

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

DANH SÁCH TUYỂN TÍNH

Nội dung

- Định nghĩa danh sách tuyến tính
- Kiểu dữ liệu trừu tượng danh sách tuyến tính
- Mảng
- Danh sách liên kết
- Ngăn xếp
- Hàng đợi

Định nghĩa danh sách tuyến tính

- Các đối tượng được lưu trữ có thứ tự
- Các đối tượng có thể lặp lại giá trị
- Quan hệ trước-sau giữa các đối tượng

Kiểu dữ liệu trừu tượng danh sách tuyến tính

- Lưu trữ các đối tượng với quan hệ trước-sau
- Kí hiệu:
 - L : danh sách
 - x : đối tượng (phần tử của danh sách)
 - p : kiểu vị trí
 - $\text{END}(L)$: hàm trả về vị trí sau vị trí của phần tử cuối cùng của danh sách L

Kiểu dữ liệu trừu tượng danh sách tuyến tính

- Thao tác
 - $\text{Insert}(x, p, L)$: chèn phần tử x vào vị trí p trên danh sách L
 - $\text{Locate}(x, L)$: trả về vị trí của phần tử x trên danh sách L
 - $\text{Retrieve}(p, L)$: trả về phần tử ở vị trí p trong danh sách L
 - $\text{Delete}(p, L)$: loại bỏ phần tử ở vị trí p trên danh sách L
 - $\text{Next}(p, L)$: trả về vị trí tiếp theo của p trên danh sách L
 - $\text{Prev}(p, L)$: trả về vị trí trước của p trên danh sách L
 - $\text{MakeNull}(L)$: thiết lập danh sách L về danh sách rỗng
 - $\text{First}(L)$: trả về vị trí đầu tiên của danh sách L

Kiểu mảng

- Các đối tượng được cấp phát liên tiếp nhau trong bộ nhớ
- Truy cập các đối tượng thông qua chỉ số (kiểu vị trí chính là giá trị nguyên xác định chỉ số)
- Việc thêm hoặc loại bỏ phần tử khỏi mảng cần thiết phải dời mảng

Kiểu mảng

- Tổ chức dữ liệu
 - **int a[100000];** // bộ nhớ lưu trữ
 - **int n;** // số phần tử của mảng (bắt đầu từ chỉ số 0)

```
void insert(int x, int p) {  
    for(int j = n-1; j >= p; j--)  
        a[j+1] = a[j];  
    a[p] = x; n = n + 1;  
}  
  
void delete(int p) {  
    for(int j = p+1; j <= n-1; j++)  
        a[j-1] = a[j];  
    n = n - 1;  
}  
  
int retrieve(int p) {  
    return a[p];  
}
```

Kiểu mảng

- Tổ chức dữ liệu
 - **int a[100000];** // bộ nhớ lưu trữ
 - **int n;** // số phần tử của mảng (bắt đầu từ chỉ số 0)

```
int locate(int x) { // vị trí đầu tiên của x trong danh sách
    for(int j= 0; j <= n-1; j++)
        if(a[j] == x) return j;
    return -1;
}

void makeNull() {
    n = 0;
}

int next(int p) {
    if(0 <= p && p < n-1) return p+1;
    return -1;
}

int prev(int p) {
    if(p > 0 && p <= n-1) return p-1;
    return -1;
}
```


Con trỏ và danh sách liên kết nối đơn

- Con trỏ: địa chỉ của biến
- `int* p`: `p` là con trỏ trỏ đến 1 biến `int`. Giá trị của `p` xác định địa chỉ của biến
- `int a; int* p = &a; // p trỏ vào biến a`
- Kiểu cấu trúc

```
typedef struct TNode{  
    int a;  
    double b;  
    char* s;  
}TNode;
```

- `TNode* q`: `q` là con trỏ trỏ đến 1 biến có kiểu `TNode`

Con trỏ và danh sách liên kết nối đơn

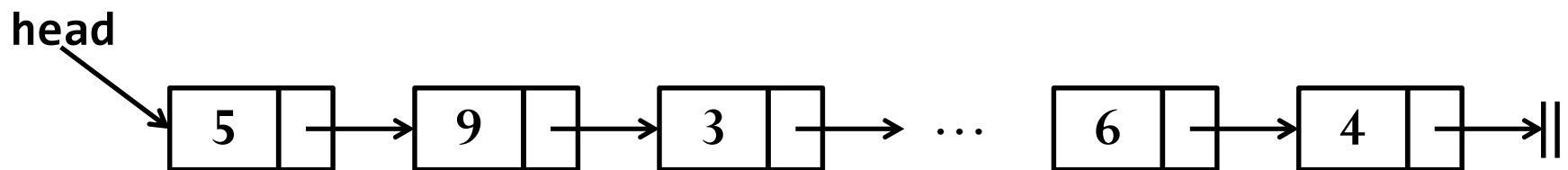
- **q->a**: truy nhập đến thành phần a của kiểu cấu trúc
- **q = (TNode*)malloc(sizeof(TNode))**: cấp phát bộ nhớ cho 1 kiểu TNode và q trỏ vào vùng nhớ được cấp phát

```
int main() {  
    int a = 123;  
    int* p;  
    p = &a;  
    *p = 456;  
    printf("a = %d\n",a);  
}
```

