**Phân tích và thiết kế thuật toán**

# **Mục lục**

[**Mục lục** 1](#_Toc136802114)

[**Đề 1** 2](#_Toc136802115)

[**Câu 1 (1.5đ) :** 2](#_Toc136802116)

[**Giải Câu 1:** 3](#_Toc136802117)

[**Câu 2 (2 đ)** 5](#_Toc136802118)

[**Giải câu 2:** 5](#_Toc136802119)

[**Câu 3 (3 đ)** 6](#_Toc136802120)

[**Giải câu 3:** 6](#_Toc136802121)

[**Câu 4 (3.5 đ)** 7](#_Toc136802122)

[**a)** **Nêu ý tưởng thuật toán tham lam giải quyết bài toán trên** 7](#_Toc136802123)

[**b)** **Minh họa bài toán với ba lô có kích thước b =130 với các đồ vật đã cho trong bảng** 7](#_Toc136802124)

[**Đề 2** 9](#_Toc136802125)

[**Câu 1 (1.5 đ)** 9](#_Toc136802126)

[**Giải câu 1:** 10](#_Toc136802127)

[**Câu 2 (2 đ)** 11](#_Toc136802128)

[**Giải câu 2:** 11](#_Toc136802129)

[**Câu 3 (3 đ)** 12](#_Toc136802130)

[**Giải câu 3** 12](#_Toc136802131)

[**Câu 4 (3.5 đ)** 13](#_Toc136802132)

[**a)** **Nêu ý tưởng thuật toán tham lam giải quyết bài toán tìm đường đi ngắn nhất** 13](#_Toc136802133)

[**b)** **Minh họa bài toán tìm đường đi ngắn nhất từ đỉnh a đến các đỉnh còn lại** 13](#_Toc136802134)

[**Giải câu 4a:** 13](#_Toc136802135)

[**Giải câu 4b:** 13](#_Toc136802136)

[**Đề 3**: 14](#_Toc136802137)

[**Câu 1 (không có )** 14](#_Toc136802138)

[**Câu 2 (2 đ ):** 14](#_Toc136802139)

[**Giải câu 2:** 14](#_Toc136802140)

[**Câu 3(3 đ)** 15](#_Toc136802141)

[**Giải câu 3:** 15](#_Toc136802142)

[**Câu 4 (3.5 đ)** 16](#_Toc136802143)

[**a)** **Nêu ý tưởng thuật toán quy hoạch đồng thời giải quyết bài toán tìm xâu chung dài nhất** 16](#_Toc136802144)

[**b)** **Minh họa bài toán với dữ liệu x, y đã cho** 16](#_Toc136802145)

[**Giải Câu 4a:** 16](#_Toc136802146)

[**Giải câu 4b:** 16](#_Toc136802147)

[**Câu 5** 18](#_Toc136802148)

[**a)** **Nêu ý tưởng thuật toán tham lam giải quyết bài toán tìm đường đi ngăn nhất** 19](#_Toc136802149)

[**b)** **Minh họa bài toán tìm đường đi ngắn nhất từ đỉnh A đến các đỉnh còn lại** 19](#_Toc136802150)

[**Câu 6:** 19](#_Toc136802151)

[**a)** **Nêu ý tưởng thuật toán tham lam giải quyết bài toán trên** 19](#_Toc136802152)

[**b)** **Minh họa bài toán với balo có kích thước b = 130 với các đồ vật cho trong bảng.** 19](#_Toc136802153)

[**Tổng quan ôn thi tự luận :** 20](#_Toc136802154)

[**Ôn quy hoạch động** 20](#_Toc136802155)

[**Bài 1: Dãy con đơn điệu dài nhất** 20](#_Toc136802156)

[**Bài 2: Bố trí phòng họp** 20](#_Toc136802157)

[**Bài 5: Balo1** 20](#_Toc136802158)

[**Bài 8. Bài toán Đổi tiền** 20](#_Toc136802159)

[**Chiến lược tham lam** 21](#_Toc136802160)

[**Bài 1: Bài toán đường đi của người giao hàng** 21](#_Toc136802161)

[**Bài 3: Bài toán phân công lao động:** 22](#_Toc136802162)

[**Bài 4:** 22](#_Toc136802163)

[**Đề tham khảo như trên** 23](#_Toc136802164)

# **Đề 1**

## **Câu 1 (1.5đ) :**

Viết lại các chương trình sau đây bằng cách sử dụng tối thiểu các cấu trúc lặp while và một sô biến phụ (không dùng if và for)

 int F (int X[], int n, int m, int k)

{ int  s1 =0, s0=0, dem1 =0 , dem0=0;

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

If ( X[i]) {

S1 ++; s0=0;

If(s1 >=k) dem1 ++;

}

Else {

S1 = 0; s0 ++;

If (s0>m) dem0++;

}

}

If (dem1 ==1  && dem0 == 1)

Return(1);

Return (0)

}

## **Giải Câu 1:**

int F(int X[], int n, int m, int k) {

int s1 = 0, s0 = 0, dem1 = 0, dem0 = 0;

int i = 0;

while (i < n) {

while (X[i]) {

s1++;

s0 = 0;

if (s1 >= k) {

dem1++;

}

i++;

if (i >= n) {

break;

}

}

while (!X[i]) {

s1 = 0;

s0++;

if (s0 > m) {

dem0++;

}

i++;

if (i >= n) {

break;

}

}

}

if (dem1 == 1 && dem0 == 1) {

return 1;

}

return 0;

}

Ý nghĩa của các phần trong đoạn code:

