HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

C Programming Basic

Stacks and Queues

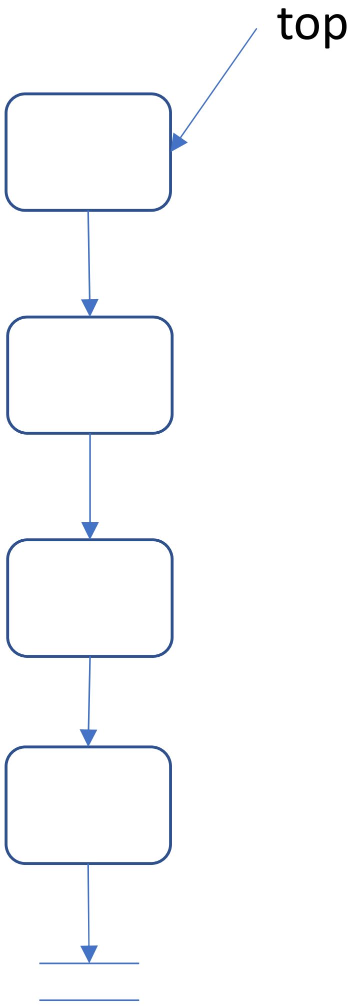
Content

- Implementation of stacks and application to the parenthesis checking problem
- Implementation of queues and application to the MAZE problem

Stacks

- Each node of the stack has following fields
  - Data: char type, representing (,), {, }, [,]
  - Pointer to the next element in the stack
  - Maintain top which is a pointer to the first element of the linked list

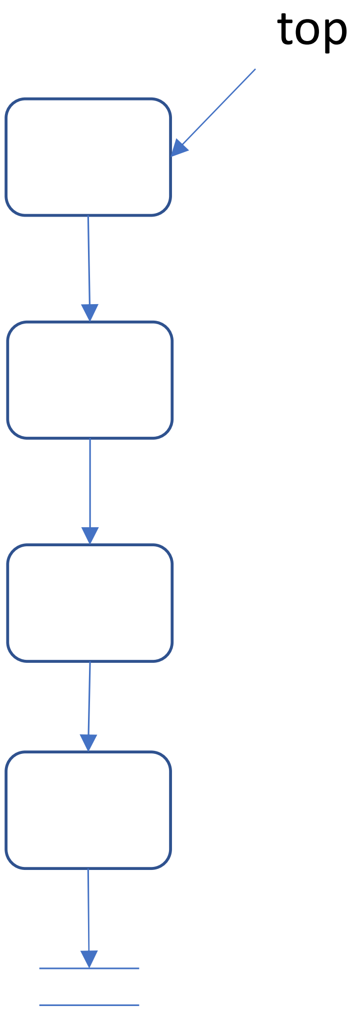
```
typedef struct Node{
    char c;
    struct Node* next;
}Node;
Node* top;
```



Stacks

- Allocate memory

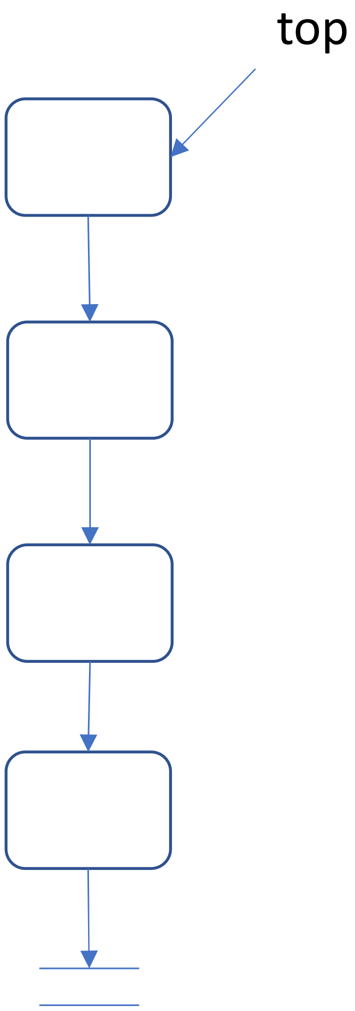
```
Node* makeNode(char x){
    Node* p = (Node*)malloc(sizeof(Node));
    p->c = x; p->next = NULL;
    return p;
}
```



Stacks

- Initialize the stack

```
void initStack(){
    top = NULL;
}
int stackEmpty(){
    return top == NULL;
}
```

