**A**

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BÁO CÁO THỰC TẬP CƠ SỞ**

**Cài đặt thuật toán tìm đường đi ngắn nhất sử dụng ma trận kề**

**GVHD:Nguyễn Thủy Đoan Trang**

**Họ và Tên: Lê Vũ Thanh Hải**

**MSSV: 60139139**

**Lớp : 60.CNTT-2**

**LỜI NHẬN XÉT**

**( Của giảng viên hướng dẫn )**

**Báo cáo tiến độ thực tập**

**Đề tài:** Cài đặt thuật toán tìm đường đi ngắn nhất sử dụng ma trận kề

Tên: Lê Vũ Thanh Hải

Lớp: 60.CNTT-2

MSSV: 60139139

**Lời mở đầu**

1. Mục tiêu cần đạt được:

* Mức 1:

1. Sử dụng ngôn ngữ C/C++ cài đặt các thuật toán tìm đường đi ngắn nhất đồ thị Disktra với bằng ma trận kề.
2. Dữ liệu vào: File văn bản DIJKSTRA.INP
   * Dòng đầu tiên chứa số đỉnh 𝑛, số cung 𝑚 của đồ thị và 2 đỉnh 𝑥, 𝑦.
   * 𝑚 dòng tiếp theo, mỗi dòng chứa 3 số 𝑢, 𝑣, 𝑤 mô tả cung (𝑢, 𝑣) có trọng số 𝑤.

Dữ liệu ra: File văn bản DIJKSTRA.OUT

* + Dòng đầu tiên ghi số đỉnh đường đi ngắn nhất đi qua (tính luôn hai đỉnh 𝑥 và 𝑦) và độ dài đường đi.
  + Dòng thứ hai ghi danh sách các đỉnh của đường đi ngắn nhất tìm được từ 𝑠 đến 𝑡.
* Mức 2: Mô phỏng bằng đồ họa.

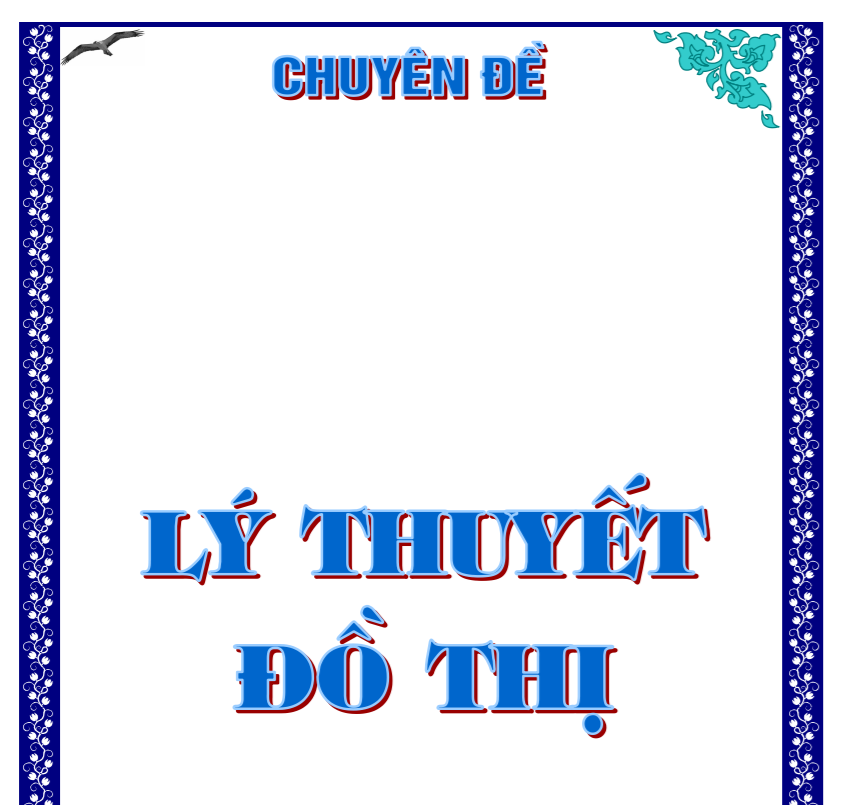
1. Lý do chọn đề tài:

-Lý do em chọn đề tài này là vì em tự cảm thấy trình độ của mình xứng với đề tài này nên chọn.

