3.6. Hàng đợi (queue)

3.6.1. Định nghĩa hàng đợi (queue).

Tập hợp 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).

3.6.2. Biểu diễn hàng đợi dựa vào mảng

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;
```

3.6.3. Các thao tác trên hàng đợi

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)).

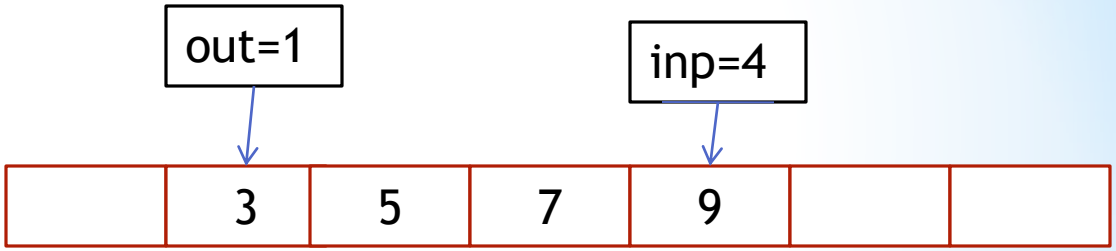
Các kiểu hàng đợi:

- **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.

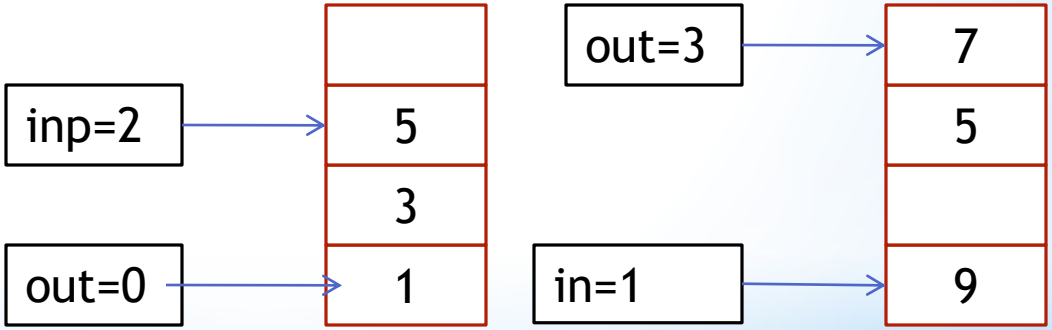
•**Hàng đợi vòng**: Các thao tác được xây dựng sao cho các ô nhớ con trỏ inp và con trỏ out đã đi qua sẽ được sử dụng lại. Đối với hàng đợi vòng, nhiều trạng thái inp<out và nhiều trạng thái inp>out.

• **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.

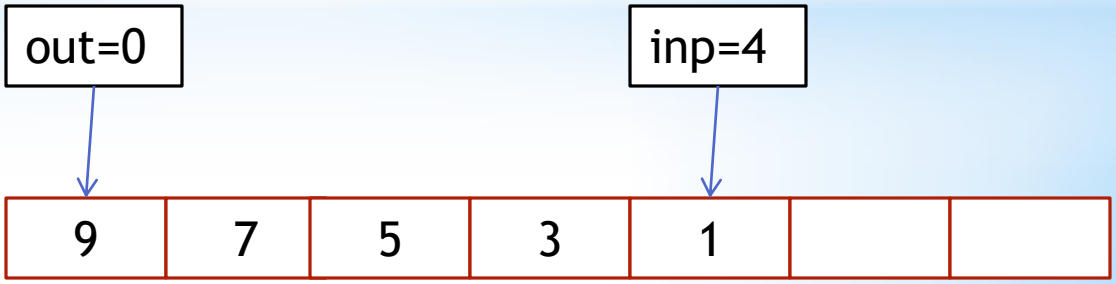
Hàng đợi tuyến tính:



Hàng đợi vòng:



Hàng đợi ưu tiên:



Biểu diễn hàng đợi:

```
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;
```

Kiểm tra tính rỗng của hàng đợi: hàng đợi tuyến tính rỗng khi nó ở trạng thái khởi đầu (inp = out = -1) hoặc khi con trỏ in = out. Như vậy, trong cả hai trường hợp inp = out thì trạng thái hàng đợi rỗng.

```
int   Empty( queue  *q) { //q là con trỏ kiểu queue
    If ( q ->inp == q ->out) //Nếu q ->inp = q ->out
        return (1); //Trả lại giá trị đúng
    return(0); //Trả lại giá trị sai
}
```

Kiểm tra tính đầy của hàng đợi: hàng đợi tuyến tính đầy khi con trỏ inp ở vị trí MAX-1 (inp = MAX -1).

