

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, \dots, G_b . Give a polynomial-time algorithm to find the shortest such path.

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 to find the shortest path from s to t on $G^* = (V, E^*)$.

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

For an $s - t$ path P in one of the graphs G_i , we define the length of P to be simply the number of edges in P , and we denote this to be $\ell(P)$. Formally, we define $\text{changes}(P_0, P_1, \dots, P_b)$ to be the number of indices i ($0 \leq i \leq b - 1$) for which $P_i \neq P_{i+1}$. Fix a constant $K > 0$. We define the cost of the sequence of paths P_0, P_1, \dots, P_b

$$\text{cost}(P_0, P_1, \dots, P_b) = \sum_{i=0}^b \ell(P_i) + K \cdot \text{changes}(P_0, P_1, \dots, P_b)$$

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

We define $\text{MinPath}(i)$ to be the recurrence for the a sequence of paths of minimum cost in G_0, \dots, G_i . Let the set of edges that exist from graphs G_a, \dots, G_b be defined as

$$E_{a,b} = \bigcap_{i=a}^b E_i$$

Let $\ell_E(E_{a,b})$ be the length of the shortest $s - t$ path in $E_{a,b}$. If a path does not exist we define $\ell_E(E_{a,b}) = \infty$.

$$\text{MinPath}(i) = \min \left(\ell_E(E_{0,i}) \cdot (i+1), \min \left(\left\{ \text{MinPath}(\phi) + \ell_E(E_{\phi+1,i}) \cdot (i-\phi) + K \mid 1 \leq \phi < i \right\} \right) \right)$$

Proof. We show by strong induction that our recurrence relation is correct. Let $P(i)$ be the predicate, "MinPath correctly computes the sequence of paths of minimum cost in G_0, \dots, G_i ". We define $i \in \mathbb{N}$.

Base Case: When $i = 0$, it is trivial to see that the correct minimum cost is $\ell_E(E_{0,0})$ as there is only one graph to consider. We see that value in the left parameter of the min function evaluates to $\ell_E(E_{0,0}) \cdot (0 + 1) = \ell_E(E_{0,0})$. The right parameter never evaluates. Thus, the min of these two values is $\ell_E(E_{0,0})$, which is the correct minimum cost. Therefore, $P(0)$ holds.

Inductive step: Suppose $P(i)$ holds for all $0 \leq i \leq k$. We show that $P(k + 1)$ holds. \square

§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 $\text{MaxPath}(v)$ 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 .

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

In the case where maximum number of disjoint paths does not contain v , we define $\text{MaxDoesNotContain}(v)$ 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 $(v, c) \in E$. We will define the set of child nodes of a vertex v as

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

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 v , there are a couple of subcases. We define $\text{MaxContains}(v, \delta)$ 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 $\text{MaxDoesNotContain}(v)$, 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.

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

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. MaxPath(v) calls MaxContains($v, 0$) and MaxDoesNotContain(v). Since v is not a leaf and $0 \neq k$, third condition holds in MaxContains. Notice that the number of nodes the sub tree where the root is c will have strictly less nodes than $n + 1$, which is amount of nodes in the sub tree where the root is v because c is a child of v . Hence, MaxContains($c, 1$) holds by our inductive hypothesis. Without loss of generality, MaxPath(c') and MaxDoesNotContain(v) holds by our inductive hypothesis. Therefore, $P(\eta + 1, \kappa)$ holds.

Inductive step for length of path: Suppose $P(\eta, \kappa)$ holds for all $n \leq \eta$ and $k \leq \kappa$. We show that $P(\eta, \kappa + 1)$ holds. MaxPath(v) calls MaxContains($v, 0$) and MaxDoesNotContain(v). We continue with the third condition in MaxContains($v, 0$). Notice that MaxContains($c, 1$) is equivalent to MaxContains($c, 0$) if we fix $k = \kappa$. Since $\kappa \leq \kappa + 1$, MaxContains($c, 0$) where $k = \kappa$ and subsequently MaxContains($c, 1$) where $k = \kappa + 1$ holds by our inductive hypothesis. Without loss of generality, MaxPath(c') and MaxDoesNotContain(v) holds by our inductive hypothesis because they will eventually reach MaxContains($v', 1$). Therefore, $P(\eta, \kappa + 1)$ holds.

By strong induction, we have proven that $P(n, k)$ holds for all $n, k \in \mathbb{N}^+$. Therefore, MaxPath is correct. \square

§2.2 Write the pseudocode for the iterative version of the algorithm to find the maximum number of players that can play at the same time.

The main idea of this algorithm is to compute bottom up the optimal cost to view event n when we start on some event i . We do this by creating a length n list which the i th index represents the most amount of events you can view, given that you go straight from the starting coordinate, 0, to event i , and visit the most maximum amount of events from i to n . We will assume that $|\text{coordinate of } n| \leq n$ and are using zero based numbering for lists.

§2.2.1 Analyze the computing complexity.

Data: $T = (V, E)$, a tree with root r ; k , length of path

Result: Maximum number of disjoint paths of size k in the tree T

```

def max( $T, k$ ):
    contains  $\leftarrow$  a map
    notContains  $\leftarrow$  a map
    maxPaths  $\leftarrow$  a map
    for  $v \in V$ :
        #base case
        if  $v$  is a leaf:
            maxPaths.insert( $v, 0$ )
            notContains.insert( $v, 0$ )
            for  $i$  from 0 to  $k - 1$ :
                contains.insert( $\langle v, i \rangle, 0$ )
            contains.insert( $\langle v, k \rangle, 1$ )
        else:
            notContainVal = 0
            for each child  $c$  of  $v$ :
                notContainVal += maxPaths.get( $c$ )
            notContains.insert( $v, notContainVal$ )
            for  $i$  from 0 to  $k-1$ :
                containsVal  $\leftarrow$  a empty list
                for each child  $c$  of  $v$ :
                    #Where  $c' \in C_v \setminus \{c\}$ 
                    val = contains.get( $c, i+1$ ) +  $\sum$  maxPaths.get( $c'$ )
                    containsVal.insert(val)
                contains.insert( $\langle v, i \rangle, \max(\text{containsVal})$ )
            contains.insert( $\langle v, k \rangle, 1 + notContains.get(v)$ )
            maxPaths.insert( $v, \max(\text{contains.get}(\langle v, 0 \rangle), notContains.get(v))$ )
    return maxPaths.get( $r$ )

```