ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

LẬP TRÌNH C CƠ BẢN

Ngăn xếp và hàng đợi

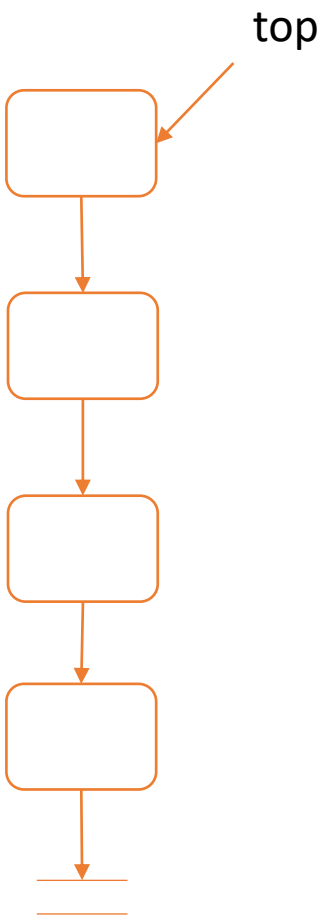
Nội dung

- Cài đặt ngăn xếp và ứng dụng vào bài toán kiểm tra biểu thức ngoặc
- Cài đặt hàng đợi và ứng dụng vào bài toán tìm đường trong mê cung

Ngăn xếp

- Sử dụng danh sách liên kết, Mỗi phần tử bao gồm
  - Data: kiểu ký tự, biểu diễn dấu ngoặc
  - Con trỏ trỏ đến phần tử tiếp theo trong danh sách liên kết
  - Đỉnh (top) của danh sách liên kết được trỏ bởi con trỏ đến đầu của danh sách liên kết

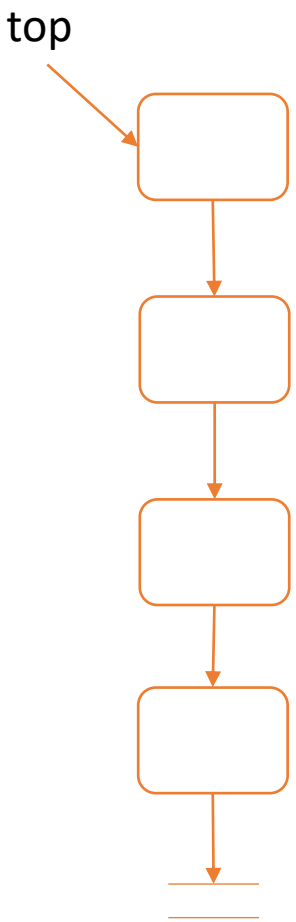
```
typedef struct Node{
    char c;
    struct Node* next;
}Node;
Node* top;
```



Ngăn xếp

- Hàm cấp phát bộ nhớ

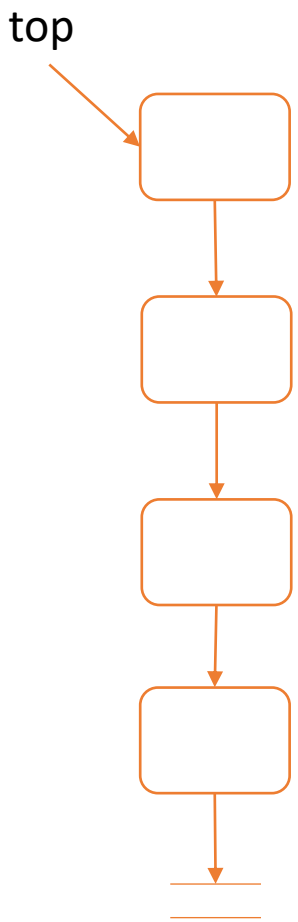
```
Node* makeNode(char x){
    Node* p = (Node*)malloc(sizeof(Node));
    p->c = x; p->next = NULL;
    return p;
}
```



