Đại học quốc gia TP. Hồ Chí Minh

Trường ĐH Khoa Học Tự Nhiên

Khoa Công Nghệ Thông Tin

Cải tiến thuật toán Bellman

Lớp : CTVP – 2011

Sinh viên : Trần Khánh Trình

MSSV : 1152002

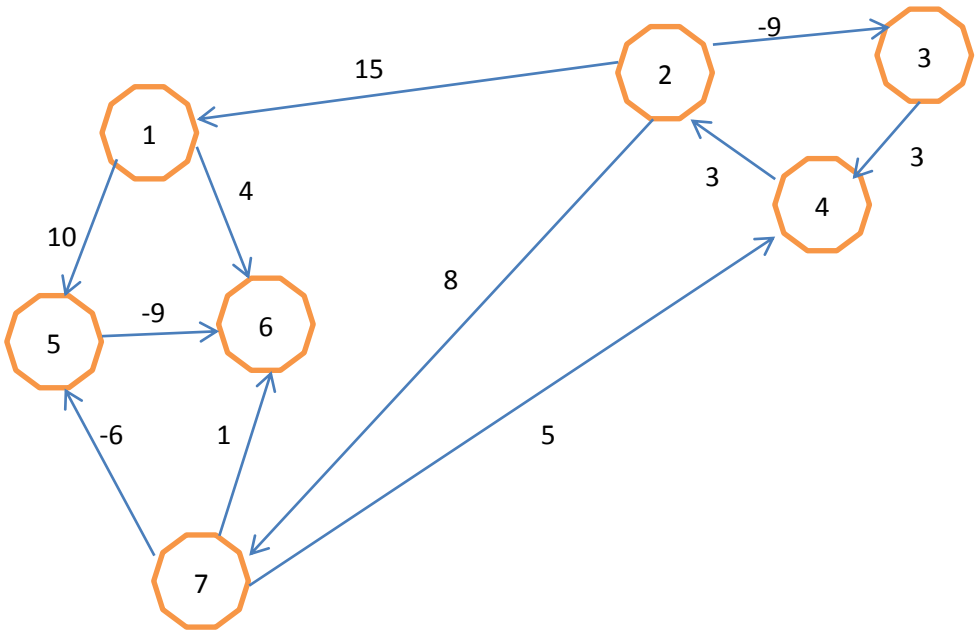
Mục lục

Phần I: Mục đích

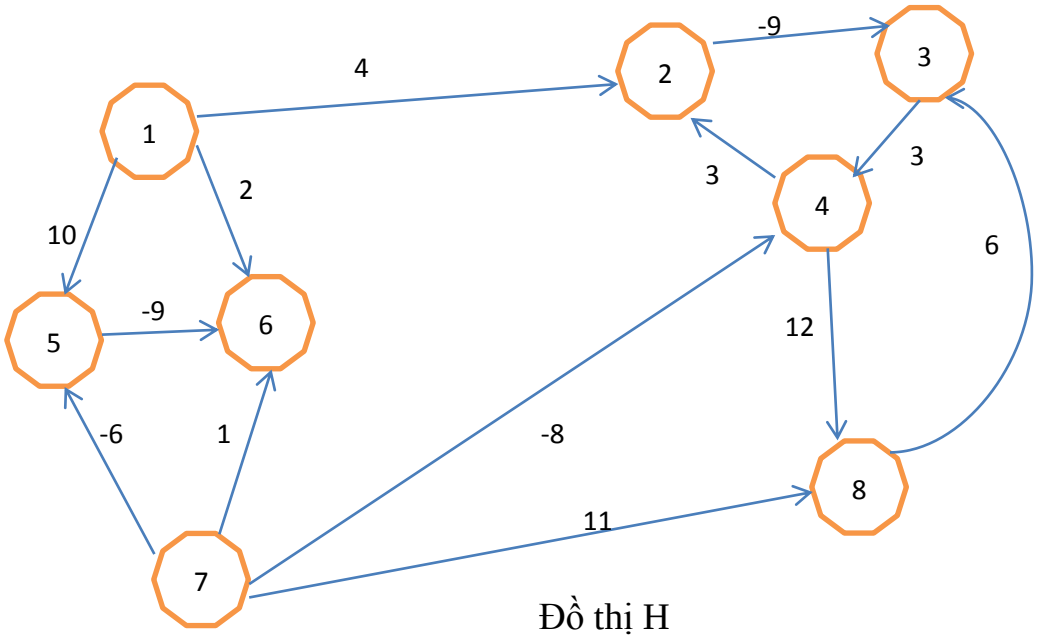
Tìm ra cách khắc phục khuyết điểm của Bellman : Không tìm ra được đường đi ngắn nhất (dù có) do gặp phải chu trình âm trong đồ thị.

