|  |  |  |
| --- | --- | --- |
| T(n) = 0(n^) | T(n) = O(\* logn) | T(n) = O (f(n)) |

\* Tham lam :

- Đổi tiền :

+ Số tờ tiền cần đổi là phép / ---- Số tiền còn lại là %

=> Sắp xếp các loại tiền giảm dần để sử dụng ít số tiền nhất.(Tức là lấy ra các tờ to trả cho khách trước đượcý).

    int a[5] = {1,5,10,25,100};

    sort(a,a+5,greater<int>());

    int Tien\_Can\_Doi;

    cin >> Tien\_Can\_Doi;

    /\*- Dùng map để lưu trữ những số tiền đã đổi -\*/

    vector<pair<int,int>> vt;

    int cents = 0;

    for(int i=0;i < 5;i++){

       if(Tien\_Can\_Doi  / a[i] > 0){

           cents += Tien\_Can\_Doi /a[i];

           vt.push\_back({Tien\_Can\_Doi / a[i], a[i]});

           Tien\_Can\_Doi = Tien\_Can\_Doi % a[i];

       }

    }

    cout << " Chung ta can phai su dung : "<< cents << " de doi" << endl;

    for (auto x : vt){

        cout << x.first <<"  "<< x.second<<endl;

    }

- Trồng cây :

+ Sắp xếp cây hoa nào mà nở lâu nhất thì sẽ cho trồng đầu tiên.

int n;

    cin >>n;

    int \*a= new int[n];

    for(int i=1;i<=n;i++){

        cin >> a[i];

    }

    sort(a+1,a+n+1,greater<int>());

    int Ngay\_No\_Som\_Nhat =0;

    for (int i=1;i<=n;i++){

        int tmp = i + a[i];

        Ngay\_No\_Som\_Nhat = max(tmp, Ngay\_No\_Som\_Nhat);

    }

    cout<< Ngay\_No\_Som\_Nhat + 1<<endl;

- Chọn môn học :

+ Thêm môn kết thúc đầu tiên vào vì nó luôn đúng (Hoặc hiểu đơn giản là cuộc họp kết thúc sớm nhất có thể.

+ Duyệt qua các môn tiếp theo, nếu xuất hiện môn học nào có thời gian bắt đầu >= thời giân kết thúc của môn học trước đó thì chọn.

    int n;

    cin >> n;

    pair<int, int> \*a = new pair<int, int>[n];

    for (int i = 0; i < n; i++) {

        cin >> a[i].first >> a[i].second;

    }

    // Sắp xếp các cặp số theo thời gian bắt đầu tăng dần

    sort(a, a + n);

    vector<pair<int, int>> result;

    // Chọn các cuộc họp không gây trùng lặp về thời gian

    int end\_time = -1;

    for (int i = 0; i < n; i++) {

        if (a[i].first >= end\_time) {

            result.push\_back(a[i]);

            end\_time = a[i].second;

        }

    }

    // In ra các cuộc họp có thể diễn ra

    cout << "- Co toi da " << result.size() << " cuoc hop co the dien ra:" << endl;

    for (auto meeting : result) {

        cout << meeting.first << " " << meeting.second << endl;

    }

- Lập lịch cũng tương tự.

- Cái ba lô :

+ Bài toán cái balo ta phải sắp xếp theo tỉ lệ giá trị / đồ vật để tìm ra cái thằng cần phải lấy nhất.

bool cmp(pair<int, int> &a, pair<int, int> &b) {

     return (double)a.second / a.first > (double)b.second / b.first;

    // return (double)a.second  > (double)b.second;

int main() {

    int n, Size\_Balo;

    cin >> n >> Size\_Balo;

    pair<int, int> \*a = new pair<int, int>[n];

    vector<pair<int,pair<int,int>> > kq;

    for (int i = 0; i < n; i++) {

        cin >> a[i].first >> a[i].second;

    }

    sort(a, a + n, cmp);

    cout << "Sau khi sap xep la: " <<endl;

    for(int i=0;i < n ;i++){

        cout << a[i].first << " "<< a[i].second<<endl;

    }

    int Gia\_Tri\_Max = 0;

    for (int i = 0; i < n; i++) {

        if (Size\_Balo > 0) {

            int So\_Lan =  Size\_Balo / a[i].first;

            Gia\_Tri\_Max += So\_Lan  \* a[i].second;

            kq.push\_back({So\_Lan ,{a[i].first, a[i].second}});

            Size\_Balo -= ( So\_Lan\* a[i].first);

        }

    }

    cout << " Gia tri max lay duoc  : " << Gia\_Tri\_Max << endl;

    cout<< "Cac do vat can lay : "<<endl;

    for(auto x : kq){

        cout  <<" So lan : "<< x.first <<" lay do co kich thuoc ;"<<x.second.first << "  trong luong " << x.second.second<<endl;

    }

    delete[] a; // Giải phóng bộ nhớ

    return 0;

}

- Đóng thùng :

+ Tạo một biến đếm số thùng đã dùng.Tiếp tục dùng một vector lưu trữ thùng ,mỗi phần tử là một vector độc lập (coi nó như một cái thùng).

+ Sắp xếp các phần tử giảm dần để lấy được ra các các vật có kích thước nặng cho vào thùng trước.

+ Đánh dấu vật đó chưa được cho vào thùng.

+ Duyệt qua các thùng xem có thùng nào có khả năng chứa đồ vật đó hay không => Nếu có khả năng thì cho vật đó vào thùng và đánh dấu check = true;

+ Nếu thùng chưa được đánh dấu tức là không có thùng nào có đủ sức để chứa

-> Ta phải tạo thùng mới để nhét nó vào

int Tong(vector<int> &X){

    int tong = 0;

    for (int i = 0;i<X.size();i++){

        tong+= X[i];

    }

    return tong;

}

void SoThung(int \*a,int n,int B){

    int SoThung = 0;

    vector<vector<int>> ThungChua;

    /\* - Sap xep cac thung giam dan - \*/

    sort(a,a+n,greater<int>());

    for (int i = 0;i<n;i++){

         bool check = false;

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

            if(Tong(ThungChua[j]) + a[i] <= B){

                ThungChua[j].push\_back(a[i]);

                check = true;

            }

        }

         /\* - Nếu mà không có thùng chứa nào thỏa  - \*/

         if(check == false){

            ThungChua.push\_back({a[i]});

            SoThung ++;

         }

    }

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

        cout << "Thung " << i + 1 << ": ";

        for (int j = 0; j < ThungChua[i].size(); j++) {

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

        }

        cout << endl;

    }

    cout<<"So thung tat ca la :"<<  SoThung<<endl;

}

- Sắp xếp công việc:

- Cho n công việc phải hoàn thành , mõi công việc thực hiện trong một đơn vị thời gian. Việc i đem lại Wi tiền thưởng nếu hoàn thành đúng hạn di.

- Tìm cách thực hiện các công việc để có lợi nhuận cao nhất mà thời gian thực hiện là ít nhất.

- Ý tưởng tham lam : Sắp xếp lại lịch theo chiều giảm dần của phần thưởng Wi . Thay đổi i và di tương ứng.

// Định nghĩa một struct để lưu thông tin về công việc

struct CongViec {

    int id;

    int deadline;

    int reward;

};

// So sánh công việc dựa trên phần thưởng giảm dần

bool cmp(const CongViec &a, const CongViec &b) {

    return a.reward > b.reward;

}

// Hàm tham lam để tìm lịch làm việc tối ưu

int maxProfit(vector<CongViec> &jobs) {

    // Sắp xếp các công việc theo thứ tự giảm dần của phần thưởng

    sort(jobs.begin(), jobs.end(), cmp);

    // Tạo mảng lưu trữ trạng thái đã sử dụng của mỗi thời điểm

    // Tạo vector có kích thước jobs.size() , ban đầu các phần tử là false

    vector<bool> slot(jobs.size(), false);

    // Tính tổng phần thưởng

    int totalProfit = 0;

    // Duyệt qua từng công việc

    for (int i = 0; i < jobs.size(); i++) {

        // Tìm một thời điểm trống để thực hiện công việc

        for (int j = min(jobs[i].deadline, (int)jobs.size()) - 1; j >= 0; j--) {

            if (slot[j]==false) {

                // Thực hiện công việc tại thời điểm j

                slot[j] = true;

                totalProfit += jobs[i].reward;

                break;

            }

        }

    }

    return totalProfit;