```
int   Full( queue  *q) { //q là con trỏ kiểu queue
      If ( q ->inp == MAX -1 ) //Nếu q ->inp = MAX-1
            return (1); //Trả lại giá trị đúng
      return(0); //Trả lại giá trị sai
}
```

Đưa dữ liệu vào hàng đợi:

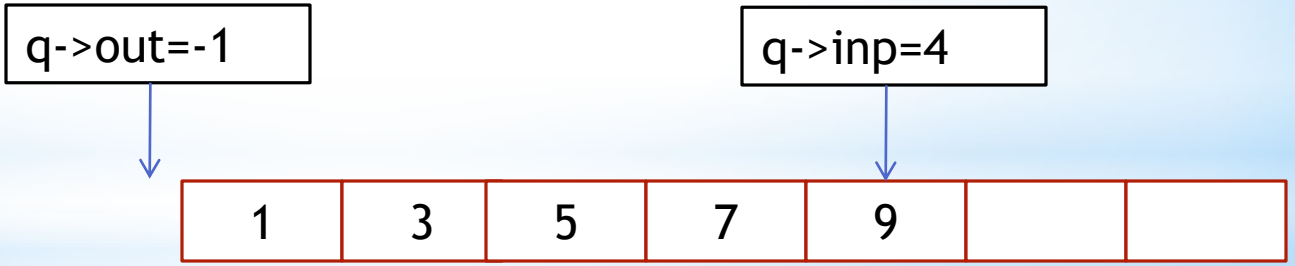
```
void Push( queue  *q, int   x) { //x là dữ liệu cần lưu trữ trong queue
      If ( !Full(q)) //Nếu đúng q chưa đầy: q->in <MAX-1
            q ->inp = (q ->in) +1; //Nhấc con trỏ in lên 1 đơn vị
            q->node[q->inp] = x; //Lưu trữ x tại vị trí q->inp
      }
      else <Thông báo tràn queue>;
}
```

Lấy dữ liệu ra khỏi hàng đợi:

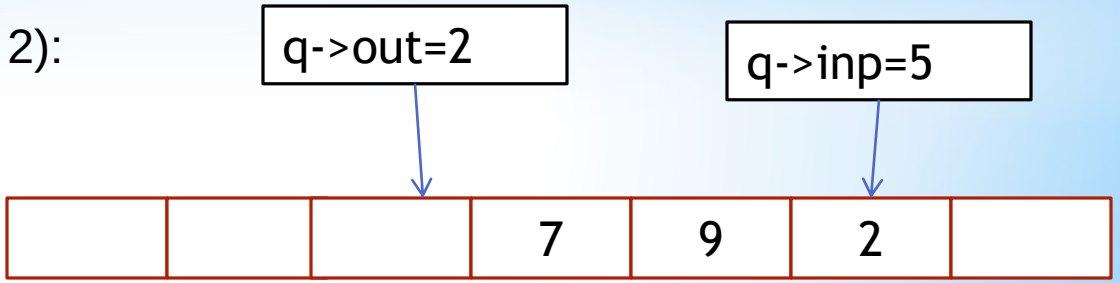
```
int Pop( queue  *q) { //q là con trỏ kiểu queue
    If ( !Empty (q)) //Nếu đúng q không rỗng
