CS 577: HW 3

Daniel Ko

Summer 2020

§1 Kleinberg Chapter 6, Q14

§1.1 Suppose it is possible to choose a single path P that is an $s-t$ path in each of the graphs $G_0, G_1, \cdots, G_b$. Give a polynomial-time algorithm to find the shortest such path.

§1.1.1 Set up the recursive formula and justify its correctness.

We define the set of edges that exist in all points in time as

$$E^* = \bigcap_{i=0}^{b} E_i$$

Since our graph is unweighted, we can perform breadth first search for the shortest path from s to t on $G^* = (V, E^*)$. We define $\delta(v, t)$ to be the length of the shortest path from v to t.

$$\delta(v,t) = \begin{cases} 1 & \textbf{when } (v,t) \in E^* \\ \infty & \textbf{when } v \text{ has been visited} \\ \min(\{1 + \delta(\phi, t) \mid (v, \phi) \in E^*\}) & \textbf{otherwise} \end{cases}$$

§1.1.2 Write the pseudocode for the iterative version of the algorithm to find the minimum cost. You are not required to write pseudocode to find the shortest path.

§1.1.3 Analyze the computing complexity.

We claim computing complexity of *asdasd* is $O(V + E)$.

§1.2 Give a polynomial-time algorithm to find a sequence of paths $P_0, P_1, \cdots, P_b$ of minimum cost, where P_i is an $s-t$ path in G_i for $i = 0, 1, \cdots, b$.

§1.2.1 Set up the recursive formula and justify its correctness.

$$\text{OPT}(i) = \min()$$

§2 Given a rooted tree $T = (V, E)$ and an integer k, find the largest possible number of disjoint paths in T, where each path has length k.

§2.1 Set up the recursive formula and justify its correctness.

We define MaxPath(v) to be the recurrence for the maximum number of disjoint paths of size k in a sub tree of T with root v. We consider two major cases, the maximum number of disjoint paths may or may not contain v.

$$\text{MaxPath}(v) = \max\Big(\text{MaxContains}(v, 0), \text{MaxDoesNotContain}(v)\Big)$$

In the case where maximum number of disjoint paths does not contain v, we define MaxDoesNotContain(v) to be the sum of the maximum number of disjoint paths of size k for each sub tree generated by the children of v, i.e. the sub trees having c as the root where $(c, v) \in E$. We will define the set of child nodes of a vertex v as

$$C_v = \Big\{c \mid (v, c) \in E\Big\}$$

Adding up all the maximum paths for each sub tree gives us the total amount of maximum paths for the entire tree.

$$\text{MaxDoesNotContain}(v) = \sum_{c \in C_v} \text{MaxPath}(c)$$

In the case where the maximum number of disjoint paths contains r, there are a couple of subcases. We define MaxContains(v, δ) to be the maximum number of disjoint paths of size k in a sub tree of T with root v, given that v is currently a part of a path of size δ, where δ is bounded by $0 \leq \delta \leq k$. The first subcase is if v is a leaf node and the current size of the path it's on is less than k, i.e $\delta < k$, there would be 0 disjoint paths of size k in this sub tree. The second subcase is when v is the last node in a path of size k, i.e. $\delta = k$. Then, the maximum paths will be the sum of the maximum paths on the sub trees generated by the children of v, or equivalently MaxDoesNotContain(v), plus the path that v is on. The third subcase covers when v is on some path that is not complete nor trivially ends on v. The maximum number of disjoint paths in the sub tree of root v currently a part of a path of size δ, will be the maximum number of disjoint paths of some sub tree $c \in C_v$ which would be part of a path of size $\delta + 1$, plus the sum of the maximum number of disjoint paths for the rest of the children of v. We try out all combinations of c and pick the maximal one.

$$\text{MaxContains}(v, \delta) = \begin{cases} 0 & \textbf{when } \delta < k \textbf{ and } v \text{ is a leaf} \\ \text{MaxDoesNotContain}(v) + 1 & \textbf{when } \delta = k \\ \max\Bigg(\Big\{\text{MaxContains}(c, \delta + 1) & \\ \qquad\qquad + \displaystyle\sum_{c' \in C_v \setminus \{c\}} \text{MaxPath}(c') \mid c \in C_v\Big\}\Bigg) & \textbf{otherwise} \end{cases}$$

Proof. We show by strong induction that our recurrence relation is correct. Let $P(n, k)$ be the predicate, "MaxPath correctly computes the maximum number of disjoint paths of size k in a sub tree of T which has n number of nodes and where v is the root node". We assume that the size of a path must be at least 1, as a path of 0 would lead to infinite amount of paths, and that a sub tree must contain at least one node, as our sub tree has a root. Hence, we define $n, k \in \mathbb{N}^+$.
Base Case: When $n = 1, k = 1$, MaxPath returns 0. MaxContains($v, 0$) returns 0 because $0 < 1$ and v is a leaf. MaxDoesNotContain(v) returns 0 because v has no children. The max of $\{0, 0\}$

equals 0. It is clear to see that there are no paths for a tree that consists of a single node. Hence, $P(1, 1)$ holds.
Inductive step for number of nodes: Suppose $P(\eta, \kappa)$ holds for all $n \leq \eta$ and $k \leq \kappa$. We show that $P(\eta + 1, \kappa)$ holds.

$\square$

§2.2 Write the pseudocode for the iterative version of the algorithm to find the minimum cost. You are not required to write pseudocode to find the shortest path.