}

Trung tâm dữ liệu thuê máy

// Định nghĩa một struct để lưu thông tin của mỗi yêu cầu thuê máy

struct RentalRequest {

    int start\_time;

    int end\_time;

    int rent\_fee;

};

// Hàm so sánh để sắp xếp các yêu cầu theo thời gian kết thúc tăng dần

bool compareEndTime(const RentalRequest &a, const RentalRequest &b) {

    return a.end\_time < b.end\_time;

}

// Hàm tham lam để tìm tổng số tiền thu được lớn nhất

int maxProfit(vector<RentalRequest> &requests) {

    // Sắp xếp các yêu cầu theo thời gian kết thúc tăng dần

    sort(requests.begin(), requests.end(), compareEndTime);

    int total\_profit = 0;

    int end\_time = 0; // Thời gian kết thúc của yêu cầu thuê máy trước đó

    // Duyệt qua từng yêu cầu thuê máy

    for (int i = 0; i < requests.size(); ++i) {

        // Nếu thời gian bắt đầu của yêu cầu hiện tại sau thời gian kết thúc của yêu cầu trước đó,

        // chấp nhận yêu cầu này và cập nhật thời gian kết thúc và tổng số tiền thu được

        if (requests[i].start\_time >= end\_time) {

            total\_profit += requests[i].rent\_fee;

            end\_time = requests[i].end\_time;

        }

    }

    return total\_profit;

}

- Lưu file đĩa từ

Bài toán này yêu cầu tìm cách lưu trữ các file trên đĩa từ sao cho tổng chi phí để truy cập tất cả các file là nhỏ nhất. Mỗi file được truy cập đúng một lần và chi phí để truy cập file thứ i là tổng dung lượng của tất cả các file từ file đầu đến file thứ i.

Ý tưởng của giải thuật tham lam là sắp xếp các file theo thứ tự tăng dần của dung lượng, sau đó lưu trữ các file theo thứ tự đó. Khi đó, việc truy cập file thứ i sẽ chỉ mất thời gian để truy cập các file có dung lượng nhỏ hơn hoặc bằng dung lượng của file thứ i.

bool compareFileSize(const pair<int, int> &a, const pair<int, int> &b) {

    return a.second < b.second;

}

// Hàm tham lam để chọn và ghi file vào đĩa từ

void GreedyFileOnTape(vector<pair<int, int>> &files) {

    // Sắp xếp các file theo dung lượng tăng dần bằng cách sử dụng hàm số sánh

    sort(files.begin(), files.end(), compareFileSize);

    cout << "Cach luu tru file voi chi phi nho nhat: ";

    // Duyệt qua từng file và ghi vào đĩa từ theo thứ tự đã sắp xếp

    for (int i = 0; i < files.size(); ++i) {

        cout << files[i].first << " "; // Ghi chỉ số của file vào đĩa từ

    }

    cout << endl;

}

int main() {

    // Khởi tạo vector chứa thông tin về các file

    vector<pair<int, int>> files = {{1, 5}, {2, 4}, {3, 6}};

    // Gọi hàm tham lam để chọn và ghi file vào đĩa từ

    GreedyFileOnTape(files);

    return 0;

}

\* Quay lui :

|  |  |  |
| --- | --- | --- |
| int x[20];  int n;  void Output(){      for (int i=0;i<n;i++){          cout<<x[i] << " ";      }      cout <<endl;  }  void  BackTracking(int i){     for(int j =0;j<=1;j++){      x[i] = j;      if(i == n){          Output();      }      else {          BackTracking(i+1);      }     }  } | int a[11];  int n;  int check[11] = {};  void IN\_RA\_CAU\_HINH(){     for (int i=1;i<=n;i++){      cout << a[i];     }     cout<<endl;  }  void Try(int i){     for (int  j =1; j<= n ;j++){         if(check[j] ==0){            check[j] = 1;            a[i] = j;            if( i == n){              IN\_RA\_CAU\_HINH();            }            else{              Try(i+1);          }          check[j] = 0;         }     }  } | int n;  int k;  int a[11];  void IN\_RA\_CAU\_HINH(){      for (int i=1;i<=k;i++){          cout << a[i];      }      cout <<endl;  }  void Try(int i){      for (int j = a[i-1] + 1 ; j<=n-k+i;j++){          a[i] = j;          if(i ==k){             IN\_RA\_CAU\_HINH();          }          else {              Try(i+1);          }      }  } |

- Kiểm tra xem từ một dãy có những số nào cộng lại với nhau thành một tập con có tổng bằng n

+ Ta sẽ dùng việc sinh nhị phân để liệt kê ra hết tất cả các trường hợp có thể có của cấu hình

+ Sau đó ta sẽ dựa vào cấu trúc của cấu hình thêm các số sao cho phù hợp :

+ Với các vị trí cấu hình = 1 thì ta sẽ cộng cái số trong mảng ban đầu trùng với cái chỉ số của cái vị trí cấu hình = 0 vào một biến và phải luôn kiểm tra cái tổng đó có bằng k không thì tại các vị trí cộng số thì ta sẽ thêm cái só được cộng vào vector.

+ Chú ý trước khi thêm một cái vector kết quả vào vector chính thì ta phải kiểm tra xem nó đã xuất hiện chưa mới thêm vào tránh trùng kết quả.

int n, Tong;

int a[100], v[100];

vector<vector<int> > vt;

void Try(int i){

    for(int j = 0; j <= 1; j++){

         a[i] = j;

         if(i == n){

            int cursum = 0;

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

                if(a[k] == 1){

                    cursum += v[k];

                }

            }

            if(cursum == Tong){

                        vector<int> tmp;

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

                            if(a[l] == 1){

                                tmp.push\_back(v[l]);

                            }

                        }

                        auto it = find(vt.begin(), vt.end(), tmp);

                        if (it == vt.end()) {

                            vt.push\_back(tmp);

                        }

                    }

         }

         else{

            Try(i + 1);

         }

    }

}

int main(){

    cin >> n >> Tong;

    for(int i = 0; i < n; i++){

        cin >> v[i];

    }

    Try(1);

    for(int i = 0; i < vt.size(); i++){

        for(int j = 0; j < vt[i].size(); j++){

            cout << vt[i][j] << " ";

        }

        cout << endl;

    }

    return 0;

}

/\*

7   12

8   6   7   5   3   10    9

- n quân hậu

+ Xếp n quân hậu trên bàn cờ kích thước n \* n sao cho không quân nào ăn được quân nào ( Nghĩa là hai quân không được xếp trên cùng một hàng, một cột, hoặc cùng một đường chéo chính hoặc cùng một đường chéo phụ ).

+ Bàn cờ 8 \* 8 thì nếu idx = 7 thì nghĩa là nó hoàn thành xong việc đặt 8 quân hậu vào bàn cờ.

+ Ta xét hai quân hậu Một(i,j) và quân hậu Hai(row,column)

Hai quân hậu cùng hàng ⬄ i = row

Hai quân hậu cùng cột ⬄ j = column

Hai quân hậu cùng đường chéo chính ⬄ i – j = row – column

Hai quân hậu cùng đường chéo phụ ⬄ i + j = row + column

Hai quân hậu cùng trên đường chéo ⬄|i – row|=|j – column|

=> Ta phải kiểm tra xem có thể đặt một quân hậu vào hàng i và cột j trên bàn cờ không, đảm bảo việc không có xung đột trên đường chéo hoặc xung đột cột giữa các quân hậu

- Ta duyệt theo từng hàng : Kiểm tra xem quân hậu đã đặt trên hàng k có cùng cột với cột j hay không ( queen[k] = j) hoặc có nằm trên đường chéo với quân hậu cần đặt hay không

int queen[n] = {0};

void Xuat(){

    for (int i = 0;i<n;i++){

        for (int j =0;j<n;j++){

            /\* Với hàng i thì quan hậu ở cột j \*/

            if(queen[i] == j){

                cout <<" \* ";

            }

            else{

                cout << " 0 ";

            }

        }

        cout << endl;

    }

    cout<<"\nBan co muon xem kq tiep theo : ";

    if(getchar() == 'y'){

        cin.ignore(1);

        cout<<"==============";

    }

    else{

        exit(1);

    }

}

bool Co\_The\_Dat(int i,int j){

   for (int k = 0;k < i;k++){

    /\* - Nếu nó vi phạm - \*/

      if(queen[k] == j || abs(k- i) ==abs(queen[k] - j) ){

        return false;