        q ->out = (q ->out) +1; //Nhấc con trỏ out lên 1 đơn vị
        int x = q->node[q->inp]; // x là giá trị node được lấy ra
        return(x); //Trả lại giá trị node loại bỏ
    }
    else { <Thông báo stack rỗng>; return (∞); }
}
```

Hoạt động của Push và Pop:

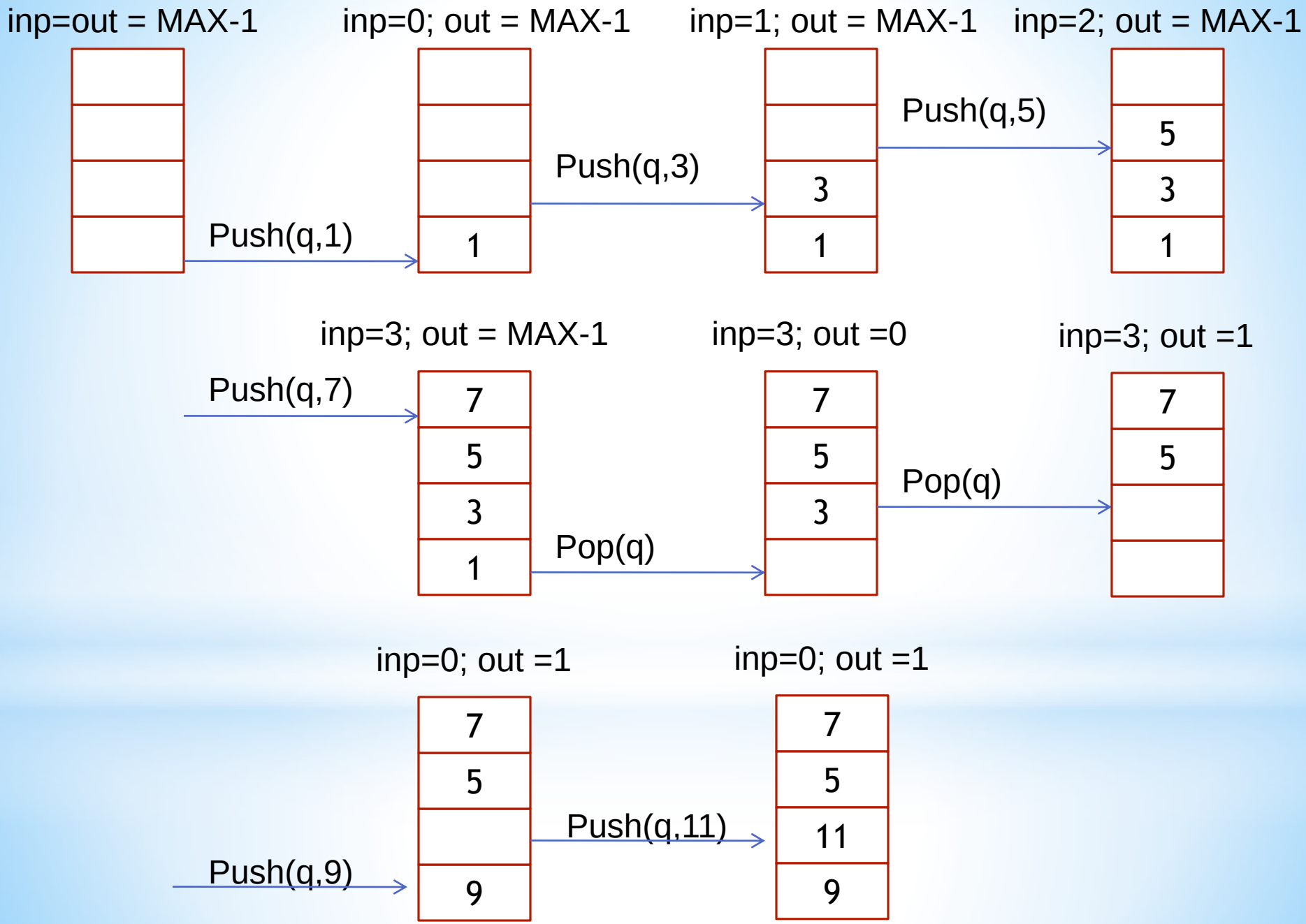
Push(q,1); Push(q,3); Push(q,5); Push(q,7); Push(q,9);



Pop (q); Pop (q); Pop(q); Push(q, 2):



Hàng đợi vòng: Hàng đợi rỗng: inp=out = MAX-1



3.6.4. Cài đặt hàng đợi bằng danh sách liên kết đơn

Cài đặt hàng đợi bằng danh sách liên kết đơn được thực hiện bằng cách:

Queue = { DSLKĐ + [Add-Top + Del-Bottom] }

Queue = { DSLKĐ + [Add-Bottom + Del-Top] }

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
        }
}
```