1. Hàm **F** nhận vào một mảng **X** gồm các phần tử kiểu **int**, kích thước mảng **n**, và hai giá trị **m** và **k** cũng kiểu **int**.
2. Biến **s1** và **s0** được sử dụng để đếm số lượng liên tiếp các giá trị **1** và **0** trong mảng **X**.
3. Biến **dem1** và **dem0** được sử dụng để đếm số lượng lần xuất hiện của một chuỗi liên tiếp **k** giá trị **1** và **m** giá trị **0** trong mảng **X**.
4. Biến **i** được sử dụng làm biến đếm để duyệt qua các phần tử trong mảng **X**.
5. Sử dụng cấu trúc lặp **while**, đoạn code duyệt qua từng phần tử trong mảng **X** và thực hiện các công việc sau:
   * Nếu phần tử là **1**, tăng giá trị **s1** lên 1 và đặt **s0** về 0. Nếu **s1** đạt giá trị **k** hoặc lớn hơn, tăng giá trị **dem1** lên 1.
   * Nếu phần tử là **0**, đặt **s1** về 0 và tăng giá trị **s0** lên 1. Nếu **s0** vượt quá giá trị **m**, tăng giá trị **dem0** lên 1.
   * Duyệt qua từng phần tử trong mảng **X** cho đến khi hết phần tử hoặc gặp điều kiện dừng.
6. Kiểm tra điều kiện: Nếu **dem1** bằng 1 và **dem0** bằng 1, trả về giá trị 1. Ngược lại, trả về giá trị 0.

Tổng quan, chương trình này đếm số lượng chuỗi liên tiếp các giá trị **1** và **0** trong mảng **X** và kiểm tra xem có duy nhất một chuỗi liên tiếp **k** giá trị **1** và một chuỗi liên tiếp **m** giá trị **0** hay không. Nếu có, trả về 1; ngược lại, trả về 0.

## **Câu 2 (2 đ)**

Trình bày ý tưởng của thuật toán sắp xếp chọn, hiện thực đoạn mã xử lý chính và đánh giá độ phức tạp thuật toán

## **Giải câu 2:**

Ý tưởng của thuật toán sắp xếp chọn (Selection Sort) là tìm kiếm phần tử nhỏ nhất trong mảng và đưa nó về đúng vị trí của nó. Thuật toán này hoạt động bằng cách chia mảng thành hai phần: phần đã sắp xếp và phần chưa sắp xếp. Ban đầu, phần đã sắp xếp không có phần tử nào và toàn bộ mảng được coi là phần chưa sắp xếp. Trong mỗi bước lặp, thuật toán tìm phần tử nhỏ nhất trong phần chưa sắp xếp và hoán đổi nó với phần tử đầu tiên của phần chưa sắp xếp. Kết quả là mỗi lần lặp, phần tử nhỏ nhất sẽ được đưa vào phần đã sắp xếp và thuật toán tiếp tục tìm phần tử nhỏ nhất trong phần còn lại của mảng chưa sắp xếp.

Đoạn mã xử lý chính cho thuật toán sắp xếp chọn trong C++ như sau:

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

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

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

// Hoán đổi phần tử nhỏ nhất với phần tử đầu tiên của phần chưa sắp xếp

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

}

Trong đoạn mã trên, hàm **selectionSort** nhận vào một mảng **arr** và số lượng phần tử **n**. Thuật toán sắp xếp chọn được thực hiện bằng cách sử dụng hai vòng lặp **for**. Vòng lặp bên ngoài duyệt qua các phần tử từ **0** đến **n-1**, đại diện cho phần đã sắp xếp. Vòng lặp bên trong tìm kiếm phần tử nhỏ nhất trong phần chưa sắp xếp, bắt đầu từ **i+1** đến **n-1**. Nếu tìm thấy phần tử nhỏ hơn phần tử tại **minIndex**, **minIndex** sẽ được cập nhật. Sau khi hoàn thành vòng lặp bên trong, phần tử nhỏ nhất được đưa vào đúng vị trí của nó bằng cách hoán đổi với phần tử đầu tiên của phần chưa sắp xếp.

Độ phức tạp của thuật toán sắp xếp chọn là O(n^2), trong đó n là số lượng phần tử trong mảng. Điều này có nghĩa là thời gian thực hiện thuật toán tăng theo bình phương số lượng phần tử. Dù rất dễ hiểu và cài đặt, nhưng thuật toán sắp xếp chọn không phải là thuật toán hiệu quả trong các trường hợp có số lượng phần tử lớn.

## **Câu 3 (3 đ)**

Phân tích các yếu tố đệ quy và hiện thực đoan mã đệ quy giải bài toán cho trước số thực x và số nguyên dương n sau đây.

 S = -1-x-(x+2)1x-cosx+x-(x+2)2x-cosx-x-(x+2)3x-cosx+…+(-1)nx-(x+2)nx-cosx

## **Giải câu 3:**

Phân tích các yếu tố đệ quy trong công thức:

* Có thể nhận thấy rằng các số hạng của công thức đều có dạng **(-1)^n \* x - (x + 2) \* n \* x - cos(x)**. Điểm chung của các số hạng này là **(x + 2) \* n \* x - cos(x)**.
* Để tính toán số hạng tiếp theo, ta cần biết số hạng trước đó và giá trị **n**.
* Các yếu tố đệ quy trong bài toán này là **(x + 2) \* n \* x - cos(x)** và **(-1)^n \* x**.