      }

   }

   return true;

}

/\* - Với mỗi cái idx thì ta sẽ đặt nó trên một dòng - \*/

void Dat\_Quan\_Hau(int idx){

    for (int j =0;j < n;j++){

        if(Co\_The\_Dat(idx,j)){

           /\*- Đặt 1 quân hậu vào vị trí cột j của hàng idex - \*/

           queen[idx] = j;

           /\* = No cung coi như một bước để đánh dấu \*/

           /\* - Nếu idx tìm đủ tất cả các hàng - \*/

           if(idx == n -1){

             Xuat();

           }

           else{

            /\* -Nhảy xuống hàng tiếp theo = \*/

            Dat\_Quan\_Hau(idx + 1);

           }

        }

    }

}

Mã đi tuần

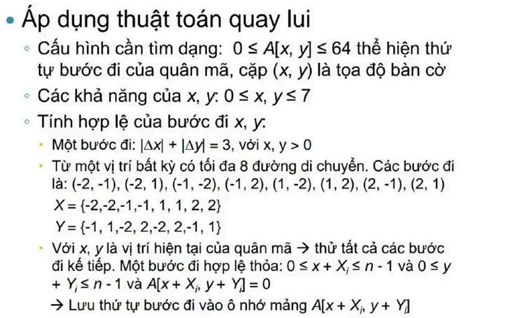
- Mã đi tuần là một bài toán tìm đường đi cho con mã

có thể đặt chân tới tất cả các điểm trên bàn cờ với mỗi

vị trí con mã chỉ được đi đến duy nhất một lần.

- Ý tưởng của bài toán này là dùng thuật toán BackTracking. Ý tưởng khá đơn giản , khi bạn đứng trước 2 con đường, bạn sẽ chọn 1 đường đê đi nếu đường đó là đường cục , bạn quay lại vị trí ban đầu và đi đường thứ hai.

- Ta nhận thấy được từ một vị trí này di chuyển sang một vị trí khác thì quân mã đi theo hình chữ l và tổng trị tuyệt đối sự thay đổi vị trí sẽ luôn bằng 3 nên mình sẽ đặt mảng có các phần tử như này ứng với sự thay đổi thỏa mãn.



#define MAX 8

using namespace std;

int a[MAX][MAX] = {0};

int X[8] = {-2,-2,-1,-1,1,1,2,2};

int Y[8] = {-1,1,-2,2,-2,2,-1,1};

int n;

int dem = 0;

void Xuat(){

    for (int i=0;i<n;i++){

        for (int j=0;j<n;j++){

            cout << a[i][j]<<" ";

        }

        cout <<endl;

    }

}

void Move(int x, int y) {

    dem++;

    a[x][y] = dem;

    for (int i = 0; i < 8; i++) {

        int u = x + X[i];

        int v = y + Y[i];

        // Nó không vượt quá cái bàn cờ với lại nó là cái vị trí chuea được đi qua

        // thì mình sẽ gọi hàm đệ quy di chuyển đến cái vị trí đó

        if (u >= 0 && u < n && v >= 0 && v < n && a[u][v] == 0) {

            Move(u, v);

        }

    }

    if (dem == n \* n) {

        cout << "Cac buoc da di : "<<endl;

        Xuat();

        exit(0);

    }

    // Đây chính là phần quay lui của bài toán tức nào nếu nó

    // Giảm cái đếm này nếu như mình không tìm được bước đi phải trả về giá trị ban

    // đầu cho nó  không thì nó sẽ không thể quay lui lại cho đến khi tìm được đường đi mới thỏa mãn

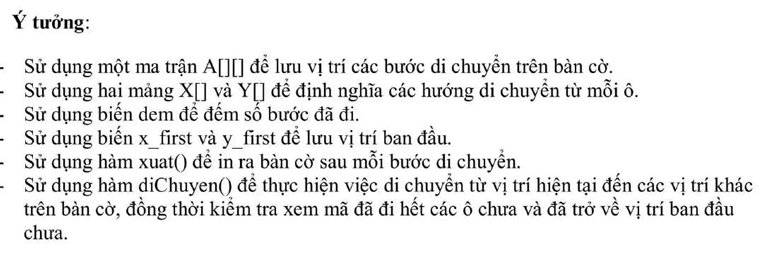
    dem--;

    a[x][y] = 0;

}

- In ra một chu trình đóng từ bài toán mã đi tuần

Bài toán "mã đi tuần theo chu trình đóng" yêu cầu tìm một chu trình đóng, tức là một đường đi mà quân mã đi qua mỗi ô trên bảng một lần duy nhất và quay lại ô ban đầu. Điều này là khác biệt so với bài toán mã đi tuần thông thường, nơi mục tiêu là di chuyển qua mỗi ô một lần duy nhất mà không cần quay lại ô ban đầu.



int A[MAX][MAX] = { 0 };

int X[8] = {-2,-2,-1,-1, 1, 1, 2, 2};

int Y[8] = {-1, 1,-2, 2,-2, 2,-1, 1};

int dem = 0;

int n, x\_first, y\_first;

void xuat() {

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++)

            printf("%2d ", A[i][j]);

        cout << endl;

    }

    cout << endl;

}

void diChuyen(int x, int y) {

    ++dem; //Tang gia tri buoc di

    A[x][y] = dem; //Danh dau da di

    for (int i = 0; i < 8; i++)    {

        //Kiem tra xem ma da di het ban co chua

        if (dem == n \* n && x == x\_first && y == y\_first) {

            cout << "Cac buoc di la: \n";

            xuat();

            exit(0); //ket thuc chuong trinh

        }

        //Neu chua di het ban co thi tao buoc di moi

        int u = x + X[i];//tao mot vi tri x moi

        int v = y + Y[i];//tao mot vi tri y moi

        //Neu hop le thi tien hanh di chuyen

        if (u >= 0 && u < n && v >= 0 && v < n && A[u][v] == 0)

            diChuyen(u, v);

    }

    //Neu khong tim duoc buoc di thi ta phai tra lai cac gia tri ban dau

    --dem;

    A[x][y] = 0;

}

- Biểu thức ZeZo :

Bài toán này yêu cầu tạo ra tất cả các biểu thức số học bằng cách sử dụng các số từ 1 đến 9 cùng với các dấu (+, -) sao cho kết quả của biểu thức là 0.

Ý tưởng của bài toán được thực hiện thông qua việc sinh tất cả các biểu thức có thể từ các số và dấu được cho sẵn, sau đó kiểm tra xem biểu thức nào đạt được kết quả là 0.Sử dụng một hàm đệ quy zero\_expr(int i) để sinh ra tất cả các biểu thức có thể bằng cách đặt các dấu (+, -) vào giữa các số.

Trong hàm zero\_expr, duyệt qua tất cả các dấu có thể đặt vào giữa các số (có 3 dấu: "+", "-", và dấu rỗng), sau đó gọi đệ quy với chỉ số i+1 để xử lý số tiếp theo.

Khi đã xử lý hết tất cả các số, thực hiện hàm test() để kiểm tra kết quả của từng biểu thức. Trong hàm test(), thực hiện tính toán kết quả của biểu thức và kiểm tra xem kết quả có bằng 0 hay không. Nếu có, lưu biểu thức đó vào mảng M để sau này in ra.

vector<string> Dau = {"+","-",""}; //Mang dau

vector<string> S = {"1","2","3","4","5","6","7","8","9"};

vector<string> D(10); //Luu dau giua cac so

vector<string> M(100); //Mang luu bieu thuc zero

int N, dem;

void test() {

    string st = "", so = "";

    int i, k = 0, l;

    long T = 0;

    int dau = 0;

    for (i = 0; i < N; i++)

        st = st + D[i] + S[i];

    l = st.length();

    while (k < l) {

        if (st[k] >= '1' && st[k] <= '9') {

            so = so + st[k];

        } else {

            if (dau == 0) {

                T = stoi(so, nullptr, 10);

            } else {

                if (dau == -1)

                    T -= stoi(so, nullptr, 10);

                else T += stoi(so, nullptr, 10);

            }

            if (st[k] == '-') dau = -1; else dau = 1;

            so = "";

        }

        k++;

    }

    if (so != "") {

        if (dau == -1) T = T - stoi(so, nullptr, 10);

        else T = T + stoi(so, nullptr, 10);

    }

    if (T == 0) {

        dem++;

        M[dem] = st;

    }

}

void zero\_expr(int i) {

    for (int j = 0; j < 3; j++) {

        D[i] = Dau[j];

        if (i == N) test();

        else zero\_expr(i + 1);

    }

}

int main()

{

    cout<<"Nhap vao so n: ";

    cin>>N;

    zero\_expr(1);

    for (int i = 0; i < dem; i++)

        cout<<M[i]<<endl;

    return 0;

}

- In ra các dãy số trong X có tổng bằng T

- Phân tích số n thành tổng của các số nguyên dương.

