CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Ngăn xếp, hàng đợi, danh sách liên kết

ThS. Nguyễn Văn Tiến

Nội dung

1. Danh sách liên kết đơn (Single linked list)
2. Danh sách liên kết kép (Double linked list)
3. Ngăn xếp (Stack)
4. Hàng đợi (Queue)

Danh sách liên kết đơn (Single linked list)

Vấn đề

Để lưu trữ tập hợp các phần tử có chung kiểu dữ liệu thường sử dụng mảng.

Mảng có ưu điểm xử lý dữ liệu nhanh do truy cập trực tiếp vào các phần tử của mảng (do tính chất cấp phát các ô nhớ liên tục)

Số lượng phần tử của mảng được cấp phát không thay đổi được trong quá trình thực thi dẫn đến thiếu bộ nhớ hoặc thừa.

Danh sách liên kết được sử dụng để khắc phục các nhược điểm này

Danh sách liên kết đơn (Single linked list)

Định nghĩa

Tập hợp các node thông tin được tổ chức rời rạc trong bộ nhớ. Trong đó, mỗi node gồm hai thành phần:

- Thành phần dữ liệu (data): dùng để lưu trữ thông tin của node.
- Thành phần con trỏ (pointer): dùng để liên kết với node dữ liệu tiếp theo

Danh sách liên kết đơn (Single linked list)

Định nghĩa

```
struct Item {
<Kiểu 1> <Thành viên 1>;
<Kiểu 2> <Thành viên 2>;
<Kiểu N> <Thành viên N>;
};
```

```
struct node {
Item infor; //Thông tin của node;
struct node *next; //con trỏ
} *Start;
```

So sánh mảng và danh sách liên kết

Nội dung	Mảng	Danh sách liên kết
Kích thước	Kích thước cố định	*Kích thước thay đổi trong quá trình thêm/ xóa phần tử*
	Cần chỉ rõ kích thước trong khi khai báo	*Kích thước tối đa phụ thuộc vào bộ nhớ*
Cấp phát bộ nhớ	Tĩnh: Bộ nhớ được cấp phát tĩnh	*Động: Bộ nhớ được cấp phát trong quá trình chạy*
Thứ tự & sắp xếp	Được lưu trữ trên một dãy ô nhớ liên tục	*Được lưu trữ trên các ô nhớ ngẫu nhiên*
Truy cập	*Truy cập tới phần tử ngẫu nhiên trực tiếp bằng cách sử dụng chỉ số mảng: O(1)*	Truy cập tới phần tử ngẫu nhiên cần phải duyệt từ đầu/cuối đến phần tử đó: O(n)
Tìm kiếm	*Tìm kiếm tuyến tính hoặc tìm kiếm nhị phân*	Chỉ có thể tìm kiếm tuyến tính

Danh sách liên kết đơn (Single linked list)

Các thao tác trên danh sách liên kết đơn

Sửa đổi nội dung của node

Duyệt danh sách liên kết

Tìm kiếm node trong danh sách liên kết

Sắp xếp các node trong danh sách liên kết

Đảo ngược danh sách liên kết

Danh sách liên kết kép (Double linked list)

Vấn đề

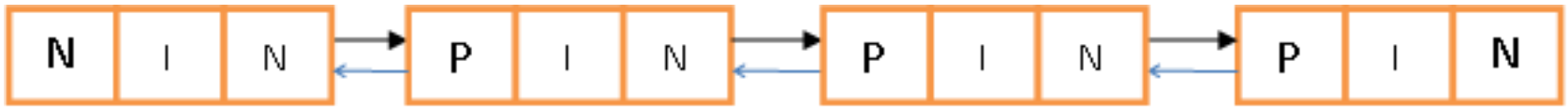
Hạn chế lớn nhất đối với danh sách liên kết đơn là vấn đề tìm kiếm. Phương pháp tìm kiếm duy nhất ta có thể cài đặt được trên danh sách liên kết đơn là tìm kiếm tuyến tính.

