Câu 1:

* Mô tả :

1. Nhập vào số tiền cần đổi
2. Nhập số các đồng xu
3. Nhập mảng các đồng xu cần đổi
4. Duyệt qua từng đồng xu cần đổi rồi lấy số tiền cần đổi chia cho mệnh giá xu ta được số xu đổi tại mệnh giá đó
5. Lấy sô tiền chia dư cho mệnh giá xu tại I ta được số tiền còn lại
6. Lặp cho đến khi số tiền = 0 hoặc ko thể chia được nữa

* Code:

#include <iostream>

using namespace std;

void doiTien(int soTien, int loaiXu[], int n ){

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

int solgXu = soTien / loaiXu[i];

if(solgXu > 0){

cout<<"SỐ lượng xu "<<loaiXu[i]<<": "<<solgXu<<endl;

soTien %= loaiXu[i];

cout<<"số tiền còn lại: "<<soTien<<endl;

}

}

}

int main()

{

int soTien;

cout<<"nhập số tiền cần đổi: ";

cin>>soTien;

int n;

cout<<"nhập số loại xu: ";

cin>>n;

int loaiXu[n];

cout<<"Nhập giá trị các loại xu: "<<endl;

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

cin>>loaiXu[i];

}

doiTien(soTien, loaiXu, n);

return 0;

}

* Đánh giá thuật toán:
* Vòng lặp có độ phức tạp là O(n)
* T(n) = O(n)

Câu 2

| **Mặt hàng** | **Trọng lượng** | **Giá trị** |
| --- | --- | --- |
| A | 18 | 25 |
| B | 15 | 24 |
| C | 10 | ? |

Giả sử sức chứa của cái túi là **M = 20** (trọng lượng).

1. Mô tả chi tiết thuật toán

• Nhập dữ liệu: Người dùng nhập sức chứa của túi (sucChua) và thông tin về mỗi mặt hàng, bao gồm tên (ten), trọng lượng (trongLuong), và giá trị (giaTri).

• Sắp xếp mảng mặt hàng: Sử dụng thuật toán sắp xếp nổi bọt để sắp xếp mảng matHangs theo tỷ lệ giá trị/trọng lượng giảm dần.

• Chọn mặt hàng: Duyệt qua mảng đã sắp xếp và chọn mặt hàng theo quy tắc sau: Nếu mặt hàng có trọng lượng nhỏ hơn hoặc bằng sức chứa còn lại của túi, chọn toàn bộ mặt hàng. Nếu mặt hàng có trọng lượng lớn hơn sức chứa còn lại của túi, chọn một phần của mặt hàng để đảm bảo sức chứa của túi được sử dụng hết.

• In ra màn hình danh sách mặt hàng được chọn và tổng giá trị của túi.

#include <iostream>

#include <string>

using namespace std;

struct MatHang {

    string ten;   // Tên của mặt hàng (thay đổi từ char thành string để lưu chuỗi tên)

    int trongLuong; // Trọng lượng của mặt hàng

    int giaTri;     // Giá trị của mặt hàng

};

// Hàm so sánh tỷ lệ giá trị/trọng lượng giữa hai mặt hàng

bool soSanhMatHang(const MatHang &a, const MatHang &b) {

    return ((double)a.giaTri / a.trongLuong) > ((double)b.giaTri / b.trongLuong);

}

// Hàm sắp xếp danh sách mặt hàng theo tỷ lệ giá trị/trọng lượng giảm dần (Sử dụng thuật toán sắp xếp nổi bọt)

void sapXepNoiBot(MatHang matHangs[], int soLuongMatHang) {

    for (int i = 0; i < soLuongMatHang - 1; ++i) {

        for (int j = 0; j < soLuongMatHang - i - 1; ++j) {

            // So sánh nếu mặt hàng j có tỷ lệ giá trị/trọng lượng nhỏ hơn mặt hàng j+1

            if (!soSanhMatHang(matHangs[j], matHangs[j + 1])) {

                swap(matHangs[j], matHangs[j + 1]); // Đổi chỗ nếu cần

            }

        }

    }

}

// Hàm giải quyết bài toán chọn mặt hàng sao cho tổng giá trị tối đa mà không vượt quá sức chứa

void caiTuiDangPhanSo(MatHang matHangs[], int soLuongMatHang, int sucChua) {

    sapXepNoiBot(matHangs, soLuongMatHang); // Sắp xếp các mặt hàng theo tỷ lệ giá trị/trọng lượng

    double tongGiaTri = 0.0;  // Tổng giá trị của các mặt hàng trong túi

    int trongLuongHienTai = 0;  // Trọng lượng hiện tại của túi

    cout << "Danh sach mat hang duoc chon:" << endl;

