|  |
| --- |
| #include <cmath>  #include <cstdio>  #include <vector>  #include <iostream>  #include <algorithm>  using namespace std;  int main() {  int n;  cin >> n;  int i;  int a[n], b[n-2];  for (i=0; i<n; i++)  cin >> a[i];    for (i=0; i<n-2; i++){  b[i] = a[i]+ a[i+1] + a[i+2];  }  int min = b[0];  for (i=0; i<n-2; i++){  if (b[i] < b[0])  min = b[i];  }  cout << min;  return 0;  } |

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int n;

cin >> n;

long long a[n];

long long m = 0;

long long sum=0;

for (int i=0; i<n; i++) cin >> a[i];

for (int i=0; i<n; i++){

if (a[i] > 0) sum =0;

else sum += a[i];

m = min(sum, m);

}

cout << m;

return 0;

}

#include <iostream>

using namespace std;

int main() {

int n;

cin >> n;

int a[n];

for (int i = 0; i < n; i++) {

cin >> a[i];

}

int sum = 0;

int max\_sum = 1e9;

for (int i = 0; i < n; i++) {

sum += a[i];

if (sum < max\_sum) {

max\_sum = sum;

}

if (sum > 0) {

sum = 0;

}

}

cout << max\_sum;

return 0;

}

#include<bits/stdc++.h>

#define faster() ios\_base:: sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

using namespace std;

int main(){

faster();

int n;

cin >> n;

int a[n],b[n];

set<int> s,s1;

for(int i=0;i<n;i++){

cin >> a[i];

s.insert(a[i]);

b[i] =' ';

}

int dem = 0;

for(set<int>::iterator it = s.begin();it!=s.end();it++){

if( \*it < 0 ) {

dem++;

break;

}

}

if(dem ==0 ) {

for(set<int>::iterator it = s.begin();it!=s.end();it++){

cout << \*it << endl;

break;

}

}

else {

dem = 0;

int sum=0;

for(int i=0;i<n;i++){

if(a[i] <= 0 ) sum += a[i];

else if(a[i] > 0) {

b[i] = sum;

s1.insert(b[i]);

dem++;

sum = 0;

}

}

for(set<int>::iterator it = s1.begin();it!=s1.end();it++){

cout << \*it << endl;

break;

}

}

return 0;

}

|  |
| --- |
| #include <bits/stdc++.h>  #include <string.h>  using namespace std;  int main()  { int n; char tt='y';  cin >> n;  if (n <= 1){  cout <<"Invalid";  return 0;}  char node[n];  for (int i=0; i<n; i++) cin >> node[i];  cout <<"Nhap vao duong di\n";  int a[n][n];  for (int i=0; i<n; i++){  for (int j=0; j<n; j++)  a[i][j] =0;  }    while(1) {  string p; int x=-1,y=-1;  cin >> p;  for (int i=0; i<n; i++){  if (node[i] == p[0]) x=i;  if (node[i] == p[1]) y=i;  }  if (x>=0 && y>=0 && x<n && y<n) a[x][y] = 1;  cin >> tt; if (tt == 'n') break;  }    cout << "\\ ";  for (int i=0; i<n; i++) cout << node[i] << " ";  cout << endl;  for (int i=0; i<n; i++){  cout << node[i] <<" ";  for (int j=0; j<n; j++){  cout << a[i][j] <<" ";  }  cout << endl;    }  } |

int n; char tt='y';

cin >> n;

if (n <= 1){

cout <<"Invalid";

return 0;}

char node[n];

for (int i=0; i<n; i++) cin >> node[i];

cout <<"Nhap vao duong di\n";

int a[n][n];

for (int i=0; i<n; i++){

for (int j=0; j<n; j++)

a[i][j] =0;

}

while(1) {

string p; int x=-1,y=-1;

cin >> p;

for (int i=0; i<n; i++){

if (node[i] == p[0]) x=i;

if (node[i] == p[1]) y=i;

}

if (x>=0 && y>=0 && x<n && y<n) a[x][y] = 1;

cin >> tt; if (tt == 'n') break;

}

cout << "\\ ";

for (int i=0; i<n; i++) cout << node[i] << " ";

cout << endl;

for (int i=0; i<n; i++){

cout << node[i] <<" ";

for (int j=0; j<n; j++){

cout << a[i][j] <<" ";

}

cout << endl;

}

vector trong C++ là một kiểu dữ liệu tương tự như mảng, nhưng có thêm tính năng tự động thay đổi kích thước khi cần. Vector cung cấp một số tính năng hữu ích như lưu trữ số lượng phần tử linh hoạt hơn, thêm, xóa và truy cập phần tử nhanh hơn, và nhiều hơn nữa.

