Programming Technology: Vector of STL C++

Tran Viet Khoa
tvkhoa.husc@gmail.com

Hue University— Mar 2020

Mục lục

1 Giới thiệu

- Vector là một lớp mẫu trong STL của C ++. Lớp vectơ là lớp chứa (container) lưu trữ dữ liệu kiểu danh sách tuyến tính theo mô hình cấp phát động nên linh hoạt hơn mảng tĩnh.
- Lập trình với vector rất hiệu quả vì là lớp mẫu nên có rất nhiều phương thức dựng sẵn làm việc được với nhiều kiểu dữ liệu khác nhau.
- Có thể tạo các mảng động mà không cần phải cấp phát và thu hồi vùng nhớ bằng cách sử dụng toán tử new và delete khi dùng với con trỏ.
- Vector là một lớp chứa cung cấp khả năng sử dụng mảng mềm dẻo, có kiểm soát phạm vi truy cập tốt với kích thước tùy ý.

Bài viết này sẽ đề cập đến các vấn đề sau của vector, bao gồm:

- Định nghĩa và Khai báo
- Các phương thức của vector
- Một số thuật toán trên <algorithm> dùng cho vector.

2 Định nghĩa và khai báo

2.1 Định nghĩa

Về mặt ý nghĩa vector hoàn toàn tương tự như mảng được cấp phát động thông qua con trỏ, tuy nhiên nó được tổng quát hóa bằng khái niệm lớp (class) của lập trình hướng đối tượng và bổ sung thêm kỹ thuật lập trình mẫu. Người lập trình sẽ dùng nó cực kỳ linh hoạt với các ưu điểm:

- Không quan tâm đến bộ nhớ (về việc cấp phát và thu hồi).
- Có thể lưu cho bất kỳ đối tượng dữ liệu nào, số nguyên, số thực, chuỗi ký tự, cặp (pair), cấu trúc (struct) nhờ vào thiết kế mẫu.
- Có thể sử dụng được các thuật toán mẫu (thư viện <algorithm>) để xử lý cho bất kỳ đối tượng nào lưu trữ trên vector. Ví dụ, bạn có thể sắp xếp một danh sách các số nguyên, danh sách họ tên bằng hàm sort().
- Được hỗ trợ bởi lớp duyệt (Iterator) cho nên việc bắt lỗi truy cập phạm vi chặt chẽ hơn.

Để lưu trữ cho đối tượng nào trên vector ta cần xác định chính xác đối tượng đó để khai báo cho đúng. Bỏ qua cách dùng template<typename T> [1], vì bài viết dành cho người bắt đầu nên khi sử dụng vector ta cần biết chính xác đối tượng cần lưu trữ. Và C++ cung cấp cú pháp giúp ta định nghĩa được vector lưu trữ như sau:

cú pháp:

```
typedef vector<int> VI;
typedef pair<int,int> PII;
typedef vector<PPI> VP;
typedef vector <VI> Matrix;
```

Trong đó:

1. Định nghĩa kiểu VI là vector chứa các số nguyên.
2. Định nghĩa kiểu bộ (pair) hai số nguyên PII.
3. Định nghĩa kiểu VP là vector chứa các bộ.
4. Định nghĩa kiểu Matrix là vector của các vector (hay nói cách khác là mảng của các mảng).

Bạn không cần phần định nghĩa như trên mà khai báo trực tiếp cũng được. Tuy nhiên, nếu định nghĩa trước thì các khai báo sau này sẽ thống nhất hơn, dễ dò tìm và sửa lỗi hơn.

[1] Chi tiết sẽ trình bày bài sau

2.2 Khai báo và khởi tạo

cú pháp:

```
<TYPE> vecName;
```

Trong đó:

- TYPE: Là các kiểu dữ liệu vector đã định nghĩa ở trên.
- vecName: Tên biến vector.
- Việc khai báo này thường khai trong hàm main(), trong hàm hay là tham số của hàm.
- Vì linh động nên ta có thể gán, thay đổi số lượng bất kỳ lúc nào, cũng như việc điền dữ liệu khởi tạo ban đầu.

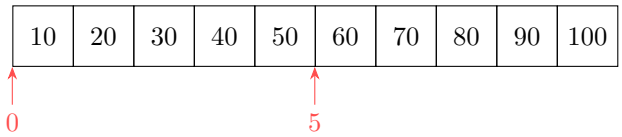
Ví dụ: Khai báo một vector có 4 phần tử với giá trị mỗi phần tử là 100.

```
VI v(4,100);
```

Ví dụ: Đoạn mã sau thể hiện trong bộ nhớ như hình vẽ.

index.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
int main()
{
   VI a;
   for (int i = 1; i <= 10; i++)    a.push_back(i * 10);
   cout << "\nReference operator a[] : a[0] = " << a[0];
   cout << "\nat : a.at(5) = " << a.at(5);
   return 0;
}
```



Hình 1: vector và chỉ số

Chương trình trên in ra hai giá trị 10, 60 của vector tại hai vị trí index là 0 và 5.
Ví dụ: Định nghĩa, khai báo và khởi tạo một ma trận hai $A_{N.M}$ với các phần tử của ma trận bằng -1.

matrix.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef vector < vector <int> > Matrix;
int main()
{
   int N, M;   cin >> N >> M;
   Matrix  A;
   A.assign(N , vector <int> (M, -1));
   for (int i = 0; i < N; ++i) {
      for (int j = 0; j < M; ++j)
         cout << A[i][j] << " ";
      cout << '\n';
   }
   return 0;
}
```

3 Các phương thức của vector

3.1 Truy cập phần tử

1. Dùng phép toán tải bội [idx]: Trả về phần tử tại vị trí chỉ mục *idx*, ví dụ v[0] trả về phần tử đầu tiên của vector.
2. Hàm at(idx): Trả về phần tử tại vị trí chỉ mục *idx* tương tự phép toán trên.
3. Hàm front(): Trả về phần tử đầu vector.
4. Hàm back(): Trả về phần tử cuối vector.