**Tài liệu đã tìm kiếm:**

+ Lý thuyết đồ thị: thầy Lê Minh Hoàng

https://www.hnue.edu.vn/Portals/0/TeachingSubject/hongntcntt/07b6e3d3-6727-489d-a0c5-c81f5f24daa1ly-thuyet-do-thi---le-minh-hoang.pdf



**+** Lý thuyết đồ thị: tiến sỹ Ngô Hữu Phúc và thạc sỹ Vi Bảo Ngọc

<https://fit.lqdtu.edu.vn/files/FileMonHoc/Ly%20Thuyet%20do%20thi.pdf>



**Tìm hiểu GITHUB:**

B1: GITHUB.com => Login

B2: Tạo kho lưu trữ

B3: Tải github

B4: Mở github vừa tải về => Clone a repository => Kho lưu trữ

**CHƯƠNG I: LÝ THUYẾT**

**1.MA TRẬN LIỀN KỀ (MA TRẬN KỀ):**

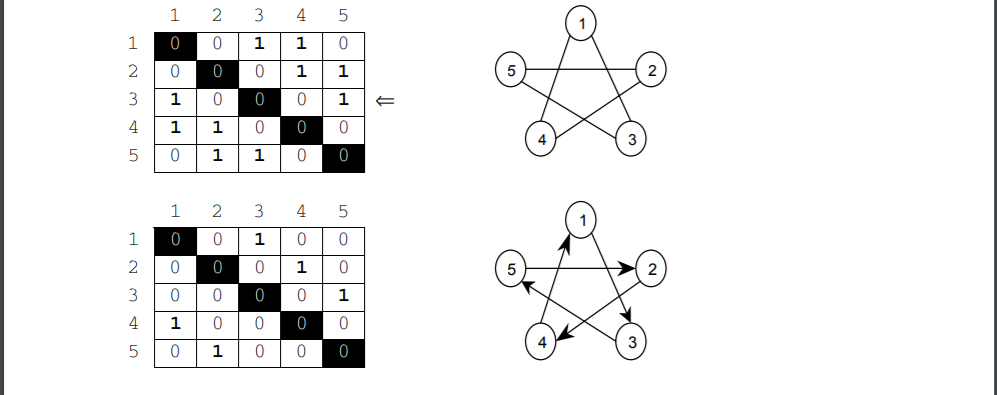
Giả sử G = (V, E) là một đơn đồ thị có số đỉnh (ký hiệu |V|) là n, Không mất tính tổng quát có thể coi các đỉnh được đánh số 1, 2, ..., n. Khi đó ta có thể biểu diễn đồ thị bằng một ma trận vuông A = [aij] cấp n. Trong đó:

• aij = 1 nếu (i, j) ∈ E

• aij = 0 nếu (i, j) ∉ E

• Quy ước aii = 0 với ∀i; Đối với đa đồ thị thì việc biểu diễn cũng tương tự trên, chỉ có điều nếu như (i, j) là cạnh thì không phải ta ghi số 1 vào vị trí aij mà là ghi số cạnh nối giữa đỉnh i và đỉnh j

Ví dụ:



Các tính chất của ma trận liền kề:

1. Đối với đồ thị vô hướng G, thì ma trận liền kề tương ứng là ma trận đối xứng (aij = aji), điều này không đúng với đồ thị có hướng.

2. Nếu G là đồ thị vô hướng và A là ma trận liền kề tương ứng thì trên ma trận A: Tổng các số trên hàng i = Tổng các số trên cột i = Bậc của đỉnh i = deg(i)

3. Nếu G là đồ thị có hướng và A là ma trận liền kề tương ứng thì trên ma trận A:

• Tổng các số trên hàng i = Bán bậc ra của đỉnh i = deg+ (i)

• Tổng các số trên cột i = Bán bậc vào của đỉnh i = deg- (i)

Trong trường hợp G là đơn đồ thị, ta có thể biểu diễn ma trận liền kề A tương ứng là các phần tử logic. aij = TRUE nếu (i, j) ∈ E và aij = FALSE nếu (i, j) ∉ E

Ưu điểm của ma trận liền kề:

• Đơn giản, trực quan, dễ cài đặt trên máy tính

• Để kiểm tra xem hai đỉnh (u, v) của đồ thị có kề nhau hay không, ta chỉ việc kiểm tra bằng một phép so sánh: auv ≠ 0.