Đoạn mã đệ quy giải bài toán trên bằng ngôn ngữ C++:

#include <cmath>

double recursiveFunction(double x, int n) {

if (n == 0) {

return -1 - x - cos(x);

}

double term = recursiveFunction(x, n - 1);

double recursiveTerm = pow(-1, n) \* x - (x + 2) \* n \* x - cos(x);

return term + recursiveTerm;

}

Trong đoạn mã trên:

* Hàm **recursiveFunction** nhận vào số thực **x** và số nguyên dương **n**.
* Nếu **n** bằng 0, chúng ta đã đạt đến điều kiện dừng và trả về giá trị của số hạng đầu tiên trong công thức.
* Nếu **n** khác 0, chúng ta tính giá trị của số hạng hiện tại bằng cách gọi đệ quy và tính toán số hạng đệ quy. Sau đó, cộng giá trị số hạng hiện tại và số hạng đệ quy và trả về kết quả.

Độ phức tạp của đoạn mã đệ quy này phụ thuộc vào số lần đệ quy và thời gian tính toán của mỗi lần đệ quy. Trong trường hợp tổng quát, độ phức tạp sẽ là O(n), với n là giá trị của tham số **n**.

## **Câu 4 (3.5 đ)**

Cho n đồ vật, đồ vật thứ I có trọng lượng **wi** và giá trị **c0** I = 1,2,3,..n hãy tìm cách sắp xếp các đồ vật này vào cái túi có dung lượng b, sao cho tổng trọng lượng của các đồ vật được chất vào túi không quá b, đồng thời tổng giá trị của chúng là lớn nhất

Cho dữ liệu

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Đồ vật (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Kích Thước (**wi**) | 30 | 40 | 30 | 25 | 40 | 50 | 20 | 20 | 30 | 10 |
| Giá trị(c**i)** | 50 | 80 | 70 | 50 | 90 | 90 | 30 | 50 | 50 | 10 |

1. **Nêu ý tưởng thuật toán tham lam giải quyết bài toán trên**

**Giải câu 4a:**

Ý tưởng của thuật toán tham lam (greedy algorithm) để giải quyết bài toán này như sau:

1. Tính giá trị trung bình của mỗi đồ vật, bằng cách chia giá trị của đồ vật cho trọng lượng của đồ vật đó.
2. Sắp xếp các đồ vật theo giá trị trung bình giảm dần.
3. Khởi tạo tổng trọng lượng hiện tại và tổng giá trị hiện tại ban đầu là 0.
4. Bắt đầu từ đồ vật có giá trị trung bình cao nhất, thêm đồ vật này vào túi nếu trọng lượng hiện tại cộng với trọng lượng của đồ vật đó không vượt quá dung lượng của túi. Cập nhật tổng trọng lượng hiện tại và tổng giá trị hiện tại.
5. Lặp lại bước 4 cho đến khi không còn đồ vật nào hoặc trọng lượng hiện tại vượt quá dung lượng của túi.
6. Trả về tổng giá trị hiện tại là giá trị tối ưu của túi.

Thuật toán tham lam giải quyết bài toán này dựa trên việc chọn đồ vật có giá trị trung bình cao nhất để thêm vào túi. Điều này đảm bảo rằng giá trị của túi sẽ càng lớn càng tốt. Tuy nhiên, thuật toán tham lam không đảm bảo tìm được lời giải tối ưu toàn cục trong mọi trường hợp, mà chỉ đưa ra một lời giải gần đúng.

1. **Minh họa bài toán với ba lô có kích thước b =130 với các đồ vật đã cho trong bảng**

**Giải câu 4b:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Item

{

int value;

int weight;

double rate;

int selected;

};

int Knapsack\_greedy(std::vector<int> &w, std::vector<int> &v, int s)

{

int remainingSize = s;

vector<Item> items(w.size());

for (size\_t i = 0; i < items.size(); i++)

{

items[i] = {v[i], w[i], static\_cast<double>(v[i]) / static\_cast<double>(w[i]), 0};

}

for (size\_t i = 0; i < items.size() - 1; i++)

{

for (size\_t j = 0; j < items.size() - i - 1; j++)

{

if (items[j].rate < items[j + 1].rate)

{

swap(items[j], items[j + 1]);

}

}

}

for (size\_t i = 0; i < items.size(); i++)

{

items[i].selected = remainingSize / items[i].weight;

remainingSize = remainingSize - items[i].selected \* items[i].weight;

}

int ans = 0;

for (const auto &item : items)

{

if (item.selected > 0)

{

cout << item.selected << " (value: " << item.value << ", weight: " << item.weight << ")" << endl;

ans += item.value \* item.selected;

}

}

return ans;

}

int main()

{

vector<int> w = {30, 40, 30, 25, 40, 50, 20, 20, 30, 10};

vector<int> v = {50, 80, 70, 50, 90, 90, 30, 50, 50, 10};

int s = 130;

int res = Knapsack\_greedy(w, v, s);

cout << res << endl;

return 0;

}

# **Đề 2**

## **Câu 1 (1.5 đ)**

Viết lại chương trình sau đây bằng cách sử dụng tối thiểu cấu trúc lặp while và một số biến phụ (không dùng lệnh if và for):

#include <stdio.h>

#include<conio.h>

Int dem = 0, A[] = {4,3,7,8,6,2,1} ,n =7, OK = 1;

Void Result (void) {

Printf (“\n Ket qua buoc %d: “, ++dem);

For (int I = 0; i<n ; i++) Printf (“%d”, A[i]);

}

Int main (void){

For (int I =0; i<=n-2; i++){

If (OK){

If (A[i] > A[i+1])

Int t = A[i]; A[i] = A[i+1]; A[i+1] =t;

}

}

Else{

If(  A[i] <   A[i+1] ) {

Int t = A[i] ; A[i] = A[i+1];  A[i+1] = t;

}

}

Result ();

OK = !OK;

}

Getch(); return(0);

