The Greedy algorithms

Câu 1: Đề xuất giải thuật tham lam giải bài toán trả  
tiền máy ATM?

#include <iostream>

#include <stdlib.h>

using namespace std;

int main() {

int n;

do

{

cout << "moi ban nhap so tien: ";

cin >> n;

} while (n < 10000);

if (n % 10000 != 0)

{

cout << "so tien phai chia het cho 10.000\n";

return 1;

}

int motTram = n / 100000;

n = n % 100000;

int namMuoi = n / 50000;

n = n % 50000;

int haiMuoi = n / 20000;

n = n % 20000;

int muoi = n / 10000;

cout << "Ket qua: \n";

cout << motTram << " to 1 tram + " << namMuoi << " to 50.000 + "

<< haiMuoi << " to 20.000 + " << muoi << " to 10.000" << endl;

return 0;

}

Câu 2: Cài đặt thuật toán Dijkstra. Đánh giá độ phức  
tạp bằng thực nghiệm và so sánh với lý thuyết

#include <iostream>

using namespace std;

#include <limits.h>

// số đỉnh

#define V 9

// tìm đỉnh có khoảng cách min, từ tập các đỉnh chưa có trong cây đường đi ngắn nhất

int minDistance(int dist[], bool sptSet[])

{

// khởi tạo giá trị min

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

// in ra mảng khoảng cách đã xây dựng

void printSolution(int dist[])

{

cout << "Vertex \t Distance from Source" << endl;

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

cout << i << " \t\t" << dist[i] << endl;

}

void dijkstra(int graph[V][V], int src)

{

int dist[V]; // Mảng output. dist[i] : khoảng cách ngắn nhất từ src đến i

bool sptSet[V]; //true nếu đỉnh i được bao gồm trong cây đường đi ngắn nhất hoặc khoảng cách ngắn nhất từ src đến i

// được hoàn thiện

// Khởi tạo tất cả các khoảng cách là INFINITE và stpSet[] là false

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

dist[i] = INT\_MAX, sptSet[i] = false;

// Khoảng cách của đỉnh nguồn so với chính nó luôn bằng 0

dist[src] = 0;

// Tìm đường đi ngắn nhất cho tất cả các đỉnh

for (int count = 0; count < V - 1; count++) {

//Chọn đỉnh có khoảng cách tối thiểu từ tập hợp các đỉnh chưa được xử lý. u luôn bằng src trong lần lặp đầu tiên.

int u = minDistance(dist, sptSet);

// Đánh dấu đỉnh đã chọn là đã duyệt

sptSet[u] = true;

//Cập nhật giá trị dist của các đỉnh liền kề của đỉnh đã chọn.

for (int v = 0; v < V; v++)

//update dist[v] chỉ khi không có trong sptSet, có một cạnh từ u đến v

//và tổng trọng số của đường dẫn từ src đến v đến u nhỏ hơn giá trị hiện tại của dist[v]

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

// in mảng khoảng cách đã xây dựng

printSolution(dist);

}

int main()

{

int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

dijkstra(graph, 0);

return 0;

}

Câu 3: bài toán cái túi

#include <iostream>

using namespace std;

struct dovat

{

int w;

int c;

};

void replace(dovat& a, dovat& b)

{

dovat tmp = a;

a = b;

b = tmp;

}

void greedy1(dovat a[], int n)

{

int i, j;

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

{

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

{

if (a[i].c < a[j].c)

{

replace(a[i], a[j]);

}

}

}

}

void greedy2(dovat a[], int n)

{

int i, j;

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

{

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

{

if (a[i].w > a[j].w)

{

replace(a[i], a[j]);

}

}

}

}

void greedy3(dovat a[], int n)

{

int i, j;

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

{

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

{

if (a[i].c / a[i].w <a[j].c / a[j].w)

{

replace(a[i], a[j]);

}

}

}

}

void nhapdovat(dovat a[], int n, int &b)

{

cout << "trong luong tui: ";

cin >> b;

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

{

cout<<"khoi luong va gia tri cua vat thu" <<i+1<<": ";

cin >> a[i].w >> a[i].c;

}

}

void chondovat(dovat a[], int n, int b)

{

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

{

if (b >= a[i].w)

{

cout << "chon do vat co khoi luong =" << a[i].w <<"gia tri bang " << a[i].c <<endl;

b = b - a[i].w;

}

}

}

int main()

{

dovat a[3];

int b;

nhapdovat(a, 3, b);

greedy1(a, 3);

chondovat(a, 3, b);

return 0;

}