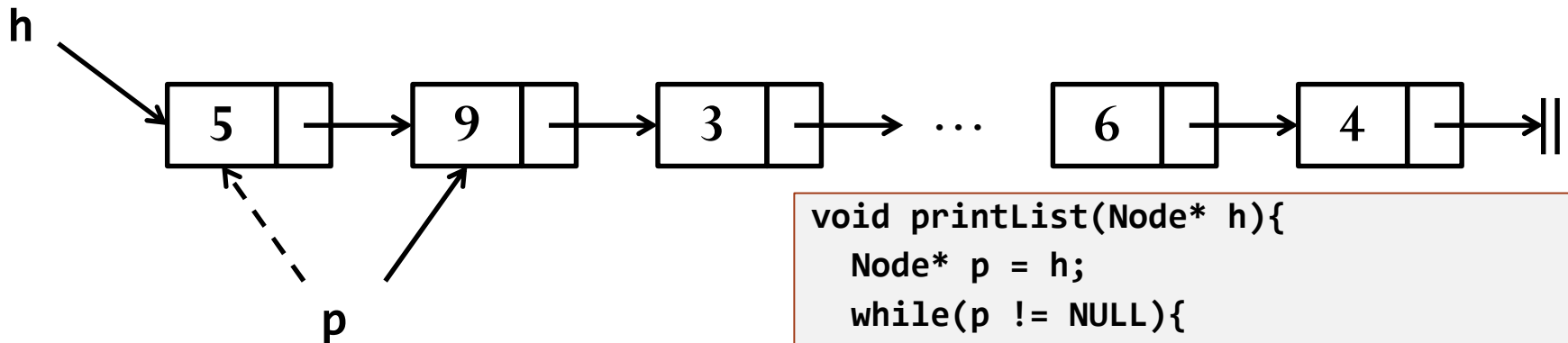
Danh sách liên kết nối đơn

- Mỗi phần tử, ngoài trường dữ liệu, có thêm con trỏ để trỏ đến phần tử tiếp theo

```
struct Node{  
    int value;  
    Node* next;  
};  
Node* head;
```



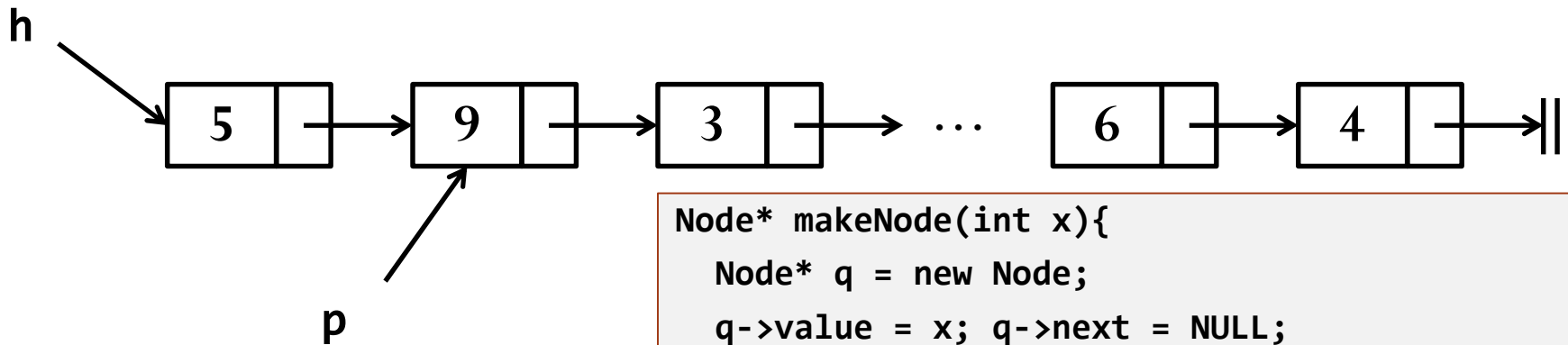
Danh sách liên kết nối đơn



```
void printList(Node* h){
    Node* p = h;
    while(p != NULL){
        printf("%d ", p->value);
        p = p->next;
    }
}

Node* findLast(Node* h){
    Node* p = h;
    while(p != NULL){
        if(p->next == NULL) return p;
        p = p->next;
    }
    return NULL;
}
```

Danh sách liên kết nối đơn



```
Node* makeNode(int x){
```

```
    Node* q = new Node;
```

```
    q->value = x; q->next = NULL;
```

```
    return q;
```

```
}
```

```
Node* insertAfter(Node* h, Node* p, int x){
```

```
    if(p == NULL) return h;
```

```
    Node* q = makeNode(x);
```

```
    if(h == NULL) return q;
```

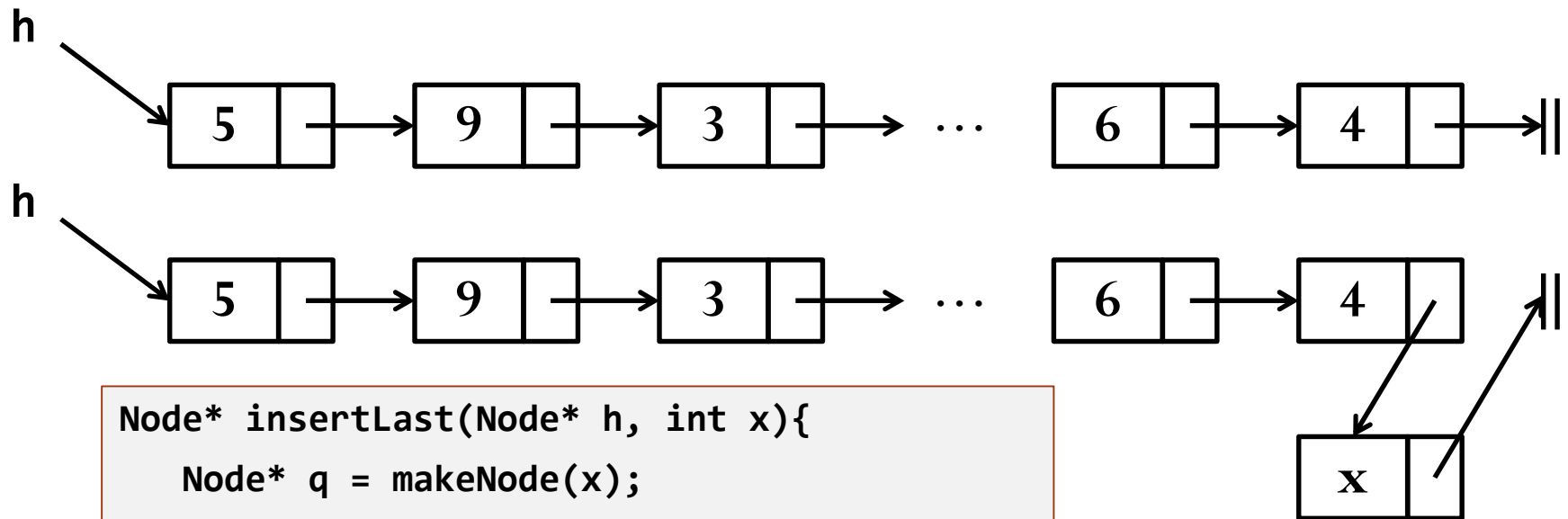
```
    q->next = p->next;
```

```
    p->next = q;
```

```
    return h;
```