}

## **Giải câu 1:**

Đoạn code sử dụng cấu trúc lặp while và một số biến phụ (không dùng lệnh if và for):

#include <stdio.h>

int dem = 0, A[] = {4,3,7,8,6,2,1}, n = 7, OK = 1;

void Result(void) {

printf("\nKet qua buoc %d: ", ++dem);

int i = 0;

while (i < n) {

printf("%d ", A[i]);

i++;

}

}

int main(void) {

int i = 0;

while (i <= n - 2) {

if (OK) {

int t = A[i];

A[i] = A[i + 1];

A[i + 1] = t;

} else {

if (A[i] < A[i + 1]) {

int t = A[i];

A[i] = A[i + 1];

A[i + 1] = t;

}

}

i++;

}

Result();

OK = !OK;

getch();

return 0;

}

* Đoạn code bắt đầu bằng các khai báo và khởi tạo biến. **dem** đếm số bước sắp xếp, **A** là mảng chứa các phần tử cần sắp xếp, **n** là kích thước của mảng, **OK** là biến điều kiện để xác định thứ tự sắp xếp (1: tăng dần, 0: giảm dần).
* Hàm **Result** được sử dụng để in ra kết quả sau mỗi bước sắp xếp. Trong hàm này, biến **dem** được tăng lên và một vòng lặp while được sử dụng để duyệt qua mảng A và in ra từng phần tử.
* Hàm **main** chứa phần chính của chương trình. Một vòng lặp while được sử dụng để thực hiện sắp xếp mảng A. Vòng lặp này sẽ chạy cho đến khi **i** đạt đến **n - 2** (điều kiện để kết thúc vòng lặp vì chỉ cần thực hiện **n - 1** lần sắp xếp để mảng được sắp xếp).
* Trong vòng lặp while, kiểm tra giá trị của biến **OK**. Nếu **OK** là 1, thực hiện hoán đổi giá trị của hai phần tử liền kề (A[i] và A[i+1]) nếu A[i] > A[i+1]. Ngược lại, nếu **OK** là 0, thực hiện hoán đổi nếu A[i] < A[i+1].
* Sau khi hoàn thành vòng lặp while, gọi hàm **Result** để in ra kết quả sau khi sắp xếp.
* Tiếp theo, biến **OK** được đảo ngược bằng cách gán **!OK**, để chuyển đổi thứ tự sắp xếp (tăng dần hoặc giảm dần).
* Cuối cùng, sử dụng **getch()** để đợi người dùng nhấn một phím trước khi chương trình kết thúc.

Chương trình trên sử dụng thuật toán sắp xếp nổi bọt để sắp xếp mảng A. Trong mỗi lần lặp, nếu hai phần tử liền kề không đúng thứ tự, chúng được hoán đổi vị trí. Sau mỗi lần lặp, phần tử lớn nhất (hoặc nhỏ nhất) sẽ được đưa về cuối mảng. Quá trình này lặp lại cho đến khi mảng được sắp xếp.

## **Câu 2 (2 đ)**

Trình bày ý tưởng của thuật toán sắp xếp chọn, hiện thực đoạn mã xử lý chính và đanh giá độ phức tạp thuật toán.

## **Giải câu 2:**

Giống câu 2 đề 1

## **Câu 3 (3 đ)**

Phân tích các yếu tố đệ quy và hiện thực đoạn mã đệ quy giải bài toán cho trước số thực x và số nguyên dương n sau đây:

S=-1+sinx-22x3-sinx-23x3-1+sinx-24x3-2-sinx-25x3-3+…+-1n+1sinx-2n+1x3-(n-1)

## **Giải câu 3**

Phân tích các yếu tố đệ quy trong công thức:

* Có thể nhận thấy rằng các số hạng của công thức có dạng **(-1)^(n+1) \* sin(x) - 2^(n+1) \* x^3 - (n-1)**. Điểm chung của các số hạng này là **sin(x) - 2^(n+1) \* x^3 - (n-1)**.
* Để tính toán số hạng tiếp theo, ta cần biết số hạng trước đó và giá trị **n**.
* Các yếu tố đệ quy trong bài toán này là **sin(x) - 2^(n+1) \* x^3 - (n-1)** và **(-1)^(n+1) \* sin(x)**.

Đoạn mã đệ quy giải bài toán trên bằng ngôn ngữ C++:

#include <cmath>

double recursiveFunction(double x, int n) {

if (n == 0) {

return -1 + sin(x) - pow(2, 2) \* pow(x, 3) - (n - 1);

}

double term = recursiveFunction(x, n - 1);

double recursiveTerm = sin(x) - pow(2, n + 1) \* pow(x, 3) - (n - 1);

return term + pow(-1, n + 1) \* recursiveTerm;

}

Trong đoạn mã trên:

* Hàm **recursiveFunction** nhận vào số thực **x** và số nguyên dương **n**.
* Nếu **n** bằng 0, chúng ta đã đạt đến điều kiện dừng và trả về giá trị của số hạng đầu tiên trong công thức.
* Nếu **n** khác 0, chúng ta tính giá trị của số hạng hiện tại bằng cách gọi đệ quy và tính toán số hạng đệ quy. Sau đó, cộng giá trị số hạng hiện tại và số hạng đệ quy, nhân với **(-1)^(n+1)** và trả về kết quả.

Độ phức tạp của đoạn mã đệ quy này phụ thuộc vào số lần đệ quy và thời gian tính toán của mỗi lần đệ quy. Trong trường hợp tổng quát, độ phức tạp sẽ là O(n), với n là giá trị đầu vào.