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

        if (trongLuongHienTai + matHangs[i].trongLuong <= sucChua) {

            // Nếu có đủ chỗ trong túi, chọn toàn bộ mặt hàng

            trongLuongHienTai += matHangs[i].trongLuong;

            tongGiaTri += matHangs[i].giaTri;

            cout << "Chon toan bo mat hang " << matHangs[i].ten << " (Trong luong: "

                 << matHangs[i].trongLuong << ", Gia tri: " << matHangs[i].giaTri << ")" << endl;

        }

    }

    cout << "Tong gia tri cua tui la: " << tongGiaTri << endl;  // In tổng giá trị của các mặt hàng được chọn

}

int main() {

    int soLuongMatHang = 3;  // Số lượng mặt hàng

    MatHang matHangs[] = {

        {"Mat hang 1", 10, 60},  // Mặt hàng 1 (Tên, Trọng lượng, Giá trị)

        {"Mat hang 2", 20, 100},

        {"Mat hang 3", 30, 120},

        {"Mat hang 4", 40, 240},

        {"Mat hang 5", 50, 200}

    };

    int sucChua = 31;

    caiTuiDangPhanSo(matHangs, soLuongMatHang, sucChua);  // Tính toán và in ra kết quả

    return 0;

}

4. Đánh giá thời gian thực hiện của thuật toán

Các phép so sánh, tính toán: O(1)

Nhập dữ liệu: Thời gian nhập dữ liệu từ bàn phím có độ phức tạp tuyến tính O(n), với n là số lượng mặt hàng.

Sắp xếp nổi bọt: Độ phức tạp là O(n^2) trong trường hợp xấu nhất.

Chọn mặt hàng: Vòng lặp duyệt qua mảng sắp xếp có độ phức tạp tuyến tính O(n).

In kết quả: Độ phức tạp tuyến tính O(n).

Thời gian thực hiện thuật toán trên như sau: T(1) = O(1) T(n) = O(1) + O(n) + O(n^2) + O(n) + O(n)

Vậy T(n) = O(n^2) với n là số lượng mặt hàng

Câu 4:

1. Mô tả :

* Tìm dãy con có tổng lớn nhất kết thúc tại vị trí k trong dãy, khi đó có 2 trường hợp:
* Dãy con chỉ chứa phần thử thứ k. So sánh best với phần tử thứ k để cập nhật giá trị lớn nhất
* Dãy con chứa phần tử thứ k và dãy con kết thúc ở phần thử thứ k - 1. Cập nhật tổng mới dựa trên tổng của dãy con k - 1 và cộng thêm phần tử thứ k. So sánh tổng mới với best để cập nhật giá trị lớn nhất.

1. Code:

#include <bits/stdc++.h>

using namespace std;

void findSubArrayMax(int arr[], int n) {

int best = INT\_MIN, sum = 0;

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

sum = max(arr[i], sum + arr[i]);

best = max(best, sum);

}

cout << "Tong lon nhat cua day con la: " << best << "\n";

}

void findSubArrayMaxWithIndices(int arr[], int n) {

int best = INT\_MIN, sum = 0;

int best\_start = 0, best\_end = 0, current\_start = 0;

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

if (sum + arr[i] < arr[i]) {

current\_start = i;

sum = arr[i];

} else {

sum += arr[i];

}

if (best < sum) {

best = sum;

best\_start = current\_start;

best\_end = i;

}

}

cout << "Tong lon nhat cua day con la: " << best << "\n";

cout << "Vi tri bat dau: " << best\_start << ", vi tri ket thuc: " << best\_end << "\n";

}

int main() {

int n;

cout << "Nhap so luong phan tu cua mang: ";

cin >> n;

int arr[n];

cout << "Nhap cac phan tu cua mang: ";

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

cin >> arr[i];

}

findSubArrayMax(arr, n);

findSubArrayMaxWithIndices(arr, n);

return 0;

}

1. Thực hiện bài toán với 1 bộ dữ liệu cụ thể bằng thuật toán đã xây dựng

* Nhập vào dãy {-2,1,-3,4,-1,2,1,-5,-4}
* Int n = 9;
* Int best = INT\_MIN, sum = 0;

i arr[i] sum best

0 -2 max(-2, 0 + -2) = -2 max(-∞, -2) = -2

1 1 max(1, -2 + 1) = 1 max(-2, 1) = 1

2 -3 max(-3, 1 + -3) = -2 max(1, -2) = 1

3 4 max(4, -2 + 4) = 4 max(1, 4) = 4

4 -1 max(-1, 4 + -1) = 3 max(4, 3) = 4

5 2 max(2, 3 + 2) = 5 max(4, 5) = 5

6 1 max(1, 5 + 1) = 6 max(5, 6) = 6

7 -5 max(-5, 6 + -5) = 1 max(6, 1) = 6

8 4 max(4, 1 + 4) = 5 max(6, 5) = 6

Kết quả cuối best = 6 start i=3 end i=6

1. Đánh giá thời gian thuật toán

Chủ yếu là vòng lặp có thời gian là O(n):

