CHƯƠNG 5: NGĂN XẾP (STACK) & HÀNG ĐỢI (QUEUE)

Mục tiêu của chương

Nắm vững:

- Các khái niệm, định nghĩa ngăn xếp, hàng đợi.

- Cách khai báo ngăn xếp bằng mảng, bằng danh sách liên kết, các thao tác cơ bản như kiểm tra ngăn xếp rỗng, đưa phần tử vào ngăn xếp, lấy phần tử ra khỏi ngăn xếp.

- Cách khai báo hàng đợi bằng mảng, bằng danh sách liên kết và các thao tác cơ bản trên hàng đợi.

- Vận dụng lý thuyết giải các bài tập ngăn xếp, hàng đợi.

Nội dung của chương

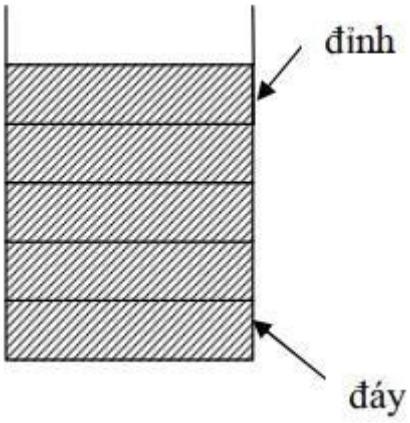
Nghiên cứu các bước thực hiện các thao tác cơ bản trên ngăn xếp, hàng đợi, minh họa từng bước thực hiện trên các bài tập cụ thể.

5.1. Ngăn xếp (Stack)

Ngăn xếp (stack) là một kiểu danh sách tuyến tính đặc biệt mà việc thêm vào một phần tử hoặc loại bỏ một phần tử chỉ được thực hiện ở một đầu của danh sách.

Đầu của ngăn xếp, nơi thực hiện các thao tác đưa thêm phần tử vào hoặc lấy phần tử ra, còn được gọi là đỉnh của ngăn xếp. Có thể coi ngăn xếp như một kho dữ liệu làm việc theo cơ chế LIFO (Last In First Out) vào sau ra trước.

Người ta còn gọi ngăn xếp là danh sách được đẩy từ trên xuống. Các phần tử có thể được thêm vào ngăn xếp bất kỳ khi nào, nhưng phần tử được lấy ra trước là phần tử được thêm vào sau cùng.



Hình 5.1: Minh họa ngăn xếp

5.1.1. Cấu trúc

Có hai 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).

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 5.1: 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;
```

5.1.2. Các phép xử lý

Các thao tác xây dựng trên stack theo cơ chế Last-In-First-Out bao gồm:

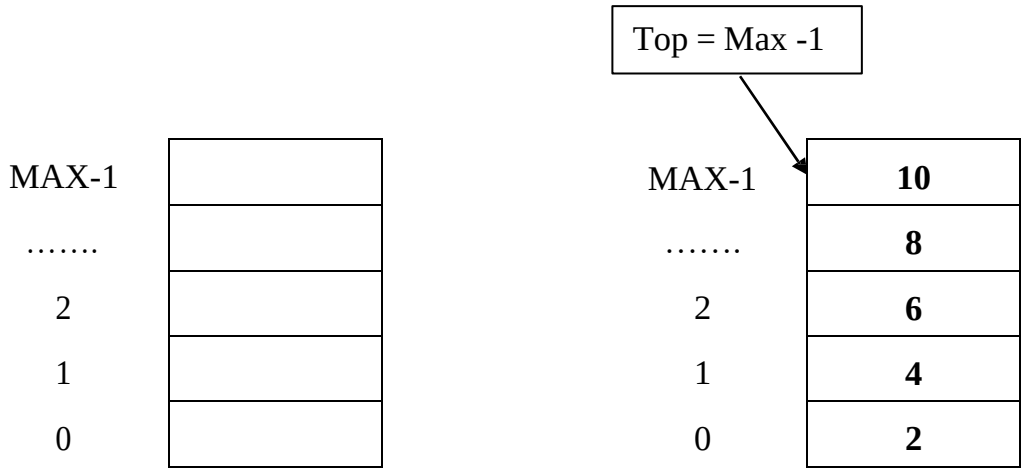
• Kiểm tra tính rỗng của stack (Empty(stack s)). Khi ta muốn lấy dữ liệu ra khỏi ngăn xếp thì nó chỉ được thực hiện khi và chỉ khi ngăn xếp không rỗng.

• Kiểm tra tính đầy của ngăn xếp (Full(stack s)). Khi ta muốn đưa dữ liệu vào ngăn xếp thì nó chỉ được thực hiện khi và chỉ khi ngăn xếp chưa tràn.

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

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

Ví dụ 5.2: Trạng thái rỗng, trạng thái đầy của stack.



Hình 5.2: Minh họa trạng thái ngăn xếp

5.1.2.1. Các thao tác stack trên mảng

▪ ***Kiểm tra tính rỗng của stack:***

Tất cả các thao tác trên stack chỉ được thực hiện tại vị trí con trỏ top. Vì vậy ta quy ước tại vị trí top = -1 stack ở trạng thái rỗng. Thao tác được thực hiện như sau:

```
typedef struct {
        int   top; //con trỏ top
        int   node[MAX]; //Các node của stack
```

```
} stack
int Empty(stack *s ){ //s là con trỏ đến stack
        if ( stack→top == -1 ) // Nếu top =-1
                return (1); //Hàm trả lại giá trị đúng
        return(0); //Hàm trả lại giá trị sai
}
```

▪ ***Kiểm tra tính đầy của stack:***

Khi ta muốn lấy dữ liệu ra khỏi ngăn xếp thì ngăn xếp phải chưa tràn. Vì biểu diễn dựa vào mảng, do đó tại vị trí top = MAX -1 thì stack ở trạng thái đầy. Thao tác được thực hiện như sau:

```
typedef struct {
        int   top; //Con trỏ top
        int   node[MAX]; //Các node của stack
} stack;
int Full(stack *s ){ //s là con trỏ đến stack
        if ( stack →top == MAX-1 ) // Nếu top = MAX -1
                return (1); //Hàm trả lại giá trị đúng
        return(0); //Hàm trả lại giá trị sai
}
```

▪ ***Đưa dữ liệu vào stack:***

Khi muốn đưa dữ liệu vào ngăn xếp thì ta phải kiểm tra ngăn xếp có đầy (tràn) hay không? Nếu ngăn xếp chưa đầy, thao tác sẽ được thực hiện. Nếu ngăn xếp đã đầy, thao tác không được thực hiện. Thao tác được thực hiện như sau:

```
typedef struct {
        int   top; //Con trỏ top
        int   node[MAX]; //Các node của stack
} stack;
void  Push(stack *s, int  x) { //x là node cần thêm vào stack
        if ( !Full (s)) // Nếu stack chưa tràn
                s →top = (s →top) +1; //Tăng con trỏ top lên 1 đơn vị
                tode[s →top] = x; //Lưu trữ x tại vị trí top
        }
        else <thông báo tràn stack>;
}
```

▪ ***Lấy dữ liệu ra khỏi stack:***

Khi muốn lấy dữ liệu ra khỏi ngăn xếp thì ta phải kiểm tra ngăn xếp có rỗng hay không? Nếu ngăn xếp không rỗng, thao tác sẽ được thực hiện. Nếu ngăn xếp rỗng, thao tác không được thực hiện. Thao tác được thực hiện như sau:

```
typedef struct {
        int   top; //Con trỏ top
        int   node[MAX]; //Các node của stack
} stack;
int  Pop(stack *s ){ //s là con trỏ đến stack
        if ( !Empty (s)) { // Nếu stack không rỗng
                x = Node[s →top]; //x là nội dung node bị lấy ra
                s →top = (s →top) -1; //giảm con trỏ top 1 đơn vị
                return (x); //Trả lại x là node bị loại bỏ
        }
else {<thông báo stack rỗng>; return (∞); }
}
```

5.1.2.2. Các thao tác trên stack dựa danh sách liên kết đơn

Lớp các thao tác trên stack được xây dựng như sau:

```
struct node{//Biểu diễn stack
        int info; //Thành phần thông tin của node
        struct node *link; //Thành phần con trỏ của node
}*Stack;
class stack_list{ //Mô tả lớp stack_list
        public:
                node *push(node *, int); //Thêm node
                node *pop(node *); //Loại bỏ node
                void traverse(node *);//Duyệt stack
                stack_list(){  Stack = NULL; } //Constructor của lớp
};
```

▪ ***Thêm node vào stack:***

```
node *stack_list::push(node *Stack, int item){ //Thêm node vào stack
        node *tmp; tmp = new (struct node);
        tmp→info = item;
        tmp→link = Stack;
        Stack = tmp;
        return Stack;
```

```
}
```

▪ ***Loại bỏ node ra khỏi stack:***

```
node *stack_list::pop(node *Stack){ node *tmp;
        if (Stack == NULL) cout<<"Stack rỗng"<<endl;
        else { tmp = Stack;
                cout<<“Node đã bị loại bỏ: "<<tmp->info<<endl;
                Stack = Stack→link; free(tmp);
        }
return Stack;
}
```

▪ ***Duyệt stack:***

```
void stack_list::traverse(node *Stack){
        node *ptr; ptr = Stack;
        if (Stack == NULL) cout<<"Stack rỗng"<<endl;
        else { cout<<“Các phần tử của stack :"<<endl;
                while (ptr != NULL)  {
                cout<<ptr→info<<endl; ptr = ptr→link;
                }
        }
}
```

5.1.3. Ứng dụng ngăn xếp

5.1.3.1. Ứng dụng đổi cơ số

▪ **Cấu trúc dữ liệu của ngăn xếp**

```
#define Maxlength 16
struct Stack
{
   int  E[Maxlength];
   int  top;
 }
struct Stack S;
```

▪ **Giải thuật**

```
void Convert(int n)
{ int r;
  while (n!=0)
```

```
    {
        r= n%2;
        Push(s, r);
        n= n/2;
     }
    cout <<  "\n Ma nhi phan: ";
    while (Emty(s))
     {
         Pop(s, r);
         cout << r ;
      }
  }
```

Ví dụ 5.3: Đổi số M từ cơ số 10 sang cơ số 8.

Thực hiện chia liên tiếp cho 8, lấy số dư và viết theo thứ tự ngược lại. Việc chia sẽ dừng khi được thương là 0. Giả sử, cho M = 7189 ở cơ số 10, cần đổi M ra cơ số 8.

7189	chia 8 được 898	dư 5
898	chia 8 được 112	dư 2
112	chia 8 được 14	dư 0
14	chia 8 được 1	dư 6
1	chia 8 được 0	dư 1

Kết quả ở cơ số 8, M được viết là 16025. Như vậy, số dư được tạo ra sau sẽ viết trước, số dư được tạo ra trước sẽ viết sau.

5.1.3.2. Ứng dụng tính giá trị của biểu thức số học

▪ Biểu thức số học với cách viết thông thường (infix notation) là toán tử nằm ở giữa hai toán hạng.

▪ Biểu thức số học Balan: dạng biểu thức số học theo ký pháp hậu tố (postfix notation) và tiền tố (prefix notation) mà được gọi chung là ký pháp Balan.

Ví dụ 5.4: Tính giá trị của biểu thức chỉ chứa các toán hạng và phép toán cộng, trừ, nhân, chia được viết theo ký pháp Balan.

Cách viết biểu thức số học theo ký pháp Balan là cách viết mà các toán hạng được viết trước, các toán tử được viết sau.

Chẳng hạn, biểu thức được viết theo ký pháp Balan

13 7 + 10 9 5 – – × ở dạng trung tố sẽ là **(13 + 7) × (10 – (9 – 5))**.

Ví dụ 5.5: Biểu diễn các biểu thức thức số học dạng hậu tố

a)

$a + b \Longleftrightarrow a\ b\ +$

$a - b \Longleftrightarrow a\ b\ -$

$b - a * b \Longleftrightarrow a\ b\ a\ *\ -$

$b * a / b \Longleftrightarrow a\ b\ *\ b\ /$

$(P) \Longleftrightarrow P$

b)

$(a + b * c) - (a / b + c) =$

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

$= (abc\ * +) - (ab\ / c +)$

$= abc\ * + - ab\ / c +$

▪ Thuật toán chuyển đổi biểu thức trung tố P thành biểu thức hậu tố

Bước 1 (Khởi tạo): stack = $\varnothing$; Out = $\varnothing$; Bước 2 (Lặp):

for each $x \in P$ do

2.1. Nếu x =‘ (‘ : Push(stack, x);

2.2. Nếu x là toán hạng: x $\Longrightarrow$ Out; 2.3. Nếu x $\in$ { +, - , *, / }

y = get(stack);

a) Nếu priority(x)³priority(y): Push(stack, x);

b) Nếu priority(x)<priority(y):

y = Pop(stack); y $\Longrightarrow$ Out; Push(stack, x);

c) Nếu stack = $\varnothing$: Push(stack, x);

2.4. Nếu x =‘)’:

y = Pop(stack);

While (y!=‘(’) do

y $\Longrightarrow$ Out; y = Pop(stack); EndWhile;

EndFor;

Bước 3 (Hoàn chỉnh biểu thức hậu tố):

