Chapter 9

STACK, QUEUE

Stacks

LIFO: Last In, First Out.

Restricted form of list: Insert and remove only at front of list.

Notation:

- Insert: PUSH
- Remove: POP
- The accessible element is called TOP.

Stack ADT

```
typedef int BOOL ;
typedef int Elem;
  // Initialize the stack
  void init(Stack);
  // Reinitialize the stack
  void clear(Stack);
  // Push an element onto the top of the stack.
  BOOL push(Stack, Elem);
  // Remove the element at the top of the stack.
  BOOl pop(Stack, Elem*);
  // Get a copy of the top element in the stack
  BOOl topValue(Stack, Elem*);
  // Return the number of elements in the stack.
  int length(Stack);
```

Array-Based Stack

```
// Array-based stack implementation
int size;        // Maximum size of stack
  int top;         // Index for top element
  Elem *listArray; // Array holding elements
```

Issues:

- Which end is the top?
- Where does "top" point to?
- What is the cost of the operations?

Linked Stack

```
// Linked stack implementation
node_ptr top; // Pointer to first elem
  int currentSize; // Count number of elems
```

What is the cost of the operations?

How do space requirements compare to the array-based stack implementation?

Array-Based Stack Class (1)

```
#define TRUE 1
#define FALSE 0
typedef int BOOL ;


typedef int Elem;

typedef struct STACK{
  int size ;      // Maximum size of stack
  int top  ;       // Index for top element
  Elem *listArray ; // Array holding elements

}*Stack;
```

Array-Based Stack Class (2)

```
void init_size(Stack stack, int mysize)
{   stack->size = mysize;
    stack->top = 0;
    stack->listArray =
      (Elem*)malloc((mysize)*sizeof(Elem)); }
void init(Stack stack)
{    init_size(stack, 200);}
void clear(Stack stack)
{   stack->size = 0;
    stack->top = 0;
    if( stack->listArray != NULL )
    { free(stack->listArray); stack->listArray =
NULL; }}
```

Array-Based Stack Class (3)

```
BOOL push(Stack stack, Elem e)
{   if ( stack->top < stack->size ){
        stack->listArray[ stack->top++ ] = e;
        return TRUE;
    }
    return FALSE;}
BOOL pop(Stack stack, Elem* e)
{    if ( stack->top > 0  ){
        *e = stack->listArray[ stack->top - 1 ];
        stack->top --;
        return TRUE;
    }
    return FALSE;}
```

Array-Based Stack Class (4)

```
 // Get a copy of the top element in the stack
BOOL topValue(Stack stack, Elem* e)
{    if ( stack->top > 0  ){
          *e = stack->listArray[ stack->top - 1 ];
          return TRUE;
    }
    return FALSE;}
  // Return the number of elements in the stack.
int length(Stack stack)
{
    return stack->top;
}
```