3.6.5.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;
}
```

3.6.5. Ứng dụng của hàng đợi

- Biểu diễn tính toán
- Duyệt cây
- Duyệt đồ thị.
- Xây dựng các thuật toán lập lịch.

Ví dụ 1. Bài toán người sản xuất và nhà tiêu dùng.

Ví dụ 2. Thuật toán Best Fit.

Ví dụ 3. Thuật toán Best Availble.

Ví dụ 4. Bữa tối của 5 nhà triết học (The Dinner of Five Philosophers).

Ví dụ 5. Thuật toán Round-Robin.

3.7. Hàng đợi ưu tiên

3.7.1. **Định nghĩa**. 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).

3.7.2. 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
};
```

3.7.3. 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.

Lớp biểu các thao tác trên hàng đợi ưu tiên được cài đặt bằng C++ như sau:

Biểu diễn node của hàng đợi ưu tiên:

```
struct node{
        int priority; //Mức độ ưu tiên của node
        int info; //Thông tin của node
        struct node *link; //Liên kết của node
};
```

Lớp thao tác trên hàng đợi ưu tiên:

```
class Priority_Queue{
  private:      node *front; //thành phần private
  public:
        Priority_Queue(){ front = NULL;  } //Constructor lớp Priority_Queue
        void Push(int item, int priority); //Đưa phần tử queue
        void Pop(void) ; //Loại bỏ phần tử có độ ưu tiên cao nhất
        void display(); //Hiển thị hàng đợi ưu tiên
};
```

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;
                }
        }
}
```

3.8. Hàng đợi hai điểm cuối (double ended queue):

3.8.1.Định nghĩa

Hàng đợi dqueue(*hàng đợi hai điểm cuối*) là loại hàng đợi đặc biệt cho phép thêm và loại bỏ phần tử tại điểm cuối của hàng đợi.

3.8.2. Biểu diễn dqueue

```
struct node {
        int info; //Thông tin của node
        node *next; //Con trỏ đến node tiếp theo
        node *prev;//Con trỏ đến node sau nó

}*head, *tail;
```

head: là con trỏ đến đỉnh đầu; tail là con trỏ đến đỉnh cuối.

3.8.3. Ứng dụng của dqueue

- dqueue hỗ trợ các thao tác cho cả stack và queue.
- dqueue hỗ trợ các phép quay theo chiều kim đồng hồ hoặc bán nửa kiem đồng hồ.
- Các thao tác thêm và loại bỏ được thực hiện hiệu quả với độ phức tạp hằng số.

3.8.4. Ngôn ngữ lập trình

- STL của C++ hỗ trợ các thao tác trên dqueue.
- Java cung cấp interface cho dqueue.

3.8.5. Các thao tác trên dqueue

Các thao tác trên dqueu bao gồm:

- Insert-Front(): thêm một node vào đầu dqueue.
- Insert-Last(): thêm một node cuối đầu dqueue.
- Del-Front(): loại một node ở đầu dqueue.
- Del-Last(): Loại một node ở cuối dqueue
- Display-Front(): hiển thị dqueue bắt đầu từ đầu trước
- Display-Last(): hiển thị dqueue bắt đầu từ đầu sau
- GetFront(): nhận node ở đầu dqueue
- GetLasst(): nhận node cuối dqueue.
- isEmpty(): kiểm tra tính rỗng của dqueue.
- isFull(): kiểm tra tính tràn của dqueue.

Dưới đây là một cách cài đặt cho lớp dqueue:

Biểu diễn dqueue:

```
struct node{
        int info; //Thông tin của node
        node *next; //Thành phần con trỏ trước
        node *prev; //Thành phần con trỏ sau
}*head, *tail;
```

Mô tả lớp dqueue:

```
class dqueue {
    public:
        int top1, top2; //top1 là con trỏ vào; top2 là con trỏ ra.
        void insert_front(); //Thêm node vào đầu trước của dqueue
        void insert_last(); //Thêm node vào đầu sau của dqueue
        void del_front(); //Loại node ở đầu trước của dqueue
        void del_last(); //Loại node vào đầu trước của dqueue
        void display_front(); //Hiển thin nội dung dqueue bắt đầu từ đầu trước
        void display_last(); //Hiển thin nội dung dqueue bắt đầu từ đầu tsau
        dqueue(){ //Constructor của dqueue
                top1 = 0;
                top2 = 0;
                head = NULL;
                tail = NULL;
        }
};
```

Thao tác Insert-front:

```
void dqueue::insert_front(){ struct node *temp;int value;
        if (top1 + top2 >= 50){
                cout<<"Tràn hàng đợi"<<endl;return;
        }
        if (top1 + top2 == 0){ //Nếu dqueue rỗng
                cout<<"Nhập giá trị node: ";cin>>value;
                 head = new (struct node); head->info = value;
                head->next = NULL; //Thiết lập liên kết trước cho head
                head->prev = NULL; //Thiết lập liên kết tsau cho head
                tail = head; // Node trước và node sau là một
                top1++; //top1 tăng lên 1 đơn vị
        }
        else {    cout<<"Nhập giá trị node: "; cin>>value;
                temp = new (struct node); temp->info = value;
                temp->next = head; //Thiết lập liên kết trước cho temp
                temp->prev = NULL; // Thiết lập liên kết sau cho temp
                head->prev = temp; Thiết lập liên kết sau cho head
                head = temp; //Node bắt đầu bây giờ chính là temp
                top1++;
        }
}
```

Thao tác Insert-Last:

```
void dqueue::insert_last(){ struct node *temp;    int value;
        if (top1 + top2 >= 50){
                cout<<"Tràn hàng đợi"<<endl; return;
        }
        if (top1 + top2 == 0){ //Nếu dqueue rỗng
                cout<<"Nhập giá trị node: "; cin>>value;
                head = new (struct node);head->info = value;
                head->next = NULL;//Thiết lập liên kết trước cho head
                head->prev = NULL; //Thiết lập liên kết sau cho head
                tail = head; //Node đầu và node cuối là một
                top1++;
        }
        else {  cout<<"Nhập giá trị node: ";cin>>value;
                temp = new (struct node);temp->info = value;
                temp->next = NULL; //Thiết lập liên kết trước cho tmp
                temp->prev = tail; //Thiết lập liên kết sau cho tmp
                tail->next = temp; //Node sau cũ được trỏ đến tmp
                tail = temp; //Node sau mới bây giờ là tmp
                top2++;
        }
}
```

Thao tác Del-Front:

```
void dqueue::del_front(){
        struct node *tmp =head;
        if (top1 + top2 <= 0){ //Nếu dqueue rỗng
                cout<<"Deque rỗng"<<endl;
                return;
        }
        head = head->next; //head được di chuyển lên một node
        head->prev = NULL; //Ngắt liên kết sau của head
        top1--; free (tmp);
}
```

Thao tác Del-Last:

```
void dqueue::del_last(){struct node *tmp =tail;
        if (top1 + top2 <= 0){
                cout<<"Deque Underflow"<<endl;
                return;
        }
        tail = tail->prev;
        tail->next = NULL;
        top2--; free(tmp);
}
```

Thao tác Display-Front:

```
void dqueue::display_front(){ struct node *temp;
        if (top1 + top2 <= 0){
                cout<<“Dqueue rỗng"<<endl;return;
        }
        temp = head; //temp trỏ đến head
        cout<<“Bắt đầu duyệt từ node đầu:"<<endl;
        while (temp != NULL){
                cout<<temp->info<<" “; temp = temp->next;
        }
 }
```

Thao tác Display-Last:

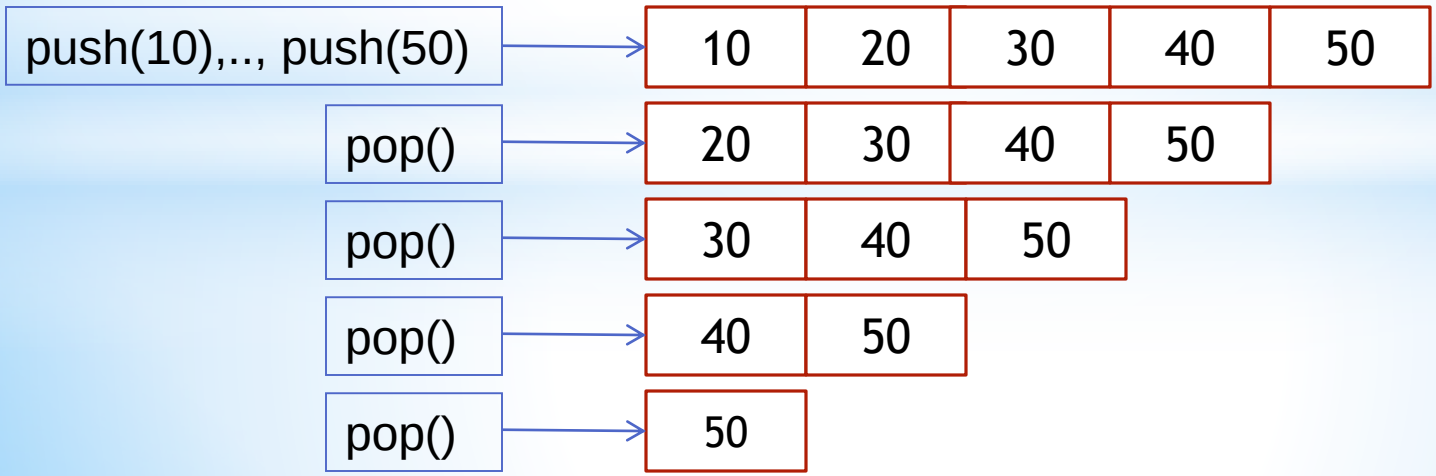
```
void dqueue::display_last(){ struct node *temp;
        if (top1 + top2 <= 0){
                cout<<“Hàng đợi rỗng"<<endl;return;
        }
        cout<<“Duyệt từ node cuối:"<<endl;
        temp = tail; //temp trỏ đến tail
        while (temp != NULL){
                cout<<temp->info<<" “; temp = temp->prev;
        }
}
```

Queue STL: để sử dụng các thao tác trên hang đợi dựa vào STL, ta sử dụng các hàm được mô tả trong #include <queue>.

QUEUE STL		
STT	**Thao tác**	**Ý nghĩa**
1	q.empty()	Kiểm tra hang đợi q có rỗng hay không. Trả lại giá trị true nếu q rỗng, ngược lại trả lại giá trị false.
2	q.size()	Trả lại một số là số lượng phần tử của hang đợi q.
3	q.front()	Truy cập vào phần tử vào trước tiên hàng đợi q.
4	q.back()	Truy cập vào phần tử vào sau cùng hàng đợi q.
5	q.push(item)	Đưa item vào hàng đợi q.
6	q.pop()	Loại phần tử ra khỏi hang đợi q theo luật FIFO
7	q.qwap(q1)	Đổi các phần tử của hang đợi q1 thành các phần tử của q và ngược lại.
8		

Ví dụ:

```
#include <iostream>
#include <queue>
using namespace std;
int main(void){ queue <int> q;
        for(int i=1; i<=5; i++)  {  q.push(i*10);  }
        cout<<"Kích cỡ hàng đợi:"<<q.size()<<endl;
        cout<<“Phần tử đầu tiên:"<<q.front()<<endl;
        cout<<“Phần từ cuối cùng:"<<q.back()<<endl;
        cout<<“Trạng thái hàng đợi:"<<q.empty()<<endl;
        cout<<“Loại bỏ phần tử:";
        while(!q.empty()){
                int t = q.front();cout<<t<<"  ";  q.pop();
        }
}
```

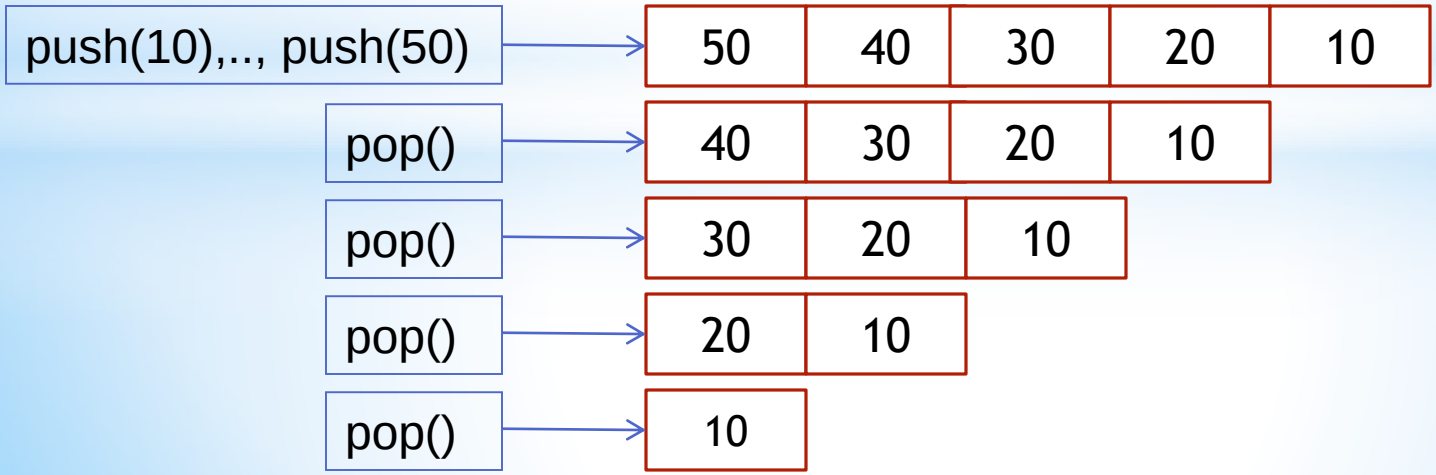


Priority Queue STL: để sử dụng các thao tác trên hàng đợi ưu tiên, ta sử dụng các hàm được mô tả trong #include <queue>.

QUEUE STL		
STT	**Thao tác**	**Ý nghĩa**
1	pq.empty()	Kiểm tra hàng đợi ưu tiên pq có rỗng hay không. Trả lại giá trị true nếu pq rỗng, ngược lại trả lại giá trị false.
2	pq.size()	Trả lại một số là số lượng phần tử của hang đợi ưu tiên pq.
3	pq.top()	Truy cập vào phần tử đầu tiên hàng đợi ưu tiên pq.
4	pq.push(item)	Đưa item vào hàng đợi ưu tiên pq.
5	pq.pop()	Loại phần tử ra khỏi hang đợi ưu tiên q theo luật FIFO
6	pq.qwap(pq1)	Đổi các phần tử của hang đợi pq1 thành các phần tử của pq và ngược lại.

Ví dụ: priority queue