Ví dụ:

Ta sẽ quan sát 2 đồ thị sau đây :



Đồ thị G



Đồ thị H

Xét đồ thị G:

- Đồ thị có chu trình âm, ta sẽ quan sát các đường đi bắt đầu từ đỉnh x = 1;
- Ta dễ dàng thấy từ 1 không đến được chu trình âm 2-3-4-2 do đó trong trường hợp này Bellman xài tốt.

Trong khi đó, đồ thị H cũng có chu trình âm tương tự như trong đồ thị G và ta cũng khởi động từ x = 1.

- Trong trường hợp này thì khác từ 1 đi thẳng vào chu trình âm được.
- Chạy thuật toán Bellman :
    - Gặp chu trình âm nên sẽ dừng lại.
    - Trong khi 1 số đỉnh như 5,6,7 vẫn có đường đi ngắn nhất từ 1 đến chúng.

⇨ Bellman gốc không giải quyết được trường hợp này, do đó ta phải cải tiến Bellman để xử lý được các trường hợp tương tự.

Phần II : Hướng giải quyết

- Ta quan sát thấy trên đồ thị H : các đỉnh 2, 3, 4, 8 có 1 đặc điểm chung như sau : nằm trên chu trình âm hoặc từ chu trình âm đi đến chúng.(Ta sẽ gọi đây là các đỉnh loại B và dĩ nhiên số đỉnh còn lại sẽ là đỉnh loại A).
- Để giải quyết vấn đề trên, trước tiên, ta cần xác định xem từ đỉnh ta chọn làm đỉnh bắt đầu x (ở trên ta chọn x = 1) có gặp chu trình âm hay không.
    - Nếu không gặp : Bellman gốc sẽ chạy tốt trong trường hợp này.
    - Nếu gặp, ta sẽ sử dụng thuật toán Bellman cải tiến (sẽ được nói ở phần sau).
- Khi x đi gặp chu trình âm, ta sẽ tìm cách loại bỏ hết các đỉnh loại B mà x gặp phải, sau đó, ta sẽ ứng dụng Bellman (hoặc Dijkstra với đồ thị mới, đồ thị chỉ toàn đỉnh loại A). Do đó, từ x sẽ không còn đường đi đến đỉnh loại B và công việc bây giờ chỉ là tìm đường đi ngắn nhất từ x đến đỉnh loại A mà thôi.

Phần III: Thuật toán

Để cải tiến thuật toán Bellman, vấn đề mấu chốt là ta phải tìm cách đưa tất cả các đỉnh trên chu trình âm hoặc từ chu trình âm đi ra để loại chúng ra khỏi đồ thị. Ta chắc chắn sẽ có 1 điều sau:

- Tổng các trọng số âm của đồ thị (ta sẽ gọi là MinCost), sẽ là trọng số nhỏ nhất có thể có của bất kỳ đỉnh nào trong đồ thị. Do đó, nếu đỉnh nào nằm trong chu trình âm, do khuyết điểm của Bellman sẽ

làm cho đỉnh đó có trọng số nhỏ hơn cả MinCost thì chắc chắn nó nằm trên chu trình âm.