Nhược điểm của ma trận liền kề:

• Bất kể số cạnh của đồ thị là nhiều hay ít, ma trận liền kề luôn luôn đòi hỏi n2 ô nhớ để lưu các phần tử ma trận, điều đó gây lãng phí bộ nhớ dẫn tới việc không thể biểu diễn được đồ thị với số đỉnh lớn.

Với một đỉnh u bất kỳ của đồ thị, nhiều khi ta phải xét tất cả các đỉnh v khác kề với nó, hoặc xét tất cả các cạnh liên thuộc với nó. Trên ma trận liền kề việc đó được thực hiện bằng cách xét tất cả các đỉnh v và kiểm tra điều kiện auv ≠ 0. Như vậy, ngay cả khi đỉnh u là **đỉnh cô lập** (không kề với đỉnh nào) hoặc **đỉnh treo** (chỉ kề với 1 đỉnh) ta cũng buộc phải xét tất cả các đỉnh và kiểm tra điều kiện trên dẫn tới lãng phí thời gian.

**2.THUẬT TOÁN DIJKSTRA:**

Trong trường hợp trọng số trên các cung không âm, thuật toán do Dijkstra đề xuất dưới đây hoạt động hiệu quả hơn nhiều so với thuật toán Ford-Bellman. Ta hãy xem trong trường hợp này, thuật toán Ford-Bellman thiếu hiệu quả ở chỗ nào: Với đỉnh v ∈ V, Gọi d[v] là độ dài đường đi ngắn nhất từ S tới v. Thuật toán Ford-Bellman khởi tạo d[v] = c[S, v]. Sau đó tối ưu hoá dần các nhãn d[v] bằng cách sửa nhãn theo công thức: d[v] := min(d[v], d[u] + c[u, v]) với ∀u, v ∈ V. Như vậy nếu như ta dùng đỉnh u sửa nhãn đỉnh v, sau đó nếu ta lại tối ưu được d[u] thêm nữa thì ta cũng phải sửa lại nhãn d[v] dẫn tới việc d[v] có thể phải chỉnh đi chỉnh lại rất nhiều lần. Vậy nên chăng, tại mỗi bước không phải ta xét mọi cặp đỉnh (u, v) để dùng đỉnh u sửa nhãn đỉnh v mà sẽ chọn đỉnh u là đỉnh mà không thể tối ưu nhãn d[u] thêm được nữa. Thuật toán Dijkstra (E.Dijkstra - 1959) có thể mô tả như sau:

**Bước 1:** Khởi tạo Với đỉnh v ∈ V, gọi nhãn d[v] là độ dài đường đi ngắn nhất từ S tới v. Ta sẽ tính các d[v]. Ban đầu d[v] được khởi gán bằng c[S, v]. Nhãn của mỗi đỉnh có hai trạng thái tự do hay cố định, nhãn tự do có nghĩa là có thể còn tối ưu hơn được nữa và nhãn cố định tức là d[v] đã bằng độ dài đường đi ngắn nhất từ S tới v nên không thể tối ưu thêm. Để làm điều này ta có thể sử dụng kỹ thuật đánh dấu: Free[v] = TRUE hay FALSE tuỳ theo d[v] tự do hay cố định. Ban đầu các nhãn đều tự do.

**Bước 2:** Lặp Bước lặp gồm có hai thao tác:

1. Cố định nhãn: Chọn trong các đỉnh có nhãn tự do, lấy ra đỉnh u là đỉnh có d[u] nhỏ nhất, và cố định nhãn đỉnh u.

2. Sửa nhãn: Dùng đỉnh u, xét tất cả những đỉnh v và sửa lại các d[v] theo công thức:

d[v] := min(d[v], d[u] + c[u, v])

Bước lặp sẽ kết thúc khi mà đỉnh đích F được cố định nhãn (tìm được đường đi ngắn nhất từ S tới F); hoặc tại thao tác cố định nhãn, tất cả các đỉnh tự do đều có nhãn là +∞ (không tồn tại đường đi).

