CHƯƠNG 5

Đường đi ngắn nhất

Đường đi ngắn nhất

- Bài toán tìm đường đi
- Các thuật toán tìm đường đi ngắn nhất
  - Thuật toán Dijkstra
  - Thuật toán Ford-Bellman
  - Thuật toán Floyd

Bài toán tìm đường đi

- ***Bài toán***: Cho đồ thị G = (*V, E*) và hai đỉnh *a, b*. Tìm đường đi ngắn nhất (*nếu có*) đi từ đỉnh *a* đến đỉnh *b* trong đồ thị G.
- **Ý nghĩa thực tế**:

  Bài toán giúp chúng ta chọn các hành trình tiết kiệm nhất (quãng đường, thời gian, chi phí ...) trong giao thông, lập lịch thi công công trình một cách tối ưu, xử lý trong truyền tin ...

Bài toán tìm đường đi (tt)

- Xét mô hình hệ thống đường hàng không:
  - Thành phố → đỉnh
  - Chuyến bay → cạnh
  - Trọng số: khoảng cách giữa các TP, thời gian của chuyến bay, tiền vé
- Xét mô hình mạng máy tính:
  - Máy tính → đỉnh
  - Đường truyền → cạnh
  - Trọng số: khoảng cách giữa các MT, thời gian đáp ứng qua mạng của máy.

Bài toán tìm đường đi (tt)

- Xác định đường đi ngắn nhất giữa 2 đỉnh của đồ thị là xác định tổng trọng số của các cạnh trên đường đi.

Thuật toán Dijkstra

- Do E.Dijkstra người Hà Lan đề xuất
- **Bài toán**: Cho đồ thị có trọng số không âm. Tìm đường đi (sơ cấp) ngắn nhất từ đỉnh s đến các đỉnh còn lại.

Thuật toán Dijkstra (tt)

- **Tổ chức dữ liệu**:
  - a[n][n]: ma trận trọng số.
  - d[]: lưu khoảng cách từ đỉnh s đến các đỉnh còn lại.
  - chuaxet[]: đánh dấu các đỉnh đã có độ dài tối ưu từ đỉnh bắt đầu s hay chưa.
  - truoc[]: nếu truoc[i]=j thì đỉnh j là đỉnh ngay trước đỉnh i trên hành trình đường đi ngắn nhất.

Thuật toán Dijkstra (tt)

- **Thuật toán**

B1: Khởi tạo

  - d[i]=a[s][i]
  - Truoc[i]=s
  - Chuaxet[v]=false

  } với i=1,n

  - Gán d[s]=0, chuaxet[s]=true

Thuật toán Dijkstra (tt)

B2. Vòng lặp

- *Tìm đỉnh u chưa xét và có d[u] nhỏ nhất.*
- *Nếu không tìm được đỉnh u thoả đk (hoặc u==y là đỉnh đích của đường đi) → thoát vòng lặp. Ngược lại:*
  - *đánh dấu chuaxet[u]=true*
  - *sửa lại giá trị lưu trong mảng d cho các đỉnh v kề với đỉnh u vừa tìm được theo công thức:*

    nếu ***d[v] > d[u]+a[u][v]*** *thì:*

    d[v]=d[u]+a[u][v]

    truoc[v]=u

Thuật toán Dijkstra (tt)

B3. Tìm và ghi kết quả

- D[i] chính là độ dài đường đi ngắn nhất từ đỉnh s đến *i*. Lần ngược theo mảng *truoc* để in ra kết quả đường đi.

Thuật toán Dijkstra (tt)

- VD

Thuật toán Ford-Bellman

- **Bài toán**: Cho đồ thị có trọng số, không có chu trình âm.
- Tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại của đồ thị.

