Bài toán người đi du lịch

1. Nêu bài toán:

Một người đi du lịch muốn tham quan n thành phố T1 ...Tn. Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố đi qua đúng 1 lần rồi quay trở lại thành phố xuất phát.

Gọi Cịj là chi phí đi từ thành phố Ti  đến thành phố Tj. Hãy tìm một hành trình thỏa mãn yêu cầu bài toán sao cho chi phí là nhỏ nhất.

1. Mô tả chi tiết thuật toán:

* Ý tưởng

Gọi là một hoán vị của {1,…,n} thì một hành trình thỏa yêu cầu bài toán có dạng: .

Nên có tất cả n! hành trình như thế.

Nếu ta cố định một thành phố xuất phát, chẳng hạn , thì có (n-1)! hành trình.

Bài toán chuyển về dạng:

Tìm Min{f(a2,…,an) : (a2,..,an) là hoán vị của {2,..,n}}

Với f(a1,..,an) =

Cách giải bài toán sẽ kết hợp đánh giá nhánh cận trong quá trình liệt kê phương án của thuật toán quay lui.

* Thiết kế bài toán

Try(i)

For(j=2n)

If(chấp nhận được)

{

Xác định  theo j;

Ghi nhận trạng thái mới;

If(i==n)

Cập nhật lời giải tối ưu;

Else

{

Xác định cận g();

If(g() )

Try(i+1);

}

Trả lại trạng thái cũ cho bài toán;

}

* Nếu ta cố định xuất phát từ , ta duyệt vòng lặp từ j=2
* Đánh giá nhánh cận:

Đặt : Cmin = Min{:i,j {1,…,n}}

Giả sử vào các bước i ta tìm được lời giải bộ phận cấp i là (x1,…,xi), tức là đã đi qua đoạn đường , tương ứng với chi phí:

Để phát triển hành trình bộ phận này thành một hành trình đầy đủ, ta còn phải đi qua n-i+1 đoạn nữa, gồm n-I thành phố còn lại và đoạn quay lại .

Do chi phí mỗi một trong n-i+1 đoạn còn lại không nhỏ hơn Cmin, nên hàm đánh giá cận có thể xác định như sau:

g( ) =

* Điều kiện chấp nhận được của j là thành phố chưa đi qua.
* Xác định theo j bằng câu lệnh gán : = j

Cập nhật trạng thái mới: Daxet[j] = 1.

Cập nhật lại chi phí sau khi tìm tìm được : S=S+

* Cập nhật lời giải tối ưu:

Tính chi phí hành trình vừa tìm được:

Tong = S+;

Nếu (Tong <) thì

Lgtu = x;

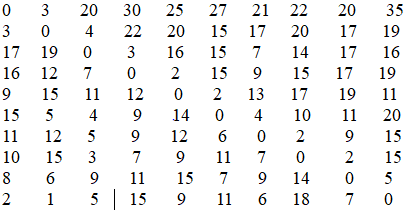
= Tong;

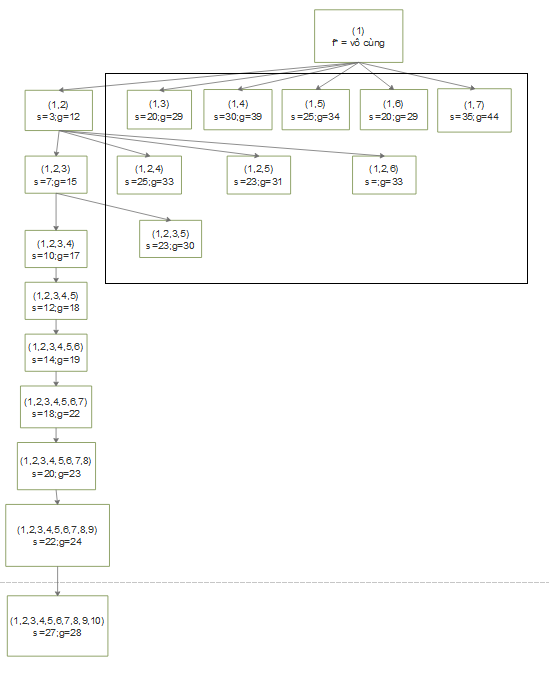
Thao tác hủy bỏ trạng thái: Daxet[j] = 0

Trả lại chi phí cũ : S=S-

1. Thực hiện bài toán.

Ma trận chi phí.





