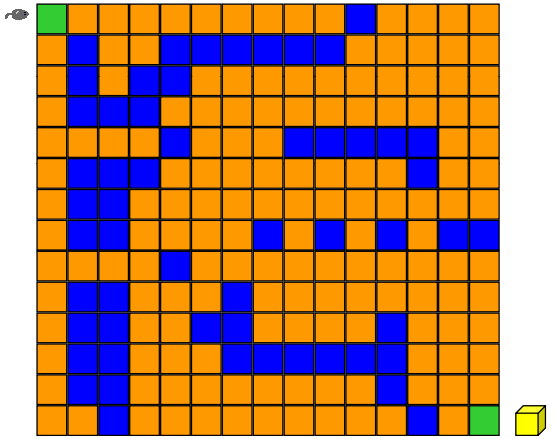


Trace System Stack

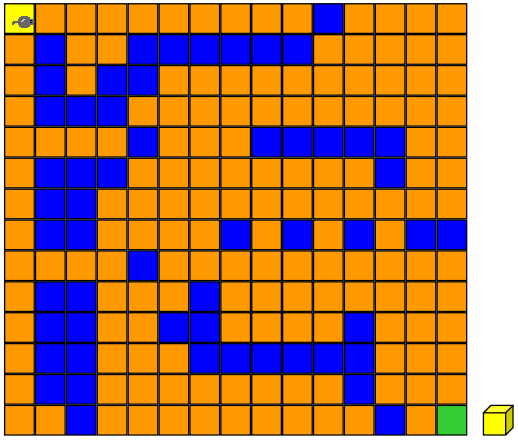
```
double f()
{
  int a=1,b=1;
  return a+b;
}
void main()
{
  int c=2, d;
  d=f();
  d=d+d;
}
```



Rat In A Maze

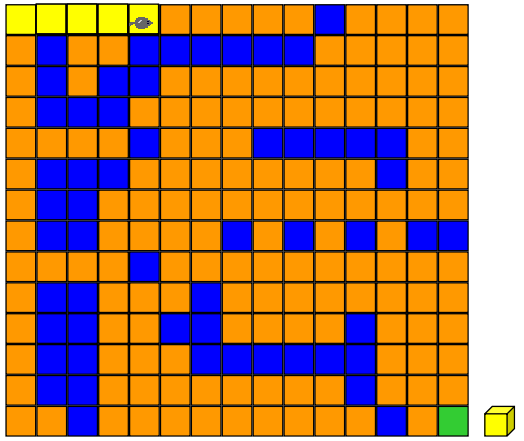


Rat In A Maze



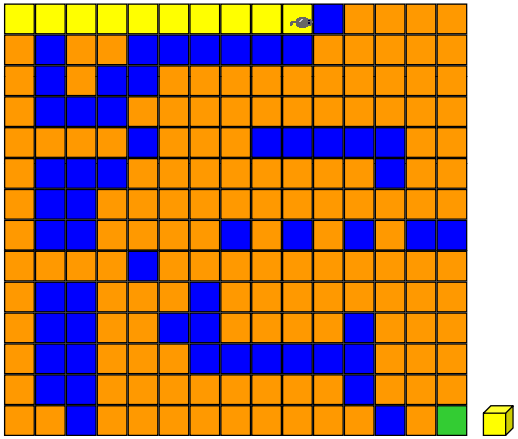
- Move order is: right, down, left, up
- Block positions to avoid revisit.

Rat In A Maze



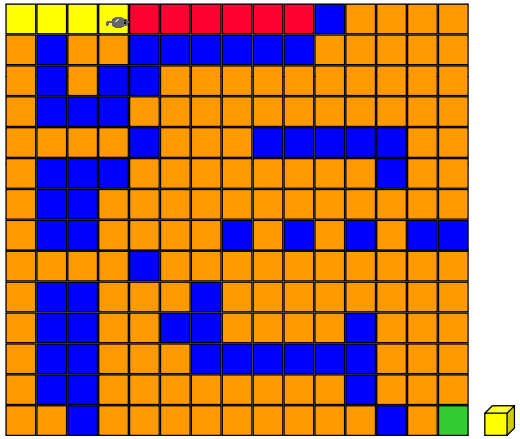
- Move order is: right, down, left, up
- Block positions to avoid revisit.

