HW 7: Extra problems

Instructors: Har-Peled

CS/ECE 374A: Intro. Algorithms & Models of Computation, Fall 2024 Version: **1.0**

1 (100 PTS.) Trees have needs, but then who doesn't?

We are given a tree $T = (V, E)$ with n vertices. Assume that the degree of all the vertices in T is at most 3. You are given a function $f : V \to \{0, 1, 2, 3\}$. The task is to compute a subset X of edges, such that for every node $v \in V$, there are at least $f(v)$ distinct edges in X that are adjacent to v.

Describe an algorithm, as fast as possible, that computes the minimum size set $X \subseteq E$ that meets the needs of all the nodes in the tree. The algorithm should output both $|X|$ and X itself.

2 (100 PTS.) Trust but shadow.

A classical tactic in war, is for a small force to "shadow" a bigger force of the enemy. So, assume we have a main army moving along a sequence $p_1, \ldots, p_n$ of locations. Initially, the army start at p_1, and every day it moves forward to the next location, spending the night there. The shadow army, similarly, knowing their enemy path, has a sequence of locations $q_1, q_2, \ldots, q_m$ that it is planning to travel through.

The shadow army, being much smaller, can move much faster than the main army. In particular, it can move through as many locations as it wants in one day.

The important thing with shadowing, is that the shadow army should not be too close to the main army when they both camp at night (because that would trigger a battle, which would be bad). Similarly, the shadow army should not be too far from the main army, as then it can not keep track of it.

The task at hand is to come up with a schedule for the shadow army. In the beginning of ith day, the main army is at location p_i, and the shadow army is at location $q_{\pi(i)}$ (π is what you have to compute). We require that $\pi(1) = 1$, $\pi(i + 1) \geq \pi(i)$, and $\pi(n) = m$. The locations $p_1, \ldots, p_n$ and $q_1, \ldots, q_m$ are points in the plane (and are the input to the algorithm), and the distance between two locations is the Euclidean distance between them.

2.A. (40 PTS.) An interval $[x, y] \subseteq \mathbb{R}$ is ***feasible*** if there exists a valid schedule π, such that, for all i, we have $\|p_i - q_{\pi(i)}\| \in [x, y]$. Given such an interval $[x, y]$, describe a dynamic program algorithm, as fast as possible, that uses as little space as possible, that decides if $[x, y]$ is feasible (no need to output the schedule).

2.B. (30 PTS.) Describe an algorithm, as fast as possible, that computes the maximal x, such that the interval $[x, \infty]$ is feasible.

2.C. (30 PTS.) The ***instability*** of an interval $[x, y]$, with $0 < x < y$, is the ratio y/x. Describe an algorithm, as fast as possible, that computes the interval with minimum instability, among all feasible intervals.

3 (100 PTS.) Superstring theory.

You are given a DFA $M = (Q, \Sigma, \delta, s, A)$ with n states, where $|\Sigma| = O(1)$.

For two strings $x, y \in \Sigma^*$, the string x is a ***superstring*** of y, if one can delete characters from x and get y.

Let w be a given input string with m characters.

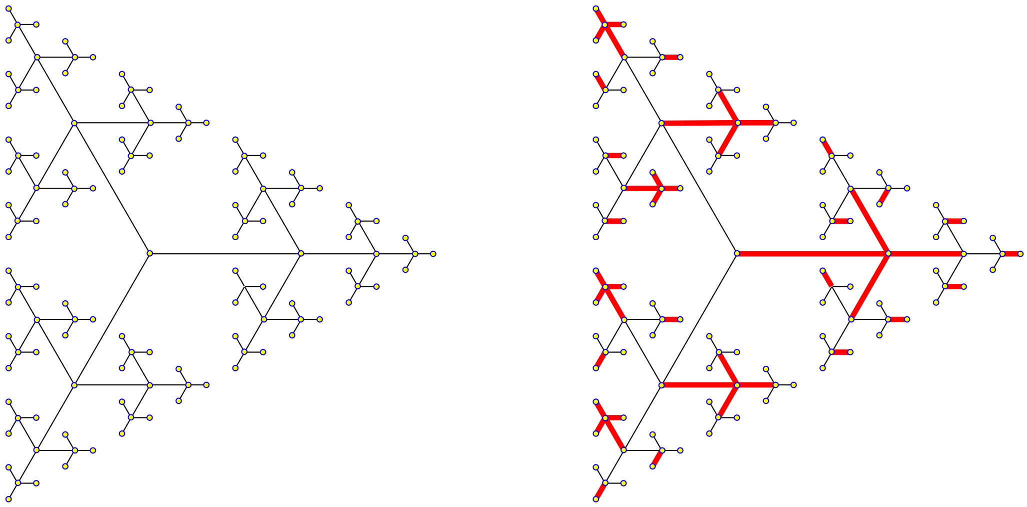
3.A. (25 PTS.) Let $q, q' \in Q$ be two states of M. Prove, that the shortest string w'', such that $\delta^*(q, w'') = q'$ is of length at most $n - 1$.