```
While (stack ≠ ∅ ) do
        y = Pop(stack); yÞOut;
EndWhile;
```

Bước 4 (Trả lại kết quả):

Return(Out).

Ví dụ 5.6: Chuyển biểu thức P = a+b*(c-d)^e sang biểu thức hậu tố.

P	Stack	Out
a		a
+	+	a
b	+	a b
*	+ *	a b
(	+ * (	a b
c	+ * (	a b c
-	+ * (-	a b c
d	+ * (-	a b c d
)	+ * (	a b c d -
	+ *	a b c d -
^	+ * ^	a b c d -
e	+ * ^	a b c d - e
	+ *	a b c d - e ^
	+	a b c d - e ^ *
		a b c d - e ^ * +

P = a b c d - e ^ * +

Bảng 3: Minh họa các bước thực hiện của biểu thức P

▪ **Thuật toán tính toán giá trị biểu thức hậu tố**

Bước 1 (Khởi tạo):

stack = $\varnothing$;

Bước 2 (Lặp):

for each

$x \in P$ do

2.1. Nếu x là toán hạng:

Push(stack, x);

2.2. Nếu x $\in$ { +, - , *, / }

a) TH2 = Pop(stack, x);

b) TH1 = Pop(stack, x);

c) KQ = TH1⊗TH2;

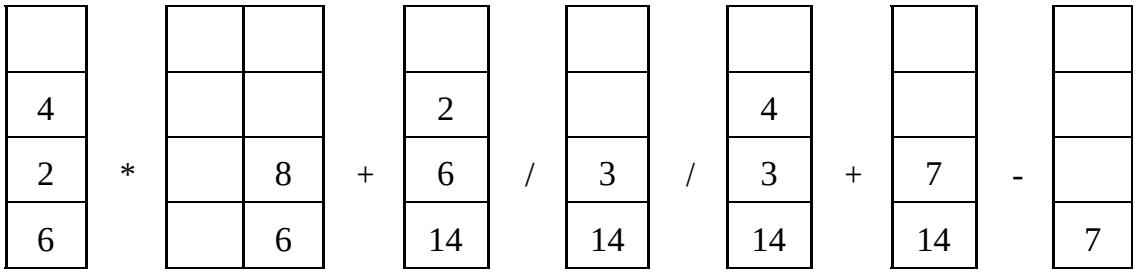
d) Push (stack, KQ);

EndFor;

Bước 4 (Trả lại kết quả):

Return(Pop(stack)).

Ví dụ 5.6: P = 6 2 4 * + 6 2 / 4 + -



Ví dụ 5.7: Tính giá trị của biểu thức M = "13 7 + 10 9 5 - - ×".

Quá trình đọc và làm việc trong ngăn xếp được minh hoạ trong bảng sau:

M_i	Stack	Out
13	Đưa 13 vào S	13
7	Đưa 7 vào S	7
+	Lấy 7 và 13 ra, tính 13+7, đưa kết quả vào S	20
10	Đưa 10 vào S	20, 10
9	Đưa 9 vào S	20, 10, 9
5	Đưa 5 vào S	20, 10, 9, 5
-	Lấy 5 và 9 ra, tính 9-5, đưa kết quả vào S	20, 10, 4
-	Lấy 4 và 10 ra, tính 10-4, đưa kết quả vào S	20, 6
×	Lấy 6 và 20 ra, tính 20×6, đưa kết quả vào S	120

Bảng 4: Minh họa các bước thực hiện của biểu thức M

5.2. Hàng đợi (Queue)

Hàng đợi (queue) là một kiểu danh sách tuyến tính mà việc thêm vào một phần tử hoặc loại bỏ một phần tử được thực hiện ở hai đầu khác nhau của danh sách.

Việc thêm phần tử được thực hiện ở một đầu của danh sách, việc loại bỏ được thực hiện ở đầu kia của danh sách. Như vậy, khác với ngăn xếp, hàng đợi là một kho dữ liệu làm việc theo cơ chế FIFO (First In First Out) vào trước ra trước.

5.2.1. Cấu trúc

Có hai phương pháp biểu diễn hàng đợi:

- Biểu diễn liên tục: sử dụng mảng.

- Biểu diễn rời rạc: sử dụng danh sách liên kết.

Trong trường hợp sử dụng mảng, mỗi node của hàng đợi được biểu diễn như một cấu trúc gồm 3 thành viên như sau:

```
typedef struct {
    int inp; // Con trỏ inp biểu diễn lối vào của hàng đợi
    int out; // Con trỏ out biểu diễn lối ra của hàng đợi
    int node[MAX]; //Các node thông tin của hàng đợi
} queue;
```

5.2.2. Các phép xử lý

Các thao tác trên hàng đợi bao gồm:

• Kiểm tra tính rỗng của hàng đợi (Empty(queue q)). Thao tác này được sử dụng khi ta cần đưa dữ liệu vào hàng đợi.

• Kiểm tra tính đầy của hàng đợi (Empty(queue q)). Thao tác này được sử dụng khi ta muốn lấy dữ dữ liệu ra khỏi hàng đợi.

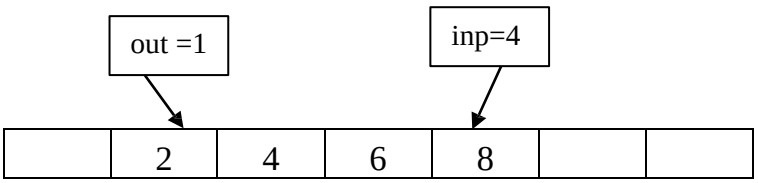
• Thao tác đưa dữ liệu vào hàng đợi (Push(queue q, int x)).

• Thao tác lấy dữ liệu ra khỏi hàng đợi (Pop(queue q)).

Các kiểu hàng đợi:

- Hàng đợi tuyến tính:

  Các thao tác được xây dựng sao cho con trỏ inp luôn có giá trị lớn hơn con trỏ out (inp > out). Đối với hàng đợi tuyến tính này, các ô nhớ con trỏ inp và con trỏ out đã đi qua sẽ không được sử dụng lại.



  Các thao tác được xây dựng sao cho con trỏ inp luôn có giá trị lớn hơn con trỏ out (inp > out) và các ô nhớ con trỏ inp đi qua sẽ được phép sử dụng lại bằng cách luôn ngầm định con trỏ out luôn thực hiện tại vị trí out = 0.

- Hàng đợi vòng: Các thao tác được xây dựng sao cho các ô nhớ con trỏ inp và con trỏ out đã đi qua sẽ được sử dụng lại. Đối với hàng đợi vòng, nhiều trạng thái inp<out và nhiều trạng thái inp>out.
- Hàng đợi ưu tiên: mỗi phần tử của hàng đợi được gắn với một độ ưu tiên sao cho phần tử nào có độ ưu tiên cao sẽ được lưu trữ gần con trỏ out nhất.

Cũng như đối với ngăn xếp, ta có thể dùng cấu trúc mảng một chiều hoặc danh sách liên kết để biểu diễn cấu trúc hàng đợi.

5.2.2.1. Biểu diễn hàng đợi bằng danh sách liên kết đơn:

```
struct node {
        int data;
        node *next;
}*front = NULL,*rear = NULL,*p = NULL,*np = NULL;
```

▪ ***Thao tác đưa phần tử vào hàng đợi*:**

```
void push(int x) {np = new node; //Cấp phát bộ nhớ cho np
        np→data = x; np→next = NULL; //Thiết lập liên kết cho np
        if(front == NULL) { //Nếu lối vào là rỗng
                front = rear = np; //
                rear→next = NULL;
        }
        else  {
                rear→next = np; rear = np;
                rear→next = NULL;
        }
}
```

▪ ***Thao tác lấy phần tử ra khỏi hàng đợi:***

```
int remove() { //Lấy phần tử ra khỏi hàng đợi
         int x;
         if(front == NULL) { //Nếu hàng đợi rỗng
                 cout<<“Hàng đợi rỗng“<<endl;
         }
         else  { //Nếu hàng đợi không rỗng
                 p = front; //q là node sau cùng
                 x = p→data; //Nội dung node q
                 front = front->next; //Chuyển front đến node kế tiếp
                 delete(p);//Giải phóng p
                 return(x); //Trả lại kết quả
         }
}
```

5.2.2.2. Cài đặt hàng đợi vòng bằng mảng

▪ ***Biểu diễn lớp hàng đợi vòng***:

```
class Circular_Queue { //Mô tả lớp Circular_Queue
```

```
    private:
        int *cqueue_arr;
        int front, rear; //Lối vào và lối ra hàng đợi
    public:
    circular_Queue(){ //Constructor
        cqueue_arr = new int [MAX];
        rear = front = -1;
    }
    void  Push(int item);
    void  Pop(void);
    void  Display(void);
}
```

▪ ***Thao tác Push*:**

```
void Circular_Queue :: Push(int item) { //Thêm phần tử vào hàng đợi vòng
    if ((front == 0 && rear == MAX-1) || (front == rear+1)) {
            cout<<“Tràn hàng đợi“<<endl; return;
    }
    if (front == -1) { front = 0; rear = 0; }
    else {
            if (rear == MAX – 1)   rear = 0;
            else  rear = rear + 1;
    }
    cqueue_arr[rear] = item
    }
```

▪ ***Thao tác Pop***:

```
void Circular_Queue :: Pop() { //Loại phần tử khỏi hàng đợi vòng
   if (front == -1) {cout<<“Queue rỗng“<<endl; return ;}
   cout<<“Phần tử bị loại bỏ là : "<<cqueue_arr[front]<<endl;
   if (front == rear) { front = -1; rear = -1; }
   else  {
        if (front == MAX – 1)   front = 0;
        else   front = front + 1;
   }
}
```

▪ ***Thao tác hiển thị hàng đợi vòng*:**

```
void Circular_Queue :: Display() { //Hiển thị hàng đợi vòng
    int front_pos = front, rear_pos = rear;
    if (front == -1) { cout<<"Hàng đợi rỗng"<<endl; return; }
    cout<<"Các phần tử của hàng đợi :";
    if (front_pos <= rear_pos) {
            while (front_pos <= rear_pos) {
                cout<<cqueue_arr[front_pos]<<" "; front_pos++;
            }
     }
    else {
            while (front_pos <= MAX – 1) {
                cout<<cqueue_arr[front_pos]<<" "; front_pos++;
            }
            front_pos = 0;
            while (front_pos <= rear_pos) {
                cout<<cqueue_arr[front_pos]<<" "; front_pos++;
            }
       }
       cout<<endl;
}
```

5.2.2.3. Hàng đợi ưu tiên

Hàng đợi ưu tiên (Priority Queue) là một mở rộng của cấu trúc dữ liệu hàng đợi có những đặc tính sau:

• Mỗi phần tử của hàng đợi được gắn với độ ưu tiên của nó.

• Node có độ ưu tiên cao hơn sẽ đứng trước node có độ ưu tiên thấp hơn trong hàng đợi.

• Nếu hai phần tử có độ ưu tiên giống nhau thì thứ tự ưu tiên tuân theo luật của hàng đợi thông thường (vào trước ra trước).

Biểu diễn