+ Ta nhận thấy 4 phân tích thành :

4 3 1 2 2 2 1 1 1 1 1 1

+ Những số đằng sau luôn nhỏ hơn những số đằng trước ( Duyệt từ cuối về đầu bắt đầu từ n

+ Các thành phần của mỗi cấu hình là khác nhau.

=> Ta sẽ tạo một biến k đại diện cho các khả năng dành cho thành phần x[i] trong dãy số và theo dõi liên tục tổng của cấu hình và sau khi ta cập nhật 1 cấu hình thì ta phải cập nhật lại tổng.

int n;

int a[10];

void out (int i){

    for (int k =1 ;k<= i;k++){

       cout << a[k] << " ";

    }

    cout<<endl;

}

void Try (int i ,int k,int cursum ){

   /\* - Duyệt từng khả năng được chấp nhận - \*/

    for (int j = k; j >= 1; j--){

        if(cursum + j <= n){

            a[i] = j ;

            cursum +=j;

            if(cursum == n){

                  out(i);

            }

            else {

                  Try(i+1,j,cursum);

            }

      /\* -Tái tạo bài toán \*/

            cursum -=j;

        }

    }

}

- Dãy con tăng dài nhất

Sử dụng một mảng result để lưu trữ dãy con tăng dài nhất.

Sử dụng một biến max\_length để lưu độ dài của dãy con tăng dài nhất.Tạo một hàm đệ quy để thử tất cả các dãy con có thể bắt đầu từ mỗi phần tử trong mảng.Trong hàm đệ quy, bạn cần duyệt qua từng phần tử trong mảng và xây dựng các dãy con tăng bắt đầu từ phần tử đó.Mỗi lần thêm một phần tử vào dãy con, bạn cần kiểm tra xem dãy con đó có tăng không và có dài hơn dãy con tăng tốt nhất hiện tại không.Khi hết các phần tử hoặc không thể thêm phần tử nào khác vào dãy con, bạn cần lưu lại dãy con tăng dài nhất và độ dài của nó.Quay lui và thử các phần tử tiếp theo.

vector<int> result; // lưu trữ dãy con tăng dài nhất

int max\_length = 0; // lưu độ dài của dãy con tăng dài nhất

void TangDaiNhat(vector<int>& A, int idx, vector<int>& current) {

    if (idx == A.size()) {

        if (current.size() > max\_length) {

            max\_length = current.size();

            result = current;

        }

        return;

    }

    // Thử thêm phần tử A[idx] vào dãy con tăng hiện tại

    if (current.empty() || A[idx] >= current.back()) {

        current.push\_back(A[idx]);

        TangDaiNhat(A, idx + 1, current);

        current.pop\_back();

    }

    // Không thêm phần tử A[idx] vào dãy con tăng hiện tại

    TangDaiNhat(A, idx + 1, current);

}

int main() {

    vector<int> A = {1, 6, 5, 4, 7, 8, 2, 9, 0, 10};

    vector<int> current;

    TangDaiNhat(A, 0, current);

    cout << "Dãy con tăng dài nhất: ";

    for (int num : result) {

        cout << num << " ";

    }

    cout << endl;

    cout << "Độ dài: " << max\_length << endl

- Stdocu :

Xác định điểm khởi đầu: Bắt đầu từ góc trên bên trái của bảng Sudoku, tức là hàng 0, cột 0.Kiểm tra điều kiện dừng: Nếu chúng ta đã đi qua tất cả các ô trên bảng, tức là hàng cuối cùng và cột cuối cùng, thì chúng ta đã tìm thấy một lời giải và có thể kết thúc thuật toán. Trong trường hợp này, trả về true.Thử từng giá trị từ 1 đến 9 cho ô hiện tại:Kiểm tra xem giá trị có hợp lệ hay không. Điều này được thực hiện bằng cách kiểm tra xem giá trị đã tồn tại trong hàng, cột và vùng 3x3 tương ứng với ô hiện tại hay không.Nếu giá trị là hợp lệ, thì gán giá trị đó vào ô hiện tại và tiếp tục thử các ô tiếp theo.Nếu giá trị không hợp lệ, thử giá trị tiếp theo.Quay lui nếu không tìm được giải pháp: Nếu không có giá trị nào từ 1 đến 9 hợp lệ cho ô hiện tại, quay lại ô trước đó (nếu có) và thử giá trị khác cho ô đó. Điều này giúp thuật toán thử tất cả các giá trị có thể cho mỗi ô.Trả về false nếu không tìm thấy lời giải: Nếu không thể tìm thấy một giá trị hợp lệ cho một ô nào đó, thuật toán trả về false, cho biết rằng không có lời giải cho bài toán Sudoku này.In ra lời giải nếu có: Nếu thuật toán kết thúc và trả về true, in ra bảng Sudoku đã giải.Thuật toán quay lui sẽ thử từng giá trị có thể cho mỗi ô và thử tất cả các phương án cho đến khi tìm ra lời giải hoặc xác định rằng không có lời giải.

const int N = 9;

// hàm in ra bình thường

void print(int a[N][N]){

    for(int i = 0; i < N; i++){

        for(int j = 0; j < N; j++){

            cout << a[i][j] << " ";

        }

        cout << endl;

    }

}

bool check(int board[N][N], int row, int col, int x){

    // Check xem trong hang hien tai da ton tai so x chua, neu ton tai se tra ve false

    for(int j = 0; j < N; j++){

        if(board[row][j] == x){

            return false;

        }

    }

    // Check xem trong cot hien tai da ton tai so x chua, neu ton tai se tra ve false

    for(int i = 0; i < N; i++){

        if(board[i][col] == x){

            return false;

        }

    }

    // chỉ số hàng và chỉ số cột của ô gốc tọa độ của ô đơn vị

    // Chỉ số hàng của ô ở gốc tọa độ của 1 ô đơn vị = j -j %3

    int startRow =row -  row % 3;

    int startCol =col- col % 3;

    // Check xem trong vung don vi hien tai da ton tai so x chua, neu ton tai thi tra ve false

    for(int i = startRow; i < 3+ startRow; i++){

        for(int j = startCol; j <3+ startCol; j++){

            if(board[i][j] == x){

                return false;

            }

        }

    }

    return true;

}

bool backTrack(int board[N][N], int row, int col){

    // Check xem nó đi quá cái ô cuối cùng của bàn cờ chưa

    if(row == N - 1 && col == N){

        return true;

    }

    if(col == N){

        row++;

        col = 0;

    }

    if(board[row][col] > 0){

        return backTrack(board, row, col + 1);

    }

    for(int j = 1; j <= N; j++){

        if(check(board, row, col, j)){

            board[row][col] = j;

            if(backTrack(board, row, col + 1)){

                return true;

            }

        }

        board[row][col] = 0;

    }

    return false;

}

\* Chia để trị :

- Lũy thừa chia để trị :

long power(int a, int n) {

    long x;

    if (n == 1)

        return a;

    else {

        x = power(a, n / 2);    //chia nguyen

        if (n % 2 == 0)         //n chan

            return x \* x;

        else                    //n le

            return x \* x \* a;

    }

}

- Sắp xếp trộn

void merge(int a[], int n, int left, int mid, int right) {

    int b[n];

    int i = left, j = mid + 1;

    int k = 0;

    while (i <= mid && j <= right) {

        if (a[i] < a[j]) {

            b[k++] = a[i++];

        } else {

            b[k++] = a[j++];

        }

    }

    while (i <= mid) {

        b[k++] = a[i++];

    }

    while (j <= right) {

        b[k++] = a[j++];

    }

    for (i = right; i >= left; i--) {

        a[i] = b[--k];

    }

}

void mergeSort(int a[], int n, int left, int right) {

    if (left < right) {

        int mid = (left + right) / 2;

        mergeSort(a, n, left, mid);

        mergeSort(a, n, mid + 1, right);

        merge(a, n, left, mid, right);

