طراحی الگوریتم

۳ 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.

Problem

we are given **a weighted**, **directed** graph *G = (V, E)*

with a weight function $\boldsymbol{w} : \boldsymbol{E} \rightarrow \boldsymbol{R}$ that maps edges to real-valued weights.

We wish to find, for **every pair** of vertices $u, v \in V$, **a shortest (least-weight) path** from *u* to *v*, where the weight of a path is the sum of the weights of its constituent edges.

We typically want the **output in tabular** form:

the entry in u's row and v's column should be the weight of a shortest path from u to v.

Solve by SSP
(Bellman-Ford an Dijkstra's algorithm)

We can solve an all-pairs shortest-paths problem **by running a single-source shortest-paths algorithm |*V*| times**, once for each vertex as the source.

- If all edge weights are nonnegative
  - we can use Dijkstra's algorithm.
    - min-priority queue : the running time is $O(V^3 + V\ E) = O(V^3)$.
    - binary min-heap : the running time of $O(V\ E\ \lg V)$,
    - Fibonacci heap : the running time of $O(V^2 \lg V + V\ E)$.
- If negative-weight edges are allowed
  - we must run the slower Bellman-Ford algorithm
    - The resulting running time is $O(V^2 E)$,

- Unlike the single-source algorithms, which assume an adjacency-list representation of the graph, most of the algorithms in this topic (All-Pairs Shortest Paths) use an **adjacency-matrix** representation.

- (Johnson's algorithm for sparse graphs uses adjacency lists.)

Assumption

we assume that the **vertices are numbered $1, 2, \ldots, |V|$,** so that

the input is an $n \times n$ matrix $W = (w_{ij})$ **representing the edge weights** of an n-vertex directed graph $G = (V, E)$.

- $w_{ij} =$
  - **0** if $i = j$,
  - the **weight of directed edge** (i, j) if $i = j$ and $(i, j) \in E$,
  - ∞ if $i = j$ and $(i, j) \in E$.

Output: D and $\mathbf{\Pi}$

- The **tabular output** of the all-pairs shortest-paths algorithms presented in this chapter is **an $\boldsymbol{n} \times \boldsymbol{n}$ matrix** $\boldsymbol{D} = (d_{ij})$,
- where entry d_{ij} contains the weight of a shortest path from vertex i to vertex j .
- If we let $\boldsymbol{\delta}(\boldsymbol{i}, \boldsymbol{j})$ denote the shortest path weight from vertex i to vertex j, then

  $\boldsymbol{d_{ij}} = \boldsymbol{\delta}(\boldsymbol{i}, \boldsymbol{j})$ at termination.

- To solve the all-pairs shortest-paths problem on an input adjacency matrix, we need to compute not only the shortest-path weights but also a ***predecessor matrix*** $\Pi = (\pi_{ij})$, where
  - π_{ij} is NIL if either $i = j$ or there is no path from i to j , and otherwise
  - π_{ij} is the predecessor of j on some shortest path from i.

Print a path

PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)

if $i = j$

    then print i

    else if $\pi_{ij} = NIL$

        then print no path from i to j exists

        else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_{ij})

            print j

the all-pairs shortest-paths problem

a dynamic-programming algorithm based on matrix multiplication

the steps of a dynamic-programming algorithm

1. **Characterize the structure of an optimal solution.**
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution in a bottom-up fashion.
4. Construct an optimal solution from computed information.

The structure of a shortest path

- **All subpaths of a shortest path are shortest paths**
- Consider a shortest path p from vertex i to vertex j , and suppose that p contains at most m edges.
  - Assuming that there are no negative-weight cycles, m is finite.
- If $i = j$, then *p* has weight 0 and no edges.
- If vertices *i* and *j* are distinct, then **we decompose path p into**

$$i \overset{p'}{\rightsquigarrow} k \rightarrow j$$

- p' is a shortest path from *i* to *k*, and so $\boldsymbol{\delta(i, j) = \delta(i, k) + w_{kj}}$. (p' now contains at most $m - 1$ edges)

the steps of a dynamic-programming algorithm

1. Characterize the structure of an optimal solution.
2. **Recursively define the value of an optimal solution**.
3. Compute the value of an optimal solution in a bottom-up fashion.
4. Construct an optimal solution from computed information.

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

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j , \\ \infty & \text{if } i \neq j . \end{cases}$$

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

$$
\begin{aligned}
l_{ij}^{(m)} &= \min\left(l_{ij}^{(m-1)}, \min_{1 \le k \le n} \left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right) \\
&= \min_{1 \le k \le n} \left\{l_{ik}^{(m-1)} + w_{kj}\right\} .
\end{aligned}
$$

the steps of a dynamic-programming algorithm

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. **Compute the value of an optimal solution in a bottom-up fashion.**
4. Construct an optimal solution from computed information.

