Network Flow Algorithms: Exercise Solutions

Yuchong Pan

December 19, 2020

1 Preliminaries: Shortest Path Algorithms

Exercise 1.1. Let i_k be the vertex selected at the k^{th} iteration of Dijkstra's algorithm. We prove by induction that at the beginning of the k^{th} iteration, $d(v) \in \{d(i_k), d(i_k)+1, \infty\}$ for all $v \in V$ not marked yet. At the beginning of the first iteration, s is selected, and any $v \in V$ has $d(v) = \infty$; this proves the base case.

Suppose that at the beginning of the k^{th} iteration, $d(v) \in \{d(i_k), d(i_k)+1, \infty\}$ for all $v \in V$ not marked yet. Let $(i_k, j) \in A$ such that j is not marked yet. If $d(i_k) + c(i_k, j) = d(i_k) + 1 < d(j)$, then we set $d(j) \leftarrow d(i_k) + 1$; otherwise, $d(j)$ remains the same, and hence $d(j) \in \{d(i_k), d(i_k)+1, \infty\}$. If $d(i_{k+1}) = d(i_k)$, then we are done; otherwise, $d(i_{k+1}) = d(i_k) + 1$, and $d(v) \in \{d(i_k)+1, \infty\} = \{d(i_{k+1}), \infty\}$ for all $v \in V$ not marked yet. This completes the induction step.

Now, consider the k^{th} iteration. If $d(i_k) + c(i_k, j) = d(i_k) + 1 < d(j)$ for some $(i, j) \in A$, then $d(j)$ was ∞, and we set $d(j) \leftarrow d(i_k) + 1$. Since $d(v) \in \{d(i_k), d(i_k)+1, \infty\}$ for all $v \in V$ not marked yet, then we can process j after any $v \in V$ not marked yet such that $d(v) < \infty$ is processed. Therefore, we can maintain a queue of all $v \in V$ not marked yet such that $d(v) < \infty$, and we push j to the tail of the queue if $d(j)$ is updated. The adapted algorithm is given in Algorithm 1. It is clear that Algorithm 1 runs in $O(m)$ time.

$q \leftarrow$ *new queue()*
$d(i) \leftarrow \infty$ for all $i \in V$
$p(i) \leftarrow$ **null** for all $i \in V$
$d(s) \leftarrow 0$
q.add(s)
while *not q.empty?* **do**
    $i \leftarrow$ *q.remove()*
    for $j \in V$ *such that* $(i, j) \in A$ **do**
        if $d(j) > d(i) + 1$ **then**
            $d(j) \leftarrow d(i) + 1$
            $p(j) \leftarrow i$
            q.add(j)

Algorithm 1: Adapted Dijkstra's algorithm where $c(i, j) = 1$ for all $(i, j) \in A$.

Exercise 1.2. ($\Longrightarrow$) Suppose for the sake of contradiction that there exists a negative-cost cycle C reachable from s. Let $v \in V(C)$. Let $\mathcal{P}$ be the set of s-v paths. Let $P_0 \in \mathcal{P}$. Let P' be a v-v path along C. Then P_0 appended by any copy of P' is an s-v path. Since $c(P') := \sum_{e \in E(P')} c(e) < 0$, then $\{c(P) := \sum_{e \in E(P)} c(e) : P \in \mathcal{P}\}$ is not bounded below. Hence, there are no simple shortest s-v paths.

($\Longleftarrow$) Suppose that there are no negative-cost cycles reachable from s. Let $i \in V$. Let $\mathcal{P}$ be the set of simple s-i paths. Since a simple s-i path consists of at most n distinct vertices, then $|P| \leq n! < \infty$. Let $P^* = \arg\min_{P \in \mathcal{P}} c(P) := \arg\min_{P \in \mathcal{P}} \sum_{e \in E(P)} c(e)$. Let P be a non-simple s-i path. Then P contains a cycle C. Since there are no negative-cost cycles reachable from s, then $c(C) := \sum_{e \in E(C)} c(e) \geq 0$. This implies that removing all occurrences of C from P yields a simple s-i path P' with $c(P) \geq c(P') \geq \min\{c(P) : P \in \mathcal{P}\} = c(P^*)$. Hence, $c(P^*) \leq c(P)$ for any s-i path P, regardless of whether P is simple or not. This shows that P^* is the shortest s-i path, and P^* is simple.

Exercise 1.3. (a) Let $G = (V, A)$ be a DAG. Suppose for the sake of contradiction that any $v \in V$ has at least an arc directed into it. Let $v_0 \in V$. Starting from v_0, we form a path backwards by following an edge directed into the vertices. By the pigeonhole principle, this forms a path with repeated vertices, say $\{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)\}$, where $v_k = v_j$ for some $j \in \{0, \ldots, k-2\}$. Then $\{(v_j, v_{j+1}), (v_{j+1}, v_{j+2}), \ldots, (v_{k-1}, v_k)\}$ is a directed cycle, a contradiction.

(b) Let $G = (V, A)$. Let $c : A \to \mathbb{R}$ be the edge costs. We give Algorithm 2.

```
q ← new queue()
d(i) ← ∞ for all i ∈ V
p(i) ← null for all i ∈ V
d(s) ← 0
q.add(s)
while not q.empty? do
    i ← q.remove()
    for j ∈ V such that (i, j) ∈ A do
        if d(j) > d(i) + c(i, j) then
            d(j) ← d(i) + c(i, j)
            p(j) ← i
        if not q.contains(j) then
            q.add(j)
```

Algorithm 2: `DAGShortest(`G`, `c`, `s`)` for finding the shortest s-i path for each $i \in V$ in a DAG G.

Since G is a DAG, then G does not contain negative-cost cycles. By Exercise 1.2, there are simple shortest paths from s to each $i \in V$. We will prove that Algorithm 2 determines the length $d(i)$ of the shortest s-i path for all $i \in V$ by induction on the "passes." Pass 0 ends after s is added to the queue, and pass k ends after any $i \in V$ such that the shortest s-i path uses at most k edges. The base case is trivial since the only s-s path is of length 0.

Suppose that after pass k for some k, $d(i)$ is the length of the shortest s-i path for any $i \in V$ such that the shortest s-i path uses at most k edges. Let $i \in V$ such that the shortest s-i path P uses $k+1$ edges. Let (j, i) be the last edge on P. Let P' be the subpath of P up to j. Then P' is the shortest s-j path. By the induction hypothesis, $c(P') = d(j)$. Therefore, $d(i)$ is set to $d(j) + c(j, i) = c(P') + c(j, i) = c(P)$ when Algorithm 2 processes j. This proves the claim.

(c) Let $G = (V, A)$. Let $c : A \to \mathbb{R}$ be the edge costs. We give Algorithm 3. Let $i \in V$. Note that $\max\{c(P) : P \text{ is an } s\text{-}i \text{ path}\} = -\min\{-c(P) : P \text{ is an } s\text{-}i \text{ path}\}$.

$(d', p) \leftarrow$ `DAGShortest(`$G, -c, s$`)`
$d(i) \leftarrow -d'(i)$ for all $i \in V$

Algorithm 3: `DAGLongest(`G`, `c`, `s`)` for finding the longest s-i path for each $i \in V$ in a DAG G.