    }

}

- Tìm phần tử lớn nhất trong mảng

int FindMax( int a[] ,int left,int right){

    if(left == right ){

        return a[left]; // Phần tử duy nhất thì

    }

    else {

        int mid  = (left + right )/2;

        int x = FindMax(a,left,mid);

        int y = FindMax(a,mid+1,right);

        return max(x,y);

    }

}

- Tìm phần tử nhỏ nhất trong mảng

int FindMin( int a[] ,int left,int right){

    if(left == right ){

        return a[left]; // Phần tử duy nhất

    }

    else {

        int mid  = (left + right )/2;

        int x = FindMin(a,left,mid);

        int y = FindMin(a,mid+1,right);

        return min(x,y);

    }

}

- Tìm kiếm nhị phân

template<class T1>

T1 Timkiemnhiphan(T1 \*a,T1 n,T1 search) {

    T1 left = 0;

    T1 right = n - 1;

    while (left <= right) {

        T1 mid = (left + right)/2;

        if (a[mid] == search) {

            return mid;

        } else if (a[mid] < search) {

            left = mid + 1;

        } else if (a[mid] > search) {

            right=mid - 1;

        }

    }

    return -1;

}

- Tìm ước chung lớn nhất của hai số theo giải thuật Euclid

Thuật toán Euclid dựa trên một quy tắc đơn giản: ước chung lớn nhất của hai số cũng chính là ước chung lớn nhất của số nhỏ hơn và phần dư của phép chia lớn hơn cho số nhỏ hơn.

Công thức r = a % b trong đoạn mã là bước tính phần dư theo quy tắc trên. Bạn cập nhật giá trị a thành b và b thành r để chuẩn bị cho vòng lặp tiếp theo. Khi b bằng 0, vòng lặp kết thúc và giá trị a chính là gcd của a và b, do a chứa giá trị cuối cùng không bị chia hết cho b.

int gcd(int a, int b) {

    int r;

    while(b != 0) {

        r = a % b;  //Phep chia du

        a = b;

        b = r;

    }

    return a;

}

int main()

{

    int a, b;

    cout<<"Nhap so a: ";cin>>a;

    cout<<"Nhap so b: ";cin>>b;

    int c = gcd(a, b);

    cout<<"Uoc so chung lon nhat cua "<<a<<" va "<<b<<" la: "<<c;

}

- Sắp xếp chèn

- Ý tưởng : Lấy phần tử thứ  a[j] chèn vào dãy các phần tử a[1]...a[j - 1] sao cho ta được dãy

a[1] ... a[j] được sắp xếp. Trong đó dãy a[1]... a[j -1 ] đã được sắp.

void insertionSort(int a[], int n) {

    for (int i = 1; i <= n-1; ++i) {

        int key = a[i];

        int j = i - 1;

        while (j >= 0 && a[j] > key) {

            a[  j + 1] = a[j];

            j--;

        }

        a[j + 1] = key;

    }

}

- Quicksort :

Chọn một phần tử bất kỳ của dãy làm dãy S2 (phần tử này được gọilà phần tử chốt - pivot).Thực hiện chuyển các phần tử có

khóa ≤ phần tử chốt về bên trái và các phần tử > phần tử chốt về bênphải, sau đó đặt phần tử chốt về đúng vị trí của nó trong dãy.

int Partition(int a[],int left,int right){

      int pivot = a[right];

       int i = left - 1;

       for (int j=left ; j< right ; j++){

        // Tim phan tu nho hon pivot thi tang i len sau do doi cho voi doi cho voi a[i]

        if(a[j] <= pivot){

            i++;

            swap(a[i] , a[j]);

        }

       }

       swap(a[i+1], a[right]);

       return i;

}

void QuichSort(int a[],int left ,int right ){

    // Neu mang chi co mot phan tu thoi

    if(left >= right ){

        return ;

    }

    // Neu mang co hai phan tu

    else if(left + 1 == right){

        if(a[left] > a[right]){

           swap(a[left],a[right]);

        }

    }

    else{

       int p = Partition(a,left,right);

       QuichSort(a, left,p);

       QuichSort(a,p+2,right);

    }

}

- Tìm kiếm theo chiều sâu :

- Tìm kiếm theo chiều rộng :

|  |  |
| --- | --- |
| /\* - n là số đỉnh, m là số cạnh -\*/  int n,m;  vector<int> adj[1001];  /\* - Mảng kiểm tra việc thăm - \*/  bool visited[1001];  void Input(){       cin >> n >> m;       for (int i=0;i<m;i++){          int x,y;          cin >> x >> y;          adj[x].push\_back(y);          adj[y].push\_back(x);       }      memset(visited,false,sizeof(visited));  }  void BFS (int u){      /\* - Tạo hàng đợi và push thằng đầu tiên vào -  \*/      queue<int> q;      q.push(u);      visited[u] = true;      while (!q.empty()){          /\* - Lấy ra thằng đầu tiên và duyệt những thằng xung quanh - \*/          int v = q.front();          q.pop();          cout << v << " ";          /\* - Duyệt những thằng kề nếu thấy chưa thăm thì push vào hàng đợi -\*/          for (auto x: adj[v]){              /\* - Nếu mảng chưa được thăm - \*/              if(visited[x] == false){                 q.push(x);                 visited[x]= true;              }          }        }  }  int main(){    Input();    BFS(1);  } | /\* - n là số đỉnh , m là số cạnh - \*/  int n, m; để lưu danh sách kề - \*/  vector <int> adj[1001];  /\* - Mảng check đã thăm chưa - \*/  bool visited[1001];  void input(){      cin >> n >> m;      for (int i=0;i<m;i++){          int x,y;          cin >> x >> y;          adj[x].push\_back(y);          adj[y].push\_back(x);      }      /\* - Chưa có đỉnh nào được check - \*/      memset(visited,false,sizeof(visited));  }  void DFS (int u){      cout << u<<" ";      /\* - Đánh dấu là u đã được thăm - \*/      visited[u] = true;      for (auto v : adj[u]){            /\* - Nếu đỉnh v chưa được thăm - \*/            if(visited[v] == false){                DFS(v);            }      }  }  int main(){      input();      DFS(1);  } |

- Tìm dãy con liên tục có tổng lớn nhất

int Day\_Con\_Co\_Tong\_Max(int a[], int n)

{

   int max\_so\_far = a[0];

   int curr\_max = a[0];

   int begin = 0, end = 0;

   for (int i = 1; i < n; i++)

   {

        if (a[i] > curr\_max + a[i]) {

            curr\_max = a[i];

            begin = i; end = i;

        } else

            curr\_max = curr\_max + a[i];

        if (max\_so\_far < curr\_max) {

            max\_so\_far = curr\_max;

            end = i;

        }

   }

    cout<<"Day con lien tiep co tong lon nhat: ";

    for (int i = begin; i<=end; i++)

        cout<<a[i]<<" ";

    cout<<endl;

   return max\_so\_far;

}

int main()

{

    int n = 6;

    int a[] = {-12, 7, 11, -8, 7, 9};

    cout << "Tong lon nhat: " << Day\_Con\_Co\_Tong\_Max(a,n) << endl;

    return 0;

}

- Tìm tổng của dãy con lớn nhất :

int maxSubArraySum(int a[], int n)

{

   int max\_so\_far = a[0];

   int curr\_max = a[0];

   for (int i = 1; i < n; i++)

   {

        curr\_max = max(a[i], curr\_max+a[i]);

        max\_so\_far = max(max\_so\_far, curr\_max);

   }

   return max\_so\_far;

}

int main()

{

    int n = 8;

    int a[] = {1,3,-5,2,3,4,-1,0};

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

    return 0;

}

- Tìm dãy con liên tục có tổng nhỏ nhất

void minSubArraySum(int a[], int n)

{

    int min\_so\_far = a[0];

    int curr\_min = a[0];

    int begin = 0, end = 0;

    int temp\_begin = 0;

    for (int i = 1; i < n; i++) {

        if (a[i] < curr\_min + a[i]) {

            curr\_min = a[i];

            temp\_begin = i;

        } else

            curr\_min = curr\_min + a[i];

        if (min\_so\_far > curr\_min) {

            min\_so\_far = curr\_min;

            begin = temp\_begin;

            end = i;

        }

    }

    cout << "Dãy con có tổng nhỏ nhất: ";

    for (int i = begin; i <= end; i++)

        cout << a[i] << " ";

    cout << endl;

}

- Tìm tổng nhỏ nhất của đoạn con liên tục

int minSubArraySum(int a[], int n)