Danh sách liên kết kép được sử dụng để khắc phục nhược điểm này

Danh sách liên kết kép (Double linked list)

Định nghĩa

```
struct node {
    Item   Infor; //thành phần dữ liệu của node
    struct  node *next; //thành phần con trỏ trước
    struct  node *prev; //thành phần con trỏ sau
}*start; //đây là danh sách liên kết kép
```



Danh sách liên kết kép (Double linked list)

Các thao tác trên danh sách liên kết kép

- Tạo node rời rạc có giá trị value cho danh sách liên kết kép.
- Thêm node vào đầu danh sách liên kết kép.
- Thêm node vào cuối danh sách liên kết kép.
- Thêm node vào vị trí pos trong danh sách liên kết kép.
- Loại node tại vị trí pos của danh sách liên kết kép.
- Sắp xếp nội dung các node của danh sách liên kết kép.
- Tìm kiếm node có giá trị value trên danh sách liên kết kép.
- Duyệt trái danh sách liên kết kép.
- Duyệt phải danh sách liên kết kép.
- Đảo ngược các node trong danh sách liên kết kép.

Ngăn xếp (Stack)

Định nghĩa

Ngăn xếp (stack) là một danh sách nhưng các phép toán chỉ được thực hiện ở một đỉnh của danh sách.

Tập hợp các node thông tin được tổ chức liên tục hoặc rời rạc nhau trong bộ nhớ và thực hiện theo cơ chế vào sau ra trước LIFO (Last – In – First – Out)

Ngăn xếp (Stack)

