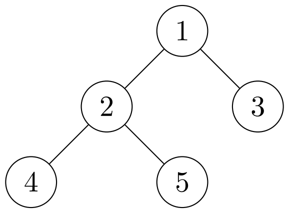
CSE421: Design and Analysis of Algorithms May 9, 2022

Shayan Oveis Gharan Section 6

P1) You are given a tree T where every node i has weight $w_i \geq 0$. Design a polynomial time algorithm to find the weight of the largest weight independent set in T. For example, suppose in the following picture $w_1 = 3, w_2 = 1, w_3 = 4, w_4 = 3, w_5 = 6$. the maximum independent set has nodes $3, 4, 5$ with weight $4 + 3 + 6 = 13$.



Solution: Make the tree T rooted at an arbitrary node r, and consider the corresponding outward rooted tree. For any vertex v, let $T(v)$ denote the subtree of T rooted at v. So, $T(r)$ is the whole tree T. For a node v define $OPT(v)$ to be the weight of the maximum independent set in the tree $T(v)$. So, the solution of the problem is $OPT(r)$. The algorithm runs in polynomial time. This is because first we only have n many sub-problems one for every possible sub-tree of T and we compute the value of each subproblem only once. Furthermore, we spend at most $O(n)$-time to find the value of each sub-problem because we only look at the children and grand-children of the given node v.

```
Function MaxInd(v)
    If $v$ is a leaf then return $OPT(v) = w_v$
    if $OPT(v)$ is not yet computed then
        Let $a = \sum_{u \text{ is a child of } v} \text{MaxInd}(u)$, $b = \sum_{u \text{ is a grand-child of } v} \text{MaxInd}(u)$
        $OPT(v) \leftarrow \max\{a, b + w_v\}$
    end
    Return $OPT(v)$
```

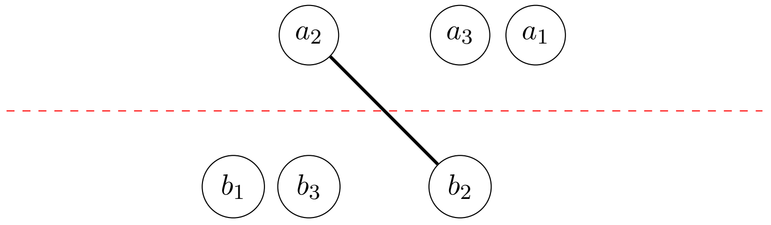
It remains to prove the correctness. Given a node v let us study $OPT(v)$. In the proof we use the following simple fact: If G is a disconnected graph, then the maximum (weighted) independent set of G is the sum of the maximum (weighted) independent sets of connected components of G.

Case 1: v is not in the independent set chosen in $OPT(v)$. In this case there is no interaction between the independent sets chosen in subtrees of the children of v, because these subtrees are not connected to each other. So, $OPT(v)$ is nothing but the sum of the maximum weight independent set of each of the subtrees of the children of v. So, $OPT(v) = \sum_{u \text{ is a child of } v} OPT(u)$.

Case 2: v is in independent set chosen in $OPT(v)$. Let I be the Optimum independent set. So $v \in I$. But then, all children of v are not in I because I is an independent set. Now, look at the subtrees rooted at grand-children of v. there are no interactions between these subtrees because there these subtrees are not connected to each other. So, $OPT(v)$ would choose the maximum independent set in each of them and $OPT(v) = w_v + \sum_{u \text{ is a grand-child of } v} OPT(u)$. Therefore,

$$OPT(v) = \begin{cases} w_v & \text{if } v \text{ is a leaf} \\ \max\{\sum_{u \text{ is a child of } v} OPT(u), w_v + \sum_{u \text{ is a grand-child of } v} OPT(u)\} & \text{otherwise.} \end{cases}$$

P2) A country has $2n$ cities; n of them are on a line north of the river with x-coordinates $a_1, \dots, a_n$ and n of them are on a line south of the river with x-coordinates $b_1, \dots, b_n$. You can assume no two cities in the north have the same coordinates and no two in the south have the same coordinates. We want to make maximum number of bridges between north and south. A bridge is a direct line connecting the i-th city in the north to the i-th city in the south, i.e., a_i to b_i. Design a polynomial time algorithm that outputs the maximum number of bridges we can build such that no two bridges cross each other. For example if $a_1 = 5, a_2 = 2, a_3 = 4$ and $b_1 = 1, b_2 = 4, b_3 = 2$ then, the maximum number of bridges is 1.



Algorithm: We sort the cities north of the river, so we assume perhaps after renaming that $a_1 \le a_2 \le \cdots \le a_n$. Furthermore, suppose for each i, $N(i)$ is the location a_i before sorting, i.e., that means that we can only make bridge between a_i and $b_{N(i)}$. Then, we run the following algorithm: I remark that there is also another way to solve this problem by reducing it to the

```
Set $M[i,0] = 0$ and $M[0,i] = 0$ for all $1 \le i \le n$;
for $i = 1 \to n$ do
    for $j = 1 \to n$ do
        Suppose $N(i)$ has the $k$-th smallest x-coordinate south of river;
        if $N(i) > k$ then
            $M[i,j] \leftarrow M[i-1,j]$;
        end
        else
            $M[i,j] \leftarrow \max\{M[i-1,j], 1 + M[i-1,k-1]\}$;
        end
    end
end
Return $M[n,n]$;
```

longest increasing subsequence problem.

Runtime: There are n^2 many subproblems and it takes $O(1)$ to solve every subproblem. So, the algorithm runs in $O(n^2)$.

Correctness: Let $OPT(i, j) =$ be maximum number of non-crossing bridges that we can draw between the cities $a_1, \ldots, a_i$ and the j-th leftmost cities in the south such that no two bridges cross with the constraint that each a_i can only have a bridge to $b_{N(i)}$. For a base case notice that if $i = 0$, i.e., we have no cities in the north or $j = 0$ that we have no cities in the south $OPT(i, j) = 0$. Now, we discuss how to solve $OPT(i, j)$. We guess the last decision that OPT takes, i.e., whether to build a bridge between $a_i, b_{N(i)}$ or not:

Case 1) $OPT(i, j)$ doesn't have a bridge between $a_i, b_{N(i)}$. Then, the i-th city in the north is useless, and $OPT(i, j)$ will simply be the maximum number of bridges between the $i - 1$ leftmost cities in the north and j leftmost cities in the south, i.e., $OPT(i - 1, j)$.

Case 2) $OPT(i, j)$ draws a bridge between $a_i, b_{N(i)}$. Suppose that $N(i)$ has the k-th smallest x-coordinate among all cities south of the river. First, notice that this bridge can be built only if $k \geq j$, i.e., if $k > j$ we can only be in case 1. If $k \leq j$ and we draw a bridge between $a_i, b_{N(i)}$ then we cannot make any bridge between cities $a_1, \ldots, a_{i-1}$ and the $j - k$ cities that come after $b_{N(i)}$ in the south; any such bridge would cross the $a_i \leftrightarrow b_{N(i)}$ bridge. Thus, we can only have bridges between cities $a_1, \ldots, a_{i-1}$ and the left $k - 1$ cities in the south, i.e., we need to take $OPT(i - 1, k - 1)$.

Taking the best of the two cases proves the correctness.