{

    int min\_so\_far = a[0];

    int curr\_min = a[0];

    for (int i = 1; i < n; i++) {

        curr\_min = min(a[i], curr\_min + a[i]);

        min\_so\_far = min(min\_so\_far, curr\_min);

    }

    return min\_so\_far;

}

int main()

{

    int n = 8;

    int a[] = {1, 3, -5,-10, 2, 3, 4, -1, 0};

    cout << "Tổng nhỏ nhất của đoạn con liên tục: " << minSubArraySum(a, n) << endl;

    return 0;

}

-fibonaci :

long Fibonacci(int n)

{

    if (n < 2) return n;

    long n1 = 1, n2 = 0; // luu co so vao cac bien

    for (int i = 2; i < n; i++) {

        long n0 = n1 + n2;

        n2 = n1;

        n1 = n0;

    }

    return n1 + n2;

}

long Fibonacci(int n)

{

    if (n < 2)

        return n;

    else return Fibonacci(n - 1) + Fibonacci(n - 2);

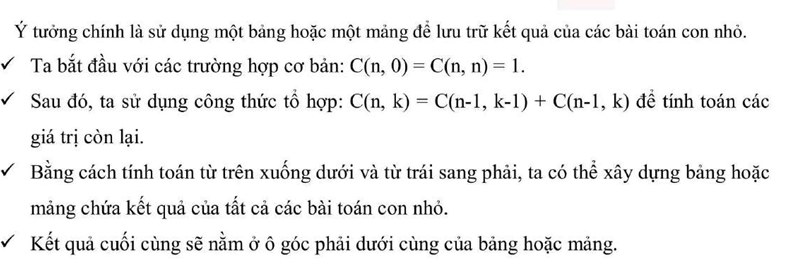
}

\* Quy hoạch động :

- Tính tổ hợp

+ Ta sẽ phải dựa vào tam giác pascal để tìm ra được quy luật của việc tính toán tổ hợp .Tam giác pascal sẽ có cấu tạo gồm các hàng n và mỗi hàng sẽ bao gồm k phần tử chạy từ 0 trở đi -> Cấu tạo nên một tổ hợp cnK .

+ Hai kết quả bên trên cộng lại sẽ ra kết quả bên dưới



int a[1005][1005];

int Tinh\_To\_Hop(int n, int k) {

    for (int i = 0; i <= n; i++) {

        a[i][0] = 1;

        a[i][i] = 1;

    }

    for (int i = 2; i <= n; i++) {

        for (int j = 1; j < i; j++) {

            a[i][j] = a[i - 1][j - 1] + a[i - 1][j];

        }

    }

    return a[n][k];

}

int main() {

    int n, k;

    cout << "Nhap k va n: ";

    cin >> k >> n;

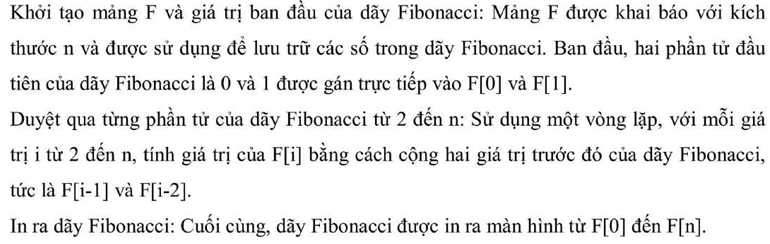
    cout << "To hop C\_k\_n"<<Tinh\_To\_Hop(n, k) << endl;

    return 0;

}

- Dãy fibonaci

Từ định nghĩa ta thấy Fn chỉ phụ thuộc vào Fn−1 và Fn−2, chính vì thế ở mỗi bước lặp chúng ta chỉ cần lưu lại hai giá trị này là đủ



int fibonacci(int n) {

    if (n < 2)

        return n;

    int n1 = 1, n2 = 0; // Lưu các số cơ sở vào các biến

    for (int i = 2; i < n; i++) {

        int n0 = n1 + n2;

        n2 = n1;

        n1 = n0;

    }

    return n1 + n2;

}

int main() {

    int n;

    cout << "Nhap so thu tu cua so Fibonacci: ";

    cin >> n;

    cout << "So Fibonacci thu " << n << " la: " << fibonacci(n) << endl;

- Kiểm tra xem có tạo được tổng S từ tập con không

Ý tưởng chính của giải thuật này là sử dụng một mảng đánh dấu để lưu trạng thái của việc có thể tạo ra tổng các giá trị từ tập hợp đã cho. Cụ thể:

Khởi tạo một mảng dp với kích thước (S+1), trong đó S là tổng cần tạo. Các phần tử của mảng dp ban đầu được gán giá trị false.

Gán dp[0] bằng true, vì luôn có thể tạo ra tổng 0 từ tập hợp rỗng.

Duyệt qua từng phần tử trong tập hợp các số nguyên đã cho. Đối với mỗi phần tử a[i], duyệt ngược từ S đến a[i].

Tại mỗi bước duyệt, kiểm tra xem có thể tạo ra tổng j từ tập hợp các số cho đến phần tử thứ i hay không. Điều này được thực hiện bằng cách kiểm tra giá trị của dp[j - a[i]]. Nếu dp[j - a[i]] là true, tức là có thể tạo ra tổng j - a[i] từ các số trước đó, ta có thể cập nhật dp[j] thành true, bởi vì ta có thể tạo ra tổng j từ tập hợp các số bằng cách thêm phần tử a[i] vào.

Cuối cùng, kiểm tra giá trị của dp[S]. Nếu dp[S] là true, tức là có thể tạo ra tổng S từ tập hợp các số đã cho, in ra 1; ngược lại, in ra 0.

Tóm lại, ý tưởng của đoạn code này là sử dụng một mảng đánh dấu để theo dõi trạng thái của việc có thể tạo ra tổng từ tập hợp các số đã cho, và dựa vào các trạng thái trước đó để xác định trạng thái hiện tại.

Top of Form

using ll = long long;

int main(){

int n,s; cin >> n >> s;

vector<int> a(n);

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

cin >> a[i];

}

vector<bool> dp(S+1 ,false);

dp[0] = true;

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

for(int j =S ; j>= a[i] ; j--){

if(dp[i] – a[i] == true){

dp[j] = true;

}

}

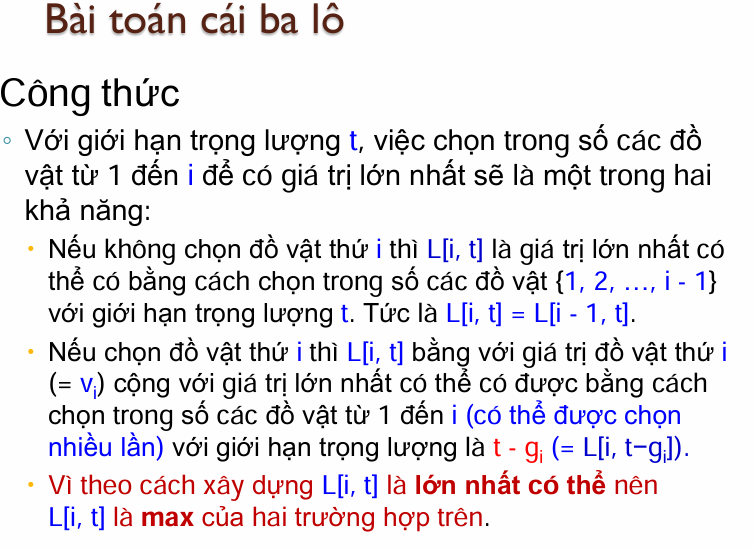
if(dp[S] ) cout<< 1<< endl;