* T(n) = O(n)

Câu 5:

1. Mô tả :

* Gọi F[m,v] là số cách phân tích số v thành tổng của các số nguyên dương ≤ m. Như vậy  
  có chứa ít nhất 1 số m trong phép phân tích. Khi đó nếu trong cách phân tích ta bỏ đi số  
  m thì ta sẽ được các cách phân tích số v – m thành tổng các số nguyên dương ≤ m.
* Ta có công thức quy hoạch động sau:
* F[0,k]=1 ; k: 1 ≤ k ≤ n
* i: 1 ≤ i ≤ n: Ɐ  
  F[i,k] = F[i,k-1] + F[i-k,k] ; k: 1 ≤ k ≤ i. Ɐ  
  F[i,k] = F[i,k-1] ; k: i+1 ≤ k ≤ n. Ɐ

Kết quả là F[n][n]

1. Code :

#include <iostream>

using namespace std;

const int MAX\_N = 100;

// Hàm tính số cách phân tích n thành tổng của các số nguyên không lớn hơn n

void calculateWays(int n, int f[MAX\_N][MAX\_N]) {

for (int k = 1; k <= n; k++)

f[0][k] = 1; // Trường hợp cơ sở

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

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

if (k <= i)

f[i][k] = f[i][k - 1] + f[i - k][k];

else

f[i][k] = f[i][k - 1];

}

}

}

int main() {

int n, f[MAX\_N][MAX\_N] = {0}; // Khởi tạo mảng với giá trị 0

// Nhập giá trị n

cout << "Vui long nhap n (n < 100): ";

cin >> n;

// Kiểm tra đầu vào

if (n < 0 || n >= 100) {

cout << "Vui long nhap so nguyen n trong khoang (0, 100)." << endl;

return 1;

}

// Tính toán số cách

calculateWays(n, f);

// Xuất kết quả

cout << "So cach: " << f[n][n] << endl;

return 0;

}

// có thể tối ưu xuống O(n):

f[0] = 1; // Có duy nhất 1 cách phân tích 0: không có số nào // Tính số cách phân tích for (int k = 1; k <= n; k++) { for (int i = k; i <= n; i++) { f[i] += f[i - k]; } }

1. Thực hiện bài toán với 1 bộ dữ liệu cụ thể bằng thuật toán đã xây dựng

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 | 2 | 3 | 3 |
| 3 | 1 | 1 | 2 | 3 | 4 | 5 |
| 4 | 1 | 1 | 2 | 3 | 5 | 6 |
| 5 | 1 | 1 | 2 | 3 | 5 | 7 |

Nhập vào số nguyên 5:

Theo bảng ta thấy có 7 cách phân tích 5 thành tổng các số nhỏ hơn 5

1. Đánh giá độ phức tạp thuật toán:

* 2 vòng lặp lồng nhau thì độ phức tạp là O(n2)
* T(n) = O(n2)

Câu 6:

1. Mô tả :

Hàm Ok(x2, y2):• Hàm kiểm tra xem có thể đặt quân hậu tại vị trí (x2, y2) trên bàn cờ mà  
không bị tấn công hay không.  
• Kiểm tra xem có quân hậu nào trên cùng một cột y2 không, và có quân hậu  
nào trên cùng một đường chéo với (x2, y2) không.  
Nếu có bất kỳ quân hậu nào tấn công, hàm trả về false, ngược lại trả về true.  
Hàm Xuat(n):• Hàm này in ra một lời giải, tức là cách đặt quân hậu trên từng dòng của bàn cờ.  
Hàm Try(i, n):• Hàm này là hàm quay lui chính để thử tất cả các cách đặt quân hậu trên dòng thứ  
i.  
• Vòng lặp for thử đặt quân hậu vào từng cột trên dòng i.  
• Nếu có thể đặt quân hậu tại một vị trí j trên dòng i mà không bị tấn công, thì đặt  
quân hậu và gọi đệ quy cho dòng tiếp theo i+1.  
• Nếu không tìm được vị trí phù hợp, hàm sẽ quay lui (backtrack) để thử các vị trí  
khác cho dòng i-1.  
Hàm main():• Trong hàm main(), biến n được đặt là 8, tượng trưng cho bàn cờ 8x8 và số quân  
hậu là 8

1. Code

#include <iostream>

#include <cmath>

using namespace std;

// Mảng a[] lưu cột mà mỗi con hậu nằm trên từng hàng.