## **Câu 4 (3.5 đ)**

Cho đồ thị vô hướng sau:

1. **Nêu ý tưởng thuật toán tham lam giải quyết bài toán tìm đường đi ngắn nhất**

### **Giải câu 4a:**

Ý tưởng của thuật toán tham lam (greedy algorithm) để giải quyết bài toán tìm đường đi ngắn nhất (shortest path) giữa hai đỉnh trong đồ thị như sau:

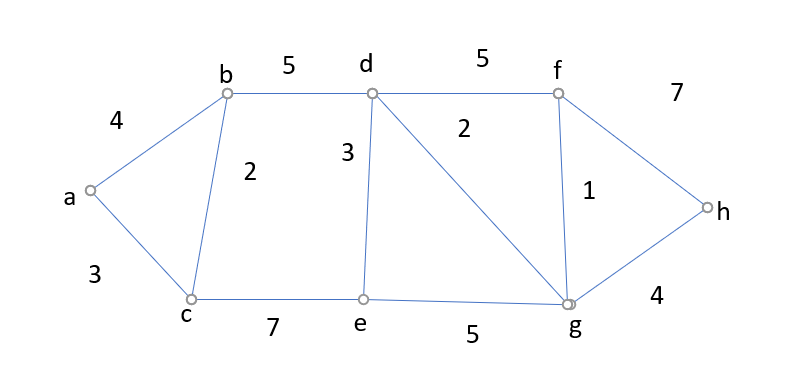
1. Khởi tạo một đỉnh xuất phát và một đỉnh đích.
2. Khởi tạo một tập hợp chứa các đỉnh đã được xét.
3. Khởi tạo một hàng đợi ưu tiên (priority queue) để lưu trữ các đỉnh đang xét, sắp xếp theo độ dài đường đi tăng dần.
4. Gán độ dài đường đi từ đỉnh xuất phát đến chính nó bằng 0 và gán độ dài đường đi từ đỉnh xuất phát đến các đỉnh còn lại là vô cùng.
5. Thêm đỉnh xuất phát vào hàng đợi ưu tiên.
6. Lặp lại cho đến khi hàng đợi ưu tiên trống:

* Lấy đỉnh có độ dài đường đi nhỏ nhất từ hàng đợi ưu tiên.
* Nếu đỉnh đó chưa được xét, thêm vào tập hợp các đỉnh đã xét.
* Duyệt qua các đỉnh kề với đỉnh đó:
* Nếu độ dài đường đi từ đỉnh xuất phát tới đỉnh kề hiện tại lớn hơn độ dài đường đi từ đỉnh xuất phát tới đỉnh hiện tại cộng với trọng số của cạnh nối giữa hai đỉnh, cập nhật độ dài đường đi và đỉnh trước đó của đỉnh kề.
* Thêm đỉnh kề vào hàng đợi ưu tiên với độ dài đường đi đã được cập nhật.

1. Trả về đường đi ngắn nhất từ đỉnh xuất phát đến đỉnh đích

Thuật toán tham lam giải quyết bài toán tìm đường đi ngắn nhất dựa trên việc lựa chọn đỉnh có độ dài đường đi nhỏ nhất trong hàng đợi ưu tiên để mở rộng đường đi từ đỉnh xuất phát. Bằng cách tiếp tục mở rộng các đường đi từ đỉnh đã được xét, thuật toán sẽ tìm ra đường đi ngắn nhất từ đỉnh xuất phát đến đỉnh đích. Tuy nhiên, thuật toán tham lam không đảm bảo tìm được lời giải tối ưu toàn cục trong mọi trường hợp, mà chỉ đưa ra một lời giải gần đúng.

1. **Minh họa bài toán tìm đường đi ngắn nhất từ đỉnh a đến các đỉnh còn lại**



### **Giải câu 4b:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B.lặp | a | b | c | d | e | f | g | h |
| k.tạo | 0;a\* |  |  |  |  |  |  |  |
| 1 | - | 4;a | 3;a\* |  |  |  |  |  |
| 2 | - | 4;a | - |  | 10;c |  |  |  |

# **Đề 3**:

## **Câu 1 (không có )**

## **Câu 2 (2 đ ):**

Trình bày ý tưởng của thuật toán sắp xếp nổi bọt, hiện thực đoạn mã xử lý chính và đánh giá độ phức tạp thuật toán

## **Giải câu 2:**

Ý tưởng của thuật toán sắp xếp nổi bọt (Bubble Sort) là duyệt qua mảng và so sánh từng cặp phần tử liền kề. Nếu phần tử đứng trước lớn hơn phần tử đứng sau, ta hoán đổi vị trí của hai phần tử đó. Quá trình này lặp lại cho đến khi không có sự hoán đổi nào xảy ra trong một vòng lặp, tức là mảng đã được sắp xếp.

Đoạn mã sau đây minh hoạ cách hiện thực thuật toán sắp xếp nổi bọt:

#include <iostream>

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

bool swapped;

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

swapped = false;

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

if (arr[j] > arr[j + 1]) {

std::swap(arr[j], arr[j + 1]);

swapped = true;

}

}

// Nếu không có sự hoán đổi trong một vòng lặp, dừng thuật toán

if (!swapped) {

break;

}

}

}

int main() {

int arr[] = {4, 3, 7, 8, 6, 2, 1};

int n = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, n);

std::cout << "Mang da sap xep: ";

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

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

}

std::cout << std::endl;

return 0;

}

Trong đoạn mã trên:

* Hàm **bubbleSort** nhận vào mảng **arr** và kích thước **n** của mảng.
* Sử dụng hai vòng lặp for lồng nhau. Vòng lặp bên ngoài (**i**) lặp qua từng phần tử của mảng, và vòng lặp bên trong (**j**) lặp qua các cặp phần tử liền kề để so sánh và hoán đổi.
* Nếu phần tử đứng trước (**arr[j]**) lớn hơn phần tử đứng sau (**arr[j + 1]**), ta hoán đổi vị trí của hai phần tử đó bằng hàm **std::swap**.
* Biến **swapped** được sử dụng để kiểm tra xem có sự hoán đổi nào xảy ra trong vòng lặp không. Nếu không có hoán đổi nào, ta biết rằng mảng đã được sắp xếp và có thể dừng thuật toán.
* Trong hàm **main**, một mảng được khởi tạo và sắp xếp bằng cách gọi hàm **bubbleSort**.
* Cuối cùng, mảng sau khi sắp xếp được in ra màn hình.