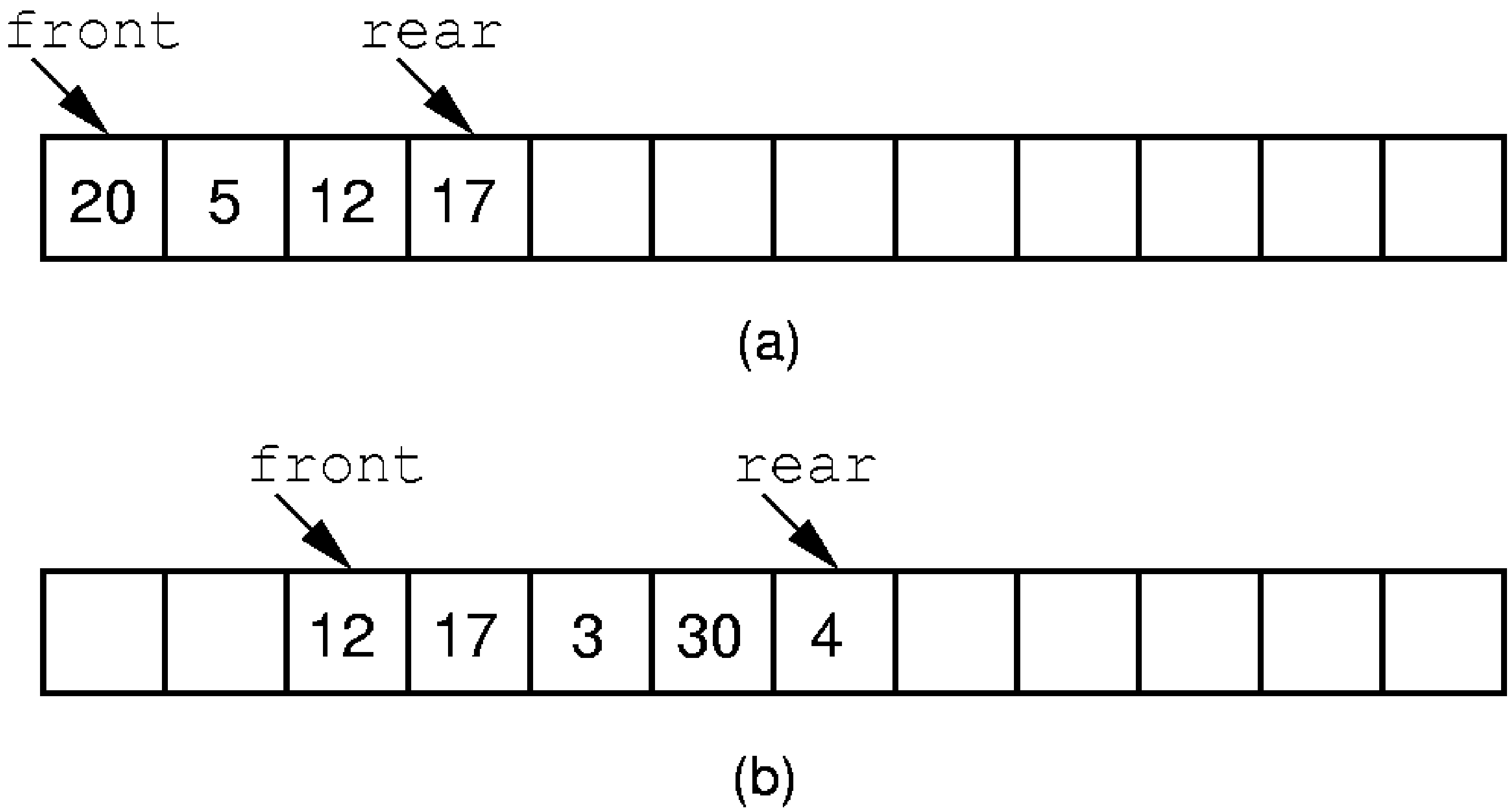
Queues

FIFO: First in, First Out

Restricted form of list: Insert at one end, remove from the other.

Notation:
- Insert: Enqueue
- Delete: Dequeue
- First element: Front
- Last element: Rear