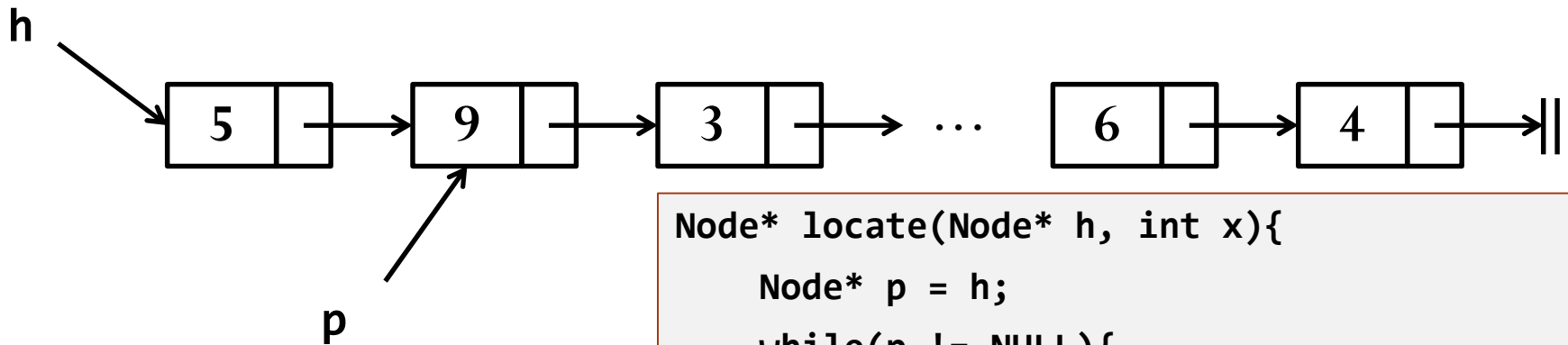
```
}
```

Danh sách liên kết nối đơn



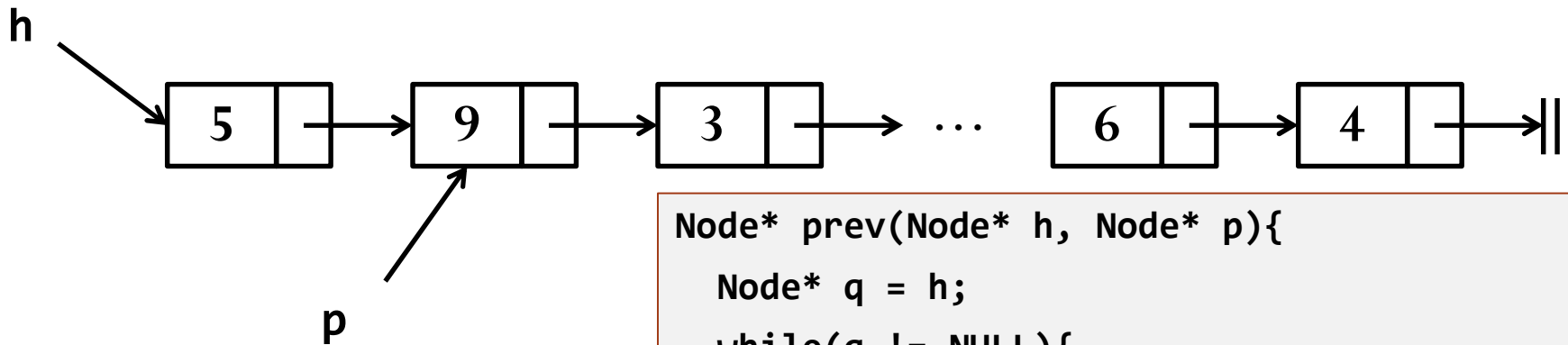
```
Node* insertLast(Node* h, int x){  
    Node* q = makeNode(x);  
    if(h == NULL)  
        return q;  
    Node* p = h;  
    while(p->next != NULL)  
        p = p->next;  
    p->next = q;  
    return h;  
}
```

Danh sách liên kết nối đơn



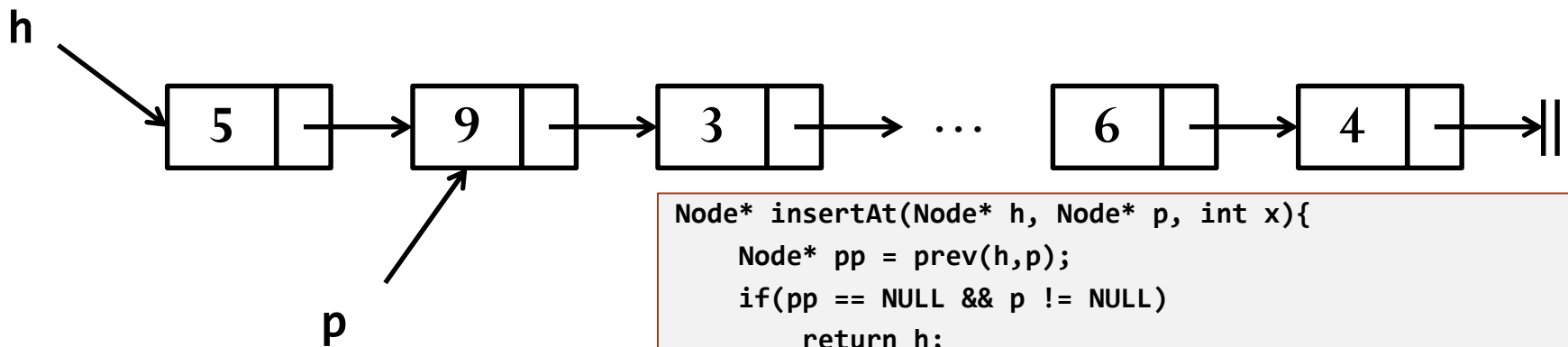
```
Node* locate(Node* h, int x){  
    Node* p = h;  
    while(p != NULL){  
        if(p->value == x) return p;  
        p = p->next;  
    }  
    return NULL;  
}
```