Độ phức tạp của thuật toán sắp xếp nổi bọt là O(n^2), với n là kích thước của mảng.

## **Câu 3(3 đ)**

Phân tích các yếu tố đệ quy và hiện thực đoạn mã đệ quy giải bài toán cho trước số thực x và sô nguyên dương n sau đây:

S = -x+1+(2x-1)1tanx-2+(2x-1)1tanx+3-(2x-1)1tanx+…+ (-1)n+1n+(2x-1)ntanx

## **Giải câu 3:**

Phân tích các yếu tố đệ quy trong công thức:

* Có thể nhận thấy rằng các số hạng của công thức có dạng **(2x-1)k \* tan(x+2k-1)**. Điểm chung của các số hạng này là **(2x-1)k \* tan(x+2k-1)**.
* Để tính toán số hạng tiếp theo, ta cần biết số hạng trước đó và giá trị **n**.
* Các yếu tố đệ quy trong bài toán này là **(2x-1)k \* tan(x+2k-1)** và **(-1)^(n+1) \* n**.

Đoạn mã đệ quy giải bài toán trên bằng ngôn ngữ C++:

#include <cmath>

double recursiveFunction(double x, int n, int k) {

if (k == n + 1) {

return pow(-1, n + 1) \* n + pow(2 \* x - 1, n) \* tan(x);

}

double term = recursiveFunction(x, n, k + 1);

double recursiveTerm = pow(2 \* x - 1, k) \* tan(x + 2 \* k - 1);

return term + recursiveTerm;

}

## **Câu 4 (3.5 đ)**

Xâu con chung của 2 xâu ký tự x và y là xâu ký tự mà các ký tự của nó đều xuất hiện trên cả x và y, đảm bảo giữ nguyên thứ tự trong x và y và không cần liền mạch kề nhau.

Cho x = “KHOAHOC”, y =”HOAHONG”

1. **Nêu ý tưởng thuật toán quy hoạch đồng thời giải quyết bài toán tìm xâu chung dài nhất**
2. **Minh họa bài toán với dữ liệu x, y đã cho**

**Giải Câu 4a:**

Ý tưởng thuật toán quy hoạch đồng thời (Dynamic Programming) để giải quyết bài toán tìm xâu con chung dài nhất của hai xâu ký tự x và y như sau:

1. Tạo ma trận đường chéo kích thước (m+1) x (n+1), với m là độ dài của x và n là độ dài của y.
2. Khởi tạo giá trị ban đầu cho ma trận đường chéo: các phần tử trên hàng đầu tiên và cột đầu tiên được gán giá trị 0.
3. Duyệt qua từng phần tử của ma trận từ trái qua phải và từ trên xuống dưới.
   * Nếu x[i-1] (ký tự tại vị trí i-1 trong x) bằng y[j-1] (ký tự tại vị trí j-1 trong y), ta gán giá trị của phần tử đó bằng giá trị phần tử ở đường chéo phía trên góc trái cộng thêm 1.
   * Nếu không bằng nhau, ta gán giá trị của phần tử đó bằng giá trị lớn nhất của hai phần tử ở hàng phía trên và cột bên trái.
4. Kết quả cuối cùng sẽ là giá trị của phần tử ở góc phải dưới của ma trận, tức là dp[m][n].

### **Giải câu 4b:**

Minh họa bài toán với dữ liệu x = "KHOAHOC", y = "HOAHONG":

H O A H O N G

0 0 0 0 0 0 0 0

K 0 0 0 0 0 0 0 0

H 0 1 1 1 1 1 1 1

O 0 1 1 1 2 2 2 2

A 0 1 2 2 2 2 2 2

H 0 1 2 2 2 3 3 3

O 0 1 2 2 2 3 3 3

C 0 1 2 2 2 3 3 3

Giá trị cuối cùng là 3, cho thấy xâu con chung dài nhất của "KHOAHOC" và "HOAHONG" là "HOA".

Đoạn mã C++ để tính xâu con chung dài nhất của hai xâu ký tự x và y:

#include <iostream>

#include <string>

#include <vector>

std::string longestCommonSubstring(const std::string& x, const std::string& y) {

int m = x.length();

int n = y.length();

std::vector<std::vector<int>> dp(m + 1, std::vector<int>(n + 1, 0));

int maxLength = 0;

int endIndex = 0;

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

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

if (x[i - 1] == y[j - 1]) {

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

if (dp[i][j] > maxLength) {

maxLength = dp[i][j];

endIndex = i - 1;

}

}

}

}

return x.substr(endIndex - maxLength + 1, maxLength);

}

int main() {

std::string x = "KHOAHOC";

std::string y = "HOAHONG";

std::string commonSubstring = longestCommonSubstring(x, y);

std::cout << "Longest Common Substring: " << commonSubstring << std::endl;

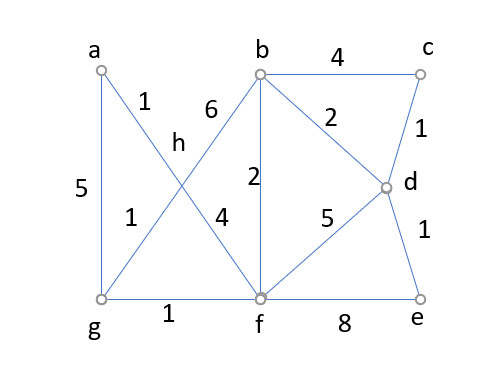
return 0;

}

Độ phức tạp của thuật toán này là O(m \* n), với m và n lần lượt là độ dài của x và y.