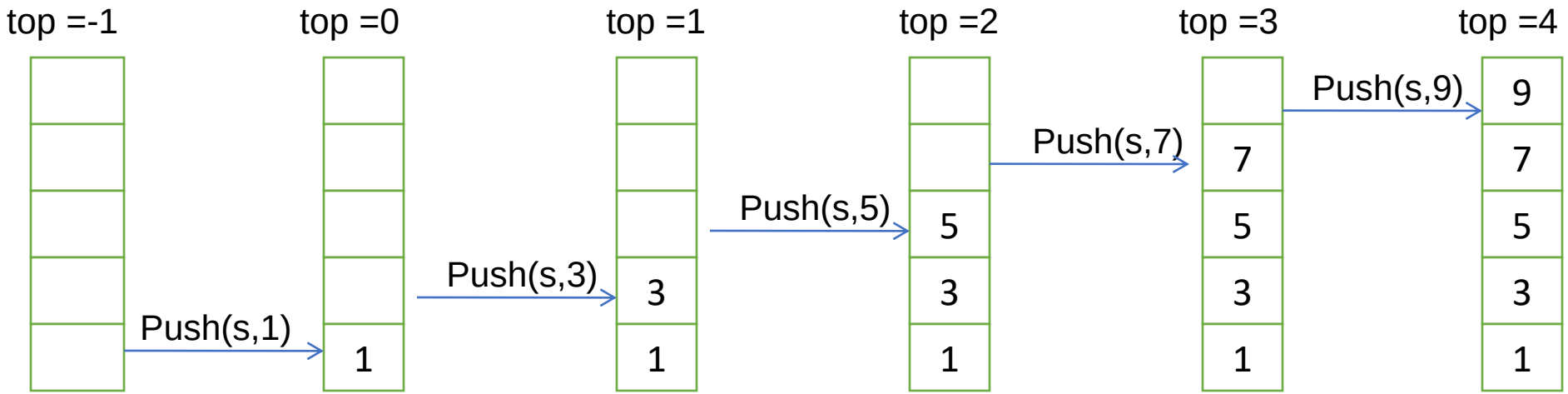
Phương pháp biểu diễn ngăn xếp

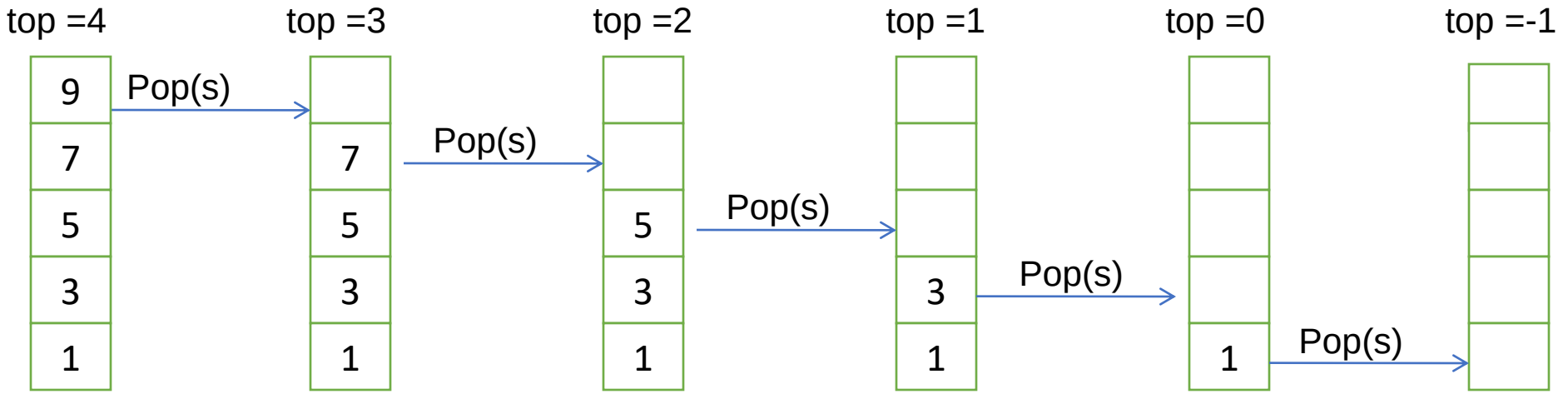
Biểu diễn liên tục: các phần tử dữ liệu của ngăn xếp được lưu trữ liên tục nhau trong bộ nhớ (Mảng).

Ví dụ. Biểu diễn ngăn xếp dựa vào mảng.

```
typedef struct {
int     top; //Đỉnh đầu của stack nơi diễn ra mọi thao tác
int     node[MAX]; //Dữ liệu lưu trữ trong stack gồm MAX phần tử
} Stack;
```

Ngăn xếp (Stack)





Ngăn xếp (Stack)

Phương pháp biểu diễn ngăn xếp

Biểu diễn rời rạc: các phần tử dữ liệu của ngăn xếp được lưu trữ rời rạc nhau trong bộ nhớ (Danh sách liên kết).

Ví dụ. Biểu diễn ngăn xếp dựa vào danh sách liên kết.

```
struct StackNode {
        int data;
        StackNode* next;
};
```

Ngăn xếp (Stack)

Các thao tác trên ngăn xếp

Kiểm tra tính rỗng của stack (Empty(stack s)).

Kiểm tra tính đầy của ngăn xếp (Full(stack s)).

Đưa dữ liệu vào ngăn xếp (Push(stack s, item x)). Chỉ được thực hiện khi và chỉ khi ngăn xếp chưa tràn.

Đưa dữ liệu vào ngăn xếp (Pop(stack s). Chỉ được thực hiện khi và chỉ khi ngăn xếp không rỗng.

Ngăn xếp (Stack)

Ưu điểm

Gọi n là số phần tử của ngăn xếp

Không gian sử dụng là O(n)

Mỗi thao tác thực hiện trong thời gian O(1)

Hạn chế khi cài đặt bằng mảng

Kích thước tối đa của ngăn xếp phải được chỉ định trước và không thể thay đổi

Cố push phần tử mới vào ngăn xếp đã đầy sẽ sinh ngoại lệ do cài đặt (implementation-specific exception)

Ngăn xếp (Stack)

Sử dụng stack trong C++ STL

Khai báo

stack <type> s;

Các phương thức cơ bản

empty() – Kiểm tra rỗng – Time Complexity : O(1)

size() – Lấy số lượng phần tử – Time Complexity : O(1)

top() – Lấy giá trị phần tử ở đỉnh – Time Complexity : O(1)

push(g) – Thêm phần tử – Time Complexity : O(1)

pop() – Loại phần tử khỏi đỉnh – Time Complexity : O(1)

Ngăn xếp (Stack)

Ứng dụng của ngăn xếp

Khử đệ quy

Kiểm soát dấu ngoặc

Biểu diễn và chuyển đổi giữa các cách biểu diễn phép toán

Duyệt cây, duyệt đồ thị…

Ví dụ kiểm tra dấu ngoặc cân xứng

Mỗi ngoặc mở "(", "[", "{" phải được cặp với một ngoặc đóng ")", "]", "}" tương ứng.