Có thể đặt câu hỏi, ở thao tác 1, tại sao đỉnh u như vậy được cố định nhãn, giả sử d[u] còn có thể tối ưu thêm được nữa thì tất phải có một đỉnh t mang nhãn tự do sao cho d[u] > d[t] + c[t, u]. Do trọng số c[t, u] không âm nên d[u] > d[t], trái với cách chọn d[u] là nhỏ nhất. Tất nhiên trong lần lặp đầu tiên thì S là đỉnh được cố định nhãn do d[S] = 0. Bước 3: Kết hợp với việc lưu vết đường đi trên từng bước sửa nhãn, thông báo đường đi ngắn nhất tìm được hoặc cho biết không tồn tại đường đi (d[F] = +∞).

**3.THUẬT TOÁN DIJKSTRA VÀ CẤU TRÚC HEAP**

Nếu đồ thị có nhiều đỉnh, ít cạnh, ta có thể sử dụng danh sách kề kèm trọng số để biểu diễn đồ thị, tuy nhiên tốc độ của thuật toán DIJKSTRA vẫn khá chậm vì trong trường hợp xấu nhất, nó cần n lần cố định nhãn và mỗi lần tìm đỉnh để cố định nhãn sẽ mất một đoạn chương trình với độ phức tạp O(n). Để tăng tốc độ, người ta thường sử dụng cấu trúc dữ liệu Heap để lưu các đỉnh chưa cố định nhãn. Heap ở đây là một cây nhị phân hoàn chỉnh thoả mãn: Nếu u là đỉnh lưu ở nút cha và v là đỉnh lưu ở nút con thì d[u] ≤ d[v].

(Đỉnh r lưu ở gốc Heap là đỉnh có d[r] nhỏ nhất).

Tại mỗi bước lặp của thuật toán Dijkstra có hai thao tác: Tìm đỉnh cố định nhãn và Sửa nhãn.

• Thao tác tìm đỉnh cố định nhãn sẽ lấy đỉnh lưu ở gốc Heap, cố định nhãn, đưa phần tử cuối Heap vào thế chỗ và thực hiện việc vun đống (Adjust)

• Thao tác sửa nhãn, sẽ duyệt danh sách kề của đỉnh vừa cố định nhãn và sửa nhãn những đỉnh tự do kề với đỉnh này, mỗi lần sửa nhãn một đỉnh nào đó, ta xác định đỉnh này nằm ở đâu trong Heap và thực hiện việc chuyển đỉnh đó lên (UpHeap) phía gốc Heap nếu cần để bảo toàn cấu trúc Heap.

Cài đặt dưới đây có Input/Output giống như trên nhưng có thể thực hiện trên đồ thị 5000 đỉnh, 10000 cạnh, trọng số mỗi cạnh ≤ 10000.

**CHƯƠNG II: PHÂN TÍCH, THIẾT KẾ**

1, Phân tích dữ liệu, thuật toán:

Dữ liệu vào: File văn bản DIJKSTRA.INP

+ Dòng đầu tiên chứa số đỉnh 𝑛, số cung 𝑚 của đồ thị và 2 đỉnh 𝑥, 𝑦.

𝑚 dòng tiếp theo, mỗi dòng chứa 3 số 𝑢, 𝑣, 𝑤 mô tả cung (𝑢, 𝑣) có trọng số 𝑤.

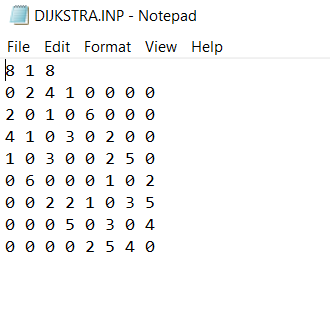
Dữ liệu ra: File văn bản DIJKSTRA.OUT

+ Dòng đầu tiên ghi số đỉnh đường đi ngắn nhất đi qua (tính luôn hai đỉnh 𝑥 và 𝑦) và độ dài đường đi.

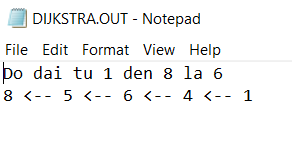
+ Dòng thứ hai ghi danh sách các đỉnh của đường đi ngắn nhất tìm được từ 𝑠 đến 𝑡.

Ví dụ:

Đầu vào:



Đầu ra:



2, Mô tả thuật toán và ý tưởng:

+ **Mô tả thuật toán:** Cho đồ thị vô hướng G = (V, E), tìm đường đi ngắn nhất từ điểm 1 đến điểm 8.

+ **Ý tưởng thuật toán:** sử dụng thuật toán Dijkstra

- Nhập dữ liệu đầu vào và đầu ra theo đề tài giảng viên giao.

