طراحی الگوریتم

۵ آذر ۹۸

ملکی مجد

Topic	Reference
Recursion and Backtracking	Ch.1 and Ch.2 JeffE
Dynamic Programming	Ch.3 JeffE and Ch.15 CLRS
Greedy Algorithms	Ch.4 JeffE and Ch.16 CLRS
Amortized Analysis	Ch.17 CLRS
Elementary Graph algorithms	Ch.6 JeffE and Ch.22 CLRS
Minimum Spanning Trees	Ch.7 JeffE and Ch.23 CLRS
Single-Source Shortest Paths	Ch.8 JeffE and Ch.24 CLRS
All-Pairs Shortest Paths	Ch.9 JeffE and Ch.25 CLRS
Maximum Flow	Ch.10 JeffE and Ch.26 CLRS
String Matching	Ch.32 CLRS
NP-Completeness	Ch.12 JeffE and Ch.34 CLRS

All-Pairs Shortest Paths

the problem of finding shortest paths between all pairs of vertices in a graph.

The Floyd-Warshall algorithm

Use dynamic-programming

negative-weight edges may be present,

but we assume that there are no negative-weight cycles

The structure of a shortest path

definition + assumption

- an ***intermediate*** vertex of a simple path *p* is any vertex of *p* other than *start* and *destination*

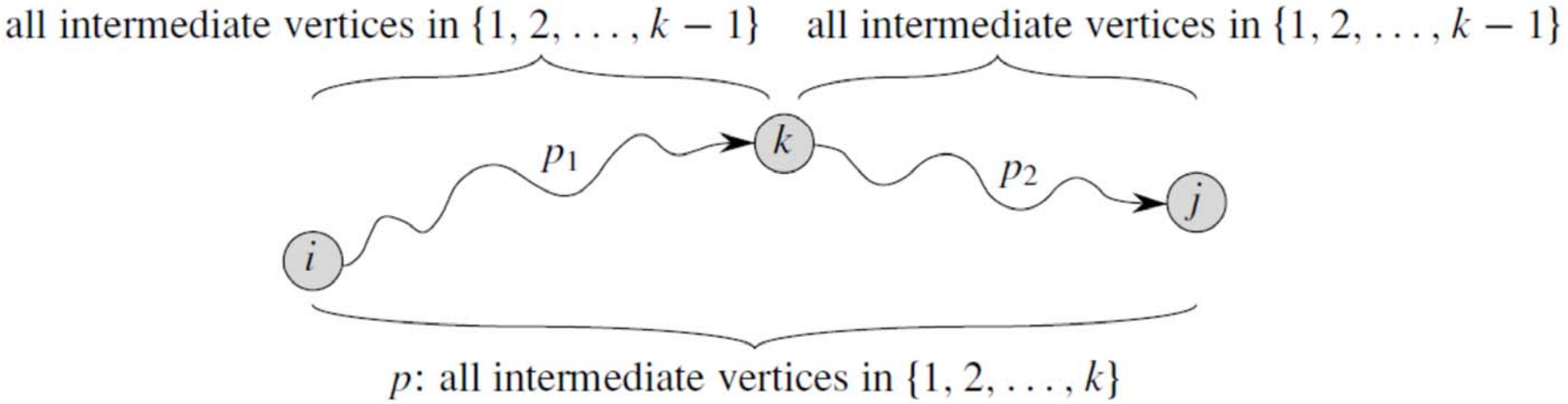
- For any pair of vertices $i, j \in V$, consider all paths from i to j whose **intermediate vertices are all drawn from {1, 2, . . . , *k*},** and let p be a minimum-weight path from among them.

- **whether or not** *k* is an intermediate vertex of path *p*
  - *Yes*
  - *No*

The structure of a shortest path relationship

- If *k* **is not** an intermediate vertex of path *p*
  - all intermediate vertices of path *p* are in the set $\{1, 2, \ldots, k - 1\}$.

- If *k* **is** an intermediate vertex of path *p*
  - we break *p* down into paths p1 from *i to k and path p*2 from k to *j*
  - *p1* is a shortest path from *i* to *k* with all intermediate vertices in the set $\{1, 2, \ldots, k - 1\}$.
  - *p*2 is a shortest path from vertex *k* to vertex *j* with all intermediate vertices in the set $\{1, 2, \ldots, k - 1\}$.


all intermediate vertices in {1, 2, . . . , k − 1}
all intermediate vertices in {1, 2, . . . , k − 1}
k
p1
p2
i
j
p: all intermediate vertices in {1, 2, . . . , k}

A recursive solution to the all-pairs shortest-paths problem

$$d_{ij}^{(0)} = w_{ij}$$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 , \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 . \end{cases}$$

$$d_{ij}^{(n)} = \delta(i, j)$$

Computing the shortest-path weights bottom up

```
FLOYD-WARSHALL(W)
1  n ← rows[W]
2  D^(0) ← W
3  for k ← 1 to n
4       do for i ← 1 to n
5            do for j ← 1 to n
6                 do d_ij^(k) ← min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
7  return D^(n)
```

Example
let's solve togerther!

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

result

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Time complexity

- the algorithm runs in time $\Theta(n3)$
- code is tight, with no elaborate data structures, and so the constant hidden in the Θ-notation is small.
- the Floyd-Warshall algorithm is quite practical for even moderate-sized input graphs.

Constructing a shortest path

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty \,, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \,. \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \,, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \,. \end{cases}$$

Transitive closure of a directed graph

- Given a directed graph $G = (V, E)$ with vertex set $V = \{1, 2, \ldots, n\}$, we may wish to find out whether there is a path in G from i to j for all vertex pairs $i, j \in V$.

- The ***transitive closure*** of G*:*
  - *The edge* (i, j) *means that* there is a path from vertex i to vertex j in G

Transitive closure of a directed graph

A recursive definition

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i, j) \notin E \ , \\ 1 & \text{if } i = j \text{ or } (i, j) \in E \ , \end{cases}$$

and for $k \geq 1$,

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)} \right) \ .$$

TRANSITIVE-CLOSURE(G)

```
1  n ← |V[G]|
2  for i ← 1 to n
3       do for j ← 1 to n
4            do if i = j or (i, j) ∈ E[G]
5                 then t_ij^(0) ← 1
6                 else t_ij^(0) ← 0
7  for k ← 1 to n
8       do for i ← 1 to n
9            do for j ← 1 to n
10                do t_ij^(k) ← t_ij^(k−1) ∨ (t_ik^(k−1) ∧ t_kj^(k−1))
11 return T^(n)
```

- How can the output of the Floyd-Warshall algorithm be used to detect the presence of a negative-weight cycle?

- Give an $O(V\ E)$-time algorithm for computing the transitive closure of a directed graph $G = (V, E)$.