// Ví dụ: a[1] = 3 nghĩa là con hậu ở hàng 1 nằm trên cột 3.

int a[8];

int soCach = 0; // Đếm số cách đặt hợp lệ

// Kiểm tra xem vị trí (x2, y2) có hợp lệ không

bool Ok(int x2, int y2) {

for (int i = 1; i < x2; i++) {

// Kiểm tra: cùng cột hoặc cùng đường chéo

if (a[i] == y2 || abs(i - x2) == abs(a[i] - y2))

return false; // Không hợp lệ

}

return true; // Hợp lệ

}

// In trạng thái bàn cờ

void XuatBanCo(int n) {

cout << "Trang thai ban co voi " << n << " con hau:\n";

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

for (int j = 1; j <= 8; j++) {

if (a[i] == j)

cout << "Q "; // In quân hậu

else

cout << ". "; // Ô trống

}

cout << endl;

}

cout << endl;

}

// Hàm quay lui để đặt quân hậu

void Try(int i, int n) {

for (int j = 1; j <= 8; j++) { // Thử đặt ở cột 1 -> 8

if (Ok(i, j)) { // Nếu đặt được

a[i] = j; // Đặt quân hậu tại cột j

if (i == n) { // Nếu đã đặt đủ n quân hậu

XuatBanCo(n); // In bàn cờ

soCach++; // Đếm số cách đặt

} else {

Try(i + 1, n); // Đặt tiếp hàng tiếp theo

}

}

}

}

int main() {

int target;

cout << "Nhap so con hau can dat (5, 6, 7): ";

cin >> target;

// Giới hạn số con hậu cần đặt

if (target < 5 || target > 7) {

cout << "Chi ho tro dat tu 5 den 7 con hau!" << endl;

return 1;

}

Try(1, target); // Bắt đầu đặt từ hàng 1

cout << "Da dat xong " << target << " con hau voi " << soCach << " cach." << endl;

return 0;

}

1. Thực hiện bài toán với 1 bộ dữ liệu cụ thể bằng thuật toán đã xây dựng

Với 5 quân hậu thì có 568 cách đặt

1. Đánh giá độ phức tạp của thuật toán

Ta có hàm Ok độ phức tạp là O(n)

Hàm Try có độ phức tạp là O(n^n) vì mỗi lần đệ quy thì hàm Ok sẽ được gọi

Các câu lệnh điều kiện, in xuất,… có độ phức tạp là O(1)

Thời gian thực hiện giải thuật trên như sau:

T(1) = O(1)

T(n) = O(n^n)

Vậy T(n) = O(n^n) với n là kích thước của bàn cờ

Câu 8:

1. Mô tả thuật toán

• Chia để trị  
♣ Kiểm tra điều kiện dừng  
o Nếu batDau bằng ketThuc tức là đã giảm đến mức cơ bản,  
trả về giá trị của batDau  
♣ Chia khoảng:  
o Giá trị trung bình giua của khoảng [batDau, ketThuc]  
♣ Đệ quy bên trái và bên phải:  
o Gọi đệ quy cho hai khoảng con [batDau, giua] và [giua + 1,  
ketThuc] để tính giá trị giai thừa của các khoảng con này.  
o Gọi hai lần đệ quy với batDau đến giua và giua + 1 đến  
ketThuc để tìm giá trị giai thừa của các phần nhỏ hơn.  
♣ Kết hợp kết quả:  
o Nhân kết quả của hai đệ quy lại với nhau, tức là trai \* phai.  
lOMoARcPSD|50827439  
o Kết quả cuối cùng sẽ là giá trị giai thừa của khoảng  
[batDau, ketThuc].  
• Quy hoạch động  
o Tạo một mảng dp với kích thước (n+1).  
o Khởi tạo dp[0] = 1.  
o Vòng lặp i từ 1 đến n:  
♣ Gán dp[i] = i \* dp[i-1].  
o Trả về dp[n]

1. Code

#include <iostream>

using namespace std;

// Hàm tính giai thừa dùng Quy hoạch động (không dùng vector)

int giaiThuaQHD(int n) {

int\* dp = new int[n + 1]; // Khai báo mảng động

dp[0] = 1; // 0! = 1

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

dp[i] = i \* dp[i - 1];

}

int result = dp[n]; // Lưu kết quả cuối cùng

delete[] dp; // Giải phóng bộ nhớ

return result;

}

// Hàm tính giai thừa dùng Chia để trị

int tinhGiaiThuaChiaDeTri(int batDau, int ketThuc) {

if (batDau > ketThuc) return 1; // Trường hợp cơ sở: n = 0

if (batDau == ketThuc) return batDau; // Một phần tử

int giua = (batDau + ketThuc) / 2;

return tinhGiaiThuaChiaDeTri(batDau, giua) \* tinhGiaiThuaChiaDeTri(giua + 1, ketThuc);

