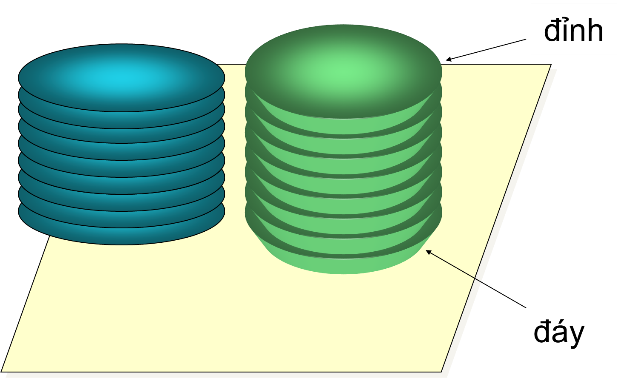
**Stack, Queue và Deque, PRIORITY QUEUE**

**I. Stack**: là một danh sách đặc biệt mà phép thêm vào hoặc loại bỏ một phần tử chỉ thực hiện tại một đầu gọi là đỉnh (Top) của Stack.



Như vậy, stack hoạt động theo cơ chế vào sau ra trước (Last In First Out).

Do đó stack còn được gọi là danh sách kiểu **LIFO**.

**Các thao tác cơ bản trên Stack**

**1. Khai báo thư viện**

#include <stack>

**2. Khai báo biến**

Stack <kiểu dữ liệu> tên\_biến;

Ví dụ: stack<int> mystack;

Dưới đây là một ví dụ về việc khai báo và sử dụng Stack

|  |
| --- |
| #include <bits/stdc++.h >  #include <stack>  using namespace std;  int main()  {  // Empty stack  stack<int> mystack;  mystack.push(0);  mystack.push(1);  mystack.push(2);  while (!mystack.empty())  {  cout << ' ' << mystack.top();  mystack.pop();  }  } |

**3. Các phép toán trên Stack:**

**a. Kiểm tra stack rỗng hay không:**

Hàm empty() được sử dụng để kiểm tra xem bộ chứa ngăn xếp có trống hay không.

**Cú pháp:**

stackname.empty()

Đúng: nếu ngăn xếp trống

Sai: ngược lại

Độ phức tạp là O(1)).

**Đầu vào:** mystack

mystack.empty();

**Đầu ra:** Đúng

**Đầu vào:** mystack = 1, 2, 3

**Đầu ra:** Sai

***Ví dụ***

|  |
| --- |
| #include <bits/stdc++.h >  #include <stack>  using namespace std;  int main()  {  stack<int> mystack;  mystack.push(1);  if (mystack.empty()) {  cout << "True";  }  else {  cout << "False";  }  return 0;  } |

**b. Kích thước ngăn xếp**

Hàm size() được sử dụng để trả về kích thước của bộ chứa ngăn xếp hoặc số lượng phần tử trong bộ chứa ngăn xếp.

**Cú pháp:** stackname.size()

Trả về : Số phần tử trong vùng chứa.

Độ phức tạp là O(1)).

***Ví dụ***

Input : mystack = 0, 1, 2

mystack.size();

Output : 3

Input : mystack = 0, 1, 2, 3, 4, 5

mystack.size();

Output : 6

|  |
| --- |
| #include <bits/stdc++.h >  #include <stack>  using namespace std;  int main()  {  int sum = 0;  stack<int> mystack;  mystack.push(1);  mystack.push(8);  mystack.push(3);  mystack.push(6);  mystack.push(2);  while (mystack.size() > 0) {  sum = sum + mystack.top();  mystack.pop();  }  cout << sum;  return 0;  } |

**c. Thêm phần tử x vào stack:**

**Cú pháp:**  stackname.push(value);

Kết quả: Thêm một phần tử có giá trị giống như phần tử của tham số được truyền vào đầu ngăn xếp.

***Ví dụ:***

Input : mystack

mystack.push(6);

Output : 6

Input : mystack

mystack.push(0);

mystack.push(1);

Output : 0, 1

Code

|  |
| --- |
| #include <bits/stdc++.h >  #include <stack>  using namespace std;    int main()  {  // Empty stack  stack<int> mystack;  mystack.push(0);  mystack.push(1);  mystack.push(2);  while (!mystack.empty()) {  cout << ' ' << mystack.top();  mystack.pop();  }  } |

**d.** **Xem thông tin của phần tử ở đỉnh stack**

- Hàm top() được sử dụng để tham chiếu phần tử trên cùng của ngăn xếp.

* Cú pháp : stackname.top()
* Giá trị trả về: Tham chiếu trực tiếp đến phần tử trên cùng của bộ chứa ngăn xếp.
* Ví dụ:

Input : stackname.push(5);

stackname.push(1);

stackname.top();

Output : 1

Input : stackname.push(5);

stackname.push(1);

stackname.push(2);

stackname.top();

Output : 2

* Code

|  |
| --- |
| #include <bits/stdc++.h >  using namespace std;  int main()  {  stack<int> mystack;  mystack.push(5);  mystack.push(1);  mystack.push(2);  // Stack top  cout << mystack.top();  return 0;  } |

**e.** **Trích thông tin và hủy phần tử ở đỉnh stack**

- Hàm **pop()** được sử dụng để loại bỏ một phần tử khỏi đầu ngăn xếp.

* **Cú pháp:** Stackname.pop()
* Kết quả: Loại bỏ phần tử mới nhất trong ngăn xếp, đó là phần tử trên cùng.

***Ví dụ***

Input : mystack = 0, 1, 2

mystack.pop();

Output : 0, 1

Input : mystack = 0, 1, 2, 3, 4, 5

mystack.pop();

Output : 0, 1, 2, 3, 4

Code

|  |
| --- |
| #include <bits/stdc++.h >  using namespace std;  int main()  {  stack<int> mystack;  mystack.push(1);  mystack.push(2);  mystack.push(3);  mystack.push(4);  // Stack becomes 1, 2, 3, 4  mystack.pop();  mystack.pop();  // Stack becomes 1, 2  while (!mystack.empty()) {  cout << ' ' << mystack.top();  mystack.pop();  }  } |

Output

2 1

**C. BÀI TẬP ÁP DỤNG**

**1. Bài toán 1: STACK**

**1.1. Đề bài**:

Cho 1 dãy số ban đầu không có phần tử và q truy vấn gồm 1 trong 3 dạng sau:

1 u : cho phần tử có giá trị u vào cuối dãy

2 : loại bỏ phần tử ở cuối dãy ( Nếu dãy rỗng thì không làm gì cả )

3 : In ra giá trị ở cuối dãy( Nếu dãy rỗng in ra -1 )

## Giới hạn

q <= 105 , 0<= u <= 109

## Input

Dòng 1: Gồm số nguyên q – số lượng truy vấn

Dòng 2…q+1 : Mỗi dòng ghi 1 trong 3 dạng truy vấn

## Output

Với mỗi truy vấn loại 3 in ra kết quả

## Example

|  |  |
| --- | --- |
| stack.inp | Stack.out |
| 6  1 2  1 3  2  3  2  3 | 2  -1 |

**1.2. Hướng dẫn:**

* Đây là bài toán cơ bản của Stack .
* Để làm bài toán này trước hết ta cần khai báo và sử dụng 1 biến Stack, đặt tên biến là q
* Cú pháp thực hiện lệnh này được thể hiện như sau:

stack < int > q ;

* Để thực hiện thao tác cho phần tử u vào cuối dãy ta dùng lệnh push.

Cụ thể:

scanf( "%d" , &u ) ;

q.push(u) ;

- Để loại bỏ phần tử ở cuối dãy ta dùng lệnh q.pop() , trước đó ta kiểm tra xem stack có rỗng hay không bằng lệnh q.empty().

- Nếu dãy không rỗng ta in ra phần tử cuối cùng dùng lệnh

printf( "%d\n" , q.top() )

**1.3. Chương trình tham khảo:**

|  |
| --- |
| #include<bits/stdc++.h>  using namespace std ;  int n ;  stack < int > q ;  int main(){  #define NAME "STACK"  freopen( NAME".inp" , "r" , stdin ) ;  freopen( NAME".out" , "w" , stdout ) ;  scanf( "%d" , &n ) ;  while(n --){  int type ; scanf( "%d" , &type ) ;  if(type == 1) {  int u ;  scanf( "%d" , &u ) ;  q.push(u) ;  }  if(type == 2 && !q.empty()) q.pop() ;  if( type == 3 ){  if( q.empty() ) printf( "-1\n") ;  else printf( "%d\n" , q.top() ) ;  }  }  return 0 ;  } |

**2. Bài toán 2: CHUYỂN NHỊ PHÂN**

**2.1. Đề bài**:

Nhập vào một số nguyên dương n.

Hãy chuyển n thành mã nhị phân và in chuỗi đó ra màn hình.

## Example

|  |  |
| --- | --- |
| nhiphan.inp | nhiphan.out |
| 13  3 | 1101  11 |

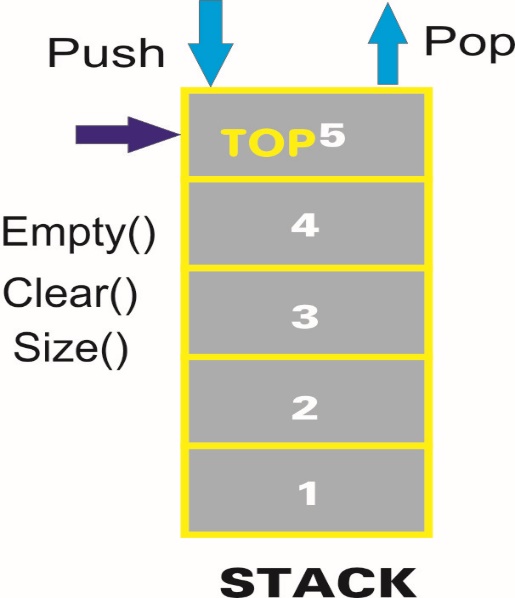
**2.2. Hướng dẫn bài tập:**

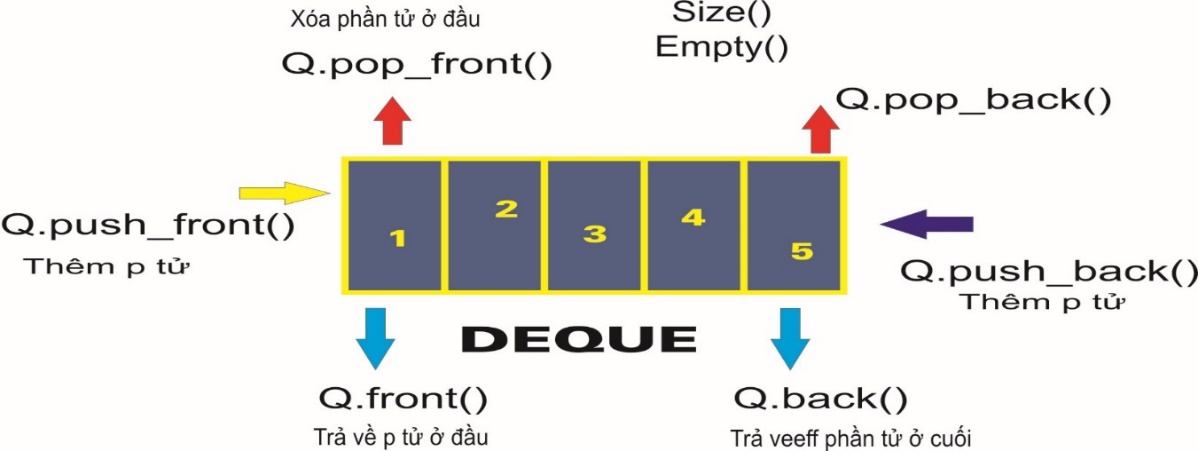
* Để chuyển một số nguyên dương n về hệ nhị phân ta làm như sau:
* Trong khi n > 0 thì ta lấy n chia cho 2 và lưu số dư của phép chia đó lại. Cứ làm vậy đến khi nào n = 0 thì dừng lại.
* Kết quả cần tìm chính là danh sách các số dư được đọc ngược lại, ta sẽ nghĩ ngay đến việc sùng **stack** để lưu các số dư này, sau đó chỉ cần in ra **stack** là được.
* Ví dụ n = 13: thì quy trình biến đổi như sau:
* Khởi tạo st = []
* Với n = 13, lấy 13 chia 2 được 6 dư 1, lưu 1 và st, st = [1].
* với n = 6, lấy 6 chia 2 được 3 dư 0, lưu 0 và st, st = [1, 0].
* với n = 3, lấy 3 chia 2 được 1 dư 1, lưu 0 và st, st = [1, 0, 1].
* Với n = 1, lấy 1 chia 2 được 0 dư 1, lưu 1 và st, st = [1, 0, 1, 1].
* Với n = 0, ta dừng lại và in ra **stack**st.

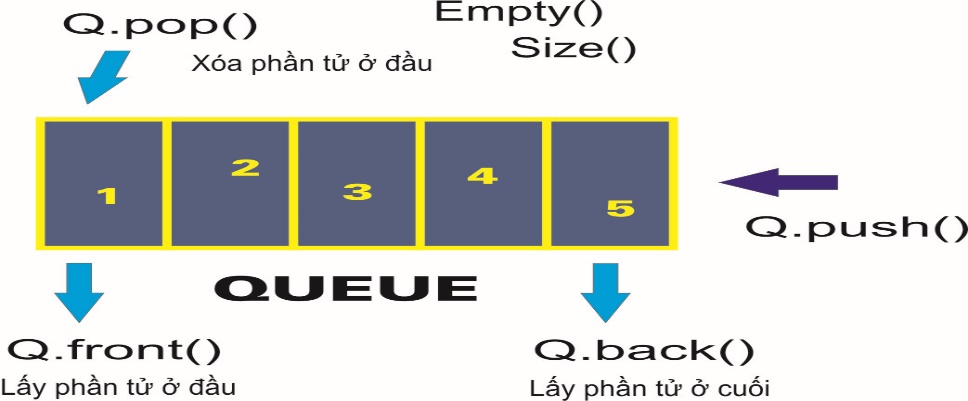
1.3. Chương trình tham khảo:

|  |
| --- |
| #include<bits/stdc++.h>  #include<stack>  using namespace std;  int main(){  stack<int> st;  int n;  cin >> n;  while(n != 0){  st.push(n%2);  n /= 2;  }  while(!st.empty()){  cout << st.top();  st.pop();  }  return 0;  } |

**Hình ảnh tóm tắt các phương thức của stack, queue, deque**

****





***II. Hàng đợi ưu tiên* *Priority queue***

* Priority queue là một loại container adaptor, được thiết kế đặc biệt để phần tử ở đỉnh luôn luôn là phần tử có độ ưu tiên lớn nhất so với các phần tử khác.
* Nó giống như một heap, mà ở đây là heap max, tức là phần tử có độ ưu tiên lớn nhất có thể được lấy ra và các phần tử khác được chèn vào bất kì.
* Khác với queue, priority\_queue không có khái nhiệm front, back mà chỉ có khái nhiệm top tương tự như stack.

|  |  |
| --- | --- |
| **equal\_to** | **Bằng (==)** |
| **not\_equal\_to** | **Không bằng (!=)** |
| **greater** | **Lớn hơn (>)** |
| **less** | **Nhỏ hơn (<)** |
| **greater\_equal** | **Lớn hơn hoặc bằng (>=)** |
| **less\_equal** | **Nhỏ hơn hoặc bằng (<=)** |

**b. Khai báo priority\_queue**

* Trong C++ không có thư viện priority\_queue, do đó, để sử dụng priority\_queue, ta cần khai báo thư viện: #include <bits/stdc++.h>

priority\_queue <int> pq;

* Ta có thể tưởng tượng priority\_queue tương tự như stack, do đó, phần tử có độ ưu tiên lớn nhất sẽ nằm về phía bên phải. Do đó phần tử độ ưu tiên cao nhất là phần tử có giá trị lớn nhất.

priority\_queue <int,vector<int>,greater<int> > myPriorityQueue ;

* Khi sử dụng phép toán greater, phần tử độ ưu tiên cao nhất là phần tử có giá trị nhỏ nhất.