- Từ đỉnh vừa tìm được, ta se tìm các đỉnh có liên quan với nó bằng phương thức Relation (sẽ được giới thiệu trong mã nguồn) , phương thức này sẽ giúp ta tìm hết các đỉnh loại B bắt nguồn từ đỉnh tìm được.

Và các bước của thuật toán sẽ như sau :

Bước 1: Tìm MinCost := Tổng các trọng số âm trong đồ thị.

Bước 2: Chạy 1 lần bằng Bellman cổ điển :

- Nếu không có chu trình âm, thì in kết quả, kết thúc chương trình
- Nếu có chu trình âm, thì sang tiếp bước 3.

Bước 3: Ta sẽ cho chạy Bellman không giới hạn số lần như sau :

- Ta sẽ tạo 1 biến First_B (đỉnh đầu tiên loại B), biến này sẽ có giá trị khởi đầu là -1;
- Trong khi First_B bằng -1,
  - Ta sẽ cập nhật lại P_row và P_NextRow.
  - Rồi trong các đỉnh, nếu tìm được đỉnh nào có trọng số < MinCost, ta đưa đỉnh đó vào tập đỉnh loại B đồng thời gán First_B là đỉnh đó.
  - Sau đó ta sẽ dùng phương thức Relation, để loại hết các đỉnh loại B.
  - Kế tiếp, ta sẽ cho chạy lại Bước 2 với đỉnh ban đầu x.

Về phần Bellman cổ điển, ta cũng có 1 thay đổi nhỏ : là chỉ chạy trên các đỉnh loại A.

Phần IV : Mã nguồn

```
#ifndef BELLMAN_H
#define BELLMAN_H
#include <vector>
#include <stdio.h>
using namespace std;
struct Vertex
{
	float length;
	int lastV;
	char extremityFlag;
	int lastVir;// Đỉnh cũ ảo của 1 đỉnh
	float lengthVir; // Trọng số ảo
};
class BellmanAlgo
{
private:
	vector <Vertex> P_Row;
	vector <Vertex> P_NextRow;
	int k, startVer;
	void InitAlg(int x);
	void ReInitAlg(int x);
	void UpdateNextRow();
	void UpdateNextRow2();
	int TwoRowsEqual();
	vector<vector<float>> ArrDistance;
	vector<vector<int>> B;
	float MinCost;
	void CalculMinCost();
	void CheckRelation(int);
	bool CheckDinhLoaiB(Vertex);
public:
	int N;
	int BellmanKhongGioiHan(int);
	vector<int> DinhLoaiB;
	float Distance(int i, int j);
	int RunAlg(int x);
	int RunAlg2(int x);
	void PrintPath(int y);
	int ReadData(char * fname);
};
#endif
```

Tập tin Bellman.h

```
int BellmanAlgo::ReadData(char * fname)
{
	FILE * f = fopen (fname,"rt");
	if(f == NULL)
		return 0;
	fscanf(f,"%d",&N);
	ArrDistance.resize(N);
	for(int i = 0;i<N; i++)
	{
		ArrDistance[i].resize(N);
		for(int j = 0; j < N; j++)
		fscanf(f,"%f",&ArrDistance[i][j]);
	}
	B.resize(N);
	for(int i= 0 ; i < N; i++)
	{
		B[i].resize(N);
		for(int j =0; j<N; j++)
			fscanf(f,"%d",&B[i][j]);
	}
	CalculMinCost();
	fclose(f);
	return 1;
                                        }
```

Hàm ReadData (Đọc dữ liệu)

```
void BellmanAlgo::CalculMinCost()
{
	MinCost = 0;
	for(int i = 0; i < N ; i++)
	{
		for(int j = 0; j < N; j++)
		if (ArrDistance[i][j]< 0)
			MinCost += ArrDistance[i][j];
	}

                                        }
```