}

int main() {

int luaChon, n;

do {

cout << "\n===== Menu =====" << endl;

cout << "1. Tinh giai thua su dung Chia de tri" << endl;

cout << "2. Tinh giai thua su dung Quy hoach dong" << endl;

cout << "0. Thoat" << endl;

cout << "Nhap lua chon cua ban: ";

cin >> luaChon;

if (luaChon == 1 || luaChon == 2) {

cout << "Nhap gia tri n (n >= 0): ";

cin >> n;

if (n < 0) {

cout << "Gia tri n khong hop le. Vui long nhap n >= 0." << endl;

continue;

}

if (luaChon == 1) {

int ketQuaChiaDeTri = tinhGiaiThuaChiaDeTri(1, n);

cout << n << "! (Chia de tri) = " << ketQuaChiaDeTri << endl;

} else {

int ketQuaQHD = giaiThuaQHD(n);

cout << n << "! (Quy hoach dong) = " << ketQuaQHD << endl;

}

} else if (luaChon == 0) {

cout << "Ket thuc chuong trinh." << endl;

} else {

cout << "Lua chon khong hop le. Vui long chon lai." << endl;

}

} while (luaChon != 0);

return 0;

}

1. Thực hiện bài toán với 1 bộ dữ liệu cụ thể bằng thuật toán đã xây dựng

Ví dụ:  
1. Tinh giai thua su dung chia de tri  
2. Tinh giai thua su dung quy hoach dong  
0. Thoat  
Nhap lua chon cua ban: 1  
Nhap gia tri n de tinh n!: 5  
5! tinh theo chia de tri = 120

1. Đánh giá độ phức tạp của thuật toán

* **Quy hoạch động:** O(n), vì cần mảng dp để lưu trữ giá trị trung gian.
* **Chia để trị:** O(log2­(n))do sử dụng ngăn xếp đệ quy ,mỗi lần chia đôi độ dài giảm đi một nửa

Câu 9:

1. Mô tả:

Hàm Swap: Để hoán đổi 2 phần tử

o Hàm partition:

* Hàm này thực hiện quá trình phân hoạch, chọn một phần tử làm

pivot (ở đây là phần tử đầu tiên của dãy).

* Duyệt qua các phần tử từ left + 1 đến right và đếm số lượng phần tử

nhỏ hơn hoặc bằng pivot.

* Hoán đổi pivot với phần tử tại vị trí left + count.
* Sử dụng hai con trỏ i và j để duyệt qua mảng và đảm bảo các phần tử

bên trái pivot là nhỏ hơn hoặc bằng, các phần tử bên phải pivot là

lớn hơn pivot.

o Hàm quickSort:

* Hàm này thực hiện sắp xếp dãy số nguyên sử dụng thuật toán

QuickSort thông qua phương pháp chia để trị.

* Nếu left không lớn hơn hoặc bằng right, tiếp tục quá trình phân

hoạch và đệ quy cho hai phần: bên trái pivot và bên phải pivot

1. Code

#include <iostream>

using namespace std;

// Hàm hoán đổi hai số

void hoanDoi(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

// Hàm chia mảng thành hai phần (Partition)

int phanHoach(int arr[], int low, int high) {

int pivot = arr[high]; // Chọn phần tử cuối làm pivot

int i = low - 1; // Chỉ số của phần tử nhỏ hơn pivot

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) { // Nếu phần tử nhỏ hơn hoặc bằng pivot

i++;

hoanDoi(arr[i], arr[j]); // Đưa phần tử đó về phía trái của pivot

}

}

// Đưa pivot về đúng vị trí

hoanDoi(arr[i + 1], arr[high]);

return i + 1; // Trả về vị trí của pivot

}

// Hàm QuickSort

void quickSort(int arr[], int low, int high) {

if (low < high) {

// Phân hoạch mảng và trả về vị trí của pivot

int pi = phanHoach(arr, low, high);

// Gọi đệ quy sắp xếp các phần tử bên trái và phải của pivot

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

// Hàm hiển thị mảng

void hienThiMang(int arr[], int n) {

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

cout << arr[i] << " ";

}

cout << endl;

}

int main() {

int n;

// Nhập số phần tử

cout << "Nhap so phan tu cua day: ";

cin >> n;

int arr[n];

cout << "Nhap cac phan tu cua day: ";

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

cin >> arr[i];

}

// Sắp xếp mảng bằng QuickSort

quickSort(arr, 0, n - 1);

// Hiển thị mảng sau khi sắp xếp

cout << "Day sau khi sap xep: ";

hienThiMang(arr, n);

return 0;