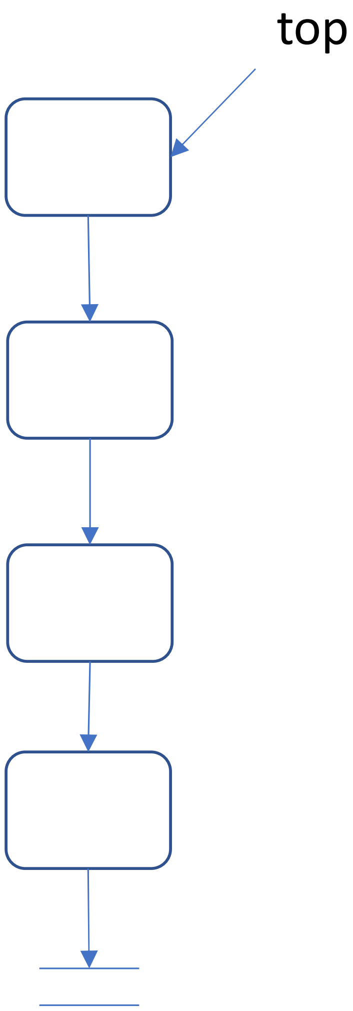


Stacks

- Push and pop operations

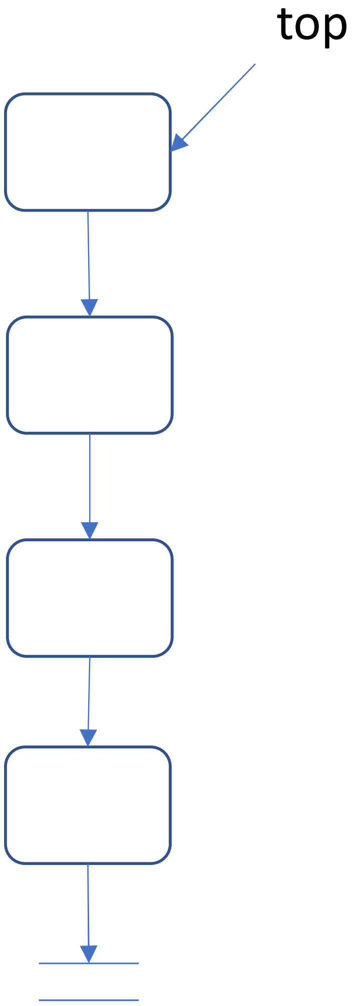
```
void push(char x){
    Node* p = makeNode(x);
    p->next = top; top = p;
}

char pop(){
    if(stackEmpty()) return ' ';
    char x = top->c;
    Node* tmp = top; top = top->next; free(tmp);
    return x;
}
```



Stacks

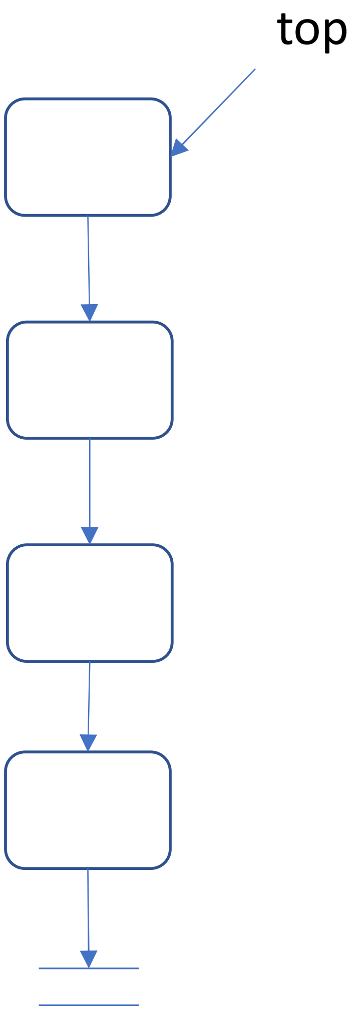
- Parenthesis expression checking
  - [({})](): true
  - ([} {}): false
- Input
  - One line contains the string (the length of the string is less than or equal to 10^6)
- Output
  - Write 1 if the sequence is correct, and write 0, otherwise