Thuật toán Ford-Bellman (tt)

- B1: Khởi tạo
- B2: Sửa nhãn khoảng cách d[]
  - Thực hiện n-2 lần sửa nhãn cho các đỉnh. Trong mỗi lần sửa, nếu (i,j) là một cạnh/cung của đồ thị và d[j]>d[i]+a[i][j] thì sửa lại d[j]=d[i]+a[i][j]
  - Sau n-2 lần sửa nhãn khoảng cách thì d[i] chính là độ dài đường đi ngắn nhất từ đỉnh s đến i.
- B3: Tìm và ghi kết quả
- Ví dụ: giáo trình tr103

Thuật toán Ford-Bellman (tt)

- VD

Thuật toán Floyd

- Tổ chức dữ liệu và khởi tạo
  - A[n][n]: ma trận trọng số
    - A_{ij}= 0 nếu i = j
    - A_{ij} = a[i,j] nếu (i,j) $\in$ E
    - A_{ij} = ∞ nếu (i,j) $\notin$ E
  - P[n][n]: ma trận các đỉnh trước
    - P_{ij} = i, trong đó P_{ij} là đỉnh trước của đỉnh j trên đường đi từ i đến j

Thuật toán Floyd (tt)

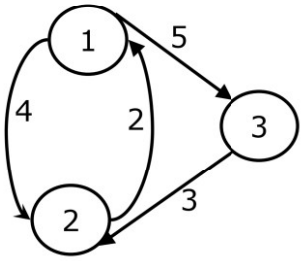
- Vòng lặp: thực hiện n lần....

```
for (k =1;  k≤ n ; k++)
    for (i =1 ;i≤ n ; i++)
        for (j =1 ;j≤ n ; j++)
            if (a[i,k] + a[k,j]  <  a[i,j])
            {   a[i,j] = a[i,k] + a[k,j];
                p[i,j] =p[k,j];
            }
```

- Kết thúc thuật toán:
  - P_{ij} = đỉnh trước của j trên đường đi ngắn nhất từ i đến đỉnh j, với chiều dài tương ứng là A_{ij}.

Thuật toán Floyd (tt)

- Ví dụ:



A_0 =

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	3	0

P_0 =

	1	2	3
1	1	1	1
2	2	2	2
3	3	3	3

Thuật toán Floyd (tt)

A_0 =

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	3	0

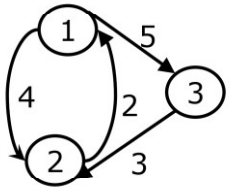
k = 1

A_1 =

	1	2	3
1	0	4	5
2	2	0	7
3	∞	3	0

P_1 =

	1	2	3
1	1	1	1
2	2	2	1
3	3	3	3



Thuật toán Floyd (tt)

A_1 =

	1	2	3
1	0	4	5
2	2	0	7
3	∞	3	0

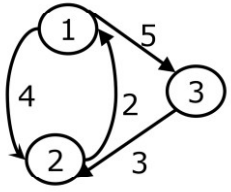
k = 2

A_2 =

	1	2	3
1	0	4	5
2	2	0	7
3	5	3	0

P_2 =

	1	2	3
1	1	1	1
2	2	2	1
3	2	3	3



Thuật toán Floyd (tt)

A_2 =

	1	2	3
1	0	4	5
2	2	0	7
3	5	3	0

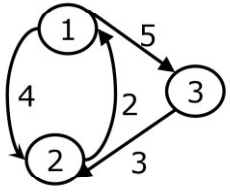
k = 3

A_3 =

	1	2	3
1	0	4	5
2	2	0	7
3	5	3	0

P_3 =

	1	2	3
1	1	1	1
2	2	2	1
3	2	3	3



Thuật toán Floyd (tt)

- Đường đi ngắn nhất từ i đến j:
  - A_{ij} cho biết độ dài đường đi
  - Dòng thứ i của ma trận P cho biết đường đi
- VD: đđnn từ 2 → 3 ?
  - P[2,3]=1: 1 là đỉnh trước của 3
  - P[2,1]=2 : 2 là đỉnh trước của 1
  - Kết quả đđnn là: 2→1→3

$A_3 =$

	1	2	3
1	0	4	5
2	2	0	7
3	5	3	0

$P_3 =$

	1	2	3
1	1	1	1
2	2	2	1
3	2	3	3

Câu hỏi???