}

1. Thực hiện bài toán với 1 bộ dữ liệu cụ thể bằng thuật toán đã xây dựng

Ví dụ:  
Nhap so luong phan tu cua day: 10  
Nhap cac phan tu cua day: 2 1 2 1 0 5 0 9 9 9  
Day sau khi sap xep: 0 0 1 1 2 2 5 9 9 9

1. Đánh giá độ phức tạp thuật toán

Hàm partition():  
o Vòng lặp đầu tiên có độ phức tạp O(n) vì duyệt qua các phần tử để đếm số  
lượng nhỏ hơn hoặc bằng pivot  
o Vòng lặp thứ 2 cũng có độ phức tạp là O(n)  
o Vì thế nên hàm partition() có độ phức tạp là O(n)  
• Hàm quickSort:  
o Đối với mỗi lời gọi đệ quy, thì độ phức tạp của quickSort là O(n) (tổng của  
các phần partition).  
o Số lượng lời gọi đệ quy là logn , n là số lượng phần tử  
o Vì thế nên hàm quickSort có độ phức tạp là O(nlogn)  
• Các câu lệnh nhập xuất,…. Có độ phức tạp là O(n) do cần lặp qua các phần tử  
• Vậy độ phức tạp của chương trình là O(nlogn)

Câu 10:

1. Mô tả:

* Nếu mảng có 1 phần tử hoặc rỗng, không cần sắp xếp.
* Chia mảng thành hai nửa:
  + - 1. Nửa trái: từ chỉ số bắt đầu đến giữa.
      2. Nửa phải: từ giữa + 1 đến chỉ số cuối.
* Gọi đệ quy MergeSort cho cả hai nửa.
* Gộp hai mảng đã sắp xếp thành một mảng sắp xếp hoàn chỉnh

1. Code

#include <iostream>

using namespace std;

// Hàm hợp nhất hai mảng con đã sắp xếp

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

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

for (int i = 0; i < n1; i++)

L[i] = A[left + i];

for (int j = 0; j < n2; j++)

R[j] = A[mid + 1 + j];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

A[k] = L[i];

i++;

} else {

A[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

A[k] = L[i];

i++;

k++;

}

while (j < n2) {

A[k] = R[j];

j++;

k++;

}

}

// Hàm MergeSort

void mergeSort(int A[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(A, left, mid);

mergeSort(A, mid + 1, right);

merge(A, left, mid, right);

}

}

// Hàm in mảng

void printArray(int A[], int size) {

for (int i = 0; i < size; i++)

cout << A[i] << " ";

cout << endl;

}

// Hàm chính

int main() {

int n;

// Nhập số phần tử của mảng

cout << "Nhập số phần tử của mảng: ";

cin >> n;

int A[n];

// Nhập các phần tử của mảng

cout << "Nhập các phần tử của mảng: ";

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

cin >> A[i];

}

cout << "Dãy ban đầu: ";

printArray(A, n);

mergeSort(A, 0, n - 1);

cout << "Dãy sau khi sắp xếp: ";

printArray(A, n);

return 0;

}

1. Thực hiện bài toán với 1 bộ dữ liệu cụ thể bằng thuật toán đã xây dựng

Nhập số phần tử của mảng: 6

Nhập các phần tử của mảng: 12 11 13 5 6 7

Dãy ban đầu: 12 11 13 5 6 7

Dãy sau khi sắp xếp: 5 6 7 11 12 13

1. Đánh giá độ phức tạp của thuật toán

Hàm mergeSort được gọi đệ quy hai lần cho mỗi nửa của mảng và sau đó gọi hàm  
merge. Do đó độ phức tạp là T(n)=2T(n/2)+O(n)  
Ở mỗi lần đệ quy trộn 2 mảng có độ phức tạp là O(n)  
Do đó độ phức tạp là T(n) = O(nlogn)

Câu 10:

1. Mô tả:

Khác 1 một ít so với bài toán cái túi phân số, Chọn đồ vật để cho vào túi sao cho tổng giá trị là lớn nhất, không vượt quá sức chứa của túi. Một đồ vật có thể được chọn nhiều lần (bài toán **không giới hạn**).

* **Sắp xếp danh sách đồ vật** theo tỷ lệ giaˊ trị/trọng lượng\text{giá trị/trọng lượng}giaˊ trị/trọng lượng giảm dần để ưu tiên chọn đồ vật có "hiệu quả" cao nhất trước.
* Áp dụng thuật toán tham lam: Thêm đồ vật vào túi cho đến khi không thể thêm nữa.