```
#include <iostream>
#include <queue>
using namespace std;
int main(void){
        //priority_queue< int, vector<int>, greater<int> > q;
        priority_queue <int> q;
        for(int i=1; i<=5; i++) q.push(i*10);
        cout<<"\nPhần tử đầu tiên:"<<q.top()<<endl;
        cout<<"\n Kích cỡ:"<<q.size()<<endl;
        cout<<"\n Trạng thái:"<<q.empty()<<endl;
        cout<<"\n Thao tác pop:";
        while(!q.empty()){
                int x = q.top(); cout<<x<<" "; q.pop();
        }
}
```

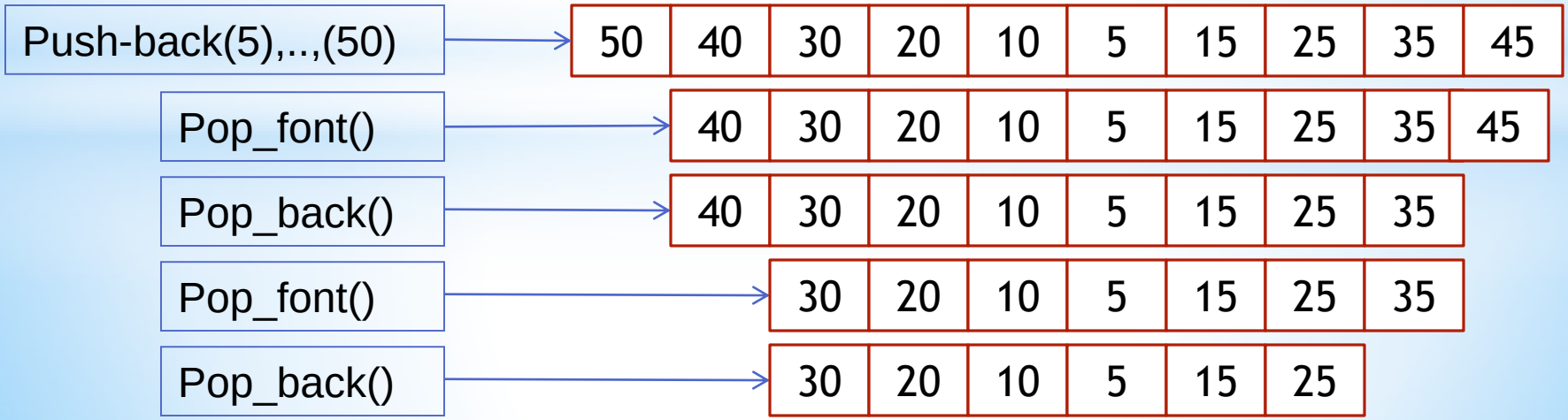


deQueue STL: để sử dụng các thao tác trên hàng đợi hai điểm cuối dequeue, ta sử dụng các hàm được mô tả trong #include <queue>.

QUEUE STL		
STT	**Thao tác**	**Ý nghĩa**
1	dq.empty()	Kiểm tra hang đợi dq có rỗng hay không. Trả lại giá trị true nếu dq rỗng, ngược lại trả lại giá trị false.
2	dq.size()	Trả lại một số là số lượng phần tử của hang đợi dq.
3	dq.front()	Truy cập vào phần tử đầu tiên hàng đợi dq.
4	dq.back()	Truy cập vào phần tử cuối cùng hàng đợi dq.
5	dq.push_front(item)	Đưa item vào đầu hàng đợi dq.
6	dq.push_back(item)	Đưa item vào cuối hàng đợi dq.
7	dq.pop_font(item)	Loại phần tử đầu tiên hàng đợi dq.
8	dq.pop_back(item)	Loại phần tử cuối cùng hàng đợi dq.

Ví dụ: priority queue