input	output
(()[][]{}){}{}[][]({[]()})	1

Stacks

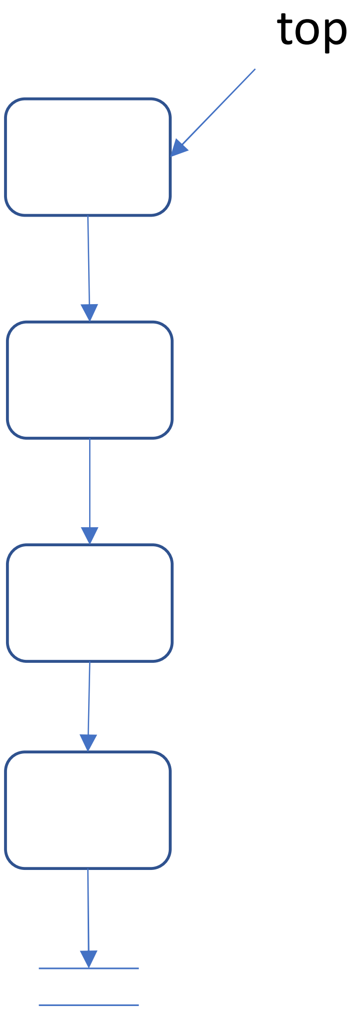
- Algorithm:
    - Initialize a stack S
    - Scan elements of the parenthesis expression
        - If meet an open parenthesis, then push it into S
        - If meet a closing parenthesis
            - If S empty, then return FALSE
            - Pop an open parenthesis out of S, if this does not match with the current closing parenthesis, then return FALSE
        - On finish: if S empty, then return TRUE, otherwise, return FALSE



Stacks

- Check the matching between an open parenthesis and a closing parenthesis

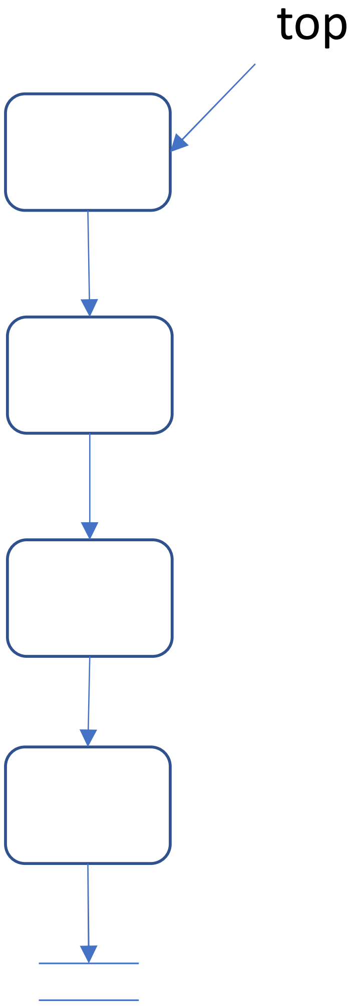
```
int match(char a, char b){
    if(a == '(' && b == ')') return 1;
    if(a == '[' && b == ']') return 1;
    if(a == '{' && b == '}') return 1;
    return 0;
}
```



Stacks

- Main algorithm

```
int check(char* s){
    initStack();
    for(int i = 0; i < strlen(s); i++){
        if(s[i] == '(' || s[i] == '[' || s[i] == '{'){
            push(s[i]);
        }else{
            if(stackEmpty()) return 0;
            char x = pop();
            if(!match(x,s[i])) return 0;
        }
    }
    return stackEmpty();
}
```