1. Code

#include <iostream>

using namespace std;

// Định nghĩa cấu trúc Đồ Vật

struct DoVat {

int index; // Số thứ tự

int trongLuong; // Trọng lượng

int giaTri; // Giá trị

};

// Hàm hợp nhất hai mảng con

void merge(DoVat arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

// Tạo mảng con

DoVat \*L = new DoVat[n1];

DoVat \*R = new DoVat[n2];

for (int i = 0; i < n1; i++) L[i] = arr[left + i];

for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];

// Hợp nhất hai mảng con theo tỷ lệ giá trị/trọng lượng

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

double tyLeL = (double)L[i].giaTri / L[i].trongLuong;

double tyLeR = (double)R[j].giaTri / R[j].trongLuong;

if (tyLeL >= tyLeR) {

arr[k++] = L[i++];

} else {

arr[k++] = R[j++];

}

}

// Sao chép các phần tử còn lại

while (i < n1) arr[k++] = L[i++];

while (j < n2) arr[k++] = R[j++];

delete[] L;

delete[] R;

}

// Hàm sắp xếp Merge Sort

void mergeSort(DoVat arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

// Hàm cài đặt giải thuật tham lam

void caiTuiThamLamKhongGioiHan(DoVat dsDoVat[], int soLuong, int sucChua) {

// Sắp xếp danh sách đồ vật theo tỷ lệ giá trị/trọng lượng giảm dần

mergeSort(dsDoVat, 0, soLuong - 1);

// Biến theo dõi trọng lượng và giá trị hiện tại

int trongLuongHienTai = 0;

int giaTriTong = 0;

cout << "Danh sach cac do vat duoc chon:" << endl;

// Lặp qua danh sách đồ vật

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

// Thêm đồ vật vào túi nếu còn chỗ

while (trongLuongHienTai + dsDoVat[i].trongLuong <= sucChua) {

trongLuongHienTai += dsDoVat[i].trongLuong;

giaTriTong += dsDoVat[i].giaTri;

cout << "Chon do vat " << dsDoVat[i].index

<< " (Trong luong: " << dsDoVat[i].trongLuong

<< ", Gia tri: " << dsDoVat[i].giaTri << ")" << endl;

}

}

// In kết quả tổng

cout << "Tong trong luong cua cac do vat duoc chon: " << trongLuongHienTai << endl;

cout << "Tong gia tri cua cac do vat duoc chon: " << giaTriTong << endl;

}

// Hàm chính

int main() {

int soLuong, sucChua;

// Nhập số lượng đồ vật và sức chứa của túi

cout << "Nhap so luong do vat: ";

cin >> soLuong;

cout << "Nhap suc chua toi da cua tui: ";

cin >> sucChua;

// Mảng lưu các đồ vật

DoVat dsDoVat[soLuong];

// Nhập thông tin từng đồ vật

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

dsDoVat[i].index = i + 1;

cout << "Nhap trong luong cua do vat " << i + 1 << ": ";

cin >> dsDoVat[i].trongLuong;

cout << "Nhap gia tri cua do vat " << i + 1 << ": ";

cin >> dsDoVat[i].giaTri;

}

// Thực hiện bài toán

caiTuiThamLamKhongGioiHan(dsDoVat, soLuong, sucChua);

return 0;

}

1. Đánh giá thời gian thực hiên

Hàm mergeSort(): với vòng lặp thứ 1 duyệt từ 1 đến n phần tử và vòng lặp thứ 2  
duyệt từ 1 đến n-1 phần tử nên có độ phức tạp là O(nlogn)  
- Sau khi danh sách được sắp xếp, thuật toán lặp qua danh sách đồ vật để thêm vào túi.

- Vòng lặp chạy qua nnn đồ vật. Trong trường hợp mỗi đồ vật có thể được thêm vào nhiều lần (khi không giới hạn số lượng), số lần lặp thêm phụ thuộc vào sức chứa WWW của túi.

- Độ phức tạp của bước này trong trường hợp xấu nhất là O(n+W/min\_weight) với min\_weight là trọng lượng nhỏ nhất của một đồ vật.Vì mergeSort() chiếm độ phức tạp đa số trong chương trình nên chương trình trên  
có độ phức là T(n) = O(nlogn) + O(n+W/min\_weight)

Câu 12:

1. Mô tả:

**Đầu vào**:

* Một dãy số nguyên dương (a0,a1,…,an−1) với n phần tử.
* Một số nguyên dương MMM, là tổng cần tìm.

**Đầu ra**:

* Tất cả các dãy con sao cho tổng của các phần tử trong dãy con bằng MMM.

**Phương pháp quay lui (backtracking)**:

* Xây dựng dần dãy con bằng cách duyệt qua từng phần tử của mảng.
* Ở mỗi bước, quyết định có thêm phần tử hiện tại vào dãy con hay không.
* Nếu tổng của dãy con bằng M, in kết quả.
* Nếu tổng vượt quá M, quay lui để thử các khả năng khác.