Danh sách liên kết nối đơn



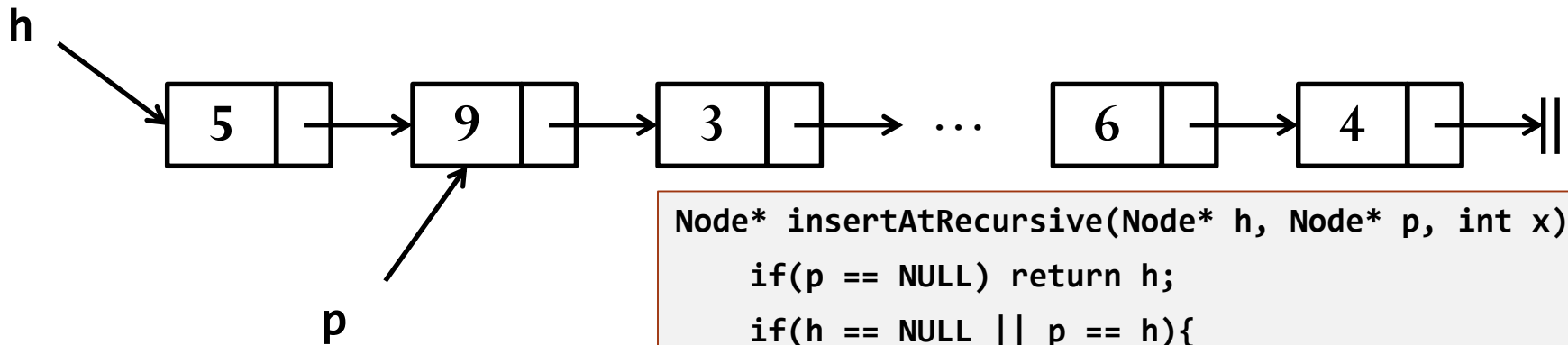
```
Node* prev(Node* h, Node* p){  
    Node* q = h;  
    while(q != NULL){  
        if(q->next == p) return q;  
        q = q->next;  
    }  
    return NULL;  
}
```


Danh sách liên kết nối đơn



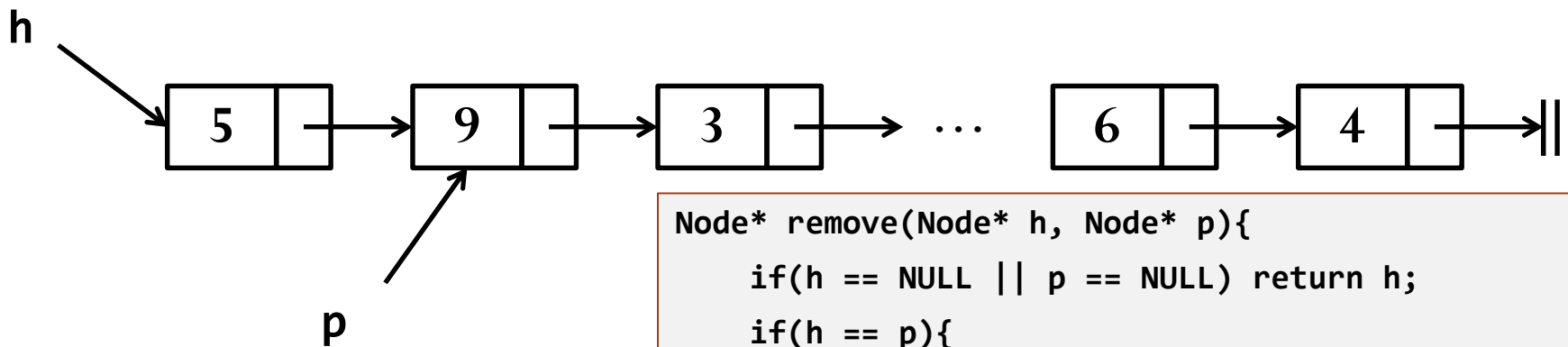
```
Node* insertAt(Node* h, Node* p, int x){
    Node* pp = prev(h,p);
    if(pp == NULL && p != NULL)
        return h;
    Node* q = new Node;
    q->value = x; q->next = NULL;
    if(pp == NULL){
        if(h == NULL){
            return q;
        }
        q->next = h;
        return q;
    }
    q->next = p;
    pp->next = q;
    return h;
}
```

Danh sách liên kết nối đơn



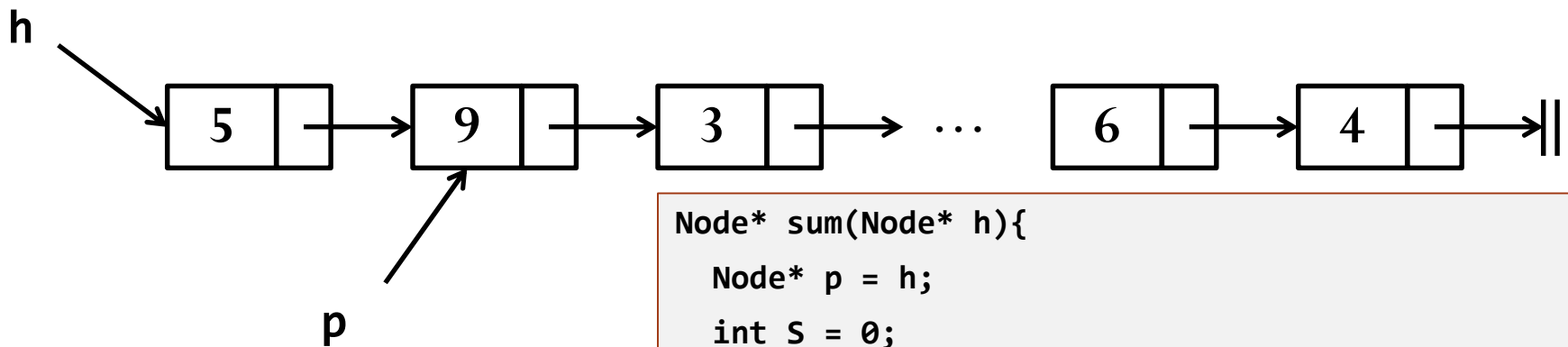
```
Node* insertAtRecursive(Node* h, Node* p, int x){  
    if(p == NULL) return h;  
    if(h == NULL || p == h){  
        return makeNode(x);  
    }else{  
        h->next = insertAtRecursive(h->next,p,x);  
        return h;  
    }  
}
```