Ví dụ

cân xứng: ( )(( )){([( )])}

không cân xứng: ((( )(( )){([( )])}

không cân xứng:)(( )){([( )])}

không cân xứng: ({[ ])}

không cân xứng: (

Ví dụ chuyển đổi trung tố hậu tố

Biểu thức Hậu tố:

a + b ↔ a b +

a - b ↔ a b -

a * b ↔ a b *

a / b ↔ a b /

(P) ↔ P

Ví dụ chuyển đổi trung tố hậu tố

$$(a+b*c)-(a/b+c)=$$
$$=(a+bc*)-(ab/+c)$$
$$=(abc*+)-(ab/c+)$$
$$=abc*+-ab/c+$$
$$=abc*+ab/c+-$$

Thuật toán infix-to-postfix (P):

Bước 1 (Khởi tạo)**:**

stack = ∅; *//stack dùng để lưu trữ các phép toán*

Out = ∅;*// out dùng lưu trữ biểu thức hậu tố*

Bước 2 (Lặp) :

For each x∈P do *// duyệt từ trái qua phải biểu thức trung tố P*

2.1. Nếu x =‘ (‘ : Push(stack, x);

2.2. Nếu x là toán hạng: x⇒Out;

2.3. Nếu x ∈ { +, -, *, / ,^}

y = get(stack); *//lấy phép toán ở đầu ngăn xếp*

a) Nếu priority(x)≥priority(y): Push(stack, x);

b) Nếu priority(x)<priority(y):

y = Po(stack); y⇒Out; Push(stack, x);

c) Nếu stack = ∅: Push(stack, x);

2.4. Nếu x =‘)’:

y = Pop(stack);

While (y!=‘(’) do

y⇒Out; y = Pop(stack);

EndWhile;

EndFor;

Bước 3(Hoàn chỉnh biểu thức hậu tố)*:*

While (stack ≠ ∅) do

y = Pop(stack); y⇒Out;

EndWhile;

Bước 4(Trả lại kết quả)*:*

Return(Out).

x∈P	Bước	Stack	Out
x =‘(‘	2.1	(	∅
x =a	2.2	(	a
x =+	2.3.a	(+	a
x =b	2.2	(+	a b
x =*	2.3.a	(+ *	a b
x = c	2.2	(+ *	a b c
x =‘)‘	2.3	∅	a b c * +
x =-	2.2.c	-	a b c * +
x =‘(‘	2.1	- (	a b c * +
x =a	2.2	- (	a b c * + a
x =/	2.2.a	- (/	a b c * + a
x =b	2.2	- (/	a b c * + a b
x =+	2.3.b	- (+	a b c * + a b/
x =c	2.2	- (+	a b c * + a b/c
x =‘)‘	2.4	∅	a b c * + a b/c + -
P = a b c * + a b/c + -			

Thuật toán tính toán giá trị biểu thức hậu tố:

Bước 1 (Khởi tạo)**:**

stack = ∅;

Bước 2 (Lặp) :

For each x∈P do

2.1. Nếu x là toán hạng:

Push(stack, x);

*2.2. Nếu x ∈ { +, -, *, / }*

a) TH2 = Pop(stack, x);

b) TH1 = Pop(stack, x);

c) KQ = TH1 ⊗ TH2;

d) Push (stack, KQ);

EndFor;

Bước 4(Trả lại kết quả)*:*

Return(Pop(stack)).

Ví dụ: P = 6 2 4 * + 6 2 / 4 + -

Bài toán phần tử bên phải đầu tiên lớn hơn

MÔ TẢ BÀI TOÁN

Cho dãy số A[] gồm N phần tử. Với mỗi A[i], bạn cần tìm phần tử bên phải đầu tiên lớn hơn nó. Nếu không tồn tại, in ra -1.

VÍ DỤ:

Dãy A[] = {4, 5, 2, 25} KẾT QUẢ: R[] = {5, 25, 25, -1}

GIẢI THUẬT:

Cách làm thông thường: độ phức tạp O(n2)

Sử dụng Stack: độ phức tạp O(n)

Bài toán phần tử bên phải đầu tiên lớn hơn

```
void xuly(int a[], int n){
    stack<int> st;
    int R[n], i;
    for(i=n-1;i>=0;i--){
        while(!st.empty() && a[i] >= st.top()) st.pop();
        if(st.empty()) R[i] = -1;
        else R[i] = st.top();
        st.push(a[i]);
    }
    for(i=0;i<n;i++) cout << R[i] << " ";
    cout << endl;
}
```