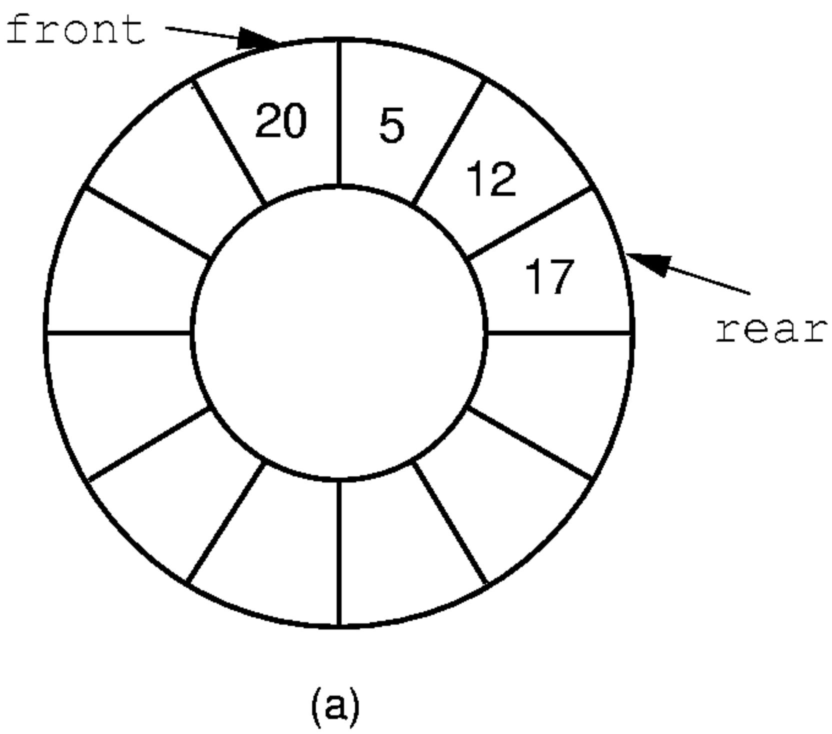
Queue Implementation (1)



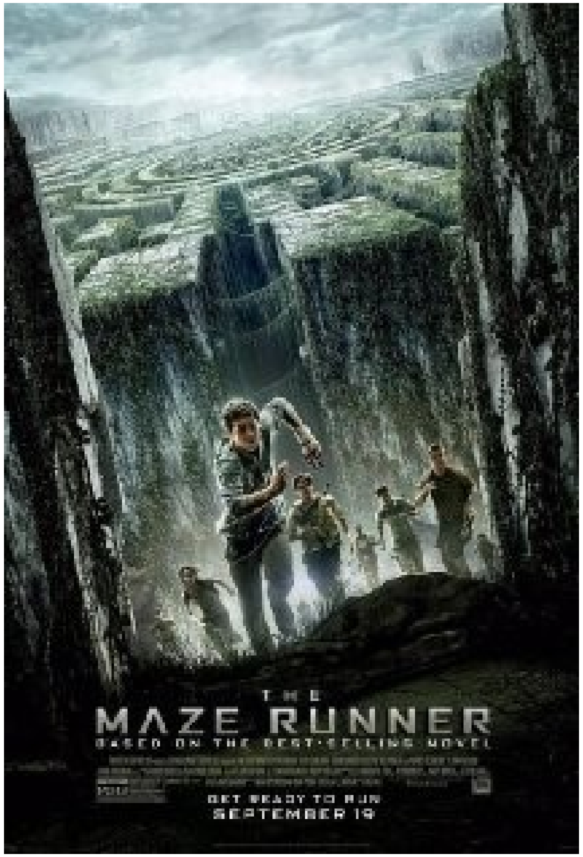
Queue ADT

```
// Initialize the queue
  void init(Queue, int);
  // Reinitialize the queue
  void clear(Queue);
  // enqueue an element into the queue.
  BOOL enqueue(Queue, Elem) ;
  // dequeue an element from the queue.
  BOOL dequeue(Queue, Elem*);
  // Get a copy of the front element in the
  queue
  bool frontValue(Queue, Elem*);
  // Return the number of elements in the
  queue.
     int length(Queue);
```

Queue Implementation (2)



The Maze Runner



How to find a way out of maze?

Problem description

- Given you a maze map, a start point(S) and an exit (E), calculate the minimum distance from the start to the exit.