1. Triển khai với chương trình C++.

#include <iostream>

#include <vector>

#include <climits>

#include <fstream>

using namespace std;

const int N = 100; *// Số đỉnh tối đa của đồ thị*

const int INF = INT\_MAX; *// Vô cùng*

int n; *// Số đỉnh của đồ thị*

int C[N+1][N+1]; *// Ma trận trọng số*

int x[N+1]; *// Lưu trữ lời giải tốt nhất*

bool Daxet[N+1]; *// Mảng đánh dấu các đỉnh đã xét*

int S; *// Chi phí lời giải hiện tại*

int Gttu; *// Lưu trữ giá trị tốt nhất hiện thời*

int Httu[N+1]; *// Lưu trữ lời giải tốt nhất*

int Cmin; *// Lưu trữ trọng số nhỏ nhất*

int k =1; *// chọn đỉnh xuất phát*

*// sao chép lời giải tối ưu hiện thời sang lời giải tối ưu nhất (cuối cùng)*

void Gan(int *x*[], int *Httu*[], int *n*)

{

    for (int i = 1; i <= *n*; i++) {

*Httu*[i] = *x*[i];

    }

}

*// hàm giải bằng phương pháp nhánh cận*

void TRYY(int *i*)

{

    for (int j = 2; j <= n; j++)

    {

        if (!Daxet[j]) *// nếu gặp đỉnh chưa xét*

        {

            x[*i*] = j; *// thử đỉnh đang xét*

            Daxet[j] = true;

            S += C[x[*i* - 1]][x[*i*]]; *// cộng chi phí thêm đỉnh đang xét*

            if (*i* == n) *// Đã duyệt hết các đỉnh*

            {

                if (S + C[x[n]][x[1]] < Gttu)

                {

                    Gttu = S + C[x[n]][x[1]];

                    Gan(x, Httu, n);

                }

            }

            else if (S + (n - *i*) \* Cmin < Gttu) *// đánh giá cận*

            {

                TRYY(*i* + 1);

            }

            S -= C[x[*i* - 1]][x[*i*]];

            Daxet[j] = false;

        }

    }

}

*// hàm khởi tạo*

void Init()

{

*// khởi tạo mảng Daxet*

        for(int i =1;i<=n;i++)

        {

            Daxet[i]=false;

        }

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

*// tìm giá trị nhỏ nhất Cmin*

        for (int i = 1; i <= n; i++)

        {

            for(int j =1;j<=n;j++)

            {

                if(Cmin > C[i][j])

                    Cmin = C[i][j];

            }

        }

        Gttu = INF; *// khởi tạo giá trị tối ưu*

        S =0;

        Daxet[k]=true;

        x[1]=k;

}

int main()

{

    ifstream inputFile("input1.txt");

    if (!inputFile)

    {

        cout << "loi mo file" << endl;

        return 1;

    }

    inputFile >> n; *// đỉnh*

*// Đọc từ file*

    for (int i = 1; i <= n; i++)

    {

        for (int j = 1; j <= n; j++)

        {

            inputFile >> C[i][j];

        }

    }

    inputFile.close();

    Init();

    TRYY(2);

    cout<<"Ma tran chi phi:"<<endl;

    for(int i =1 ;i<=n;i++)

{

    for(int j=1;j<=n;j++)

    {

        cout<<C[i][j]<<" ";

    }

    cout<<endl;

}

*// in ra màn hình*

    cout << "\nLich trinh du lich toi uu: ";

    for (int i = 1; i <= n; i++) {

        cout << Httu[i] << " ";

    }

    cout << Httu[1] << endl; *// in ra đỉnh xuất phát*

*// in ra chi phí tốt nhất*

    cout << "Chi phi toi uu: " << Gttu << endl;

    return 0;

}

1. Đánh giá độ phức tạp.

Nhận xét thấy, sử hàm TRYY thực hiện phương pháp nhánh cận để giải quyết bài toán trên. Với vòng lặp for trong hàm TRYY được thực hiện n-1 lần nên có độ phức tạp là O(n-1), mặt khác trong hàm TRYY có sử dụng gọi đệ quy hàm TRYY.

* Trong trường hợp xấu nhất, với n lần gọi đệ quy, thì lúc đó độ phức tạp O(n)= O(n)\*O(n-1) = O(n2-n) = O(n2 ). Vậy trong trường hợp xấu nhất thì độ phức tạp của thuật toán trên là O(n2).
* Trong trường hợp tốt nhất, khi đó hàm đệ quy chỉ được gọi 1 lần, lúc này độ phức tạp của thuật toán chỉ là O(n-1) = O(n).