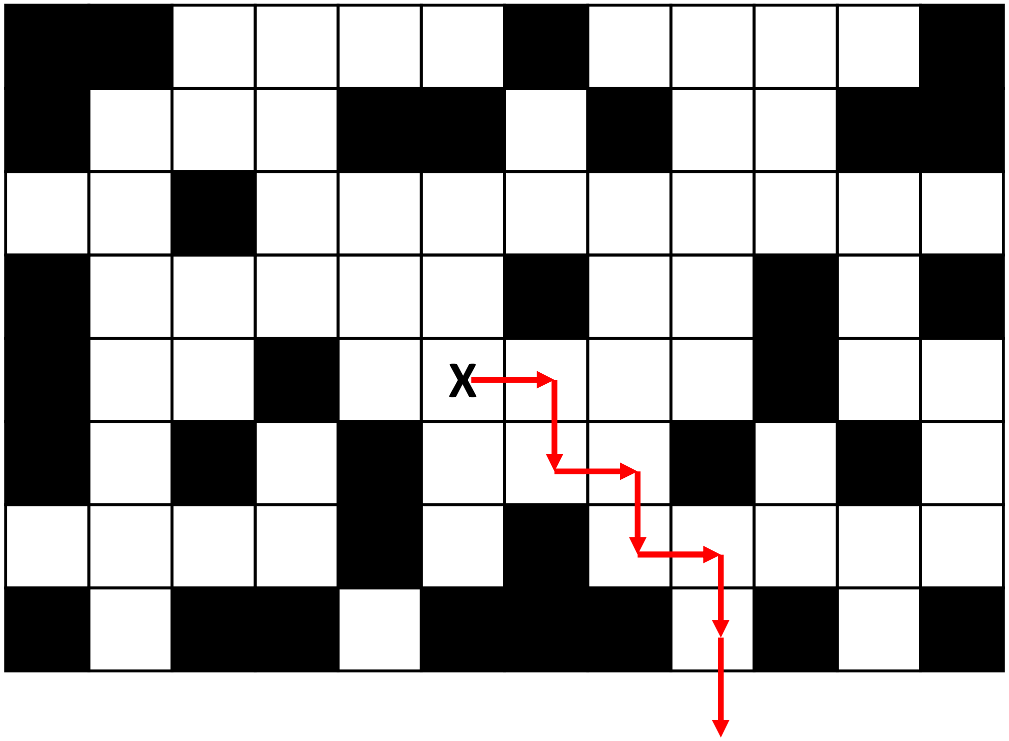


Queues

- Data structures for storing elements in a linear order
  - Push an element is performed at the end of the queue (tail)
  - Pop an element out of the queue is performed at the front of the queue (head)
- Queue can be used to solve problems of finding the shortest path in a transition graph (Breadth-First Search)

MAZE problem

- A Maze is represented by a 0-1 matrix a_{NxM} in which $a_{i,j}$ = 1 means cell (i,j) is an obstacle, $a_{i,j}$ = 0 means cell (i,j) is free.
- From a free cell, we can go up, down, left, or right to an adjacent free cell.
- Compute the minimal number of steps to escape from a Maze from a given start cell (i_0, j_0) within the Maze.



Escape the Maze after 7 steps

MAZE problem

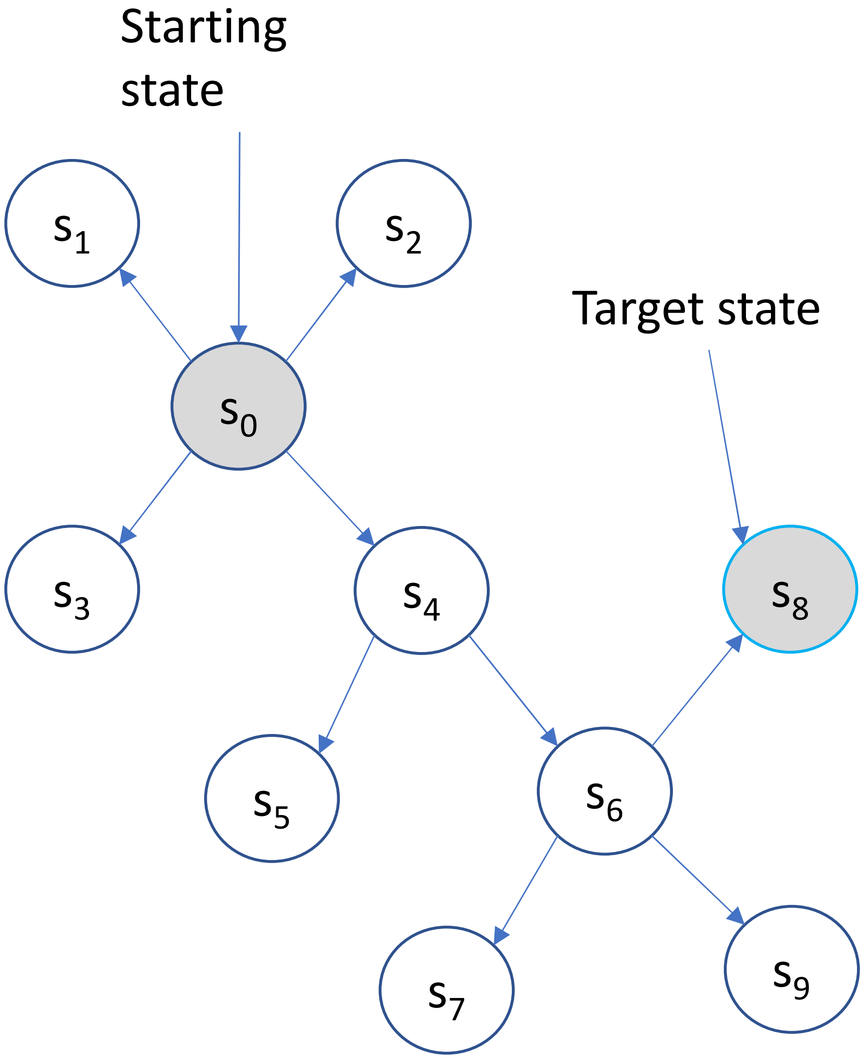
- Input
  - Line 1 contains N,M,i0,j0 ($2 \le N,M \le 900$)
  - Line i+1 (i=1,…,N) contains the i^{th} line of the matrix $a_{N \times M}$
- Output
  - Unique line contains the number minimal of steps to escape the Maze or -1 if no way to escape the Maze.