Các bài toán khác

1. Chuyển đổi giữa các dạng biểu thức:

Trung tố

Hậu tố

Tiền tố

2. Tính độ dài dãy ngoặc đúng dài nhất
3. Đếm số dấu ngoặc cần đổi chiều
4. Tính diện tích hình chữ nhật lớn nhất

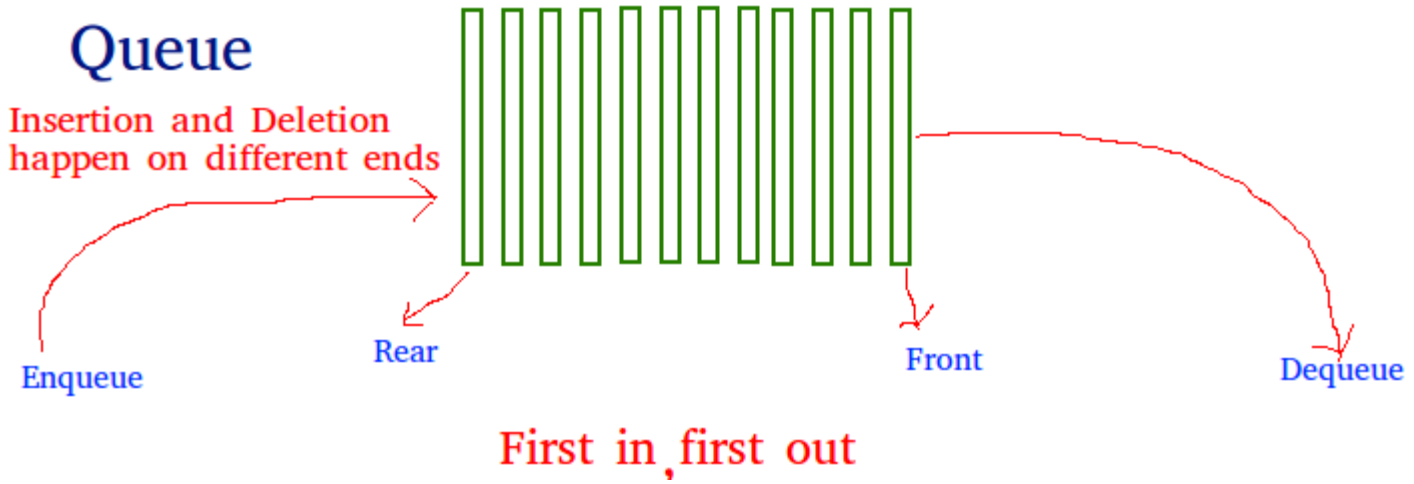
Hàng đợi (Queue)

Định nghĩa

Hàng đợi trong máy tính cũng giống như hàng đợi trong thực tế: hàng đợi mua vé tàu, vé xe, vé máy bay. Hàng đợi ứng dụng trong nhiều lĩnh vực khác nhau của khoa học máy tính

Danh sách có thứ tự của các phần tử.

Cơ chế FIFO: (First In, First Out).



Các loại hàng đợi

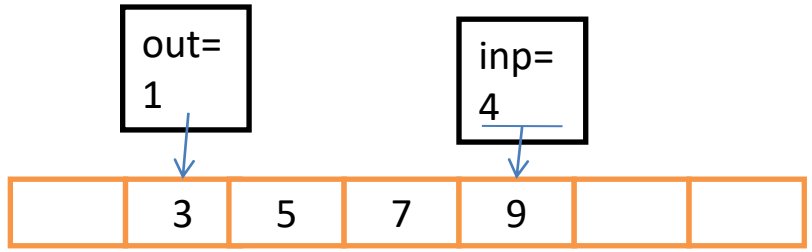
Hàng đợi (queue) là tập các node thông tin được tổ chức liên tục hoặc rời rạc nhau trong bộ nhớ và thực hiện theo cơ chế FIFO (First-In-First-Out).

Hàng đợi tuyến tính (sequential queue): hàng đợi liên tục được xây dựng theo nguyên tắc có điểm vào (inp) luôn lớn hơn điểm ra (out). Không gian nhớ sẽ không được tái sử dụng sau mỗi phép lấy phần tử ra khỏi hàng đợi.