Hàm CalculMinCost (Hàm này sẽ giúp ta tính tổng trọng số âm)

```
void BellmanAlgo::InitAlg(int x)
{

	P_Row.resize(N);
	P_NextRow.resize(N);
	DinhLoaiB.resize(N);
```

```
	k = 1;
	for(int i = 0; i < N; i++)
	{
		P_Row[i].length = 0;
		P_Row[i].extremityFlag = 1;
		P_Row[i].lastV =-1;
		DinhLoaiB[i] = 0;
		P_Row[i].lastVir = 0;
		P_Row[i].lengthVir = -1;
	}
	startVer = x;
	P_Row[x].extremityFlag = 0;
                                        }
```

Hàm InitAlg (Khởi đầu của thuật toán Bellman cổ điển)

```
void BellmanAlgo::UpdateNextRow() //Bellman cổ điển
{
	Vertex v;
	for (int  i = 0; i < N; i++)
	{
		if(DinhLoaiB[i] == 0)
		{
		v = P_Row[i];
		for(int j = 0 ; j < N ; j++)
		{
			if(B[j][i] == 0)
				continue;
			float len = Distance(j,i);
			if(P_Row[j].extremityFlag != 1 && DinhLoaiB[j] == 0 &&
DinhLoaiB[i] == 0)
			{
				len += P_Row[j].length;
				if(v.extremityFlag || v.length > len)
					{
							v.length = len;
							v.lastV = j;
							v.extremityFlag = 0;

					}
			}
		}

			P_NextRow[i] = v;
		}}}
```

Hàm UpdateNextRow(Cập nhật dòng kế của Bellman cổ điển)

```
int BellmanAlgo ::TwoRowsEqual()
{
	int Equal = 1;//true;
	for(int i = 0; i < N; i++)
	{
		if (P_Row[i].extremityFlag != P_NextRow[i].extremityFlag
			|| P_Row[i].length != P_NextRow[i].length )
		{
			Equal = 0;
		}
	}
	return Equal;
}
```

Hàm TwoRowsEqual (Kiểm tra xem 2 dòng có bằng nhau không, của Bellman cổ điển)

```
int BellmanAlgo::RunAlg(int x)
{
	if (x <0 || x >= N)
		return -1; // Nhập sai dữ liệu
	InitAlg(x);
	while (k < N)
	{

		UpdateNextRow();
		if(TwoRowsEqual())
			break;
		k++;
	for(int i = 0; i < N; i++)
		P_Row[i] = P_NextRow[i];

	}


	if(k==N)
		return -2; // Có chú trình âm
	return 1;// Chạy tốt
}
```

Hàm RunAlg(Hàm chạy chỉnh của Bellman cổ điển)

```
void BellmanAlgo::ReInitAlg(int x)
{
	k = 1;
	for(int i = 0; i < N; i++)
	{
```

```
            P_Row[i].length = 0;
            P_Row[i].extremityFlag = 1;
            P_Row[i].lastV =-1;
      }
      startVer = x;
      P_Row[x].extremityFlag = 0;
                                    }
```

Hàm ReInitAlg(Tái khởi tạo dùng trong Bellman cải tiến).

```
int BellmanAlgo::RunAlg2(int x)
{
      if (x <0 || x >= N)
            return -1;
      ReInitAlg(x);
      while (k < N)
      {

            UpdateNextRow();
            if(TwoRowsEqual())
                  break;
            k++;
      for(int i = 0; i < N; i++)
            P_Row[i] = P_NextRow[i];

      }


      if(k==N)
            return -2;
      return 1;}
```

Hàm RunAlg2(Hàm chạy Bellman cổ điển trong trường hợp tồn tại chu trình âm)

```
void BellmanAlgo::UpdateNextRow2()
{
      Vertex v;
      for (int  i = 0; i < N; i++)
      {
            if(DinhLoaiB[i] == 0)
            {
            v = P_Row[i];
            for(int j = 0 ; j < N ; j++)
            {
                  if(B[j][i] == 0)
                        continue;
                  float len = Distance(j,i);
```

```
                if(P_Row[j].extremityFlag != 1 && DinhLoaiB[j] == 0 &&
DinhLoaiB[i] == 0)
                {
                        len += P_Row[j].length;
                        if(v.extremityFlag || v.length > len)
                                {
                                        v.length = len;
                                        v.lastV = j;
                                        v.extremityFlag = 0;
                                        v.lengthVir = v.length;
                                        v.lastVir = -1;
                                }
                }
        }

                P_NextRow[i] = v;}}}
```