Ngăn xếp

- Khởi tạo ngăn xếp

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

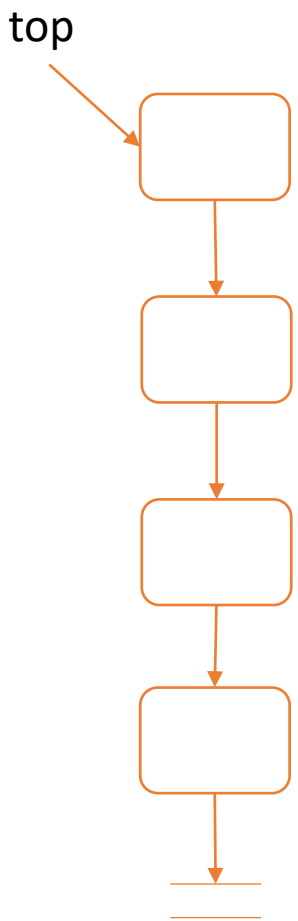


Ngăn xếp

- Thao tác thêm phần tử và lấy ra một phần tử khỏi ngăn xếp

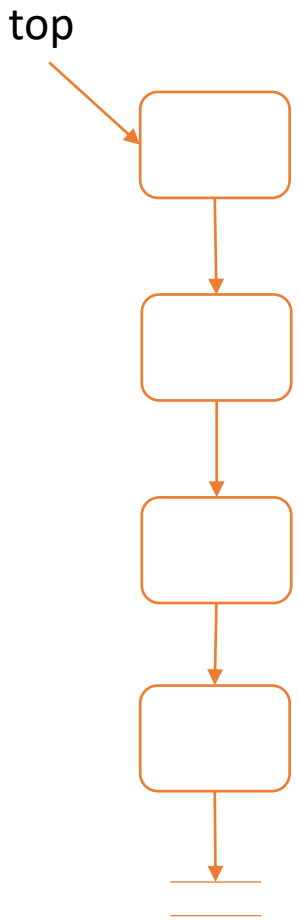
```
void push(char x){
    Node* p = makeNode(x);
    p->next = top; top = p;
}

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



Ngăn xếp

- kiểm tra tính hợp lệ của dãy ngoặc
  - [({})](): hợp lệ
  - ([} {}): không hợp lệ
- Thuật toán:
  - Sử dụng 1 ngăn xếp S
  - Duyệt dãy ngoặc từ trái qua phải
    - Nếu gặp ngoặc mở thì đẩy vào S
    - Nếu gặp ngoặc đóng
      - Nếu S rỗng thì trả về FALSE
      - Nếu S còn phần tử thì lấy 1 ngoặc mở ra khỏi S, kiểm tra đối sánh với ngoặc đóng: nếu ngoặc mở và đóng không tương ứng với nhau thì trả về FALSE
    - Kết thúc việc duyệt, nếu S vẫn còn phần tử thì trả về FALSE, ngược lại thì trả về TRUE



Ngăn xếp

- Kiểm tra xem ngoặc mở và ngoặc đóng có khớp nhau hay không

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

Ngăn xếp

- Kiểm tra tính hợp lệ của biểu thức ngoặc được biểu diễn bởi xâu ký tự s

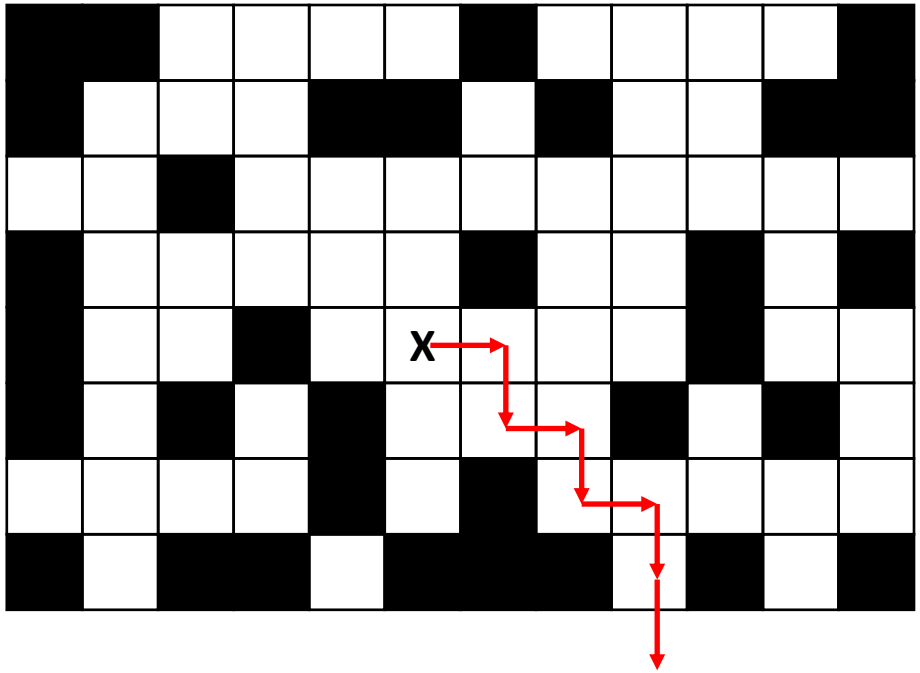
```
int check(char* s){
    initStack();
    for(int i = 0; i < strlen(s); i++){
        if(s[i] == '(' || s[i] == '[' || s[i] == '{'){
            push(s[i]);
        }else{
            if(stackEmpty()) return 0;
            char x = pop();
            if(!match(x,s[i])) return 0;
        }
    }
 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

Hàng đợi

- Hàng đợi là cấu trúc dữ liệu lưu trữ các phần tử một cách tuyến tính
  - Thêm 1 phần tử được thực hiện ở cuối hàng đợi (tail)
  - Lấy ra 1 phần tử được thực hiện ở đầu hàng đợi (head)
- Hàng đợi có thể được ứng dụng để cài đặt các bài toán tìm kiếm theo chiều rộng (thuật toán loang) trên không gian chuyển trạng thái