Computing the shortest-path weights bottom up

extend path

با استفاده از کوتاهترین میسرها به طول m-1، کوتاهترین مسیرها به طول m را محاسبه کنیم

```
EXTEND-SHORTEST-PATHS(L, W)
n ← rows[L]
let L′ = (l′_ij) be an n × n matrix
for i ← 1 to n
     do for j ← 1 to n
            do l′_ij ← ∞
               for k ← 1 to n
                   do l′_ij ← min(l′_ij, l_ik + w_kj)
return L′
```

Computing the shortest-path weights bottom up

similarity to matrix multiplication

```
EXTEND-SHORTEST-PATHS(L, W)
n ← rows[L]
let L' = (l'_ij) be an n × n matrix
for i ← 1 to n
    do for j ← 1 to n
        do l'_ij ← ∞
           for k ← 1 to n
               do l'_ij ← min(l'_ij, l_ik + w_kj)
return L'
```

$l^{(m-1)} \rightarrow a$,

$w \rightarrow b$,

$l^{(m)} \rightarrow c$,

$\min \rightarrow +$,

$+ \rightarrow \cdot$

extending shortest paths edge by edge

$$\begin{aligned}
L^{(1)} &= L^{(0)} \cdot W &= W \,, \\
L^{(2)} &= L^{(1)} \cdot W &= W^2 \,, \\
L^{(3)} &= L^{(2)} \cdot W &= W^3 \,, \\
&\vdots \\
L^{(n-1)} &= L^{(n-2)} \cdot W &= W^{n-1} \,.
\end{aligned}$$

All-Pairs Shortest Paths algorithm

SLOW-ALL-PAIRS-SHORTEST-PATHS(W)

```
n ← rows[W]
L^(1) ← W
for m ← 2 to n − 1
        Do L^(m) ← EXTEND-SHORTEST-PATHS(L^(m−1), W)
return L(n − 1)
```

$$\boldsymbol{Time\ complexity\ of\ computing\ L^{(n-1)}:\ \Theta(n^4)}$$

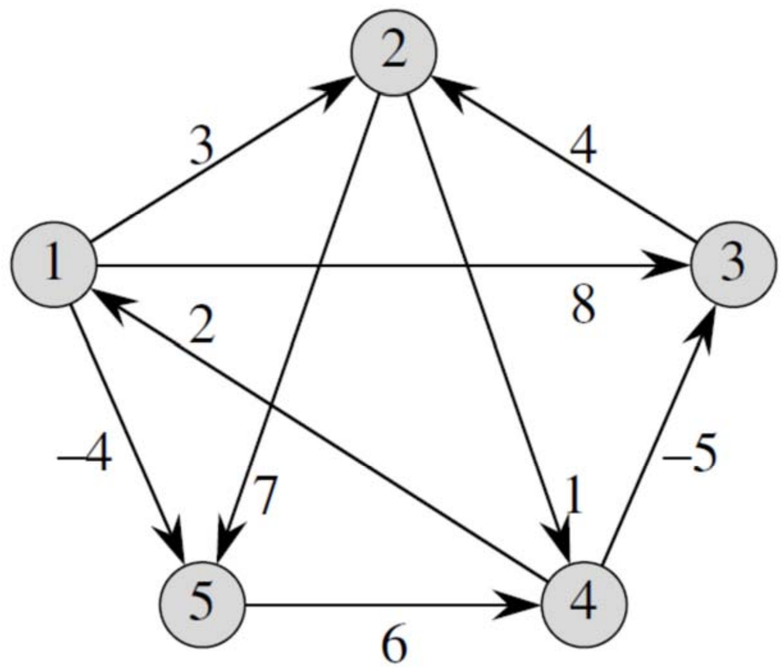
Improving the running time

$$
\begin{aligned}
L^{(1)} &= W\,, \\
L^{(2)} &= W^2 &&= W \cdot W\,, \\
L^{(4)} &= W^4 &&= W^2 \cdot W^2 \\
L^{(8)} &= W^8 &&= W^4 \cdot W^4\,, \\
&\vdots \\
L^{(2^{\lceil \lg(n-1) \rceil})} &= W^{2^{\lceil \lg(n-1) \rceil}} &&= W^{2^{\lceil \lg(n-1) \rceil - 1}} \cdot W^{2^{\lceil \lg(n-1) \rceil - 1}}\,.
\end{aligned}
$$

$\boldsymbol{\Theta(n^3 \lg n)}$ algorithm with technique of ***repeated squaring***.

```
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
n ← rows[W]
L^(1) ← W
m ← 1
while m < n − 1
      do L^(2m) ← EXTEND-SHORTEST-PATHS(L^(m), L^(m))
            m ← 2m
return L^(m)
```

example



$$
L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}
$$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Sample problem:

- Modify FASTER-ALL-PAIRS-SHORTEST-PATHS so that it can detect the presence of a negative-weight cycle.

- Give an efficient algorithm to find the length (number of edges) of a minimum length negative-weight cycle in a graph.