Danh sách liên kết nối đơn



```
Node* remove(Node* h, Node* p){  
    if(h == NULL || p == NULL) return h;  
    if(h == p){  
        h = h->next;  
        delete p;  
        return h;  
    }else{  
        h->next = remove(h->next,p);  
        return h;  
    }  
}
```

Danh sách liên kết nối đơn

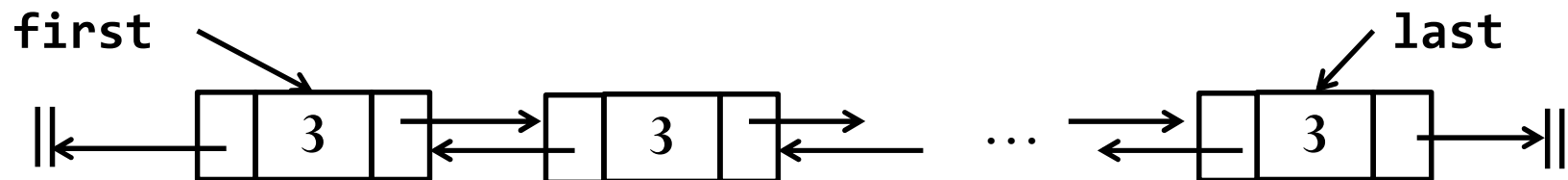
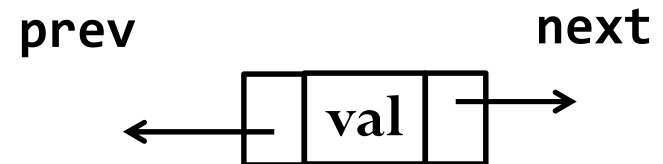


```
Node* sum(Node* h){  
    Node* p = h;  
    int S = 0;  
    while(p != NULL) {  
        S = S + p->value;  
        p = p->next;  
    }  
    return S;  
}  
  
int sumRecursive(Node* h){  
    if(h == NULL) return 0;  
    return h->value + sumRecursive(h->next);  
}
```

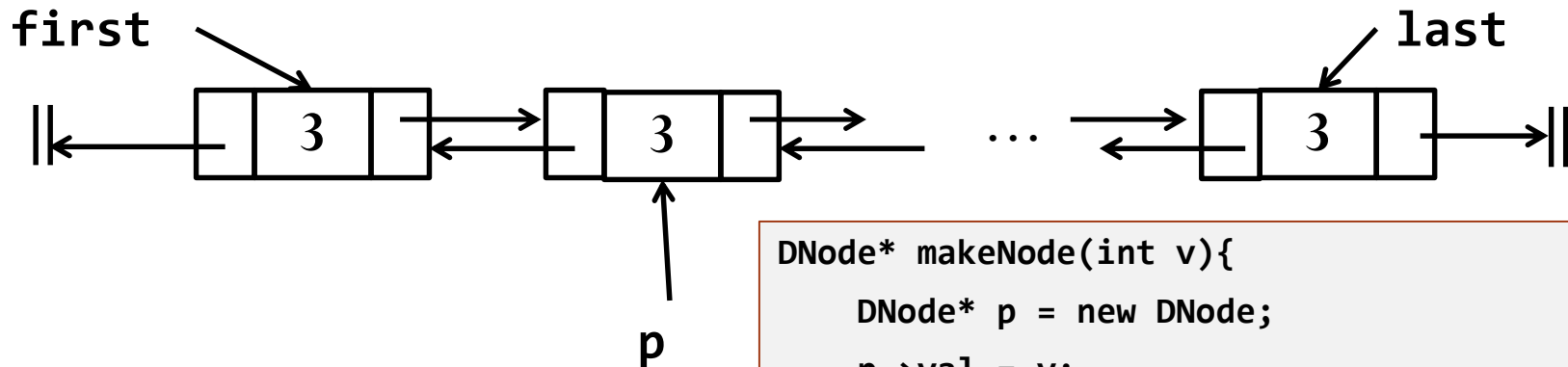
Danh sách liên kết nối đôi

```
struct DNode{  
    int val;  
    DNode* prev;  
    DNode* next;  
};
```

```
DNode* first;  
DNode* last;
```