Bài toán tìm đường trong mê cung (maze)

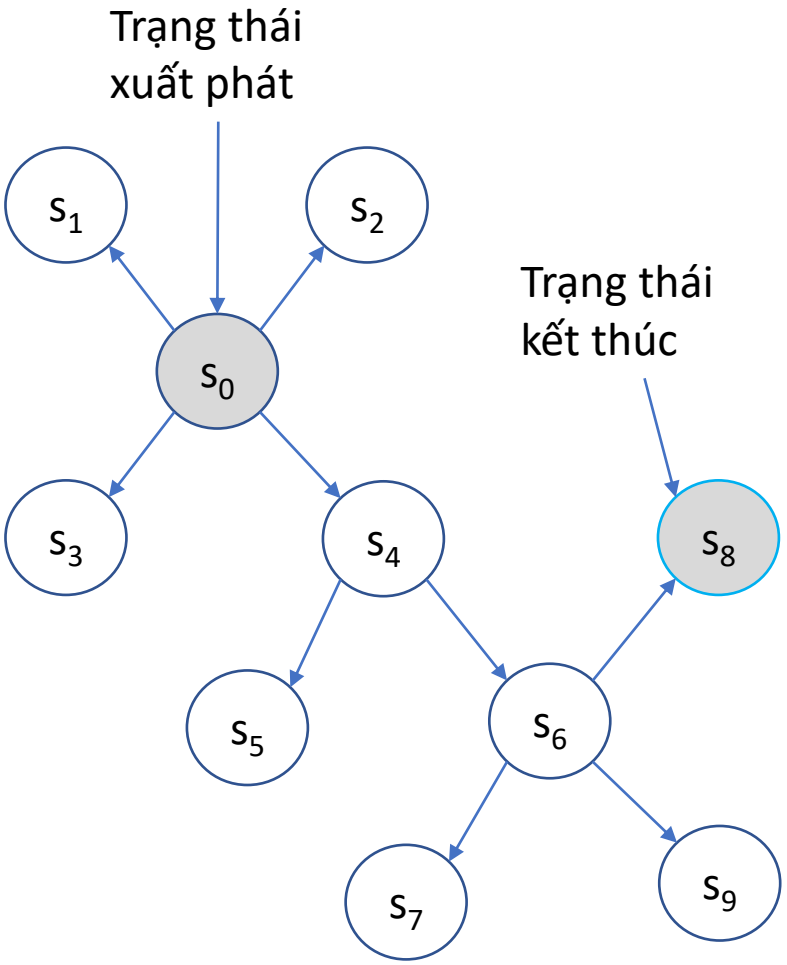
- Một mê cung được biểu diễn bởi ma trận 0-1 a_{NxM} trong đó $a_{i,j}$ = 1 có nghĩa ô (*i,j*) là tường gạch, $a_{i,j}$ = 0 có nghĩa ô (i,j) là ô trống, có thể đi vào.
- Từ một ô trống, ta có thể di chuyển sang một trong 4 ô lân cận (lên trên, xuống dưới, sang trái, sang phải) nếu ô đó cũng là ô trống.
- Tính toán đường đi từ một ô trống (i_0, j_0) thoát ra khỏi mê cũng sử dụng ít bước di chuyển nhất



Thoát ra khỏi mê cung sau 7 bước

Bài toán tìm đường trong mê cung (maze)

- Trạng thái của bài toán được biểu diễn bởi (r,c) tương ứng là chỉ số hàng và cột của một vị trí
- Thuật toán
  - Đưa trạng thái xuất phát vào hàng đợi
  - Lặp
    - Lấy 1 trạng thái ra khỏi hàng đợi, sinh ra các và đưa các trạng thái lân cận của nó vào hàng đợi nếu các trạng thái này chưa được sinh ra trước đó
  - Thuật toán kết thúc khi sinh ra được trạng thái đích



Bài toán tìm đường trong mê cung (maze)

- Định nghĩa cấu trúc dữ liệu trạng thái

```
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;
```

Bài toán tìm đường trong mê cung (maze)

```
#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;
```

Bài toán tìm đường trong mê cung (maze)

```
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;
}
```

Bài toán tìm đường trong mê cung (maze)

```
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;
}
```

Bài toán tìm đường trong mê cung (maze)

```
void input(){
    FILE* f = fopen("maze.txt","r");
    fscanf(f,"%d%d%d%d",&n,&m,&r0,&c0);
    for(int i = 1; i <= n; i++){
        for(int j =1; j <= m; j++){
            fscanf(f,"%d",&A[i][j]);
        }
    }
    fclose(f);
}
```

Bài toán tìm đường trong mê cung (maze)

```
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++;
}
```

Bài toán tìm đường trong mê cung (maze)

```
int main(){
    printf("MAZE\n");
    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];
```

Bài toán tìm đường trong mê cung (maze)

```
            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
    }
    Node* s = finalNode;
    while(s != NULL){
        printf("(%d,%d) ",s->row,s->col);
        s = s->parent;
    }
    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