```
#include <iostream>
#include <queue>
using namespace std;
int main(void) {    deque<int> dq;
    int A[]={5,10,15,20,25,30,35,40,45,50}, n=10;
    for(int i=0; i<n; i++) {
            if(A[i]%2) dq.push_back(A[i]);
            else dq.push_front(A[i]);
    }
    cout<<"So luong phan tu:"<<dq.size()<<endl;
    cout<<"Front to rear:";
    while(!dq.empty()) { //int t=dq.front();dq.pop_front();
            int t=dq.back();dq.pop_back(); cout<<t<<" ";
     }
}
```

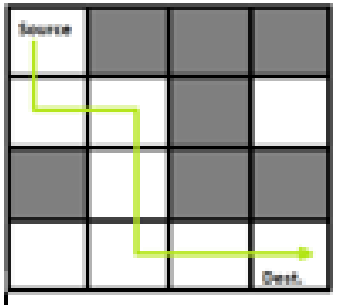


BÀI 3. DI CHUYỂN TRONG MÊ CUNG 1

Cho một mê cung bao gồm các khối được biểu diễn như một ma trận nhị phân A[N][N]. Một con chuột đi từ ô đầu tiên góc trái (A[0][0]) đến ô cuối cùng góc phải (A[N-1][N-1]) theo nguyên tắc:

- Down (D): Chuột được phép xuống dưới nếu ô dưới nó có giá trị 1.
- Right (R): Chuột được phép sang phải dưới nếu ô bên phải nó có giá trị 1.

Hãy đưa ra một hành trình của con chuột trên mê cung. Đưa ra -1 nếu chuột không thể đi đến đích.



Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất đưa vào số N là kích cỡ của mê cung; dòng tiếp theo đưa vào ma trận nhị phân A[N][N]. T, N, A[i][j] thỏa mãn ràng buộc: $1 \leq T \leq 10$; $2 \leq N \leq 10$; $0 \leq A[i][j] \leq 1$.

Output: Đưa ra tất cả đường đi của con chuột trong mê cung theo thứ tự từ điển. Đưa ra -1 nếu chuột không đi được đến đích.

Input	Output
1 4 1 0 0 0 1 1 0 1 0 1 0 0 1 1 1 1	DRDDRR

BÀI TẬP

Cho cặp số S và T là các số nguyên tố có 4 chữ số (Ví dụ S = 1033, T = 8197 là các số nguyên tố có 4 chữ số). Hãy viết chương trình tìm cách dịch chuyển S thành T thỏa mãn đồng thời những điều kiện dưới đây:

- Mỗi phép dịch chuyển chỉ được phép thay đổi một chữ số của số ở bước trước đó (ví dụ nếu S=1033 thì phép dịch chuyển S thành 1733 là hợp lệ);
- Số nhận được cũng là một số nguyên tố có 4 chữ số (ví dụ nếu S=1033 thì phép dịch chuyển S thành 1833 là không hợp lệ, và S dịch chuyển thành 1733 là hợp lệ);
- Số các bước dịch chuyển là ít nhất.

Ví dụ với S = 1033, T = 8179 trong file nguyento.in dưới đây sẽ cho ta file ketqua.out như sau:

nguyento.in	*ketqua.out*
1033 8179	6
	8179 8779 3779 3739 3733 1733 1033

Kế tục thành công với khối lập phương thần bí, Rubik sáng tạo ra dạng phẳng của trò chơi này gọi là trò chơi các ô vuông thần bí. Đó là một bảng gồm 8 ô vuông bằng nhau như Hình 1. Trạng thái của bảng các màu được cho bởi dãy kí hiệu màu các ô được viết lần lượt theo chiều kim đồng hồ bắt đầu từ ô góc trên bên trái và kết thúc ở ô góc dưới bên trái. Ví dụ: trạng thái trong Hình 1 được cho bởi dãy các màu tương ứng với dãy số (1, 2, 3, 4, 5 , 6, 7, 8). Trạng thái này được gọi là trạng thái khởi đầu.

Biết rằng chỉ cần sử dụng 3 phép biến đổi cơ bản có tên là ‘A’, ‘B’, ‘C’ dưới đây bao giờ cũng chuyển được từ trạng thái khởi đầu về trạng thái bất kỳ:

‘A’ : đổi chỗ dòng trên xuống dòng dưới. Ví dụ sau phép biến đổi A, hình 1 sẽ trở thành hình 2:

‘B’ : thực hiện một phép hoán vị vòng quanh từ trái sang phải trên từng dòng. Ví dụ sau phép biển đổi B Hình 1 sẽ trở thành Hình 3.

‘C’ : quay theo chiều kim đồng hồ bốn ô ở giữa. Ví dụ sau phép biến đổi C Hình 1 trở thành Hình 4.

Hình 1

1	2	3	4
8	7	6	5

Hình 2

8	7	6	5
1	2	3	4

Hình 3

4	1	2	3
5	8	7	6

Hình 4

1	7	2	4
8	6	3	5

Ví dụ trạng thái: 2 6 8 4 5 7 3 1 sẽ thực hiện ít nhất là 7 bước như sau:

$$B \rightarrow C \rightarrow A \rightarrow B \rightarrow C \rightarrow C \rightarrow B$$

BÀI 4. GIÁ TRỊ NHỎ NHẤT CỦA XÂU

Cho xâu ký tự S[] bao gồm các ký tự in hoa [A, B, …,Z]. Ta định nghĩa giá trị của xâu S[] là tổng bình phương số lần xuất hiện mỗi ký tự trong xâu. Ví dụ với xâu S[] = "AAABBCD" ta có $F(S) = 3^2 + 2^2 + 1^2 + 1^2 = 15$. Hãy tìm giá trị nhỏ nhất của xâu S[] sau khi loại bỏ K ký tự trong xâu.

Input:

- Dòng đầu tiên đưa vào số lượng test T ($T \leq 100$).
- Mỗi test được tổ chức thành 2 dòng. Dòng thứ nhất ghi lại số K. Dòng thứ 2 ghi lại xâu ký tự S[] có độ dài không vượt quá 10^6.

Output:

- Đưa ra giá trị nhỏ nhất của mỗi test theo từng dòng.

Input	Output
2 0 ABCC 1 ABCC	6 3