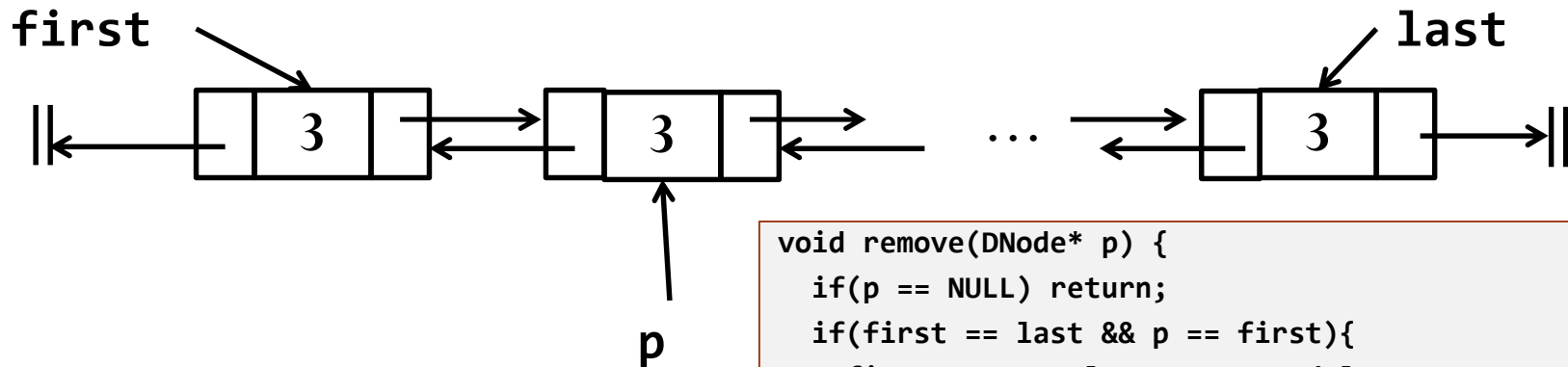


Danh sách liên kết nối đôi



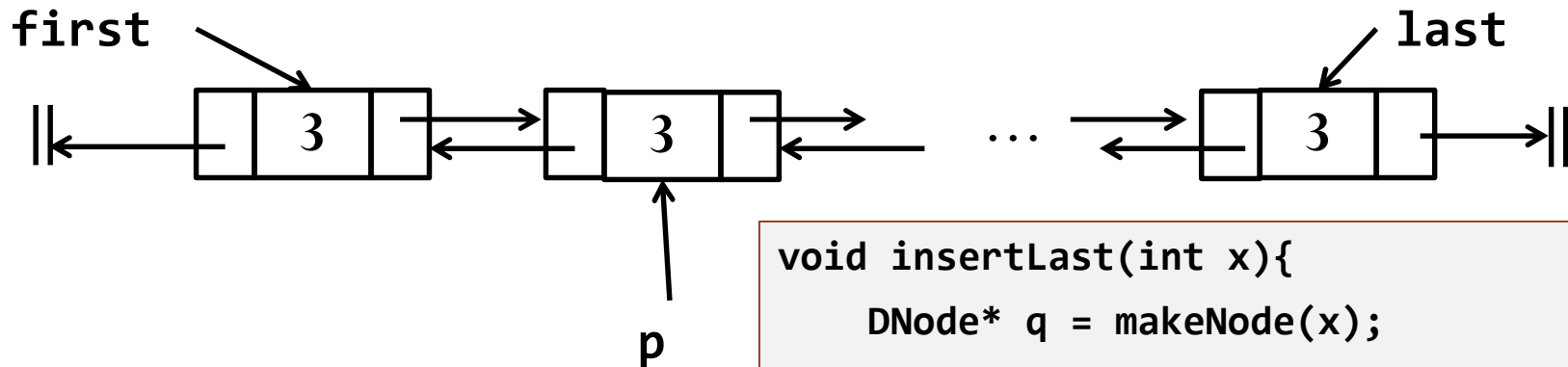
```
DNode* makeNode(int v){  
    DNode* p = new DNode;  
    p->val = v;  
    p->next = NULL;  
    p->prev = NULL;  
    return p;  
}
```

Danh sách liên kết nối đôi



```
void remove(DNode* p) {  
    if(p == NULL) return;  
    if(first == last && p == first){  
        first = NULL; last = NULL; delete p;  
    }  
    if(p == first){  
        first = p->next; first->prev = NULL;  
        delete p; return;  
    }  
    if(p == last){  
        last = p->prev; last->next = NULL;  
        delete p; return;  
    }  
    p->prev->next = p->next;  
    p->next->prev = p->prev;  
    delete p;  
}
```

Danh sách liên kết nối đôi



```
void insertLast(int x){  
    DNode* q = makeNode(x);  
    if(first == NULL && last == NULL){  
        first = q;  
        last = q;  
        return;  
    }  
    q->prev = last;  
    last->next = q;  
    last = q;  
}
```


Thư viện C++

- Kiểu list:

[http://www.cplusplus.com/
reference/list/list/](http://www.cplusplus.com/reference/list/list/)

- Phương thức

- push_front()
- push_back()
- pop_front()
- pop_back()
- size()
- clear()
- erase()

```
#include <list>
#include <stdio.h>
using namespace std;

int main(){
    list<int> L;
    for(int i = 1; i <= 10; i++)
        L.push_back(i);
    list<int>::iterator it;
    for(it = L.begin(); it != L.end();
        it++){
        int x = *it;
        printf("%d ",x);
    }
}
```

Ngăn xếp (Stack)

- Danh sách các đối tượng
- Thao tác thêm mới và loại bỏ được thực hiện ở 1 đầu (đỉnh hay **top**) của danh sách (Last In First Out – LIFO)
- Thao tác
 - Push(x, S): chèn 1 phần tử x vào ngăn xếp
 - Pop(S): lấy ra 1 phần tử khỏi ngăn xếp
 - Top(S): truy cập phần tử ở đỉnh của ngăn xếp
 - Empty(S): trả về true nếu ngăn xếp rỗng

Ngăn xếp

- Ví dụ: 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

- Ví dụ: kiểm tra tính hợp lệ của dãy ngoặc
 - [({})](): hợp lệ
 - ([} {}): không hợp lệ

```
#include <stack>
#include <stdio.h>

using namespace std;

bool match(char a, char b){
    if(a == '(' && b == ')') return true;
    if(a == '{' && b == '}') return true;
    if(a == '[' && b == ']') return true;
    return false;
}
```

Ngăn xếp

- Ví dụ: kiểm tra tính hợp lệ của dãy ngoặc
 - [({})](): hợp lệ
 - ([} {}): không hợp lệ

```
bool solve(char* x, int n){
    stack<char> S;
    for(int i = 0; i <= n-1; i++){
        if(x[i] == '[' || x[i] == '(' || x[i] == '{'){
            S.push(x[i]);
        }else{
            if(S.empty()){
                return false;
            }else{
                char c = S.top(); S.pop();
                if(!match(c,x[i]))
                    return false;
            }
        }
    }
    return S.empty();
}

int main() {
    bool ok = solve("[({})]()",8);
}
```

Hàng đợi (Queue)

- Danh sách các đối tượng với 2 đầu **head** và **tail**
- Thao tác thêm mới được thực hiện ở **tail**
- Thao tác loại bỏ được thực hiện ở **head** (First In First Out – FIFO)
- Thao tác
 - Enqueue(x, Q): chèn 1 phần tử x vào hàng đợi
 - Dequeue(Q): lấy ra 1 phần tử khỏi hàng đợi
 - Empty(Q): trả về true nếu hàng đợi rỗng

Hàng đợi

- Bài toán rót nước
 - Có 1 bể chứa nước (vô hạn)
 - Có 2 cốc với dung tích là a , b (nguyên dương) lít
 - Tìm cách đo đúng c (nguyên dương) lít nước