**c. Các phương thức thành viên**

* size(): Trả về số lượng phần tử của priority\_queue
* empty(): Trả về true(1) nếu priority\_queue rỗng, ngược lại là false (0)
* top() : Truy xuất phần tử ở đỉnh priority\_queue (phần tử có độ ưu tiên lớn nhất).
* push (const x): Thêm phần tử có giá trị x vào priority\_queue. Kích thước priority\_queue tăng thêm 1.
* pop (): Loại bỏ phần tử ở đỉnh priority\_queue. Kích thước priority\_queue giảm đi 1.

### Ví dụ: **Bài 07.** Trò chơi dân gian*(Nguồn sưu tầm)*

Ôn lại truyền thống trò chơi dân gian, học sinh các trường THPT đã thành lập ra CLB “Trò chơi dân gian”, mỗi thành viên trong CLB sẽ có trách nhiệm sưu tầm và tổ chức trò chơi đó, Khánh là một thành viên của CLB đồng thời là HSG môn tin với tối chất của một lập trình viên Khánh có một trò chơi như sau:

Có N đống sỏi xếp thành một hàng, đống thứ i có Ai viên sỏi.

Ta có thể ghép hai đống sỏi bất kỳ thành một đống và mất một chi phí bằng 5% tổng hai đống sỏi đó. Hãy tìm cách ghép N đống sỏi này thành một đống với chi phí là nhỏ nhất.

Các bạn hãy Khánh đưa ra phương án chơi tối ưu nhé!

*Input*: vào từ file văn bản DONG.INP có dạng:

* Dòng 1: Số nguyên dương N là số đống sỏi.
* Dòng tiếp theo, ghi N số nguyên dương, tương ứng là số lượng sỏi trong từng đống. Số lượng sỏi không vượt quá 10.000.

*Output*: Đưa ra file văn bản DONG.OUT có dạng:

* Ghi 1 số thực duy nhất là chi phí nhỏ nhất phải trả để ghép N đống sỏi thành 1 đống. Kết quả ghi dưới dạng 2 chữ số sau dấu thập phân.

Example:

|  |  |
| --- | --- |
| **DONG.INP** | **DONG.OUT** |
| 4  10 11 12 13 | 4.60 |
| 2  1 1 | 0.10 |

*Ràng buộc:*

* Có 50% số test còn lại ứng với 40% số điểm của bài có .
* Có 50% số test còn lại ứng với 40% số điểm của bài có .

- Sub 01: Duyệt trâu 2 vòng lặp với độ phứp tạp O(N2).

- Sub 02: với ta dễ dàng giải quyết bằng nhiều cách như sử dụng Segment Tree … Ngoài ra sử dụng priority\_queue với tính chất đặc trưng của hàng đợi ưu tiên bài toán sẽ trở lên dễ dàng hơn rất nhiều.

Chương trình tham khảo:

|  |
| --- |
| #include<bits/stdc++.h>  #define For(i,n,m) for(long i=n;i<=m;i++)  using namespace std;  priority\_queue<long long,vector<long long>,greater<long long> > q;  double kq;  long long n,x,a,b;  main()  {  //freopen("DONG.inp","r",stdin);  // freopen("DONG.out","w",stdout);  scanf("%lld",&n);  For(i,1,n)  {  scanf("%lld",&x);  q.push(x);  }  while(q.size()>1)  {  a=q.top();  q.pop();  b=q.top();  q.pop();  kq+=a+b;  q.push(a+b);  }  printf("%0.2f",kq/20);  } |

*Cảm nhận:* Bài toán rất dễ hiểu khi sử dụng priority\_queue để giải quyết, ĐPT của bài toán là O(nlog(n)).