## **Câu 5**

Cho đồ thị sau



1. **Nêu ý tưởng thuật toán tham lam giải quyết bài toán tìm đường đi ngăn nhất**
2. **Minh họa bài toán tìm đường đi ngắn nhất từ đỉnh A đến các đỉnh còn lại**

## **Câu 6:**

Cho n đồ vật, đồ vật thứ i có trọng lượng wi và giá trị ci i = 1,2,3, …… n. Hãy tìm cách xếp các đồ vật này vào cái túi có dung lượng b, sao cho tổng trọng lượng của các đồ vật được chất vào túi không quá b, đồng thời tổng giá trị của chúng là lớn nhất.

Cho dữ liệu:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Đồ vật (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Trọng lượng (Wi) | 30 | 40 | 30 | 25 | 40 | 50 | 20 | 20 | 30 | 10 |
| Giá trị (ci) | 50 | 80 | 70 | 50 | 90 | 90 | 30 | 50 | 50 | 10 |

1. **Nêu ý tưởng thuật toán tham lam giải quyết bài toán trên**
2. **Minh họa bài toán với balo có kích thước b = 130 với các đồ vật cho trong bảng.**

Giải: Ta có kỹ thuật Tham lam áp dụng cho bài toán này là:

1. Tính giá trị cho các loại đồ vật.
2. Xét các loại đồ vật theo thứ tự đơn giá từ lớn đến nhỏ.
3. Với mỗi đồ vật được xét sẽ lấy một số lượng tối đa mà trọng lượng còn lại của ba lô cho phép.
4. Xác định trọng lượng còn lại của ba lô và quay lại bước 3 cho đến khi không còn có thể chọn được đồ vật nào nữa.

# **Tổng quan ôn thi tự luận :**

## **Ôn quy hoạch động**

### **Bài 1: Dãy con đơn điệu dài nhất**

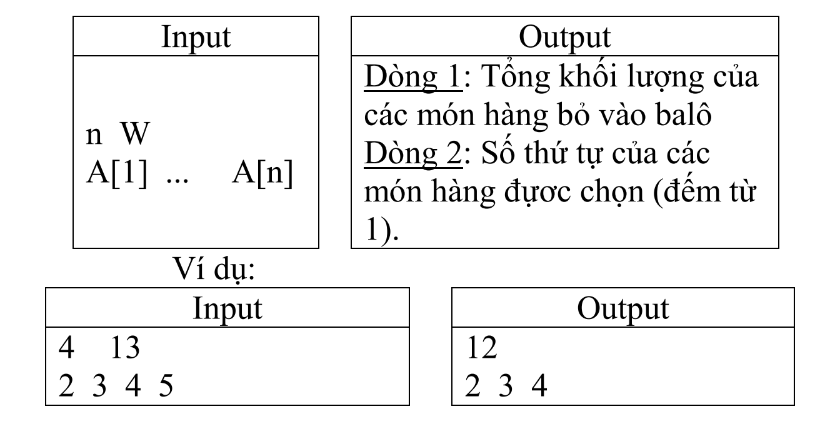
Cho dãy a1,a2,..an. Hãy tìm một dãy con tăng có nhiều phần tử nhất của dãy

### **Bài 2: Bố trí phòng họp**

Có n cuộc họp, cuộc họp thứ i bắt đầu vào thời điểm ai và kết thúc ở thời điểm bi. Do chỉ có một phòng hội thảo nên 2 cuộc họp bất kỳ sẽ được cùng bố trí phục vụ nếu khoảng thời gian làm việc của chúng chỉ giao nhau tại đầu mút. Hãy bố trí phòng họp để phục vụ được nhiều cuộc họp nhất