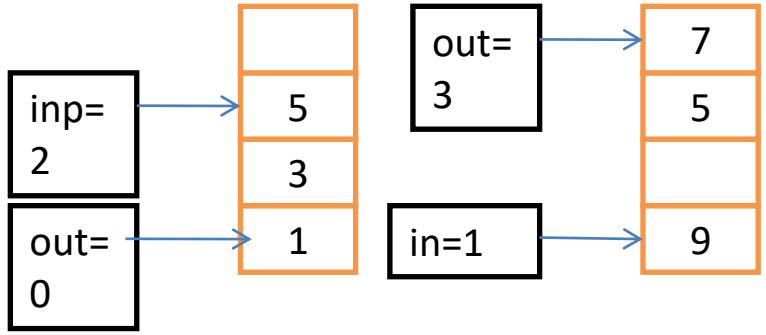
Hàng đợi vòng (circular queue): hàng đợi liên tục được xây dựng theo nguyên tắc không gian nhớ sẽ được tái sử dụng sau mỗi phép lấy phần tử ra khỏi hàng đợi.

Các loại hàng đợi

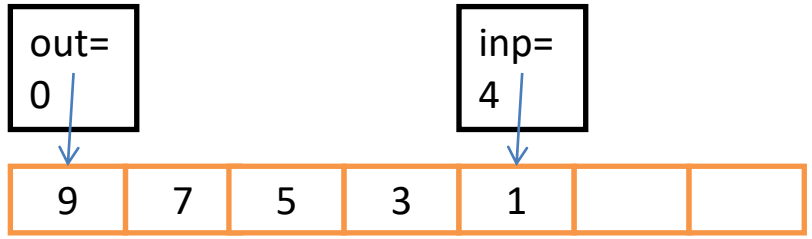
Hàng đợi tuyến tính



Hàng đợi vòng



Hàng đợi ưu tiên



Hàng đợi (Queue)

Các loại hàng đợi

Hàng đợi ưu tiên (priority queue): hàng đợi được xây dựng theo nguyên tắc phép đưa phần tử vào hàng đợi được xếp ứng với thứ tự ưu tiên của nó.

Hàng đợi hai đầu (double ended queue): hàng đợi được xây dựng theo nguyên tắc phép đưa phần tử vào và lấy phần tử ra khỏi hàng đợi được thực hiện ở hai đầu.

Hàng đợi (Queue)

Phương pháp biểu diễn hàng đợi

Biểu diễn liên tục sử dụng mảng

```
typedef struct {
int inp; // dùng để đưa phần tử vào hàng đợi
int out; // dùng để lấy phần tử ra khỏi hàng đợi
int node[MAX]; //các node thông tin của hàng đợi
} queue;
```

Hàng đợi (Queue)

Phương pháp biểu diễn hàng đợi

Biểu diễn rời rạc sử dụng danh sách liên kết đơn

```
struct node { //định nghĩa cấu trúc node
int infor;//thành phần dữ liệu
struct node *next; //thành phần liên kết
} *queue;
```

Hàng đợi (Queue)

Mô tả hàng đợi trong lập trình

Các thông tin cần thiết

- Kiểu phần tử
- Đầu vào / đầu ra / Số lượng

Các thao tác

- Khởi tạo
- bool isEmpty()
- bool isFull()
- Enqueue (hoặc push)
- Dequeue (hoặc pop)

Hàng đợi (Queue)

Sử dụng queue trong C++ STL

Cài đặt mở rộng từ deque

Khai báo

queue <type> Q;

Các phương thức cơ bản

front(): xem phần tử ở đầu

back(): xem phần tử ở cuối.

push(x): thêm vào cuối

pop(): xóa ở đầu

empty()

size()

Hàng đợi (Queue)

Ứng dụng hàng đợi

Kỹ thuật "loang" BFS

Bước 1: Đưa trạng thái xuất phát vào hàng đợi

Bước 2: Lặp đến khi hàng đợi rỗng hoặc gặp trạng thái đích

Xét trạng thái S ở đầu hàng đợi

Loại bỏ S ra khỏi hàng

Đưa các trạng thái đến được từ S vào hàng đợi

(chú ý: nếu trạng thái đã từng có trong hàng đợi trước đó thì không thể đưa vào lần 2)

Bước 3: Kết luận kết quả

Hàng đợi (Queue)

Một số bài toán sử dụng hàng đợi

Liệt kê số nhị phân

Số 0 và 9

THANK YOU FOR YOUR ATTENTION!