1. Code :

#include <iostream>

using namespace std;

const int SO\_LUONG\_TOI\_DA = 100;

// Hàm đệ quy tìm các dãy con có tổng bằng M

void timDayCon(int mang[], int soLuong, int M, int tongHienTai, int chiSoBatDau, int duongDi[], int chiSoDuongDi) {

// Điều kiện dừng: nếu tổng hiện tại bằng M

if (tongHienTai == M) {

cout << "Day con co tong " << M << ": ";

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

cout << duongDi[i] << " ";

}

cout << endl;

return;

}

// Nếu tổng vượt M hoặc hết phần tử để xét, dừng lại

if (tongHienTai > M || chiSoBatDau >= soLuong) {

return;

}

// Thử tất cả các khả năng

for (int i = chiSoBatDau; i < soLuong; ++i) {

duongDi[chiSoDuongDi] = mang[i]; // Chọn phần tử hiện tại

// Gọi đệ quy để xét các phần tử tiếp theo

timDayCon(mang, soLuong, M, tongHienTai + mang[i], i + 1, duongDi, chiSoDuongDi + 1);

duongDi[chiSoDuongDi] = 0; // Quay lui để thử khả năng khác

}

}

int main() {

int soLuong, M;

// Nhập số lượng phần tử

cout << "Nhap so luong phan tu trong day: ";

cin >> soLuong;

int mang[SO\_LUONG\_TOI\_DA];

// Nhập mảng

cout << "Nhap day so nguyen: ";

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

cin >> mang[i];

}

// Nhập tổng cần tìm

cout << "Nhap tong can tim: ";

cin >> M;

int duongDi[SO\_LUONG\_TOI\_DA] = {0};

// Gọi hàm tìm dãy con

cout << "Cac day con co tong " << M << " la:" << endl;

timDayCon(mang, soLuong, M, 0, 0, duongDi, 0);

return 0;

}

1. Đánh giá độ phức tạp

**Thời gian**: O(2^n), do duyệt qua tất cả các tổ hợp phần tử.

Câu 13:  
  
#include <iostream>

#include <vector>

using namespace std;

// Hàm quay lui

void backtrack(vector<int>& A, int M, int index, int current\_sum, vector<int>& current, vector<vector<int>>& result) {

// Nếu tổng hiện tại bằng M, lưu lại kết quả

if (current\_sum == M) {

result.push\_back(current);

return;

}

// Nếu tổng vượt quá M, dừng

if (current\_sum > M) {

return;

}

// Duyệt qua các phần tử từ vị trí hiện tại

for (int i = index; i < A.size(); i++) {

current.push\_back(A[i]); // Thêm phần tử vào kết quả tạm thời

backtrack(A, M, i + 1, current\_sum + A[i], current, result); // Gọi đệ quy

current.pop\_back(); // Loại bỏ phần tử để backtrack

}

}

// Hàm chính để tìm các dãy con

vector<vector<int>> findCombinations(vector<int>& A, int M) {

vector<vector<int>> result; // Lưu trữ kết quả

vector<int> current; // Kết quả tạm thời

backtrack(A, M, 0, 0, current, result); // Gọi hàm quay lui

return result;

}

int main() {

vector<int> A = {7, 1, 4, 3, 5, 6}; // Dãy số

int M = 11; // Tổng cần tìm

// Tìm các tổ hợp

vector<vector<int>> combinations = findCombinations(A, M);

// In kết quả

cout << "Các tổ hợp có tổng bằng " << M << " là:\n";

for (const auto& combination : combinations) {

cout << "{ ";

for (int num : combination) {

cout << num << " ";

}

cout << "}\n";

}

return 0;

}  
  
  
độ phức tạp O(n⋅2n).  
  
  
**Giải thích:**

1. **Hàm backtrack**:
   * **Tham số**:
     + A: Dãy số ban đầu.
     + M: Tổng cần tìm.
     + index: Vị trí hiện tại trong dãy số.
     + current\_sum: Tổng hiện tại của dãy con.
     + current: Dãy con tạm thời.
     + result: Lưu trữ tất cả dãy con thỏa mãn.
   * **Quy trình**:
     + Kiểm tra nếu current\_sum == M, lưu lại kết quả.
     + Nếu current\_sum > M, dừng hàm (quay lui).
     + Duyệt qua các phần tử từ index đến cuối dãy, thêm phần tử vào current và gọi đệ quy.
2. **Hàm findCombinations**:
   * Khởi tạo các biến và gọi hàm backtrack.
3. **Hàm main**:
   * Khởi tạo dãy số và tổng MMM.
   * Gọi hàm tìm tổ hợp và in kết quả.

**Kết quả ví dụ:**

Với A={7,1,4,3,5,6}A = \{7, 1, 4, 3, 5, 6\}A={7,1,4,3,5,6} và M=11M = 11M=11, chương trình in: