**BÁO CÁO PHÂN TÍCH THIẾT KẾ VÀ GIẢI THUẬT**

***Sinh Viên : Lê Anh Đức***

***Lớp: MMT14-16***

***NS: 27/8/1997***

***MSV:15150076***

**Bài 19**: Với bài toán tìm đường đi ngắn nhất trên đồ thị và thuật toán **Bellman Ford** hãy trình bày các nội dung sau:

1. Nêu bài toán;

2. Mô tả chi tiết thuật toán;

3. Đánh giá độ phức tạp thuật toán;

4. Tự xác định 2 bộ dữ liệu (với số phần tử N>=5), với mỗi bộ dữ liệu hãy thực hiện từng bước thuật toán đã mô tả ở mục 2 và ghi ra kết quả mỗi bước;

5. Viết chương trình trên C/C++

6. Viết báo cáo trình bày các nội dung từ 1-4 và kết quả thực hiện chường trình (5).

1. Nêu bài toán:

Bài toán tìm đường đi ngắn nhất nguồn đơn : là bài toán tìm một đường đi giữa 2 đỉnh bất kí của 1 đồ thì sao cho tổng trọng số các cạnh tạo nên đường đi đó là nhỏ nhất.  Định nghĩa một cách hình thức, cho trước một đồ thị có trọng số (nghĩa là một tập đỉnh *V*, một tập cạnh *E*, và một hàm trong số có giá trị [thực](https://vi.wikipedia.org/wiki/S%E1%BB%91_th%E1%BB%B1c) *f* : *E* → **R**), cho trước một đỉnh *v* thuộc *V*, tìm một đường đi *P* từ *v* tới mỗi đỉnh *v'* thuộc *V* sao cho là nhỏ nhất trong tất cả các đường nối từ *v* tới *v'*

**Thuật toán Bellman-Ford** là một [thuật toán](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n) tính các đường đi ngắn nhất nguồn đơn trong một [đồ thị có hướng có trọng số](https://vi.wikipedia.org/w/index.php?title=%C4%90%E1%BB%93_th%E1%BB%8B_c%C3%B3_h%C6%B0%E1%BB%9Bng_c%C3%B3_tr%E1%BB%8Dng_s%E1%BB%91&action=edit&redlink=1) (trong đó một số [cung](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_th%E1%BB%8B_(l%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%93_th%E1%BB%8B)) có thể có trọng số âm). [Thuật toán Dijkstra](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_Dijkstra) giải cùng bài toán này tuy nhiên Dijkstra có thời gian chạy nhanh hơn đơn giản là đòi hỏi trọng số của các cung phải có giá trị không âm.

Thuật toán Bellman Ford chạy trong thời gian *O*(*V*·*E*), trong đó *V* là số đỉnh và *E* là số cung của đồ thị.

2. Mô tả chi tiết thuật toán:

Input: + 1 Ma trận trong số .

+ Điểm băt đầu đi để đi đến các đỉnh còn lại.(nguồn)

+Một mảng lưu độ dai tu 1 dinh den cac dinh cua no

Output: + Một mảng tổng đường đi ngắn nhất từ đỉnh bắt đầu đến các đỉnh còn lại.

function BellmanFord(danh\_sách\_đỉnh, danh\_sách\_cung, nguồn)

// hàm yêu cầu đồ thị đưa vào dưới dạng một danh sách đỉnh, một danh sách cung

// hàm tính các giá trị khoảng\_cách và đỉnh\_liền\_trước của các đỉnh,

// sao cho các giá trị đỉnh\_liền\_trước sẽ lưu lại các đường đi ngắn nhất

// bước 1: khởi tạo đồ thị

for each v in danh\_sách\_đỉnh:

if v is nguồn then khoảng\_cách(v):= 0

else khoảng\_cách(v):= vô cùng

đỉnh\_liền\_trước(v):= max int

// bước 2: kết nạp cạnh

for i from 1 to (Tổng số đỉnh -1):

for each (u,v) in (Tổng số cạnh):

if (khoảng\_cách(v) > khoảng\_cách(u) + trọng\_số(u,v)):

khoảng\_cách(v):= khoảng\_cách(u) + trọng\_số(u,v)

đỉnh\_liền\_trước(v):= u

// bước 3: kiểm tra chu trình âm

for each (u,v) in danh\_sách\_cung:

if khoảng\_cách(v) > khoảng\_cách(u) + trọng\_số(u,v):

error "Đồ thị chứa chu trình âm"

2.Đánh giá đọ phức tạp của thuật toán:

Do sử dụng 2 vòng for duyệt vét cạn theo đỉnh(V)

Và 1 còn for để kết nạp cạnh (E) nên:

Độ phức tạp của thuật toán là O = O(V\*E);

Nếu tính cả bước kiểm tra đồ thị có chu trình âm hay ko : O = O(V\*E+V);

Tính đúng đắn của thuật toán:

+Thuật toán tìm dược đường đi ngắn nhất đến các đỉnh

+ tìm được trong đồ thị có tồn tại chu trình âm hay không.

4. Tự xác định 2 bộ dữ liệu (với số phần tử N>=5), với mỗi bộ dữ liệu hãy thực hiện từng bước thuật toán đã mô tả ở mục 2 và ghi ra kết quả mỗi bước;

Bộ dữ liệu 1:

-1

1

9

4

2

1

2

14

6

Tìm tất cả các đường đi từ đỉnh 0 đến đỉnh 5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| step | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | **∞** | **∞** | **∞** | **∞** | **∞** |
| 1 | 0 | 1 | **∞** | **∞** | **∞** | **∞** |
| 2 | 0 | 1 | 5 | 3 | 15 | **∞** |
| 3 | 0 | 1 | 5 | 3 | 15 | 6 |
| 4 | 0 | 1 | 5 | 3 | 9 | 5 |
| 5 | 0 | 1 | 5 | 3 | 9 | 5 |
| 6 | 0 | 1 | 4 | 3 | 9 | 5 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

- **Bước 0:** Ta đánh dấu đỉnh xuất phát (0) = 0, các đinh còn lại bằng vô cực.

- **Bước 1:**

Tại đỉnh (1) có đỉnh (0) đi vào có chi phí hiện tại 1 < chi phí trước (∞) => cập nhật lại chi phí đỉnh (1)

- **Bước 2:**

Tại đỉnh (2) có đỉnh (1) đi vào có chi phí hiện tại 5 < chi phí trước (∞) => cập nhật lại chi phí đỉnh (2) =5

Tại đỉnh (3) có đỉnh (1) đi vào có chi phí hiện tại 3 < chi phí trước (∞) => cập nhật lại chi phí đỉnh (3) =3

Tại đỉnh (4) có đỉnh (1) đi vào có chi phí hiện tại 15 < chi phí trước (∞) => cập nhật lại chi phí đỉnh (2) =15

- **Bước 3:**

Tại đỉnh (5) có đỉnh (2) đi vào có chi phí hiện tại 6 < chi phí trước (∞) => cập nhật lại chi phí đỉnh (5) =6

- **Bước 4:**

Tại đỉnh (4) có đỉnh (3) đi vào có chi phí hiện tại 9 < chi phí trước (15) => cập nhật lại chi phí đỉnh (4) =9

- **Bước 5:**

Tại đỉnh (5) có đỉnh (3) đi vào có chi phí hiện tại 5 < chi phí trước (6) => cập nhật lại chi phí đỉnh (5) =5

- **Bước 6:**

Tại đỉnh (2) có đỉnh (5) đi vào có chi phí hiện tại 4 < chi phí trước (5) => cập nhật lại chi phí đỉnh (2) =4

Bộ dữ liệu 2: cũng như đồ thị trên nhưng trong đồ thị có chu trình âm

-1

1

9

4

2

1

2

3

-16

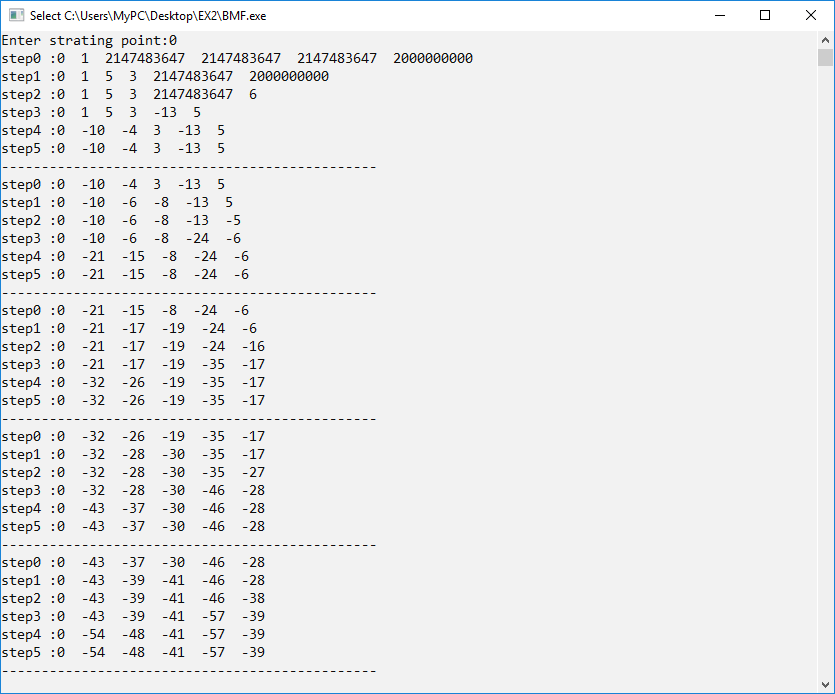
Tìm tất cả các đường đi từ đỉnh 0 đến đỉnh 5

Duyệt lần 1:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| step | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | **∞** | **∞** | **∞** | **∞** | **∞** |
| 1 | 0 | 1 | **∞** | **∞** | **∞** | **∞** |
| 2 | 0 | 1 | 5 | 3 | **∞** | **∞** |
| 3 | 0 | 1 | 5 | 3 | **∞** | 6 |
| 4 | 0 | 1 | 5 | 3 | -13 | 5 |
| 5 | 0 | -10 | -4 | 3 | -13 | 5 |
| 6 | 0 | -10 | -4 | 3 | -13 | 5 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Duyệt lần 2:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| step | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | -10 | -4 |  |  |  |
| 2 | 0 | -10 | -6 |  |  |  |
| 3 | 0 | -10 |  |  |  |  |
| 4 | 0 | -10 |  |  |  |  |
| 5 | 0 | -21 |  |  |  |  |
| 6 | 0 | -21 |  |  |  |  |



5.Viết thuật toán trên C++:

void Bell\_Man\_frod(int G[6][6], int distance[6], int start, int VerticesCount) {

// khoi tao tat cac phan tu trong mang tong trong so cua tat ca cac duong di;

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

distance[i] = MaxInt;

}

distance[start] = 0;// gan cho trong so diem bat dau la 0

// vet can tat ca cac truong hop di den cac dinh

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

{

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

{

for (int k = 0; k <VerticesCount; k++)

{

if (j != k)

{

// j o day la dinh dau

// k la dinh cuoi

int weight = G[j][k];// trong so

// nếu nhãn của đỉnh đầu + trọng số của cạnh giữa 2 đỉnh đang xét < nhãn của đỉnh cuối thì gãn nhãn của đỉnh đầu = nhãn của đỉnh đầu + trọng số của cạnh giữa 2 đỉnh đang xét

if (distance[j] != MaxInt && distance[j] + weight < distance[k] && weight != MaxInt)

distance[k] = distance[j] + weight;

}

}

}

}

// Kiem tra chu trinh co am hay khong

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

{

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

{

if (i != j) {

int weight = G[i][j];

// khi tim đc đường đến các đỉnh có tổng số bé nhất xông xuôi rồi, nếu mà nó tìm thấy một đường đi khác từ các nhãn khác mà ngắn hơn giá trị cái nãn đang có thì đồ thị này có chu trình âm

if (distance[i] != MaxInt && weight != MaxInt && distance[i] + weight < distance[j])

cout<<"Graph contains negative weight cycle.\n";

}

}

}

Print(VerticesCount,distance);

}