Hàng đợi

- Bài toán rót nước: $a = 6$, $b = 8$, $c = 4$

Bước	Thực hiện	Trạng thái
1	Đổ đầy nước vào cốc 1	(6,0)
2	Đổ hết nước từ cốc 1 sang cốc 2	(0,6)
3	Đổ đầy nước vào cốc 1	(6,6)
4	Đổ nước từ cốc 1 vào đầy cốc 2	(4,8)

Hàng đợi

- Bài toán rót nước: $a = 4$, $b = 19$, $c = 21$

Bước	Thực hiện	Trạng thái
1	Đổ đầy nước vào cốc 1	(4,0)
2	Đổ hết nước từ cốc 1 sang cốc 2	(0,4)
3	Đổ đầy nước vào cốc 1	(4,4)
4	Đổ hết nước từ cốc 1 sang cốc 2	(0,8)
5	Đổ đầy nước vào cốc 1	(4,8)
6	Đổ hết nước từ cốc 1 sang cốc 2	(0,12)
7	Đổ đầy nước vào cốc 1	(4,12)
8	Đổ hết nước từ cốc 1 sang cốc 2	(0,16)
9	Đổ đầy nước vào cốc 1	(4,16)
10	Đổ nước từ cốc 1 vào đầy cốc 2	(1,19)
11	Đổ hết nước ở cốc 2 ra ngoài	(1,0)

Hàng đợi

- Bài toán rót nước: $a = 4$, $b = 19$, $c = 21$ (tiếp)

Bước	Thực hiện	Trạng thái
12	Đổ hết nước từ cốc 1 sang cốc 2	(0,1)
13	Đổ đầy nước vào cốc 1	(4,1)
14	Đổ hết nước từ cốc 1 sang cốc 2	(0,5)
15	Đổ đầy nước vào cốc 1	(4,5)
16	Đổ hết nước từ cốc 1 sang cốc 2	(0,9)
17	Đổ đầy nước vào cốc 1	(4,9)
18	Đổ hết nước từ cốc 1 sang cốc 2	(0,13)
19	Đổ đầy nước vào cốc 1	(4,13)
20	Đổ hết nước từ cốc 1 sang cốc 2	(0,17)
21	Đổ đầy nước vào cốc 1	(4,17)

Hàng đợi

- Thiết kế thuật toán và cấu trúc dữ liệu
 - Trạng thái là bộ (x, y) : lượng nước có trong cốc 1 và 2
 - Trạng thái ban đầu $(0, 0)$
 - Trạng thái kết thúc: $x = c$ hoặc $y = c$ hoặc $x + y = c$
 - Chuyển trạng thái
 - (1) Đổ đầy nước từ bể vào cốc 1: (a, y)
 - (2) Đổ đầy nước từ bể vào cốc 2: (x, b)
 - (3) Đổ hết nước từ cốc 1 ra ngoài: $(0, y)$
 - (4) Đổ hết nước từ cốc 2 ra ngoài: $(x, 0)$
 - (5) Đổ nước từ cốc 1 vào đầy cốc 2: $(x + y - b, b)$, nếu $x + y \geq b$
 - (6) Đổ hết nước từ cốc 1 sang cốc 2: $(0, x + y)$, nếu $x + y \leq b$
 - (7) Đổ nước từ cốc 2 vào đầy cốc 1: $(a, x + y - a)$, nếu $x + y \geq a$
 - (8) Đổ hết nước từ cốc 2 sang cốc 1: $(x + y, 0)$, nếu $x + y \leq a$

Hàng đợi

- Đưa $(0,0)$ vào hàng đợi

$(0,0)$									
---------	--	--	--	--	--	--	--	--	--

- Lấy $(0,0)$ ra và đưa $(6,0)$, $(0,8)$ vào hàng đợi

$(0,0)$	$(6,0)$	$(0,8)$							
---------	---------	---------	--	--	--	--	--	--	--

- Lấy $(6,0)$ ra và đưa $(0,6)$ và $(6,8)$ vào hàng đợi

$(0,0)$	$(6,0)$	$(0,8)$	$(0,6)$	$(6,8)$					
---------	---------	---------	---------	---------	--	--	--	--	--

- Lấy $(0,8)$ ra và đưa $(6,2)$ vào hàng đợi

$(0,0)$	$(6,0)$	$(0,8)$	$(0,6)$	$(6,8)$	$(6,2)$				
---------	---------	---------	---------	---------	---------	--	--	--	--

Hàng đợi

- Đưa $(0,6)$ ra và đưa $(6,6)$ vào hàng đợi

$(0,0)$	$(6,0)$	$(0,8)$	$(0,6)$	$(6,8)$	$(6,2)$	$(6,6)$			
---------	---------	---------	---------	---------	---------	---------	--	--	--

- Lấy $(6,8)$ ra và không đưa trạng thái mới nào vào hàng đợi

$(0,0)$	$(6,0)$	$(0,8)$	$(0,6)$	$(6,8)$	$(6,2)$	$(6,6)$			
---------	---------	---------	---------	---------	---------	---------	--	--	--

- Lấy $(6,2)$ ra và đưa $(0,2)$ vào hàng đợi

$(0,0)$	$(6,0)$	$(0,8)$	$(0,6)$	$(6,8)$	$(6,2)$	$(6,6)$	$(0,2)$		
---------	---------	---------	---------	---------	---------	---------	---------	--	--

- Lấy $(6,6)$ ra và đưa $(4,8)$ vào hàng đợi

$(0,0)$	$(6,0)$	$(0,8)$	$(0,6)$	$(6,8)$	$(6,2)$	$(6,6)$	$(0,2)$	$(4,8)$	
---------	---------	---------	---------	---------	---------	---------	---------	---------	--

Hàng đợi

- Thiết kế thuật toán và cấu trúc dữ liệu
 - Hàng đợi Q để ghi nhận các trạng thái được sinh ra
 - Mảng 2 chiều để đánh dấu trạng thái đã được xét đến
 - $visited[x][y] = true$, nếu trạng thái (x, y) đã được sinh ra
 - Ngăn xếp để in ra chuỗi các hành động để đạt được kết quả
 - Danh sách L để lưu các con trỏ trỏ đến các vùng nhớ được cấp phát động (phục vụ cho việc thu hồi bộ nhớ khi kết thúc chương trình)