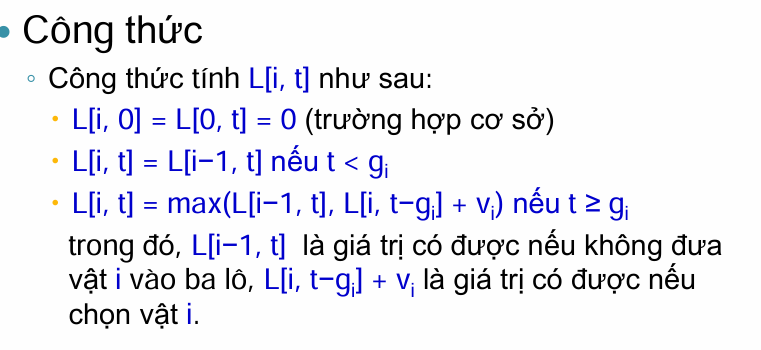
else{

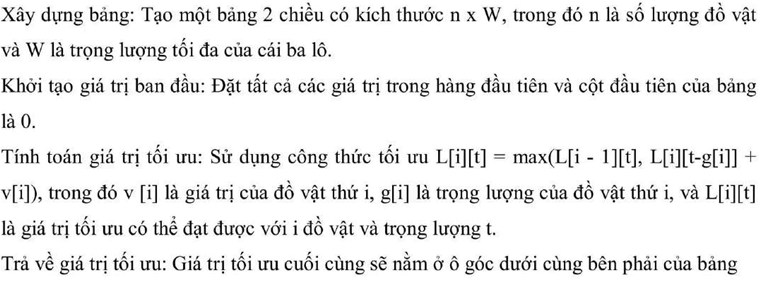
cout << 0 << endl;

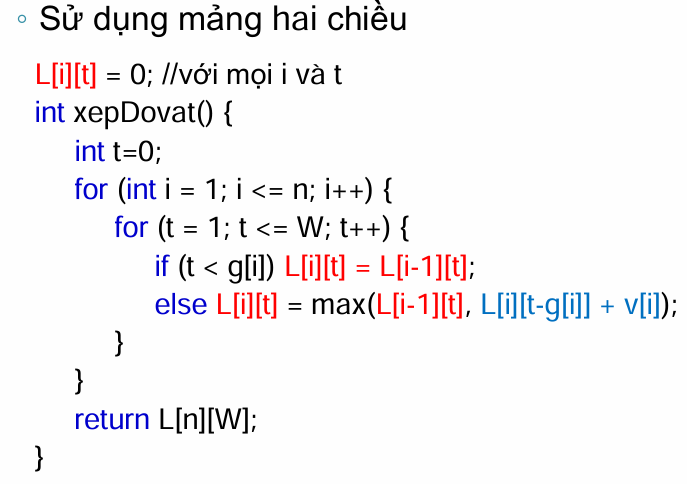
}

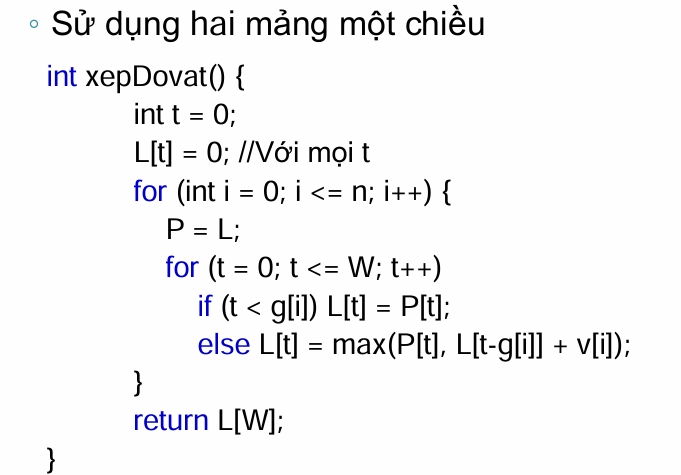
- Cái ba lô

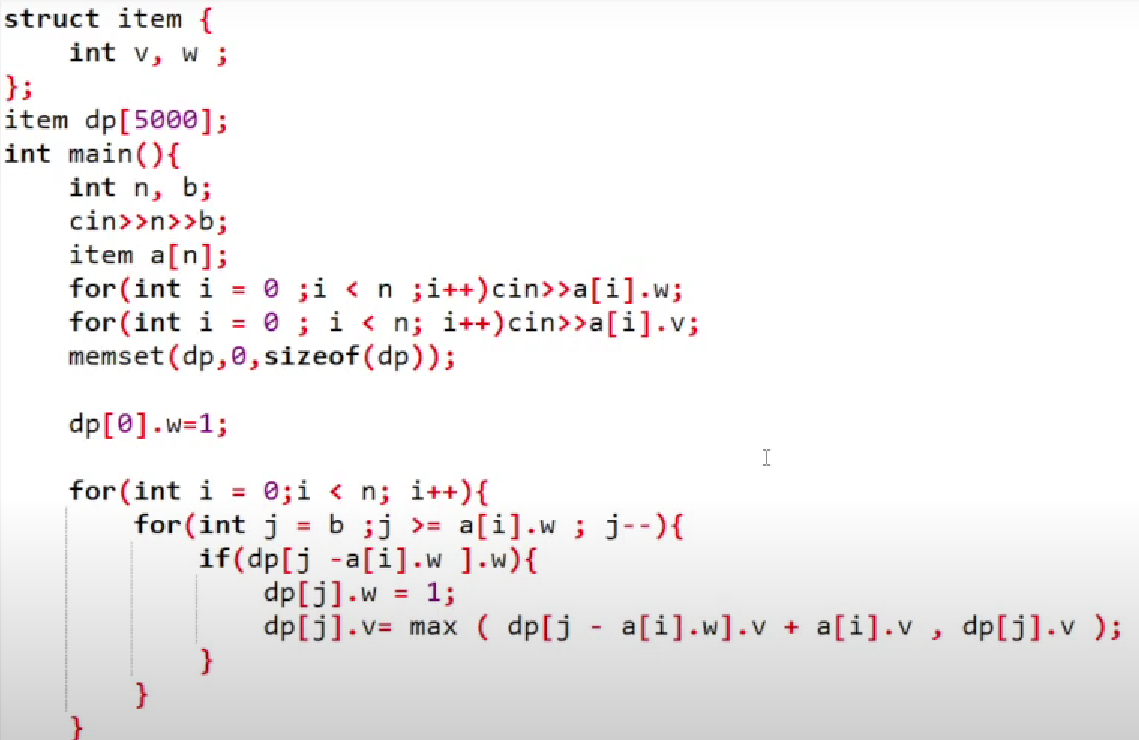


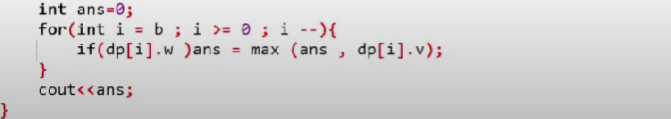












- Độ dài dãy con tăng dài nhất

Bài toán này là bài toán tìm độ dài của dãy con tăng dài nhất trong một dãy số đã cho. Ý tưởng chính của thuật toán quy hoạch động để giải quyết bài toán này là sử dụng một mảng động để lưu trữ chiều dài của dãy con tăng tại mỗi vị trí trong dãy.

Cụ thể, ta sẽ duyệt qua từng phần tử trong dãy số. Tại mỗi vị trí, ta sẽ duyệt lại qua các phần tử từ đầu đến vị trí hiện tại. Nếu phần tử tại vị trí hiện tại lớn hơn phần tử tại một vị trí trước đó và độ dài của dãy con tại vị trí đó cộng thêm 1 lớn hơn độ dài hiện tại của dãy con tại vị trí hiện tại, ta sẽ cập nhật độ dài của dãy con tại vị trí hiện tại bằng giá trị này.

int a[100], length[100],n, Do\_Dai\_Max;

int prev[100];

int main(){

    cin >> n;

    for(int i =0;i<n;i++){

        cin >> a[i];

    }

    /\* - Coi moi phan tu la mot day rieng biet - \*/

    for(int i=0;i<n;i++){

        length[i] = 1;

    /\* - Khởi tạo mảng prev với giá trị -1 để biểu thị không có phần tử trước đó - \*/

    }

   for(int i = 0; i < n; i++) {

        for(int j = 0; j < i; j++) {

            /\* Số đằng sau nhỏ hơn số đằng trước \*/

            if(a[j] < a[i]) {

                length[i] = max(length[i], length[j] + 1);

            }

        }

        /\* Cập nhật độ dài tối đa \*/

        Do\_Dai\_Max = max(Do\_Dai\_Max, length[i]);

   }

    cout << "\nDo dai max  : "<< Do\_Dai\_Max << endl;

}

int main(){

    int n;

    cin >> n;

    int \*a = new int[n];

    for(int i=0;i<n;i++){

        cin >> a[i];

    }

    int length[n];

    int Do\_Dai\_Max = 0;

    /\* - Côi mõi phần tử là một dãy tăng đần độ dài 1 - \*/

    for(int i=0;i<n;i++){

        length[i] =1;

    }

    for(int i=1;i<n;i++){

        for(int j=0;j<i;j++){

    /\* - Chỉ số sau > chỉ số trước \*/

            if(a[i] > a[j]){

              length[i] = max(length[i] , length[j] + 1);

            }

        }

       Do\_Dai\_Max = max(Do\_Dai\_Max, length[i]);

    }

    cout << Do\_Dai\_Max << endl;

}

- Dãy con tăng dài nhất

+ Bản chất của bài toán này là việc một thằng ở đằng sau sẽ tìm xem trước mình xem có thằng nào nó nhỏ hơn mình không , giả sử tìm được hai số nhỏ hơn thì nó sẽ ưu tiên cái thằng mà tại đó đang có chuỗi tăng dần dài hơn để nó nối vào sẽ được chuỗi tăng dần lớn hơn.