Rat In A Maze



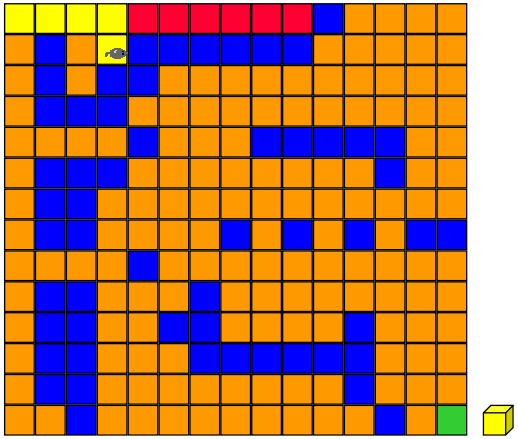
- Move backward until we reach a square from which a forward move is possible.

Rat In A Maze



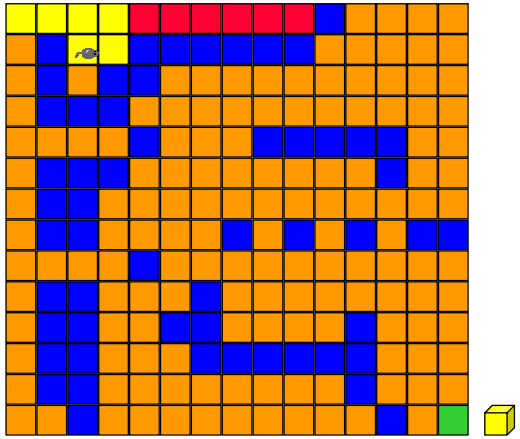
- Move down.

Rat In A Maze



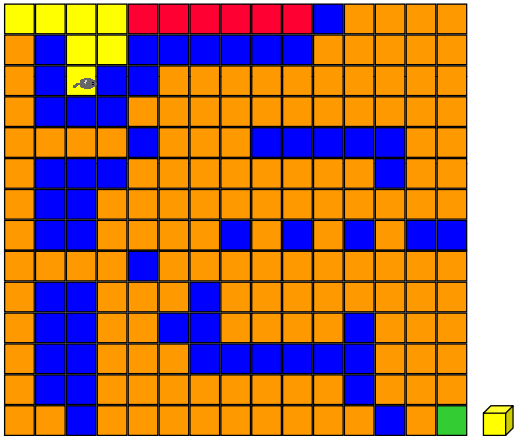
- Move left.

Rat In A Maze



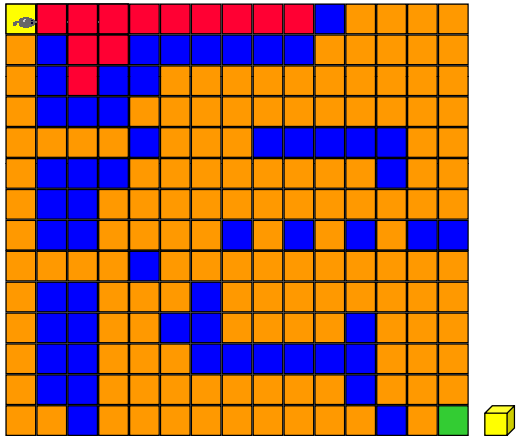
- Move down.

Rat In A Maze



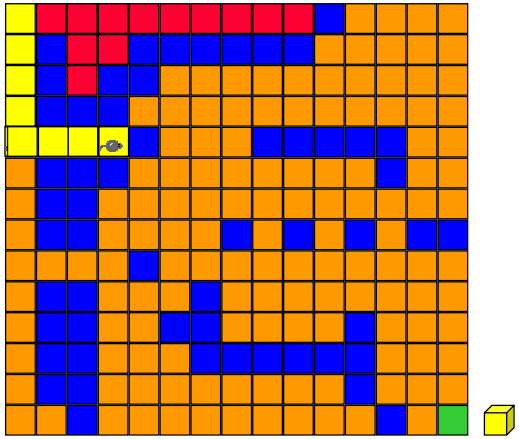
- Move backward until we reach a square from which a forward move is possible.

Rat In A Maze



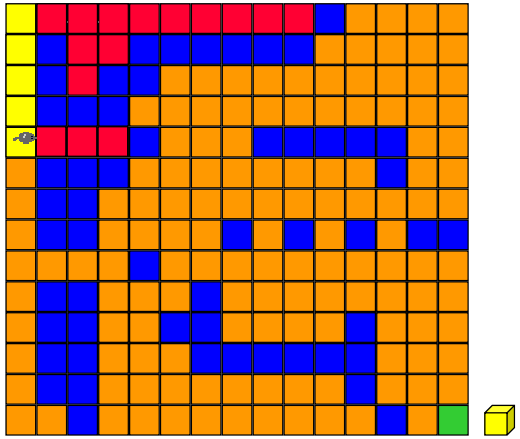
- Move backward until we reach a square from which a forward move is possible.
- Move downward.