- Đồ thị cho trong tập tin DIJKSTRA.INP.

- Dòng đầu tiên chứa số đỉnh n, và 2 đỉnh x,y

- n dòng tiếp theo, mỗi dòng chứa n số. Trong đó

Tìm đường đi ngắn nhất sử dụng ma trận kề, ma trận kề có n đỉnh là ma trận vuông G có số hàng số cột là n. G[i][j] là độ dài đường đi từ đỉnh i tới đỉnh j. Ở đây ta xét đồ thị vô hướng thì G[i][j] = G[j][i]. Độ dài từ một đỉnh tới chính nó luôn là 0 (G[i][i] = 0). Nếu giữa 2 cạnh i và j của đồ thị không tồn tại đường đi thì G[i][j] = vô cùng (∞). Tuy nhiên khi biểu diễn trong máy tính thì giá trị ∞ được đặt là 1 hằng số rất lớn hoặc là tổng các giá trị trong ma trận (tổng độ dài các cạnh).

- Các dòng được hiển thị như bảng bên dưới:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 0 | 2 | 4 | 1 | 0 | 0 | 0 | 0 |
| 2 | 2 | 0 | 1 | 0 | 6 | 0 | 0 | 0 |
| 3 | 4 | 1 | 0 | 3 | 0 | 2 | 0 | 0 |
| 4 | 1 | 0 | 3 | 0 | 0 | 2 | 5 | 0 |
| 5 | 0 | 6 | 0 | 0 | 0 | 1 | 0 | 2 |
| 6 | 0 | 0 | 2 | 2 | 1 | 0 | 3 | 5 |
| 7 | 0 | 0 | 0 | 5 | 0 | 3 | 0 | 4 |
| 8 | 0 | 0 | 0 | 0 | 2 | 5 | 4 | 0 |

4

1

3

1

6

2

2

2

3

5

5

4

1

2

Đầu tiên:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

+ Ta sẽ xuất phát từ đỉnh 0, nên giá trị tại 1 là 0 và các đỉnh còn lại là vô cùng

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | . | 2 (1,2) | 4 (1,3) | 1 (1,4) | ∞ | ∞ | ∞ | ∞ |

+ Bắt đầu từ đỉnh 1 ta thấy từ 1 có đường đi tới 3 đỉnh là 2, 3, 4. Xét trọng số (1,2) =2; (1,3) = 4; (1,4) = 3. Ta điền vào bảng, những đỉnh còn lại không có đường đi thì giữ Nguyên vô cùng. Từ bước này ta sẽ chọn đỉnh tiếp theo để đi, chọn đỉnh có giá trị trọng số nhỏ nhất ở từng bước. Ta xét đỉnh 4 vì có trọng số nhỏ nhất từ đỉnh 1 tới đỉnh 4 là 1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | . | 2 (1,2) | 4 (4,3) | . | ∞ | 3 (4,6) | 6 (4,7) | ∞ |

+ Tiếp theo đến đỉnh 4, ta thấy từ đỉnh 4 có đường đi tới 3 đỉnh là 3, 6, 7 có các trọng số lần lượt là 3, 2, 5 cộng với 1 (1,4) thành 4, 3, 6. Ta thấy giá trị trọng số nhỏ nhất ở đây là 3 (4,6), ta xét đỉnh tiếp theo là 6.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 3 | . | 2 (1,2) | 5 (6,3) | . | 4 (6,5) | . | 6 (6,7) | 8 (6,8) |

+ Tiếp là đỉnh 6, ta thấy từ đỉnh 6 có đường đi tới 4 đỉnh là 3, 5, 7, 8 có các trọng số lần lượt là 5, 4, 6, 8.Ta thấy giá trị trọng số nhỏ nhất ở đây là 4 (6,5), ta xét đỉnh tiếp theo là 5.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 4 | . | 10(5,2) | 5 (6,3) | . | . | . | 6 (6,7) | 6 (5,8) |

+ Cuối cùng là đỉnh 5, ta thấy từ đỉnh 5 có đường đi tới 3 đỉnh là 2, 8 có các trọng số lần lượt là 10, 6. Ta thấy giá trị trọng số nhỏ nhất ở đây là 6 (5,8). Ta kết thúc thuật toán tìm đường đi ngắn nhất từ đỉnh 1 tới đỉnh 8.