Hàm UpdateNextRow2(dùng trong Bellman cải tiến).

```
int BellmanAlgo::BellmanKhongGioiHan(int x)
{
      if (x <0 || x >= N)
            return -1; // Se thong bao la du lieu sai (vi dinh nhap vao
<0 va lon hon so dinh);
      int First_B = -1;
      while (First_B == -1)
      {
                UpdateNextRow2();
                for(int j = 0; j< N; j++)
                {
                        if(DinhLoaiB[j] == 0 && P_Row[j].lengthVir <
MinCost)
                        {
                                DinhLoaiB[j] = 1;
                                First_B = j;
                                //Kiem tra cac dinh co lien he voi dinh
loai B vua tim duoc
                                CheckRelation(First_B);
                                while(RunAlg2(x) == -2)
                                {
                                        BellmanKhongGioiHan(x);
                                }
                                break;
                        }
                }
            for(int i = 0; i < N; i++)
```

```
            P_Row[i] = P_NextRow[i];

                                                }
```

Hàm BellmanKhongGioiHan(Để tìm ra chu trình âm, đồng thời chạy lại hàm Bellman cổ điển cho đồ thị sau khi loại hết đỉnh B).

```
void BellmanAlgo::CheckRelation(int x)
{
      for(int i = 0; i < N; i++)
      {
            if (ArrDistance[x][i] != 0 && DinhLoaiB[i] == 0)
            {
                  DinhLoaiB[i] = 1;
                  CheckRelation(i);
            }
      }
                                    }
```

Thủ thuật CheckRelation (giúp ta tìm được tất cả các đỉnh có liên quan với đỉnh loại B ban đầu).

```
void BellmanAlgo::PrintPath(int y)
{

      if(y == startVer)
            printf("%d",y+1);
      else
      {
            int z = P_Row[y].lastV;
            if(z == -1)
            {
                  printf("Khong co duong!");
                  return;
            }
            PrintPath(z);
            printf("---> %d",y+1);      }}
```

Hàm PrintPath(giúp ta in ra đường đi (nếu không có chu trình âm).

```
float BellmanAlgo::Distance(int j, int i)
{
      return ArrDistance[j][i];
                                    }
```

Thủ thuật nhỏ giúp ta lấy giá trị trọng số giữa 2 đỉnh đã được lưu

```
#include"Bellman.h"
#include <stdio.h>
```

```
#include <conio.h>
void main()
{
	int x,y;
	char * fname = "Input.txt";
	BellmanAlgo B;
	if (B.ReadData(fname))
	{
		printf("Nhap dinh bat dau (tu 1 den n): ");
		scanf ("%d",&x);
		if(B.RunAlg(x-1) == -2)
		{
			if(B.BellmanKhongGioiHan(x-1) == -1)
				printf("Nhap sai du lieu!");
			//Chay bellman khong gioi han.
			else
			{
				for(int i = 0; i < B.N;i++)
					{
						if(B.DinhLoaiB[i] == 1)
						{
							printf ("\n Dinh %d nam trong
chu trinh am hoac tu chu trinh am di ra nen k co duong di!",i+1);
							printf("\n");
							continue;
						}
						else
						{
							printf("\n Duong di tu %d toi %d
:",x,i+1);
							B.PrintPath(i);
							printf("\n");
						}
					}
			}
		}
		else
		{
			if(B.RunAlg(x-1) == -1)
				printf("Nhap sai du lieu!");
			else
		{
			//in bthuong
			for(int i = 0; i < B.N;i++)
			{
				printf("\n Duong di tu %d toi %d :",x,i+1);
```

```
                        B.PrintPath(i);
                        printf("\n");
                  }
            }
      }
      }
      else
            printf("Khong the mo file, kiem tra lai!");
      getch();
                                    }
```

Hàm main kèm với những thủ thuật in .

==Hết==