cs 5800 - hw11

Yijing Xiao

December 2021

1 ASSP-FAST

Let $l_{i,j}^{(m)}$ be the minimum weight of any path from vertex i to vertex j that contains at most m edges. When $m = 0$, there is a shortest path from i to j with no edges if and only if $i = j$. Thus,

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

For $m \leq 1$, we compute $l_{ij}^{(m)}$ as the minimum of $l_{ij}^{(m-1)}$(the weight of a shortest path from i to j consisting of at most m edges. obtained by looking at all possible predecessors k of j. Thus, we recursively define

$$l_{ij}^{(m)} = \min(l_{ij}^{(m-1)}, \min_{1\leq k\leq n}\{l_{ik}^{(m-1)} + w_{kj}\}) = \min_{1\leq k\leq n}(l_{ik}^{(m-1)} + w_{kj})$$

The latter equality follows since $w_{jj} = 0$ for all j.

EXTEND-SP: Taking as our input the matrix $W = (w_{ij})$, we now computed a series of matrices $L^{(1)}, L^{(2)}, ..., L^{(n-1)}$, where for $m = 1, 2, ..., n-1$, we have $L^{(m)} = (l_{ij}^{(m)}$. The final matrix $L^{(n-1)}$ contains the actual shortest-path weights. Observe that $l_{ij}^{(1)} = w_{ij}$ for all vertices $i, j \in V$, and so $L^{(1)} = w$. The heart of the algorithm is, given matrices $L^{(m-1)}$ and w, returns the matrix $L^{(m)}$. That is, it extends the shortest paths computed so far by one more edges.

ASSP-FAST: In each iteration of the while loop, we compute $L^{(2m)} = (L^{(m)})^2$, starting with $m = 1$. At the end of each iteration, we double the value of m. The final iteration computes $L^{(n-1)}$ by actually computing $L^{(2m)}$ for some $n - 1 \leq 2m < 2n - 2$. By equation $\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = ...$, so $L^{(2m)} = L^{(n-1)}$. The next time the $m < n - 1$ performed, m has been doubled, so now $m \geq n - 1$, and the procedure returns the last matrix it computed.

Algorithm 1

```
function EXTEND-SP(L, w)
    n = L.rows
```

Let $L' = (l'_{ij})$ be a new $n \times n$ matrix
for $i = 1$ to n **do**
    for $j = 1$ to n **do**
        $l'_{ij} = \infty$
        for $k = 1$ to n **do**
            $l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$
        end for
    end for
end for
return L'
end function

function ASSP-FAST(w)
    $n = w.rows$
    $L^{(1)} = w$
    $m = 1$
    while $m < n - 1$ **do**
        let$L^{(2m)}$ be a new $n \times n$ matrix
        $L^{(2m)} =$ EXTEND-SP$(L^{(m)}, L^{(m)})$
        $m = 2m$
    end while
    return $L^{(m)}$
end function

---

2 Exercise 24.1-3

Algorithm 2

function INITIALIZE-SINGLE-SOURCE(G, s)
    for each vertex $v \in G.V$ **do**
        $v.d = \infty$
        $v.\pi = NIL$
    end for
    $s.d = 0$
end function

function RELAX(u, v, w)
    if $v.d > u.d + w(u, v)$ **then**
        $v.d = u.d + w(u, v)$
        $v.\pi = u$
    end if
end function

function BELLMAN-FORD(G, w, s) INITIAL-SINGLE-SOURCE(G, s)
    for $i = 1$ to $|G.V| - 1$ **do**

```
        for each v in G.V do
            T = v.d
        end for
        for each edge(u, v) ∈ G.E do
            Relax(u, v, w)
        end for
        for each edge(u, v) ∈ G.E do
            if v.d ≠ T[].d then
                if v.d > u.d + w(u, v) then
                    return FALSE
                end if
            end if
        end for
    end for
    return TRUE
end function
```

If the value of $v.d$ at the end of the loop does not change from the value of $v.d$ at the beginning of the loop, the loop is the $(m+1)th$ passes.

3 Exercise 24.2-2

The DAG-SHORTEST-PATH algorithm starts by topologically sorting the directed acyclic graph (dag) to impose a linear ordering on the vertices. If the dag contains a path form vertex u to vertex v, then u precedes v in the topological sort. Topological sort ensures that edges are arranged in a linear fashion. If the edges are arranged in a linear fashion, and use perform relaxation then the last vertex will have its edges already visited because if there exist edge (u, v) then u appears before v in the ordering. So we can perform the looping for the first $|V| - 1$ vertices because the vertices after topological sort will be ordered.

4 Exercise 24.3-4

Dijkstra's algorithm solves the single-source shortest-path problem on a weighted, directed graph $G = (V, E)$ for the case in which all edge weights are noonegative. Therefore, $w(u, v) \leq 0$ for each edge $(u, v) \in E$. Dijkstra's algorithm maintains a set s of vertices whose final shortest-path weights from the source s have already been determined. The algorithm repeatly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, add u to S, and relaxes all edges leaving u.

Algorithm 3

```
function Check-Output(G, w, s)
    Topological-Sort(G)                    ▷ find the linear order of the vertices
    Initialize-Single-Source(G, s)
```

for each edge $(u, v) \in G.E$ **do**
  Relax(u, v, w)
end for
end function

The topological sort is used to find the vertices in the linear order of consideration. The correctness of the Professor Gaedel's procedure would rely on checking the vertex weight $v.d$ is equal to minimum of (u, v). The program terminates successfully once all the edges are relaxed. The running time of initialization would take $O(V)$ and relaxation would take $O(V + E)$. Thus, the running time of algorithm would be $O(V + E)$.

5 Transitive Closure

Given a directed graph $G = (V, E)$ with vertex set $V = \{1, 2, ..., n\}$, we might wish to determine whether G contains a path from i to j for all vertex pairs $i, j \in V$. We define the transitive closure of G as the graph $G^* = (V, E^*)$, where $E^* = \{(i, j) :$ there is a path from vertex i to vertex j in $G\}$.

6 Exercise 25.1-6

FIND-PRE-MATRIX: 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 processor of j on some shortest path form i.

PRINT-ASSP: The following procedure is a modified version of the PRINT-PATH procedure from Chapter 22, prints a shortest path from vertex i to vertex j. Give the predecessor matrix Π, the PRINT-ASSP will print the vertices on a given shortest path.

Algorithm 4

function Find-Pre-Matrix(L, w)
  $n = L.rows$
  let $\Pi = (\pi_{ij})$ be a new $n \times n$ matrix
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      for $k = 1$ to n **do**
        if $L_{ij} == l_{ik} + w_{kj}$ **then**
          $\pi_{ij} = k$
        else
          $\pi_{ij} = NIL$
        end if
      end for
    end for

```
        end for
        return Π
end function

function Print-ASSP(Π, i, j)
        if i == j then
                print i
        else
                if π_ij == NIL then
                        print "no path from 'i' to 'j' exists
                else
                        Print-ASSP(Π, i, π_ij)
                        print j
                end if
        end if
end function
```

7 Exercise 25.2-1

Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set $\{1, 2, ..., k\}$. When $k = 0$, a path from vertex i to vertex j with no intermediate vertex numbered higher than 0 has no intermediate vertices at all. Such a path has at most one edge, and hence $d_{ij}^{(0)} = w_{ij}$. Following the above discussion, we define $d_{ij}^{(k)}$ recursively by

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

Becasue for any path, all intermediate vertices are in the set $\{1, 2, ..., n\}$, the matrix $D^{(n)} = d_{ij}^{(n)}$ gives the final answer: $d_{ij}^{(n)} = \delta(i, j)$ for all $i, j \in V$.

$$D^{(0)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix} \qquad D^{(1)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix} \qquad D^{(3)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix} \quad D^{(5)} = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

$$D^{(6)} = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

8 Exercise 25.2-4

The old version for computing the values $d_{ij}^{(k)}$ is: $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$. If we use $d_{ik}^{(k)}$ rather than $d_{ik}^{(k-1)}$ in the computation, then we are using subpath from i to k with all intermediate vertices in $\{1, 2, ..., k\}$. However k cannot be an intermediate vertex on a shortest path from i to k since there are no negative-weight cycles on this shortest path. Thus, $d_{ik}^{(k)} = d_{ik}^{(k-1)}$ and also applies to $d_{kj}^{(k)} = d_{kj}^{(k-1)}$. Therefore, we could simply drops all the superscripts.