Một số khác biệt giữa vector và mảng trong C++ bao gồm:

1. Kích thước của vector có thể thay đổi sau khi được tạo, còn kích thước mảng là cố định.
2. Vector có thể tự động thay đổi kích thước để chứa thêm hoặc xóa các phần tử, trong khi mảng thường phải được cấp phát lại để thay đổi kích thước.
3. Truy cập phần tử trong vector có thể được thực hiện giống như mảng, bằng cách sử dụng toán tử [] hoặc phương thức at(). Tuy nhiên, vector cung cấp một số phương thức khác để truy cập phần tử, ví dụ như front() và back().
4. Vector có thể được truyền vào các hàm và phương thức dưới dạng đối số, mà không cần phải khai báo kích thước được trước. Mảng, tuy nhiên, thường yêu cầu khai báo kích thước được trước.
5. Vector có nhiều phương thức hỗ trợ thêm, xóa và sắp xếp phần tử một cách dễ dàng hơn so với mảng.

Tổng quan, vector là một kiểu dữ liệu linh hoạt và tiện dụng hơn so với mảng trong nhiều trường hợp.

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

int maxStackHeight(int n, const vector<int>& a) {

vector<int> sorted\_a = a;

sort(sorted\_a.begin(), sorted\_a.end(), greater<int>());

vector<int> dp(n, 0);

// Base case: the maximum stack height ending at the first brick is 1

// du no co do cung = 0 thi no van la 1 vien gach

dp[0] = 1;

// Compute the maximum stack height ending at each brick using DP

for (int i = 1; i < n; i++) {

int maxHeight = 0;

for (int j = 0; j < i; j++) {

if (sorted\_a[j] >= i - j) { // the j-th brick can support the i-th brick

maxHeight = max(maxHeight, dp[j]); // update the maximum stack height

}

}

dp[i] = maxHeight + 1; // add the current brick to the stack

}

// Return the maximum stack height

return \*max\_element(dp.begin(), dp.end());

}

int main() {

int n;

cin >> n;

vector<int> a(n);

for (int i = 0; i < n; i++) {

cin >> a[i];

}

cout << maxStackHeight(n, a) << endl;

return 0;

}

Để tính giá trị `dp[i]`, hàm sẽ tìm các viên gạch có thể xếp lên đó bằng cách tìm tất cả các vị trí trong các viên gạch trước đó (0 đến `i-1`) mà **độ dài của chúng** **không lớn hơn vị trí hiện tại `i-j`**. Sau đó, sẽ chọn viên gạch ở vị trí có `dp[j]` lớn nhất và kết hợp với viên gạch tại vị trí `i` để tạo ra một chồng gạch mới.

Còn cái test case 6 7 cơ huhu làm sao giờ ☹

tại sao lại so sánh sorted\_a[j] với i - j



Trong bài toán này, độ cao của một cột gạch phụ thuộc vào độ cứng, nghĩa là nó phải được xếp trên một cột gạch có độ cứng ít nhất bằng với độ cao của nó.

Vì vậy, ở mỗi bước duyệt, chúng ta kiểm tra xem có bất kỳ cột nào được xếp trước đó mà có độ cứng lớn hơn hoặc bằng khoảng cách giữa cột đang xét và cột đó hay không. Khoảng cách giữa hai cột gạch là **`i - j`**, tức là vị trí của cột đang xét so với cột đã xếp trước đó. Độ cứng của cột đã xếp trước đó là độ cao của cột gạch đó.

Vì vậy, nếu một cột gạch có chiều cao lớn hơn hoặc bằng khoảng cách giữa nó và cột đang xét (tức là **`sorted\_a[j] >= i - j`**), cột đang xét có thể được đặt lên cột gạch đó và độ cao của chồng gạch có thể được tăng lên.

Debug xong vẫn dell hiểu j

|  |
| --- |
| #include <iostream>  #include <algorithm>  #include <vector>  using namespace std;  int maxStackHeight(int n, const vector<int>& a) {  if (n<=0) return 0;  vector<int> sorted\_a = a;  sort(sorted\_a.begin(), sorted\_a.end(), greater<int>()); // sort tang dan  vector<int> dp(n, 0);  // Base case: the maximum stack height ending at the first brick is 1  dp[0] = 1;  // Compute the maximum stack height ending at each brick using DP  for (int i = 1; i < n; i++) { //i duyet qua het mang, j duyet qua cac phan tu lon hon a[i]  int maxHeight = 0;  for (int j = 0; j < i; j++) {  cout << "i= " << i << " j= " << j << " i-j= " << i-j << " sorted\_a[j] = " << sorted\_a[j]  <<" maxHeight= " << maxHeight << " dp["<<j<<"] = " << dp[j] << endl;  if (sorted\_a[j] >= i - j) { // the j-th brick can support the i-th brick  maxHeight = max(maxHeight, dp[j]); // update the maximum stack height  }  }    dp[i] = maxHeight + 1; // add the current brick to the stack  cout << " dp["<<i<<"] = " << dp[i] << endl;  }  // Return the maximum stack height  return \*max\_element(dp.begin(), dp.end()); //tra ve so lon nhat trong mang dp  }  int main() {  int n;  cin >> n;  vector<int> a(n);  for (int i = 0; i < n; i++) {  cin >> a[i];  }  cout << maxStackHeight(n, a) << endl;  return 0;  } |