Problem description

- A map consists of 4 kinds of characters:
  - ‘S’ means the start point;
  - ‘E’ means the exit point;
  - ‘.’ means the opened-block that you can pass;
  - ‘#’ means the closed-block that you cannot pass;

A sample Map

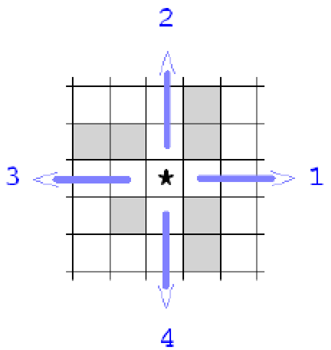
```
########
#...#..#
#.#....#
#.###..#
#S#E...#
########
```

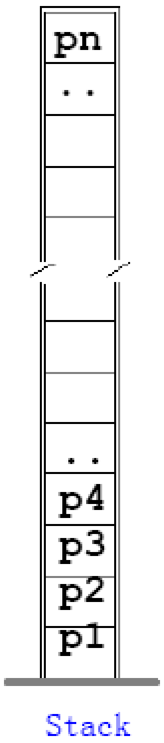
Problem description

- Requirements:
  – It is ONLY allowed to move by one step vertically or horizontally(up, down , left or right) to the next block;
  – You MUST NOT get out of the map;
  – Return -1 if given arguments can not satisfy the above requirements or you cannot find a way from ‘S’ to ‘E’.

DFS and Stack

- Depth first search
  - Go go go, until you meet a wall!
  - Backward
- Use a stack to trace the path
  - Whenever meet a wall, pop the last point
- Need to try all possible path to decide the shortest way

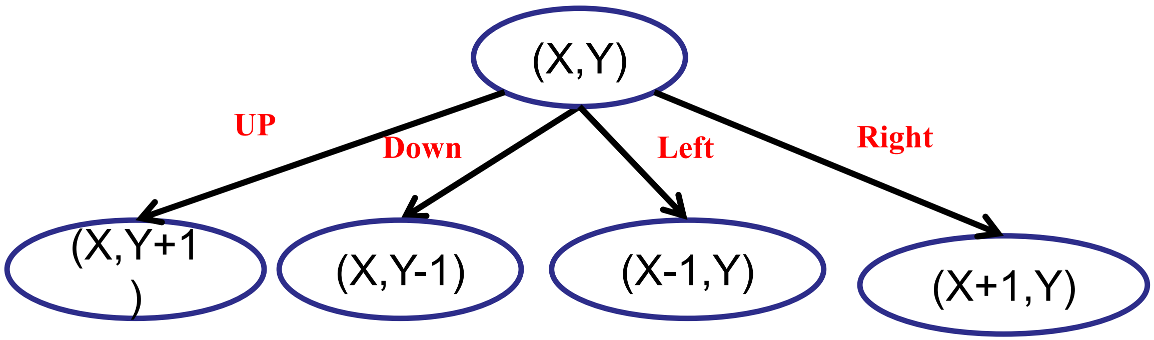




Stack based Algorithm

- Algorithm
  - Mark the Begin Node as tried. Push the Node.
  - While(Stack is not Empty){
  - Pop a node=N
  - if(N is the goal){ break;}
  - Try 4 directions of N.
  - if(a direction has a neighbor of N = M && M is not tried yet.){
  - Push M into Stack.
  - Set N as the pre-node of M
  - }
  - }
  - If(N is the Goal){
  - Output N.
  - while(N has a pre-node = M){
  - Output M;
  - N=M;
  - }
  - }
  - Else{
  - there is not any path.
  - }

Hint



- Dist[x][y] denotes the shortest distance from S point to position (x,y).
- Initially,
  - Dist[x][y] = MAX; (for (x,y) != S)
  - Dist[x][y] = 0; (for (x,y) == S)