```
struct node{ //Cấu trúc một node của hàng đợi ưu tiên
        int priority; //Mức độ ưu tiên của node
        int info; //Thông tin của node
        struct node *link; //Con trỏ liên kết của node
};
```

Thao tác trên hàng đợi ưu tiên

• Thao tác Push: đưa vào hàng đợi với mức độ ưu tiên của nó.

• Thao tác Pop: loại bỏ phần tử có mức độ ưu tiên cao nhất.

Đưa phần tử có mức độ ưu tiên vào hàng đợi:

```
void Priority_Queue::Push(int item, int priority) {
    node *tmp, *q; //Sử dụng hai con trỏ tmp và q
    tmp = new node; //Cấp phát miền nhớ cho tmp
    tmp→info = item;
    //Thiết lập nội dung cho tmp
    tmp->priority = priority; //Thiết lập độ ưu tiên cho tmp
    if (front == NULL || priority < front->priority){ //Nếu hàng đợi rỗng
            tmp->link = front; //Node đầu tiên là tmp
            front = tmp; //Hiện tại hàng đợi có một node
    }
    else { q = front; //q trỏ đến front
            while (q→link != NULL && q→link→priority <= priority)
               q=q→link; //Tìm vị trí thích hợp cho độ ưu tiên
               tmp→link = q→link; //Thiết lập liên kết cho tmp q→link
               = tmp;//thiết lập liên kết cho q
         }
}
```

Loại bỏ phần tử ra khỏi hàng đợi:

```
void Priority_Queue::Pop(){ node *tmp; //Sử dụng con trỏ tmp
    if(front == NULL) cout<<“Hàng đợi rỗng\n";
    else { tmp = front; //tmp trỏ đến front
       cout<<“Node loại bỏ là: "<<tmp->info<<endl; front = front->link;
       free(tmp);//Giải phóng tmp
    }
}
```

Hiển thị hàng đợi ưu tiên:

```
void Priority_Queue::display(){ node *ptr; ptr = front;
    if (front == NULL)  cout<<“Hàng đợi rỗng“<<endl;
    else{   cout<<“Nội dung hàng đợi:“<,endl;
            cout<<“Độ ưu tiên các node:“<<endl;
```