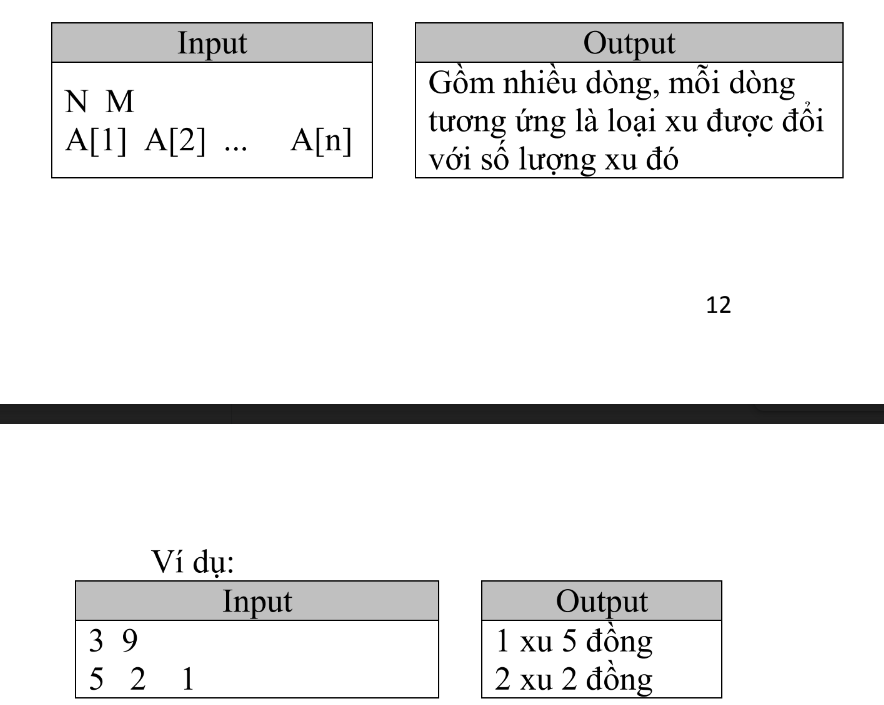
### **Bài 5: Balo1**

Cho n món hàng (n ≤ 50). Món hàng thứ i có khối lượng A[i] (số nguyên). Cần chọn những món hàng nào để bỏ vào một balô sao tổng khối lượng của các món hàng đã chọn là lớn nhất nhưng không vượt quá khối lượng W cho trước (W ≤ 100) Mỗi món chỉ chọn 1 hoặc không chọn



### **Bài 8. Bài toán Đổi tiền**

Ở đất nước nọ người ta chỉ tiêu tiền xu. Có N loại tiền xu, loại thứ i có mệnh giá là Ai đồng.Một hôm có một vị khách du lịch đến thăm, ông ta muốn đổi số tiền mình đang có mà M đồng sang loại xu của đất nước này để tiện tiêu dùng. Bên cạnh đó ông ta cũng muốn số đồng tiền đổi được là ít nhất . Bạn hãy giúp ông ta tìm cách đổi tiền yêu cầu trên.



## **Chiến lược tham lam**

### **Bài 1: Bài toán đường đi của người giao hàng**

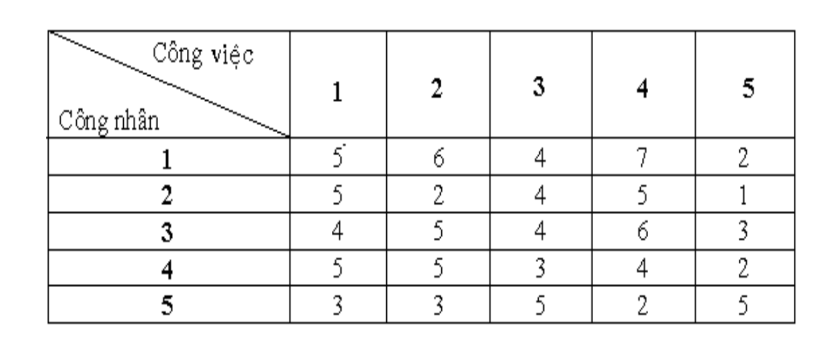
Có một người giao hàng cần đi giao hàng tại n thành phố. Xuất phát từ một thành phố nào đó, đi qua các thành phố khác để giao hàng và trở về thành phố ban đầu. Mỗi thành phố chỉ đến một lần, khoảng cách từ một thành phố đến các thành phố khác là xác định được. Khoảng cách giữa hai thành phố có thể là khoảng cách địa lý, có thể là cước phí di chuyển hoặc thời gian di chuyển. Ta gọi chung là độ dài. Hãy tìm một chu trình (một đường đi khép kín thỏa mãn điều kiện trên) sao cho tổng độ dài các cạnh là nhỏ nhất. Hay còn nói là tìm một phương án có giá nhỏ nhất. Bài toán này cũng được gọi là bài toán người du lịch.

### **Bài 3: Bài toán phân công lao động:**

Có n công nhân có thể làm n công việc.Công nhân i làm công việc j trong một khoảng thời gian tij. Phải tìm một phương án phân công như thế nào để các công việc đều được hoàn thành, các công nhân đều có việc làm, mỗi công nhân chỉ làm một công việc và mỗi công việc chỉ do một công nhân thực hiện đồng thời tổng thời gian là nhỏ nhất.

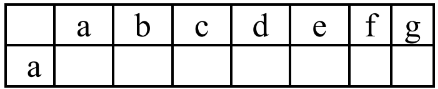
1. Mô tả kỹ thuật “tham ăn” (greedy) cho bài toán phân công lao động.

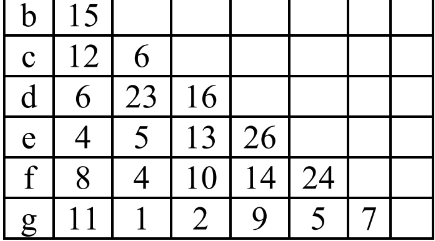
2. Tìm phương án theo giải thuật “tham ăn” cho bài toán phân công lao động được cho trong bảng sau. Trong đó mỗi dòng là một công nhân, mỗi cột là một công việc, ô (i,j) ghi thời gian tij mà công nhân i cần để hoàn thành công việc j. (Cần chỉ rõ công nhân nào làm công việc gì và tổng thời gian là bao nhiêu ).



### **Bài 4:**

Cho bài toán đường đi của người bán hàng(TSP) với 7 thành phố a, b, c, d, e, f, g. Ma trận khoảng cách giữa các thành phố được cho trong bảng sau:





* dijkstra.
* trong quy hoạch động : ôn bài túi, balo, so sánh chuỗi
* dijkstra: tìm đường đi ngắn nhất ,
* sắp xếp chèn, chọn,nổi bọt , truy hồi
* giải chuỗi : vd: sinx….., nêu thuật toán .

# **Đề tham khảo như trên**