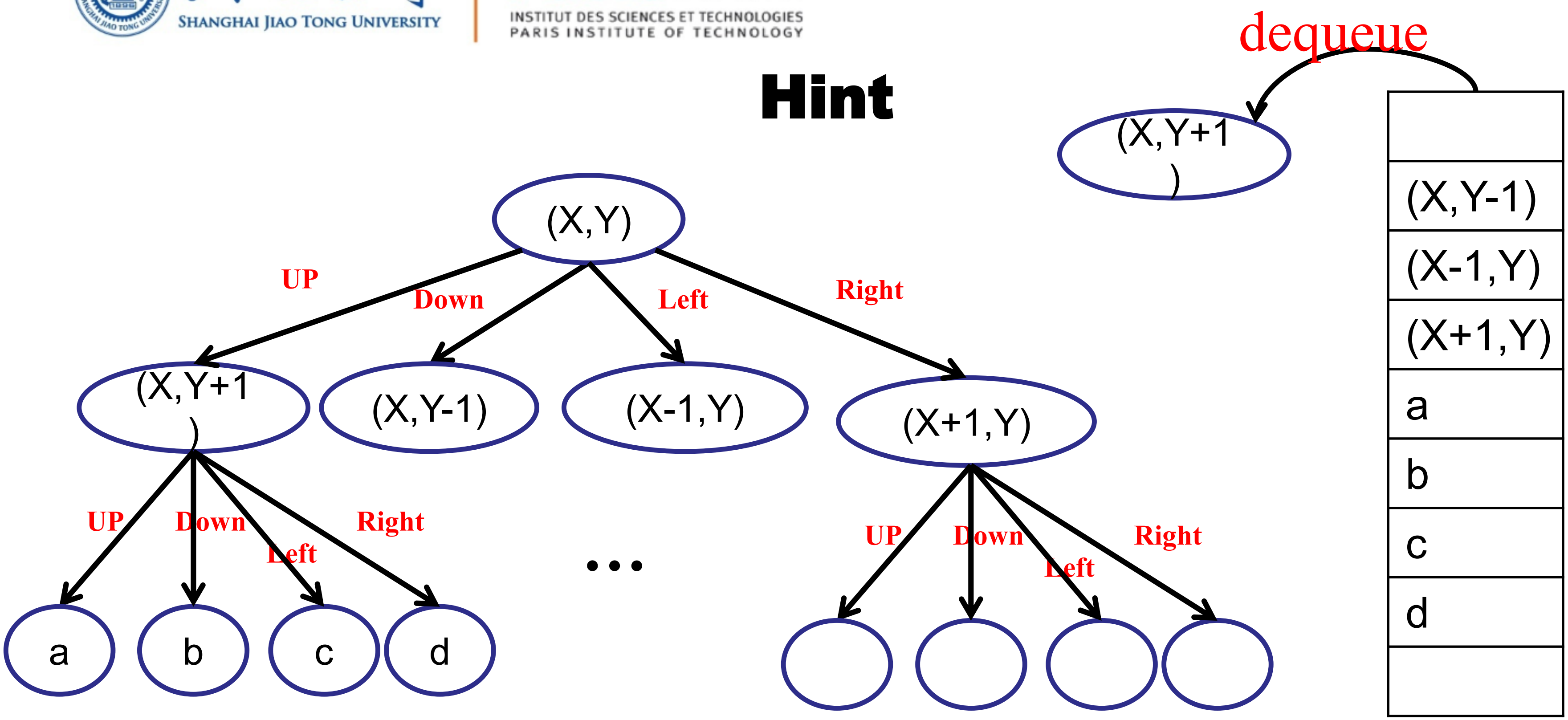
Queue and BFS

- Breadth First Search
  - Search each level
  - Done with every point, and do not come back
- Use a queue to store further points
  - Points to check
- Then you can update the distance for new position

```
If ( dist[new_x][new_y] < dist[x][y] + 1)
      dist[new_x][new_y] = dist[x][y] + 1;
```

(X,Y+1)
(X,Y-1)
(X-1,Y)
(X+1,Y)

Hint



- As long as the queue is not empty, you dequeue a position and enqueue its the next 4 positions .
- Update the distance matrix.

Algorithm

- Enqueue the root node, i.e the position of S
- Dequeue a node
  - enqueue the direct child nodes (at most 4)
  - Update the child nodes distance
- If the queue is empty, every node on the graph has been examined – quit the search.
- If the queue is not empty, repeat from Step 2.