Xét ví dụ với mã nguồn sau:

access.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
int main()
{
   VI a;
   //Điền dữ lệu vào vector theo thứ tự 10, 20, .., 100.
   for (int i = 1; i <= 10; i++)    a.push_back(i * 10);
   cout << "\nReference operator a[] : a[1] = " << a[1];
   cout << "\nat : a.at(5) = " << a.at(5);
   cout << "\nfront() : a.front() = " << a.front();
   cout << "\nback() : a.back() = " << a.back();
   return 0;
}
```

Output:
Reference operator a[] : a[1] = 20
at : a.at(5) = 60
front() : a.front() = 10
back() : a.back() = 100

Chú ý: Mặc dù làm việc với vector nhưng thói quen sử dụng phép toán [] như ở mảng vẫn được dùng nhiều ở vector.

3.2 Số lượng, kích thước phần tử

Việc quản lý số phần tử như kiểm tra vector rỗng hay không, kích thước bao nhiêu, số lượng tối đa là rất quan trọng. Lý do vector hoạt động theo nguyên lý động.

1. Hàm size(): Trả về số lượng phần tử hiện có của vector.
2. Hàm resize(n): Hàm thay đổi kích thước với n phần tử.
3. Hàm empty(): Kiểm tra vector có rỗng hay không?

Xét ví dụ với mã nguồn sau:

capacity.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
int   main()
{
   VI v;                          // Khai báo vector.
   VI::iterator  it;              // Khai báo iterator.
   for (int i=1;i<=5;i++) v.push_back(i);  // v={1,2,3,4,5}
   cout <<  v.size()  <<  endl;        // In ra  5
   v.push_back(9);                     // v={1,2,3,4,5,9}
   cout <<  v.size()  <<  endl;        // In ra  6
   v.resize(20);                  // Đổi kích thước mảng lên 20.
   cout  << v.size()<<endl;          //In ra 20.
   v.clear();                          // v={}
   cout <<  v.empty() <<  endl;       // In ra 1 (vector rỗng).
   for (int i=1;i<=5;i++)
      v.push_back(i);             // v={1,2,3,4,5}
   v.pop_back();                       // v={1,2,3,4}
   cout <<  v.size()  <<  endl;        // In ra  4
   return 0;
}
```

Output:

```
5
6
20
1
4
```

Chú ý: Hàm resize() là mở rộng hoặc thu hẹp vector, đơn vị tính là số phần tử chứ không phải là byte nhớ.

3.3 Duyệt vector

1. Hàm begin(): Trả về con trỏ lặp (iterator) trỏ đến phần tử đầu vector.
2. Hàm end(): Trả về con trỏ lặp (iterator) trỏ đến phần tử cuối vector.

3. Hàm rbegin(): Trả về con trỏ lặp thứ tự ngược (iterator) trỏ đến phần tử cuối vector.
4. Hàm rend(): Trả về con trỏ lặp thứ tự ngược (iterator) trỏ đến phần tử đầu vector.

Xét ví dụ như mã nguồn sau:

traverse.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
int   main()
{
   VI v;                          // Khai báo vector.
   VI::iterator  it;              // Khai báo iterator.
   VI::reverse_iterator  rit;     // Khai báo iterator ngược.
   for (int i = 1; i <= 5; i++) v.push_back(i);

   cout << "Output of begin and end: ";
   for (auto i = v.begin(); i != v.end(); ++i) {
      cout << *i << " ";
   }

   cout << "\nOutput of begin and end by iterator: ";
   for (it=v.begin();it!=v.end();++it){
      cout <<  *it  <<  " " ;
   }

   cout << "\nOutput of rbegin and rend: ";
   for (auto ir = v.rbegin(); ir != v.rend(); ++ir){
      cout << *ir << " ";
   }

   cout << "\nOutput of begin and end by reverse iterator: ";
   for (rit=v.rbegin();rit!=v.rend();rit++){
      cout <<  *rit  <<  " ";
   }