input	output

8 12 5 6
1 1 0 0 0 0 1 0 0 0 0 1
1 0 0 0 1 1 0 1 0 0 1 1
0 0 1 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 1 0 0 1 0 1
1 0 0 1 0 0 0 0 0 1 0 0
1 0 1 0 1 0 0 0 1 0 1 0
0 0 0 0 1 0 1 0 0 0 0 0
1 0 1 1 0 1 1 1 0 1 0 1
	7

MAZE problem

- A state of the problem is represented by (r,c) which are respectively the row and column of a position
- Search Algorithm
  - Push the starting state into the queue
  - Loop
    - Pop a state out of the queue, generate neighboring states and push them into the queue if they were not generated so far
  - The algorithm terminates when the target stated is generated



MAZE problem

```
typedef struct Node{
    int row,col;// chỉ số hang và cột của trạng thái hiện tại
    int step; // số bước di chuyển để đi từ trạng thái xuất phát đến trạng thái hiện tại
    struct Node* next; // con trỏ đến phần tử tiếp theo trong hàng đợi
    struct Node* parent;// con trỏ trỏ đến trạng thái sinh ra trạng thái hiện tại
}Node;
```

MAZE problem

- Data structures

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

Node* head, *tail;
Node* listNode[MAX*MAX];// mảng lưu các phần tử được cấp phát động, để giải phóng BN
int szList = 0;// số phần tử của listNode
int A[MAX][MAX];
int n,m;
int r0,c0;
int visited[MAX][MAX];

const int dr[4] = {1,-1,0,0};
const int dc[4] = {0,0,1,-1};
Node* finalNode;
```

MAZE problem

- Memory allocation

```
Node* makeNode(int row, int col, int step, Node* parent){
    Node* node = (Node*)malloc(sizeof(Node));
    node->row = row; node->col = col; node->next = NULL;
    node->parent = parent; node->step = step;
    return node;
}
```

MAZE problem

- Queue implemnetation

```
void initQueue(){
    head = NULL; tail = NULL;
}
int queueEmpty(){
    return head == NULL && tail == NULL;
}
```

```
void pushQueue(Node * node){
    if(queueEmpty()){
        head = node; tail = node;
    }else{
        tail->next = node; tail = node;
    }
}
Node* popQueue(){
    if(queueEmpty()) return NULL;
    Node* node = head;    head = node-
>next;
    if(head == NULL) tail = NULL;
    return node;
}
```

MAZE problem

```
void input(){
    scanf("%d%d%d%d",&n,&m,&r0,&c0);
    for(int i = 1; i <= n; i++){
        for(int j =1; j <= m; j++){
            scanf("%d",&A[i][j]);
        }
    }
}
```

MAZE problem

```
int legal(int row, int col){
    return A[row][col] == 0 && !visited[row][col];
}
int target(int row, int col){
    return row < 1 || row > n || col < 1 || col > m;
}
void finalize(){
    for(int i = 0; i < szList; i++){
        free(listNode[i]);
    }
}
void addList(Node* node){// them phan tu vao listNode de thuc hien giai phong bo nho
    listNode[szList] = node;
    szList++;
}
```

MAZE problem

```
int main(){
    input();
    for(int r = 1; r <= n; r++)
        for(int c = 1; c <= m; c++)
            visited[r][c] = 0;
    initQueue();
    Node* startNode = makeNode(r0,c0,0,NULL);
    addList(startNode);
    pushQueue(startNode);
    visited[r0][c0]= 1;
    while(!queueEmpty()){
        Node* node = popQueue();
        printf("POP (%d,%d)\n",node->row,node->col);
        for(int k = 0; k < 4; k++){
            int nr = node->row + dr[k];
            int nc = node->col + dc[k];
```

MAZE problem

```
            if(legal(nr,nc)){
                visited[nr][nc] = 1;
                Node* newNode = makeNode(nr,nc,node->step + 1, node);
                addList(newNode);
                if(target(nr,nc)){
                    finalNode = newNode; break;
                }else
                    pushQueue(newNode);
            }
        }
        if(finalNode != NULL) break;// found solution
    }
    if(finalNode != NULL) printf(“%d”, finalNode->step);
    else printf(“-1”);
    finalize();
}
```

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

soict.hust.edu.vn/ fb.com/groups/soict