Hàng đợi

- Khai báo cấu trúc dữ liệu

```
#include <stdio.h>
#include <stdlib.h>
#include <queue>
#include <stack>
#include <list>
using namespace std;

struct State{
    int x;
    int y;
    char* msg;// action to generate current state
    State* p;// pointer to the state generating current state
};

bool visited[10000][10000];
queue<State*> Q;
list<State*> L;
State* target;
int a,b,c;
```

Hàng đợi

- Các hàm khởi tạo mảng đánh dấu, đánh dấu trạng thái, kiểm tra trạng thái đích, giải phóng bộ nhớ

```
void initVisited(){
    for(int x = 0; x < 10000; x++)
        for(int y = 0; y < 10000; y++) visited[x][y] = false;
}

bool reachTarget(State* S){
    return S->x == c || S->y == c || S->x + S->y == c;
}

void markVisit(State* S){
    visited[S->x][S->y] = true;
}

void freeMemory(){
    list<State*>::iterator it;
    for(it = L.begin(); it != L.end(); it++){
        delete *it;
    }
}
```


Hàng đợi

- Hàm sinh trạng thái bởi hành động (3): đổ hết nước từ cốc 1 ra ngoài

```
bool genMove1Out(State* S){
    if(visited[0][S->y]) return false;
    State* newS = new State;
    newS->x = 0;
    newS->y = S->y;
    newS->msg = "Do het nuoc o coc 1 ra ngoai";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```

Hàng đợi

- Hàm sinh trạng thái bởi hành động (4): đổ hết nước từ cốc 2 ra ngoài

```
bool genMove2Out(State* S){
    if(visited[S->x][0]) return false;
    State* newS = new State;
    newS->x = S->x;
    newS->y = 0;
    newS->msg = "Do het nuoc o coc 2 ra ngoai";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```

Hàng đợi

- Hàm sinh trạng thái bởi hành động (5): đổ nước từ cốc 1 vào đầy cốc 2

```
bool genMove1Full2(State* S){
    if(S->x+S->y < b) return false;
    if(visited[S->x + S->y - b][b]) return false;
    State* newS = new State;
    newS->x = S->x + S->y - b;
    newS->y = b;
    newS->msg = "Do nuoc tu coc 1 vao day coc 2";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```

Hàng đợi

- Hàm sinh trạng thái bởi hành động (7): đổ nước từ cốc 2 vào đầy cốc 1

```
bool genMove2Full1(State* S){
    if(S->x+S->y < a) return false;
    if(visited[a][S->x + S->y - a]) return false;
    State* newS = new State;
    newS->x = a;
    newS->y = S->x + S->y - a;
    newS->msg = "Do nuoc tu coc 2 vao day coc 1";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```

Hàng đợi

- Hàm sinh trạng thái bởi hành động (6): đổ hết nước từ cốc 1 sang cốc 2

```
bool genMoveAll112(State* S){
    if(S->x + S->y > b) return false;
    if(visited[0][S->x + S->y]) return false;
    State* newS = new State;
    newS->x = 0;
    newS->y = S->x + S->y;
    newS->msg = "Do het nuoc tu coc 1 sang coc 2";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```

Hàng đợi

- Hàm sinh trạng thái bởi hành động (8): đổ hết nước từ cốc 2 sang cốc 1

```
bool genMoveAll121(State* S){
    if(S->x + S->y > a) return false;
    if(visited[S->x + S->y][0]) return false;
    State* newS = new State;
    newS->x = S->x + S->y;
    newS->y = 0;
    newS->msg = "Do het nuoc tu coc 2 sang coc 1";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```

Hàng đợi

- Hàm sinh trạng thái bởi hành động (1): đổ đầy nước từ bể vào cốc 1

```
bool genMoveFill1(State* S){
    if(visited[a][S->y]) return false;
    State* newS = new State;
    newS->x = a;
    newS->y = S->y;
    newS->msg = "Do day nuoc vao coc 1";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```

Hàng đợi

- Hàm sinh trạng thái bởi hành động (2): đổ đầy nước từ bể vào cốc 2

```
bool genMoveFill2(State* S){
    if(visited[S->x][b]) return false;
    State* newS = new State;
    newS->x = S->x;
    newS->y = b;
    newS->msg = "Do day nuoc vao coc 2";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```


Hàng đợi

- Hàm in chuỗi hành động để thu được kết quả

```
void print(State* target){
    printf("-----RESULT-----\n");
    if(target == NULL) printf("Khong co loi giai!!!!!!");
    State* currentS = target;
    stack<State*> actions;
    while(currentS != NULL){
        actions.push(currentS);
        currentS = currentS->p;
    }
    while(actions.size() > 0){
        currentS = actions.top();
        actions.pop();
        printf("%s, (%d,%d)\n",currentS->msg,currentS->x,
            currentS->y);
    }
}
```

Hàng đợi

- Khởi tạo, sinh trạng thái ban đầu và đưa vào hàng đợi
- Tại mỗi bước, lấy 1 trạng thái ra khỏi hàng đợi, sinh trạng thái mới và đưa vào hàng đợi

```
void solve(){
    initVisited();
    // sinh ra trang thai ban dau (0,0) va dua vao Q
    State* S = new State;
    S->x = 0; S->y = 0; S->p = NULL;
    Q.push(S); markVisit(S);
    while(!Q.empty()){
        State* S = Q.front(); Q.pop();
        if(genMove1Out(S)) break; // (0,y)
        if(genMove2Out(S)) break; // (x,0)
        if(genMove1Full2(S)) break; // (x+y-b,b) if(x+y >= b)
        if(genMoveAll12(S)) break; // (0,x+y) if(x+y <= b)
        if(genMove2Full1(S)) break; // (a,x+y-a) if (x+y >= a)
        if(genMoveAll21(S)) break; // (x+y,0) if(x+y <= a)
        if(genMoveFill1(S)) break; // (a,y)
        if(genMoveFill2(S)) break; // (x,b)
    }
}
```

Hàng đợi

- Hàm main
 - Thử nghiệm với bộ dữ liệu: $a = 4, b = 7, c = 9$

```
int main(){  
    a = 4;  
    b = 7;  
    c = 9;  
    target = NULL;  
    solve();  
    print(target);  
    freeMemory();  
}
```