+ Ta sẽ dùng một cái mảng length[i] để lưu trữ độ dài của dãy con tăng dài nhất tính tử đầu mảng (chỉ số 0) đến.

// length[i] lưu trữ độ dài max của dãy con từ đầu đến vị trí i

int a[100], length[100], prev\_index[100], n, max\_length, end\_index;

int main() {

    cin >> n;

    // Nhập các phần tử của mảng

    for(int i = 0; i < n; i++) {

        cin >> a[i];

    }

    // Khởi tạo mảng length và prev\_index

    for(int i = 0; i < n; i++) {

        length[i] = 1;       // Mỗi phần tử ban đầu là một dãy con có độ dài là 1

        prev\_index[i] = -1;  // Không có phần tử trước đó

    }

    max\_length = 0; // Khởi tạo độ dài lớn nhất của dãy con tăng

    end\_index = 0;  // Vị trí kết thúc của dãy con tăng

    // Tính độ dài dãy con tăng lớn nhất và truy vết

    for(int i = 0; i < n; i++) {

        for(int j = 0; j < i; j++) {

            if(a[j] < a[i]) { // Kiểm tra nếu phần tử đằng trước nhỏ hơn phần tử hiện tại

                if(length[j] + 1 > length[i]) { // Nếu dãy con kết thúc tại a[j] có độ dài lớn hơn

                    length[i] = length[j] + 1;  // Cập nhật độ dài dãy con tại vị trí i

                    prev\_index[i] = j;          // Lưu vị trí phần tử trước đó

                }

            }

        }

        // Cập nhật độ dài lớn nhất và vị trí kết thúc của dãy con tăng

        if(length[i] > max\_length) {

            max\_length = length[i];

            end\_index = i;

        }

    }

    // In ra độ dài lớn nhất của dãy con tăng

    cout << "Do dai lon nhat cua day con tang: " << max\_length << endl;

    // Truy vết và in ra dãy con tăng

    cout << "Day con tang co do dai lon nhat: ";

    stack<int> Day\_Tang;

    int current\_index = end\_index;

    while(current\_index != -1) {

        Day\_Tang.push(a[current\_index]);

        current\_index = prev\_index[current\_index];

    }

    while(!Day\_Tang.empty()) {

        cout << Day\_Tang.top() << " ";

        Day\_Tang.pop();

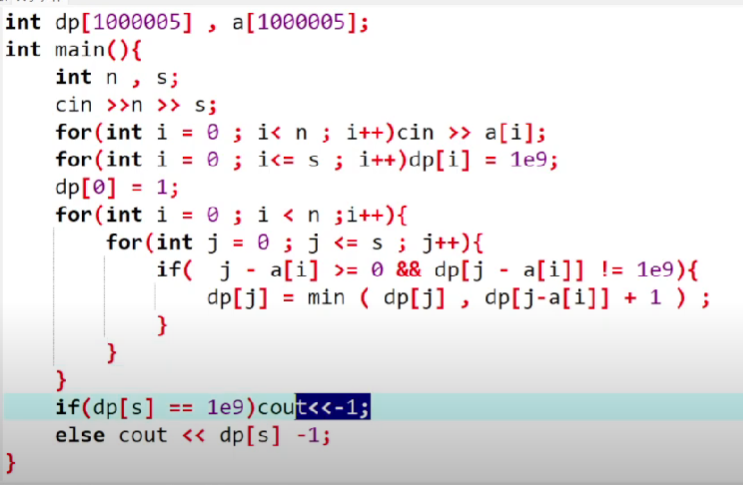
    }

    cout << endl;

    return 0;

}

- Đổi tiền 1 : Cho một dãy



tồn tại một dãy con của dãy a có tổng bằng S hay không.

Khởi tạo một mảng dp có kích thước (S+1), trong đó mỗi phần tử dp[i] sẽ đánh dấu xem có thể tạo ra tổng i từ các phần tử của dãy a hay không. Ban đầu, tất cả các phần tử của mảng dp được gán giá trị 0, nghĩa là chưa có cách nào để tạo ra tổng i.

Gán dp[0] bằng 1, vì luôn có thể tạo ra tổng 0 bằng cách không chọn bất kỳ phần tử nào từ dãy.

Duyệt qua từng phần tử a[i] của dãy. Đối với mỗi phần tử này, duyệt ngược từ S đến a[i].

Tại mỗi bước duyệt, kiểm tra xem có thể tạo ra tổng j từ các phần tử trước đó và phần tử hiện tại a[i] hay không. Nếu có thể tạo ra tổng j - a[i] từ các phần tử trước đó (dp[j - a[i]] == 1), thì có thể tạo ra tổng j từ các phần tử của dãy a bằng cách thêm phần tử a[i] vào. Do đó, gán dp[j] bằng 1.

Cuối cùng, in ra giá trị của dp[S]. Nếu dp[S] bằng 1, tức là có tồn tại một dãy con của dãy a có tổng bằng S, ngược lại, in ra 0.

Tóm lại, đoạn code này được sử dụng để kiểm tra xem có tồn tại một dãy con của dãy a có tổng bằng S hay không, thông qua việc sử dụng một mảng đánh dấu dp để theo dõi trạng thái của việc có thể tạo ra tổng từ dãy con đã chọn.

int n , S;

cin >> n >> S;

int a[n];

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

int dp[S + 1];

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

dp[i] = 0;

}

dp[0] = 1;

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

for(int j = S; j >= a[i] ; j--){

if( dp[j – a[i] ] == 1 ) dp[j] = 1;

}

}

cout << dp[S] << endl;

- Chia kẹo

Cho n𝑛 gói kẹo, gói thứ i𝑖 có ai𝑎𝑖 viên. Hãy chia các gói thành 2 phần sao cho chênh lệch giữa 2 phần là ít nhất.

- Ta sẽ tính tổng của tất cả số kẹo này, từ đó tìm ra được xem mỗi phần thì chứa vao nhiêu kẹo.

Sau khi tính tổng, ta sẽ chia tổng số lượng kẹo này cho 2 để tìm ra tổng số lượng kẹo mỗi phần cần có khi chia đều.

Sử dụng một mảng l để lưu trữ trạng thái của việc có thể tạo ra tổng j từ một phần của các gói kẹo. Ban đầu, tất cả các phần tử của mảng này được gán giá trị 0.

Sử dụng một vòng lặp while để kiểm tra và cập nhật giá trị của l. Trong mỗi bước lặp, ta duyệt qua từng gói kẹo và từng giá trị từ s đến a[i]. Nếu có thể tạo ra tổng j từ một phần của các gói kẹo, ta cập nhật l[j] thành 1.

Khi l[s] (tức là có thể tạo ra tổng s từ một phần của các gói kẹo), ta kết thúc vòng lặp và in ra tổng số kẹo trong mỗi phần, với phần thứ hai có thể tính được từ tổng số lượng kẹo của tất cả các gói trừ đi tổng số kẹo của phần thứ nhất.

int a[50], l[1000], n;

int main() {

cin >> n;

int s = 0;

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

cin >> a[i];

s += a[i];

}

int luu = s;

s = s / 2;

bool kt = true;

while (kt) {

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

for (int j = s; j >= a[i]; j--) {

if (l[j] == 0 && l[j - a[i]] == 1) {

l[j] = 1;

}

if (l[s] == 1) kt = false;

else s = s - 1;

}

}

}

cout << "Tổng số kẹo trong phần thứ nhất: " << s << endl;

cout << "Tổng số kẹo trong phần thứ hai: " << luu - s << endl;

return 0;

}

- Độ dài của xâu con chung dài nhất 