```
            while(ptr != NULL){ cout<<ptr->priority<<" "<<ptr→info<<endl;
            ptr = ptr→link;
        }
    }
}
```

5.2.3. Ứng dụng

- Trong các hệ điều hành:

+ Hàng đợi các công việc hoặc các tiến trình đang đợi để được thực hiện

+ Hàng đợi các tiến trình chờ các tín hiệu từ các thiết bị IO

+ Các file được gửi tới máy in

- Mô phỏng các hệ thống hàng đợi thời trong thực tế:

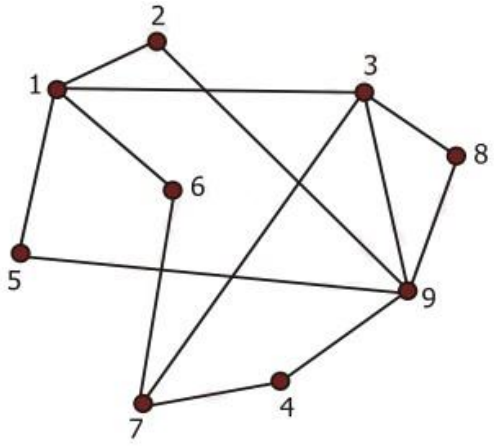
+ Các khách hàng trong các cửa hàng tạp hóa, trong các hệ thống ngân hàng

+ Các đơn đặt hàng của một công ty

+ Các cuộc gọi điện thoại hoặc các đặt hàng vé máy bay, các đặt hàng của khách hàng …

- Các ứng dụng khác: một trong những ứng dụng thường gặp nhất là trong lý thuyết đồ thị. Ngăn xếp và hàng đợi được sử dụng để duyệt đồ thị theo chiều sâu và chiều rộng.

Ví dụ, có một đồ thị được cho trong hình dưới. Vấn đề đặt ra là tìm đường đi ngắn nhất từ đỉnh 2 đến đỉnh 7.



Bằng việc sử dụng hàng đợi, ta có thể tìm được đường đi ngắn nhất từ 2 đến 7 bằng thuật toán duyệt đồ thị theo chiều rộng như sau:

- Bước 1: Khởi tạo một hàng đợi Q rỗng
- Bước 2: xếp phần tử 2 vào hàng đợi Q.
- Bước 3: lần lượt lấy các phần tử ra khỏi hàng đợi Q và xếp tất cả các đỉnh kề với nó (trừ những đỉnh đã xét) vào hàng đợi cho đến khi gặp được phần tử có giá trị bằng 7.

Nếu không còn phần tử nào trong hàng đợi mà chưa tìm được phần tử có giá trị

bằng 7 thì kết luận không có đường đi từ 2 đến 4.

Kết quả của thuật toán trên được mô tả trong bảng sau:

Thao tác	Kết quả Q
Xếp 2 vào hàng đợi	2
Lấy 1 phần tử ra khỏi hàng đợi: 2	rỗng
Xếp tất cả các phần tử kề với 2 vào Q	1, 9
Lấy 1 phần tử ra khỏi hàng đợi: 1	9
Xếp tất cả các phần tử kề với 1 vào Q	9 3 5 6
Lấy 1 phần tử ra khỏi hàng đợi: 9	3 5 6
Xếp tất cả các phần tử kề với 9 vào Q	3 5 6 4 8 (2 và 5 đã được xét nên không được xếp vào)
Lấy 1 phần tử ra khỏi hàng đợi: 3	5 6 4 8
Xếp tất cả các phần tử kề với 3 vào Q	5 6 4 8 7

Bảng 5: Minh họa các bước thực hiện duyệt đồ thị bằng hàng đợi

Như vậy sau 9 thao tác ta đã gặp được phần tử có giá trị bằng 7.

Lần ngược lại các thao tác ta sẽ thu được 1 đường đi ngắn nhất từ 2 đến 7:

- 7 được xếp vào Q do liền kề với 3
- 3 được xếp vào Q do liền kề với 1
- 1 được xếp vào Q do liền kề với 2

Vậy đường đi ngắn nhất từ 2 đến 7 tìm được là: 2 1 3 7.

Bài tập vận dụng

1. Xây dựng các thao tác trên ngăn xếp dựa vào mảng.
2. Hạn chế của cài đặt ngăn xếp bằng mảng so với danh sách liên kết.
3. Dùng bảng để minh hoạ hình ảnh của ngăn xếp được sử dụng để tính giá trị của biểu thức ((20+12)*(14/(4+3))).
4. Xây dựng tập thao tác trên hàng đợi dựa vào mảng.
5. Xây dựng tập thao tác trên hàng đợi dựa vào danh sách liên kết.
6. Cho biểu thức P = (A+B)*(C/(U+V)) dưới dạng hậu tố. Dùng bảng để minh hoạ hình ảnh của ngăn xếp được sử dụng để tính giá trị của biểu thức hậu (cho A = 30, B=15, C=14, U=4, V=3).
7. Xây dựng tập thao tác trên hàng đợi ưu tiên dựa vào mảng.
8. Xây dựng tập thao tác trên hàng đợi vòng dựa vào mảng.
9. Xây dựng tập thao tác trên hàng đợi vòng dựa vào danh sách liên kết.[84]

10. Minh họa quá trình chuyển biểu thức trung tố sau về biểu thức hậu tố:
(a/2+b) * (c - d*e) + f/g
11. Cho biểu thức hậu tố: a b c + / d - e +
Dùng bảng để minh hoạ quá trình định giá biểu thức hậu tố trên ứng với a=12, b=5, c=1, d=2, e=8.