   return 0;
}
```

Output:
Output of begin and end: 1 2 3 4 5
Output of begin and end by iterator: 1 2 3 4 5
Output of rbegin and rend: 5 4 3 2 1
Output of begin and end by reverse iterator: 5 4 3 2 1

Chú ý Không nên viết for (i=0;i<=v.size()-1;i++) Vì nếu vector v rỗng, v.size() là kiểu unsigned int, nên v.size()-1 sẽ bằng $2^{32} - 1$

3.4 Các hàm lưu trữ, sửa đổi

1. Hàm assign() – It assigns new value to the vector elements by replacing old ones
2. Hàm push_back(x): Thêm phần tử có giá trị x vào cuối vector.

3. Hàm pop_back(): Loại bỏ phần tử cuối ra khỏi vector.
4. Hàm insert(): Có 3 mẫu khác nhau. Chèn phần tử có giá trị x vào trước vị trí position. Chèn n phần tử có giá trị x vào trước vị trí position. Chèn vào trước vị trí position tất cả các phần tử trong nửa khoảng [a,b) của một vector khác.
5. Hàm erase(position): Xóa phần tử ở vị trí position.
6. Hàm clear(): Xóa vector, vector rỗng.

Xét ví dụ như mã nguồn sau:

modify.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
int   main()
{

   VI v;   // Khai báo vector.
   // Điền dữ liệu cho vector v với 5 phần tử có giá trị 10.
   v.assign(5, 10);
   // Bổ sung phần tử 15 vào cuối vector.
   v.push_back(15);
   cout << "The vector elements are: ";
   for (auto i = v.begin(); i != v.end(); ++i) {
      cout << *i << " ";
   }
   // Xóa phần tử cuối.
   v.pop_back();
   cout << "\nThe vector elements are: ";
   for (auto i = v.begin(); i != v.end(); ++i) {
      cout << *i << " ";
   }
   // Chèn phần tử 5 vào đầu vector.
   v.insert(v.begin(), 5);
   cout << "\nThe vector elements are: ";
   for (auto i = v.begin(); i != v.end(); ++i) {
      cout << *i << " ";
   }
   // Xóa phần tử đầu vector.
   v.erase(v.begin());
   cout << "\nThe vector elements are: ";
   for (auto i = v.begin(); i != v.end(); ++i) {
      cout << *i << " ";
   }
   // Xóa toàn bộ vector.
   v.clear();
   cout << "\nVector size after erase(): " << v.size();
   return 0;
}
```

Output:

```
The vector elements are: 10 10 10 10 10 15
The vector elements are: 10 10 10 10 10
The vector elements are: 5 10 10 10 10 10
The vector elements are: 10 10 10 10 10
Vector size after erase(): 0
```

4 Các thuật toán áp dụng với vector

Sức mạnh của C++ đến từ STL, viết tắt của Standard Template Library - một thư viện template cho C++ với những cấu trúc dữ liệu cũng như giải thuật được xây dựng tổng quát mà vẫn tận dụng được hiệu năng và tốc độ của C.

STL bao gồm các thuật toán cơ bản: tìm min, max, tính tổng, sắp xếp (với nhiều thuật toán khác nhau), thay thế các phần tử, tìm kiếm (tìm kiếm thường và tìm kiếm nhị phân), trộn. Toàn bộ các tính năng nêu trên đều được cung cấp dưới dạng template nên việc lập trình luôn thể hiện tính khái quát hóa cao. Nhờ vậy, STL làm cho ngôn ngữ C++ trở nên trong sáng hơn nhiều."

4.1 Cơ bản

Để hiểu rõ và sử dụng tốt các thuật toán STL cung cấp bạn cần nắm bắt các khái niệm sau:

4.1.1 Cấu trúc pair

Là cách tổ chức gộp dữ liệu dạng đơn giản như cặp (hai phần tử khác nhau) thành một.

Xét ví dụ sau:

pair.cpp

```cpp
#include <iostream>
int main()
{
    std::pair<int, std::string> p = std::make_pair(5, "srcmake");
    std::cout << p.first << " " << p.second << std::endl;
    return 0;
}
```

Output:
5 srcmake

Chú ý Gần như là một cấu trúc (struct), tuy nhiên pair là dạng mini, là một cặp với hai trường là first và second cho hai phần tử được ghép cặp với nhau.

4.1.2 Biểu thức Lambda

Từ chuẩn C++11 trở đi, một biểu thức lambda, gọi tắt là lambda, là một cách tiện lợi để định nghĩa hàm không tên ngay tại nơi nó được gọi hoặc được truyền vào như một đối số của hàm khác. Lambda thường được sử dụng để gói gọn một vài dòng code được truyền vào những thuật toán, hoặc những hàm đồng bộ (asynchronous methods).

Hiểu theo nghĩa đen đó là một cách viết khác của hàm mà không có tên (anonymous function), tăng tính đa dạng cũng như gần gũi với một số ngôn ngữ lập trình hàm hay dùng. Xét biểu thức Lambda cho việc kiểm tra xem một số i có dương hay không.

```cpp
auto srcLambda = [](int i) {
    return i > 2;
};
```

Trong đó,

- Biểu thức Lambda có tên là **srcLambda**.

- Kiểu trả về là **auto** phù hợp với việc áp dụng cho các đối tượng hay nói cách khác là có thể trả về bất kỳ kiểu dữ liệu nào giống như hàm thông thường.

- dấu ngoặc [] gọi là Mệnh đề bắt giữ (capture clause). Một biểu thức lambda có thể khai báo thêm biến mới bên trong nó (từ chuẩn C++14 trở đi), và nó còn có thể truy cập, hoặc tham chiếu đến những biến bên trong khối lệnh chứa nó. Một lambda luôn bắt đầu với cặp ngoặc vuông [ ], và những biến cần được bắt giữ (cách dùng như tham chiếu)sẽ khai báo bên trong đó. Theo quan sát phần này ít dùng.

Xét ví dụ viết biểu thức Lambda cho tính tổng hai đối tượng số, chuỗi (ghép chuỗi), ta có mã nguồn sau:

lambda.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;
int main()
{
   // Biểu thức lambda tổng quát.
   auto sum = [](auto a, auto b) {
      return a + b;
   };
   cout << sum(5, 6) << endl;        // Tổng hai số nguyên.
   cout << sum(2.0, 6.5) << endl;    // Tổng hai số thực.
   //Nối xâu.
   cout << sum(string("oj."), string("hueuni.edu.vn")) << endl;
return 0;
}
```

Output:
11
8.5
oj.hueuni.edu.vn

4.2 Các hàm xử lý dữ liệu

4.2.1 Hàm all_of

Bài tập 1: Điểm số v1

Lớp Bánh cuối năm điểm tổng kết tương đối tốt, nhiều bạn 9, 10. Tuy nhiên, Cô giáo cẩn thận kiểm tra xem có phải tất cả các bạn có điểm lớn hơn bằng 5 hay không? Anh/chị sinh viên viết chương trình kiểm tra xem các phần tử trong dãy $A = \{a_1, a_2, \ldots, a_n\}$ có các phần tử có giá trị lớn hơn bằng 5 hay không?

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng.

Output

- In **Yes** nếu thỏa điều kiện và **No** nếu ngược lại.

Samples

input	output
5 5 3 9 9 10	No

Sừ dụng hàm **all_of()** để duyệt qua các phần tử và kiểm tra điều kiện. Hàm **all_of()** kiểm tra mọi phần tử trong phạm vi có thỏa mãn điều kiệm hay không?

- Tham số của hàm là phạm vi tính từ đầu đến cuối vector và biểu thức lambda.
- Kết quả trả về true nếu tất cả các phần tử đều thỏa biểu thức lambda và false nếu ngược lại.

Mã nguồn:

allOf.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
auto lambda = [](int i) { return i>=5; };
int main() {
   int n;   cin >> n ;
   vector<int> v(n);
   for (int i = 0; i < n; i++) cin >> v[i];
   bool allGreaterThanFive   = all_of(v.begin(), v.end(), lambda);
   if(allGreaterThanFive  == 1)
      cout<< "Yes" ;
   else cout<<"No";
return 0;
}
```

Kỹ thuật Định nghĩa lại phạm vi của vector theo cú pháp #define all(x) (x).begin(),(x).end(). Và sử dụng cho hàm như sau all_of(all(v), lambda).

4.2.2 Hàm any_of

Bài tập 2: Điểm số v2

Lớp Bánh cuối năm điểm tổng kết không được tốt vì có nhiều bạn ham chơi. Tuy nhiên, Cô giáo chỉ cần có bạn có điểm 10 là hài lòng và nhờ Anh/chị sinh viên viết chương trình kiểm tra xem các phần tử trong dãy $A = \{a_1, a_2, \ldots, a_n\}$ có phần tử nào có điểm 10 hay không?

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng.

Output

- In **Yes** nếu chỉ cần có một phần tử bằng 10 và **No** nếu ngược lại.

Samples

input	output
5 5 3 9 9 10	Yes

Sừ dụng hàm **any_of()** để duyệt qua các phần tử và kiểm tra điều kiện. Hàm **any_of()** kiểm tra mọi phần tử trong phạm vi có thỏa mãn điều kiệm hay không?

- Tham số của hàm là phạm vi tính từ đầu đến cuối vector và biểu thức lambda.
- Kết quả trả về true nếu bất kỳ phần tử nào thỏa biểu thức lambda và false nếu ngược lại.

Mã nguồn:

anyof.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
auto lambda = [](int i) { return i ==10; };
int main() {
   int n;   cin >> n ;
   vector<int> v(n);
   for (int i = 0; i < n; i++) cin >> v[i];
   // Kiểm tra điều kiện lambda?
   bool anyEqualTen = any_of(v.begin(), v.end(), lambda);
   if(anyEqualTen  == 1) cout<< "Yes";
   else cout<<"No";
   return 0;
}
```

4.2.3 Hàm none_of

Bài tập 3: Điểm số v3

Lớp Bánh cuối năm điểm tổng kết không được tốt vì có nhiều bạn ham chơi. Tuy nhiên, Cô giáo rất thất vọng khi Hiệu trưởng bảo rằng tất cả đều dưới 10. Anh/chị sinh viên viết chương trình kiểm tra xem các phần tử trong dãy $A = \{a_1, a_2, \ldots, a_n\}$ phải vậy không?

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng.

Output

- In **Yes** nếu tất cả các bạn đều dưới 10 và **No** nếu chỉ cần có một bạn có điểm 10.

Samples

input	output
5 5 3 9 9 10	No

Sừ dụng hàm **none_of()** để duyệt qua các phần tử và kiểm tra điều kiện. Hàm **none_of()** kiểm tra mọi phần tử trong phạm vi có thỏa mãn điều kiệm hay không?

- Tham số của hàm là phạm vi tính từ đầu đến cuối vector và biểu thức lambda.
- Kết quả trả về true nếu tất cả các phần tử đều không thỏa biểu thức lambda và false nếu bất kỳ phần tử nào đúng điều kiện.

Mã nguồn:

noneof.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
auto lambda = [](int i) { return i >9; };
int main() {
   int n;   cin >> n ;
   vector<int> v(n);
   for (int i = 0; i < n; i++) cin >> v[i];
   bool noneGreaterThanNice = none_of(v.begin(), v.end(), lambda);
   if(noneGreaterThanNice == 1) cout<< "Yes" ;
   else cout<<"No";
   return 0;
}
```

4.2.4 Hàm find_if

Bài tập 4: Điểm số v4

Lớp Bánh cuối năm điểm tổng kết tương đối tốt, nhiều bạn 9, 10. Tuy nhiên, Cô giáo cẩn thận muốn tìm xem có học sinh nào bị điểm 5 hay không? Anh/chị sinh viên viết chương trình kiểm tra xem các phần tử trong dãy $A = \{a_1, a_2, \ldots, a_n\}$ có các phần tử có giá trị bằng 5 hay không?

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng.

Output

- In **Yes** nếu thỏa và **No** nếu ngược lại.

Samples

input	output
5 5 3 9 9 10	Yes

Sừ dụng hàm **find_if()** tìm phần tử đầu tiên thỏa điều kiện.

- Tham số của hàm là phạm vi tính từ đầu đến cuối vector và biểu thức lambda.
- Trả về con trỏ (iterator) đến phần tử tìm thấy đầu tiên trong vector.

Mã nguồn:

findif.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
auto lambda = [](int i) { return i==5; };
int main() {
   int n;   cin >> n ;
   vector<int> v(n);
   for (int i = 0; i < n; i++) cin >> v[i];
   vector<int>::iterator it = find_if(v.begin(), v.end(), lambda);
   if(it != v.end())
      cout << "Yes"<< endl;
   else
      cout << "No" << endl;
   return 0;
}
```

Thách thức Hãy in vị trí của phần tử vừa tìm thấy trong vector.

4.2.5 Hàm count_if

Bài tập 5: Điểm số v5

Lớp Bánh cuối năm điểm tổng kết tương đối tốt, nhiều bạn 9, 10. Cô giáo rất vui và muốn xem tỷ lệ 10 điểm chiếm bao nhiêu bạn. Anh/chị sinh viên viết chương trình đếm xem các phần tử trong dãy $A = \{a_1, a_2, \ldots, a_n\}$ có các phần tử có giá trị bằng 10 hay không?

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng.

Output

- In ra số bạn điểm 10 trên tổng số của lớp.

Samples

input	output
5 5 3 9 9 10	1/5

Sừ dụng hàm **count_if()** đếm xem có bao nhiêu phần tử thỏa điều kiện.

- Tham số của hàm là phạm vi tính từ đầu đến cuối vector và biểu thức lambda.
- Trả về số phần tử thỏa điều kiện đếm

Mã nguồn:

countif.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
auto lambda = [](int i) { return i ==9; };
int main() {
   int n;   cin >> n ;
   vector<int> v(n);
   for (int i = 0; i < n; i++) cin >> v[i];
   int count = count_if(v.begin(), v.end(), lambda);
   cout<<count<<"/"<<n<<endl;
   return 0;
}
```

4.3 Các hàm sao chép, sửa dữ liệu

4.3.1 Hàm copy_if

Bài tập 6: Điểm số v6

Lớp Bánh cuối năm điểm tổng kết tương đối tốt, nhiều bạn 9, 10. Cô giáo rất vui và lập một danh sách chỉ những bạn có điểm 9, 10. Anh/chị sinh viên viết chương trình sao chép các phần tử trong dãy $A = \{a_1, a_2, \ldots, a_n\}$ có các phần tử có giá trị bằng 9, 10.

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng.

Output

- Sao chép và in ra số bạn điểm 9, 10 ở bản ghi điểm mới.

Samples

input	output
5 5 3 9 9 10	9 9 10

Sừ dụng hàm **copy_if()** để copy các phần tử thỏa điều kiện lưu vào một vector mới

- Tham số của hàm là phạm vi toàn bộ của vector thứ nhất, tham số thứ hai là con trỏ đầu của vector thứ hai, cuối cùng là biểu thức lambda.
- Trả về con trỏ trỏ đến địa chỉ sau phần tử cuối của vector thứ hai.

Mã nguồn:

copyif.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define all(x) (x).begin(),(x).end()
auto lambda = [](int i) { return i >=9; };
int main() {
   int n;   cin >> n ;
   // vector thứ nhất chứa bảng điểm tổng.
   vector<int> v1(n);
   for (int i = 0; i < n; i++) cin >> v1[i];
   // Đếm xem có bao nhiêu bạn có điểm 9, 10.
   int m = count_if(all(v1), lambda);
   // Khai báo bảng điểm mới với số lượng m.
   vector<int> v2(m);
   //Copy dữ liệu.
   copy_if(all(v1), v2.begin(), lambda);
   //In bảng điểm mới.
   for_each(all(v2), [](int i) { cout << i << " "; });
   return 0;
}
```

4.3.2 Hàm transform

Bài tập 7: Điểm số v7

Sau khi có điểm, hiệu trưởng bảo cô giáo lập bảng điểm theo cách tính mới là xếp hạng điểm bằng chữ như D, D+, C, C+, B, B+, và A tùy theo thang điểm 10 của học sinh. Anh/chị sinh viên viết chương trình chuyển giúp cô giáp cho các phần tử trong dãy $A = \{a_1, a_2, \ldots, a_n\}$, theo quy tắc:

- A (8.5- 10) Giỏi.
- B+ (8.0 - 8.4) Khá giỏi.
- B (7.0 - 7.9) Khá
- C+ (6.5 - 6.9) Trung bình khá
- C (5.5 - 6,4) Trung bình.
- D+ (5.0 - 5.4) Trung bình yếu.
- D (4.0 - 4.9) Yếu.

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số thực double a_i của dãy, các số cách nhau ký tự trắng.

Output

- In ra danh sách với thang điểm mới tương ứng.

Samples

input	output
5 5 3 9 9 10	D+ D A A A

Sừ dụng hàm **transform()** để biến đổi, hàm này có thể xử lý trên vector hiện hành hoặc lưu vào vector khác. Đầu vào có thể từ một hoặc nhiều vector.

- Tham số của hàm là phạm vi toàn bộ của vector thứ nhất, tham số thứ hai là con trỏ đầu của vector thứ hai nếu có, tham số thứ ba là vector lưu, cuối cùng là biểu thức lambda.
- Trả về con trỏ trỏ đến địa chỉ sau phần tử cuối của vector lưu.

Mã nguồn:

transform.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define all(x) (x).begin(),(x).end()
auto lambda = [](double i) {
   if (i<=4.9) return "D";
   else if (i<=5.4)return "D+";
      else if (i<=6.4) return "C";
         else if (i<=6.9) return "C+";
            else if (i<=7.9) return "B";
               else if (i<=8.4) return "B+";
                  else return "A";
};
int main() {
   int n;   cin >> n ;
   vector<double> v1(n);
   vector<string > v2(n);
   for (int i = 0; i < n; i++) cin >> v1[i];
   transform(all(v1), v2.begin(), lambda);
   for_each(all(v2), [](string str) { cout << str << " "; });
   return 0;
}
```

Thách thức

Các cô của các lớp khác thấy cô Bánh làm được nên đề nghị hiệu trưởng yêu cầu cô Bánh làm giúp bảng điểm cho toàn khối 5. Đầu vào là từng bảng của mỗi lớp (2 lớp), đầu ra là bảng điểm toàn khối.

4.3.3 Hàm unique_copy

Bài tập 8: Điểm số v8

Lớp Bánh cuối năm điểm tổng kết với điểm thi đồng đều cho nên có rất nhiều bạn trùng điểm nhau. Cô giáo muốn lập một bảng điểm không có trùng nhau. Anh/chị sinh viên viết chương trình sao chép các phần tử trong dãy $A = \{a_1, a_2, \ldots, a_n\}$ sao cho không có điểm trùng lặp.

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng.

Output

- Sao chép và in ra các bạn điểm không trùng nhau ghi vào bảng điểm mới.

Samples

input	output
5 5 3 9 9 10	5 3 9 10

Sử dụng hàm **unique_copy()** để copy các phần tử thỏa điều kiện lưu vào một vector mới, bản lưu đầu không bị thay đổi, ví dụ $v_1 = \{2, 2, 3, 3, 4, 4, 4, 3\} -> v2 = \{2, 3, 4, 3\}$.

- Tham số của hàm là phạm vi toàn bộ của vector thứ nhất, tham số thứ hai là con trỏ đầu của vector thứ hai cần lưu.
- Trả về con trỏ trỏ đến địa chỉ sau phần tử cuối của vector thứ hai.

Mã nguồn:

uniquecopy.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define all(x) (x).begin(),(x).end()
int main() {
   int n;   cin >> n ;
   vector<int> v1(n), v2(n);
   for (int i = 0; i < n; i++) cin >> v1[i];
   //Khai báo biến lặp để giữ địa chỉ sau phần tử cuối.
   vector<int>::iterator ip;
   ip = std::unique_copy(all(v1), v2.begin());
   // Thay đổi kích thước cho v2, vì nếu có lặp .
   // kích thước v2<v1 và các phần tử cuối đều = 0.
   v2.resize(std::distance(v2.begin(), ip));
   // In v2.
   for_each(all(v2), [](int i) { cout << i << " "; });
   return 0;
}
```

Gợi nhớ Sử dụng nhiều biến lặp (iterator) giúp bạn có cái nhìn về con trỏ khá tốt, để sau này dễ tiếp thu danh sách liên kết (Simple List, Double List,..)

4.4 Sắp xếp và tìm kiếm

4.4.1 Hàm sort

Bài tập 9: Điểm số v9

Chuẩn bị cho lễ tổng kết Cô giáo muốn lập một bảng điểm bằng cách sắp có thứ tự theo điểm số. Anh/chị sinh viên viết chương trình sắp các phần tử trong dãy $A = \{a_1, a_2, \ldots, a_n\}$ theo thứ tự.

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng.

Output

- In ra dãy theo hai thứ tự: Tăng và giảm

Samples

input

```
5
5 3 9 9 10
```

output

```
3 5 9 9 10
10 9 9 5 3
```

Sử dụng hàm **sort()** để sắp thứ tự

- Tham số của hàm là phạm vi toàn bộ của vector sắp, có thể sử dụng hàm so sánh bằng nhiều cách khác nhau.
- Trả về void.

Mã nguồn:

sort.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define all(x) (x).begin(),(x).end()
int main()
{
   int n;   cin >> n ;
   vector<int> v(n);
   for (int i = 0; i < n; i++) cin >> v[i];
   // Sử dụng toán tử mặc định <, sắp tăng.
   sort(all(v));
   for (auto a : v) {
      cout << a << " ";
   }
   cout << '\n';
   // Dùng hàm lớn hơn (greater) để sắp giảm.
   sort(all(v), greater<int>());
   for (auto a : v) {
      cout << a << " ";
   }
   cout << '\n';
   // Sắp bằng hàm tự định nghĩa, do dùng phép toán < nên sắp tăng.
   struct {
      bool operator()(int a, int b) const {
         return a < b;
      }
   } customLess;
   sort(all(v), customLess);
   for (auto a : v) {
      cout << a << " ";
   }
   cout << '\n';
   // Sắp bằng biểu thức lambda, sắp giảm (>).
   sort(all(v), [](int a, int b) {  return a > b; });
   for (auto a : v) {
      cout << a << " ";
   }
   cout << '\n';
   return 0;
}
```

Thông tin

Sử dụng hàm **sort()** trên hoàn toàn tương tự cho mảng (array), một cấu trúc được dùng nhiều trong các bài toán thi đấu. Hàm **sort()** của STL có độ phức tạp $O(nlogn)$.

4.4.2 Hàm is_sorted

Bài tập 10: Điểm số v10

Sau khi Bánh sắp thứ tự bảng điểm cô giáo vẫn muốn kiểm tra xem bảng điểm đã có thứ tự hay chưa. Anh/chị sinh viên viết chương trình kiểm tra xem các phần tử trong dãy $A = \{a_1, a_2, \ldots, a_n\}$ đã có thứ tự hay không.

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng.

Output

- In ra **Yes** nếu có thứ tự tăng và **No** nếu ngược lại.

Samples

input	output
5 5 3 9 9 10	No

Sử dụng hàm **is_sorted()** để kiểm tra thứ tự.

- Tham số của hàm là phạm vi toàn bộ của vector cần kiểm tra.
- Trả về true nếu đã sắp tăng và false nếu ngược lại.

Mã nguồn:

issorted.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define all(x) (x).begin(),(x).end()
int main()
{
   int n;   cin >> n ;
   vector<int> v(n);
   for (int i = 0; i < n; i++) cin >> v[i];
   bool isSorted = std::is_sorted(all(v));
   if (isSorted) cout<<"Yes"; else cout << "No";
   return 0;
}
```

Thách thức

Từ bản GNU C++ 17, thuật toán **rank()** được bổ sung để sắp vị thứ các phần tử trong vector. Các phiên bản trước (14, 11,..) chưa có. Bằng phong cách lập trình STL bạn hãy viết thuật toán trên.

4.4.3 Hàm binary_search

Trả về giá trị true nếu phần tử tồn tại (tìm thấy) và false nếu ngược lại. Thuật toán được sử dụng với vector được sắp thứ tự tăng.

- Tham số của hàm là phạm vi toàn bộ của vector cần tìm.
- Trả về true nếu đã tìm thấy và false nếu ngược lại.

Mã nguồn minh họa:

bsearch.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define all(x) (x).begin(),(x).end()
int main()
{
   int n;   cin >> n ;
   vector<int> v(n);
   for (int i = 0; i < n; i++) cin >> v[i];
   //Sắp dữ liệu, mất chi phí O(nLogn).
   sort(all(v));
   int x; cin >>x;
   bool xExists = binary_search(all(v), x); // true
   if (xExists) cout<<"Yes"; else cout << "No";
   return 0;
}
```

Thông tin bổ sung

Còn rất nhiều hàm mà thư viện <algorithm> cung cấp để xử lý với các đối tượng trên lớp chứa vector. Các bạn cần tìm hiểu thêm và vận dụng vào bài toán của mình.

5 Một vài ứng dụng của vector và thuật toán

5.1 Đếm số nhỏ hơn

Bài tập 11: Đếm số nhỏ hơn

Cho dãy số $A = \{a_1, a_2, \ldots, a_n\}$. Đối với mỗi số a_i trong dãy hãy đếm xem có bao nhiêu số của A nhỏ hơn nó.

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $2 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng.

Output

- In ra dãy kết quả trên một dòng, các phần tử cách nhau ký tự trắng.

Samples

input	output
5 8 1 2 2 3	4 0 1 1 3

Note

$a_1 = 8$ lớn hơn 4 phần tử còn lại nên kết quả bằng 4. Tương tự $a_3 = 2$ lớn hơn 1 phần tử nên có kết quả bằng 1.

Lời giải:

- Phương án ngây thơ naive (không dùng chữ trâu bò bruteForce nữa) là dùng 2 vòng vặp lồng nhau và kiểm tra điều kiện để đếm, thuật toán mất $O(n^2)$ không chạy được.
- Phương án thứ hai gồm hai bước:

+ sắp thứ tự mất chi phí $O(nLogn)$.

+ tìm kiếm biên (lower_bound) mất chi phí $O(Logn)$. Tổng chi phí hai bước là $O(nLogn)$ qua được thời gian bài toán.

+ Ví dụ: xét dãy $a = \{8, 1, 2, 2, 3\}$.

Bước 1. Sắp thứ tự và lưu vào vector data ta có: $data = \{1, 2, 2, 3, 8\}$.

Bước 2. Duyệt qua các phần tử của vector a và tìm xem nó xuất hiện ở vị trí nào trong data bằng hàm lower_bound. Hàm lower_bound(range, value): trả về vị trí đầu tiên lớn hơn hoặc bằng value trong range của vector. Như vậy phần tử 8 sẽ có giá trị trả về là 4, phần từ 1 trả về vị trí 0,..

Mã nguồn như sau:

countsmall.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define all(x) (x).begin(),(x).end()
typedef vector<int> VI;
VI solve(VI& nums) {
   VI r;
   // Copy bằng hàm dựng.
   VI data(all(nums));
   //Sắp tăng.
   sort(all(data));
   //Duyệt và tìm.
   for (auto &i: nums) {
      int cnt = lower_bound(all(data), i) - begin(data);
      r.push_back(cnt);
   }
   return r;
}
int main() {
   int n;   cin >> n ;
   VI v(n), r(n,0);
   for (int i = 0; i < n; i++) cin >> v[i];
   r=solve(v);
   for (auto &i: r) {
      cout <<i << " ";
   }
   return 0;
}
```

5.2 Sắp dãy số nguyên theo số bít 1

Bài tập 12: Sắp theo bít 1

Cho dãy số $A = \{a_1, a_2, \ldots, a_n\}$. Hãy sắp dãy theo thứ tự tăng dần dựa và số lượng bít 1 của mỗi số, nếu bằng nhau về bít 1 thì tăng dần theo giá trị.

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $2 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng.

Output

- In ra dãy kết quả trên một dòng, các phần tử cách nhau ký tự trắng.

Samples

input

```
9
0 1 2 3 4 5 6 7 8
```

output

```
0 1 2 4 8 3 5 6 7
```

Lời giải: Đơn giản chỉ là sử dụng hàm sort(), vấn đề chỉ là viết biểu thức lambda cho phù hợp yêu cầu sắp.

Mã nguồn sau:

countsmall.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define all(x) (x).begin(),(x).end()
typedef vector<int> VI;
void sortByBits(VI& arr) {
   sort(all(arr), [](auto &a, auto &b) {
      int aa = bitset<32>(a).count();
      int bb = bitset<32>(b).count();
      //nếu số bít 1 của aa nhỏ hơn hoặc bằng bb, ưu tiên 1
      //và giá trị a < giá trị b, ưn tiên 2.
      return aa < bb || ((aa == bb) && (a < b));
   });
}
int main() {
   int n;   cin >> n ;
   VI v(n);
   for (int i = 0; i < n; i++) cin >> v[i];
   sortByBits(v);
   for (auto &i: v) {
      cout <<i << " ";
   }
   return 0;
}
```

Thông tin bổ sung

- Hàm bitset có thể tự viết như sau:

```cpp
  int bitSet(int n) {
     int count = 0;
     while (n > 0) {
        count += (n & 1); // Đếm bít 1.
        n >>= 1;   // Dịch phải 1 bít.
     }
     return count;
  }

```

- Hoặc dùng hàm bitset() của STL như mã nguồn trên.
- Hoặc dùng hàm __builtin_popcount().

5.3 Tích k phần tử cuối dãy

Bài tập 13: Tích k phần tử cuối dãy

Bánh muốn có một ứng dụng mô tả như sau. Ứng dụng gồm hai thao tác:
1. Bổ sung một phần tử vào danh sách.
2. Tính tích của k phần tử cuối danh sách.
Nhờ Anh/chi lập trình giúp

Input

- Dòng thứ nhất chứa số nguyên dương n là số thao tác.
- n dòng tiếp theo mỗi dòng mô tả một trong hai thao tác trên theo mẫu: $< 1x >$ là bổ sung phần tử x vào cuối danh sách, $< 2k >$ là tính tích k phần tử cuối danh sách.

Output

- In ra kết quả tích cho thao tác thứ hai.

Constrain

- $1 \leq n \leq 10^5$.
- $1 \leq k \leq 40000, 0 \leq x \leq 1000$.

Samples

input

```
10
1 3
1 0
1 2
1 5
1 4
2 2
2 3
2 4
1 8
2 2
```

output

```
20
40
0
32
```

Lời giải:
1. Thuật toán naive: Nếu gọi vector<int> data là cấu trúc lưu trữ, thuật toán rất đơn giản cho hai thao tác:
a. data.push_back(x); với độ phức tạp $O(1)$.
b. for (int i = 0; i < k; ++ i) s *= data[data.size() - i - 1]; với độ phức tạp $O(k)$
Như vậy với n thao tác độ phức tạp bài toán $O(nk)$ khó qua.
2. Thuật toán tối ưu: Sử dụng khái niệm mảng tiền tố (prefix) cho các phần tử hiện có trong vector. Khi bổ sung phải giá trị 0 ta xóa dãy kết quả chia hai tổng tiền tố cuối và tổng tiền tố tại vị trí [-k].
Mã nguồn như sau:

```
void add(int num) {
   if (num > 0) data.push_back(num * data.back());
   else data = {1};
}
int getProduct(int k) {
   return k < data.size() ? data.back() / data[data.size() - k - 1] : 0;
}
```

Như vậy phép tính tích số chỉ còn $O(1)$ và độ phức tạp tổng của thuật toán là $O(n)$.

...GOOD LUCK 4 beginner programmer...