Fundamental Algorithms
Lecture #12

Cluj-Napoca

December 17, 2018

Agenda

- Graphs
  - Single Source Shortest Path Bellman-Ford
  - Single Source Shortest Path on dags
  - All pairs Shortest Path
  - Update All pairs Shortest Path (Floyd-Warshall)
  - Transitive Closure
  - Maximum Flow (or not? Next time?)

Single Source Shortest Path

- Dijkstra’s alg. works for nonnegative costs only (Hw: show why is not applicable when negative costs are present)
- If negative costs are allowed, Bellman-Ford method is employed
- It still has limitations: no negative cycles allowed (why? Further discussion.)
- the alg. is using the relaxation technique (same as Dijkstra’s; same as Prim’s, although was not denoted as relaxation)

SSSP - Bellman-Ford alg.

```
Bellman-Ford (G, w, s)
initialize_single_source (G, s)
for i<- 1 to |V| - 1
  do for each edge (u, v) ∈ E(G)
         do relax (u, v, w)
for each edge (u, v) ∈ E(G)
     do  if d[v] > d[u] + w(u, v)
         then return false
return true
```

SSSP - Bellman-Ford alg. analysis

- The algorithm returns false in case the graph has a negative cycle. Why/how?
  - Infinite updates on negative loops are possible
  - Returns false (as potential updates are captured in the last for loop)
- Efficiency: O(VE) (the first for loop)

SSSP in dags

dag = directed acyclic graph

```
dag_shortest_path (G, w, s)
topologically sort the vertices of G
//check seminar #6 or #7 for solutions
initialize_single_source (G, s)
for each vertex u    //considered in their
                     //topological order
  do for v ∈ Adj[u]
     do relax (u, v, w)
```

SSSP in dags - analysis

- Topo sort O(V+E) (see seminar)
- The 2 for loops suggest $O(V^2)$
- Why is false?
  - Since every edge is considered exactly once, it is O(E)
  - Therefore, $O(V+E)<O(V^2)$ (straightforward version for Dijktra's)

All pairs Shortest Path

- Dynamic programming technique
- Based on the optimal structure property

If a *solution* to some *problem* is *optimal*, the *solution* of any *subproblem* the original problem contains *MUST* be *optimal* . *Same as SSSP* (oral explanation).

The converse statement is **NOT** true! (Statement p->q; converse q->p)

- So:
  - if δ (s, t) is optimal, then δ(s, v) and δ(v, t) are optimal, for whatever v in the path from s to t
  - converse **NOT** true! that is: if δ(s, v) and δ(v, t) are optimal then it does NOT follow that δ(s, t) is optimal, (Proof by contradiction - the same way as for the shortest path with greedy)
- Therefore, we have to calculate every possible path

Floyd-Warshall – the method

- Denote d_{ij}^{k} = the cost of the min path from i to j, containing as intermediate nodes ONLY nodes from the set: {1, 2, …, k}

  Considering the nodes in a given order does NOT represent a limitation. Why?
- k = 0 => no intermediate node is considered (so, it is an edge, if the “path” exists)
- $k>0$, $d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$
- Justification (blackboard)
- (it is either the former path, or a new one, passing through the new considered node, k)

Floyd-Warshall – the method – contd.

$$d_{ij}^{k} = \begin{cases} w_{ij} & \text{for } k=0 \\ \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) & \text{for } k>0 \end{cases}$$

- It is a dynamic programming pattern
- $d_{ij}^{n} = \delta(i, j)$ (i. e. min distance between i and j, considering all possible n nodes as intermediary nodes)

Floyd-Warshall – the alg.

```
Floyd-Warshall (w)
n<-|V|
D^0<-w
for k<-1 to n
  do for i<-1 to n
       do for j<-1 to n
            do d_ij^k =min(d_ij^k-1,d_ik^k-1+d_kj^k-1)
return D^n
```

$D^0 \leftarrow w$; $d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$; return D^n

Floyd-Warshall – the alg. analysis

- $O(V^3)$
- What about the memory required?
- How can be decreased the necessary amount of memory?
- Where is the actual path? Can we obtain it?

$$\pi_{ij}^{0} = \begin{cases} nil & \text{if } i=j \text{ or } w_{ij} = \infty \quad \text{(i.e. there is no edge between i, j)} \\ i & \text{if } w_{ij} < \infty \quad \text{(i.e. there is edge between i, j)} \end{cases}$$

$$\pi_{ij}^{k} = \begin{cases} \pi_{ij}^{k-1} & \text{if } d_{ij}^{k-1} <= d_{ik}^{k-1} + d_{kj}^{k-1} \\ \pi_{kj}^{k-1} & \text{if } d_{ij}^{k-1} > d_{ik}^{k-1} + d_{kj}^{k-1} \end{cases}$$

Floyd-Warshall – the alg.

```
Modified Floyd-Warshall (w)
n<-|V|
D^0<-w
initialize π
for k<-1 to n
   do for i<-1 to n
         do for j<-1 to n
               do d_ij^k =min(d_ij^(k-1),d_ik^(k-1)+d_kj^(k-1))
               if d_ij^(k-1) <= d_ik^(k-1) + d_kj^(k-1)
               then π_ij^k = π_ij^(k-1)
               else π_ij^k = π_kj^(k-1)
return D^n; π^n
```

Trace the alg (blackboard)

Transitive Closure

- Sol. #1: run Floyd-Warshall and check

  $d_{ij}^{k} \mathrel{<>} \infty$

- Means of decreasing exe time?

  Change arithmetic into logical operations

- define

$$t_{ij}^{0} = \begin{cases} 0 & \text{if } i=j \text{ or there is no edge between } i, j \\ 1 & \text{otherwhise} \end{cases}$$

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

Transitive Closure – the alg.

Transitive_closure(G)

n<-|V|

<u>for</u> i<-1 to n

  <u>do</u> <u>for</u> j<-1 to n

    <u>if</u> i=j or $(i, j) \in E$

    <u>then</u> t_{ij}^{0} <-1

    <u>else</u> t_{ij}^{0} <-0

<u>for</u> k<-1 to n

  <u>do</u> <u>for</u> i<-1 to n

    <u>do</u> <u>for</u> j<-1 to n

      <u>do</u> $t_{ij}^{k} = t_{ij}^{k-1} \vee (t_{ik}^{k-1} \wedge t_{kj}^{k-1})$

<u>return</u> T^{n}