Rat In A Maze



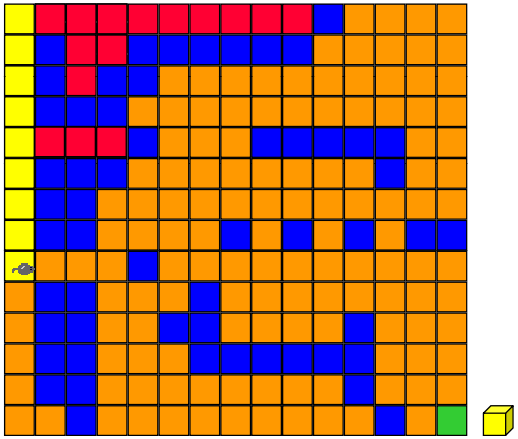
- Move right.
- Backtrack.

Rat In A Maze



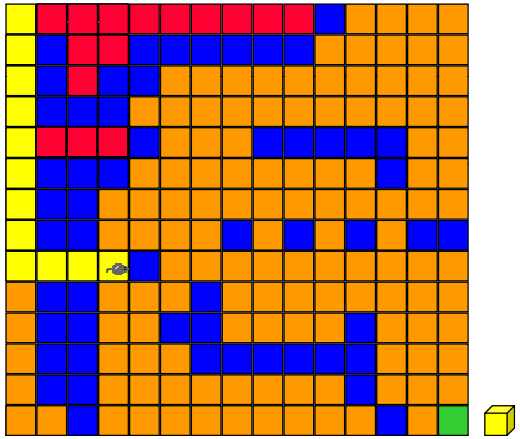
- Move downward.

Rat In A Maze



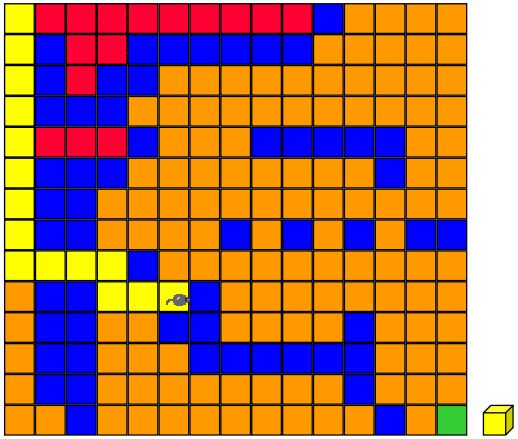
- Move right.

Rat In A Maze



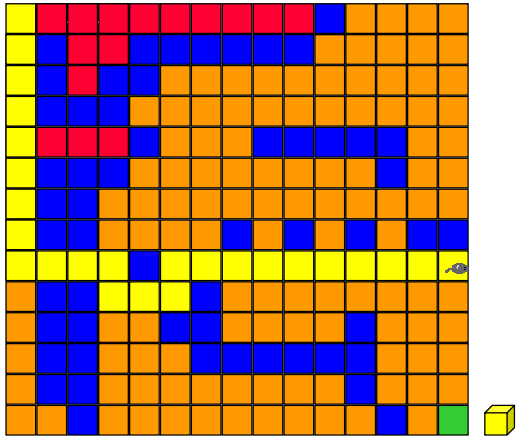
- Move one down and then right.

Rat In A Maze



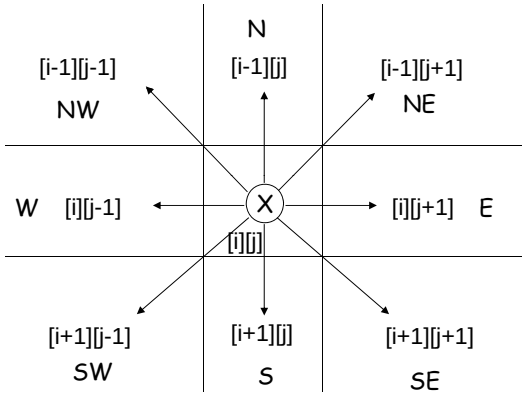
- Move one up and then right.

Rat In A Maze



- Move down to exit and eat cheese.
- Path from maze entry to current position operates as a stack.

Remark: Allowable Moves for the Rat in the Textbook



Homework

- Sec. 3.5 Exercise 1 (b) P157
  - Trace the program (find the path on the maze with stack).