3.B. (25 PTS.) Prove, that if there is a superstring x of w, such that x is accepted by M, then the shortest such superstring is of length at most $(n+1)(m+1)$, where $n = |Q|$ and $m = |w|$.

3.C. (50 PTS.) Describe an algorithm, as efficient as possible, that computes the shortest superstring z of w, such that z is accepted by M. (One can solve this problem using dynamic programming, but this is definitely not the only way.)

4 (100 PTS.) Stars in a tree.

A *star*, in a tree, is a vertex together with *all* the edges adjacent to it. A collection of stars is ***independent*** if no two stars shares a vertex. The *mass* of an independent set of stars S is the total number of edges in the stars of S.



Describe an algorithm, as efficient as possible, that computes the maximum mass of any set of independent stars in the given tree T. Here the input tree T has n vertices.

5 (100 PTS.) Exploring Narnia.

Three travelers had decided to travel to Narnia. Since they are all anti-social, they do not want to travel together. Instead, the first traveler would move from location p_1 to location p_2, and so on till arriving to location p_n (here, $P = p_1, \ldots, p_n$). A location is just a point in the plane (i.e., $p_i \in \mathbb{R}^2$). Similarly, the second traveler is going to move along $Q = q_1, \ldots, q_n$, and the third traveler is going to move along $R = r_1, \ldots, r_n$. Every day, a traveler might decide to stay in its current location, or move to the next location (the traveling between two consecutive locations takes less than a day).

By the evening of each day, the travelers arrive to their desired locations for the day, which can be thought of as a configuration $(p, q, r) \in P \times Q \times R$.

A configuration (p, q, r) is *ℓ-legal* if

$$\Delta(p, q, r) = \max\bigl(\|p - q\|, \|p - r\|, \|q - r\|\bigr) \leq \ell,$$

where $\|p - q\|$ is the Euclidean distance between the points p and q (this distance can be computed in constant time). (Intuitively, the travelers do not want to be too far from each other, so that if one of them is hurt, the others can quickly come over and help.)

5.A. (50 PTS.) Given the point sequences P, Q, R, and a parameter $\ell > 0$, describe an algorithm, as fast as possible, that decides if there is a motion planning schedule from (p_1, q_1, r_1) to (p_n, q_n, r_n) such that all the configurations used are ℓ-legal. Such a schedule is an *ℓ-feasible schedule.*

Here, it is legal for several travelers to move in the same day, but they are not allowed to move back to a previous location they already used. Furthermore, a traveler can move, in one day, only one location forward in their sequence.

5.B. (50 PTS.) Given P, Q, R, as above, describe an algorithm, as fast as possible, that computes the minimum ℓ for which there is an ℓ-feasible schedule.

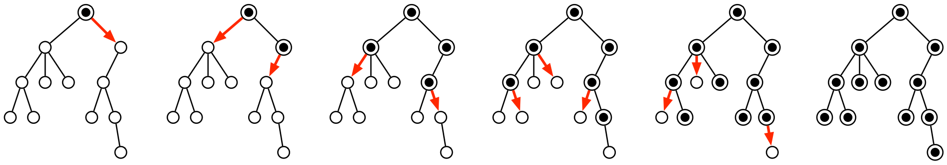
6 Suppose you are given a DFA $M = (Q, \Sigma, \delta, s, F)$ and a binary string $w \in \Sigma^*$ where $\Sigma = \{0, 1\}$. Describe and analyze an algorithm that computes the longest subsequence of w that is accepted by M, or correctly reports that M does not accept any subsequence of w.

7 Problem 6.21 in Dasgupta etal on finding the minimum sized vertex cover in a tree.

8 The McKing chain wants to open several restaurants along Red street in Shampoo-Banana. The possible locations are at $L_1, L_2, \ldots, L_n$ where L_i is at distance m_i meters from the start of Red street. Assume that the street is a straight line and the locations are in increasing order of distance from the starting point (thus $0 \le m_1 < m_2 < \ldots < m_n$). McKing has collected some data indicating that opening a restaurant at location L_i will yield a profit of p_i independent of where the other restaurants are located. However, the city of Shampoo-Banana has a zoning law which requires that any two McKing locations should be D or more meters apart. *In addition McKing does not want to open more than k restaurants due to budget constraints.* Describe an algorithm that McKing can use to figure out the maximum profit it can obtain by opening restaurants while satisfying the city's zoning law and the constraint of opening at most k restaurants. Your algorithm should use only $O(n)$ space.

9 Let $X = x_1, x_2, \ldots, x_r$, $Y = y_1, y_2, \ldots, y_s$ and $Z = z_1, z_2, \ldots, z_t$ be three sequences. A common *supersequence* of X, Y and Z is another sequence W such that X, Y and Z are subsequences of W. Suppose $X = a, b, d, c$ and $Y = b, a, b, e, d$ and $Z = b, e, d, c$. A simple common supersequence of X, Y and Z is the concatenation of X, Y and Z which is $a, b, d, c, b, a, b, e, d, b, e, d, c$ and has length 13. A shorter one is b, a, b, e, d, c which has length 6. Describe an efficient algorithm to compute the *length* of the shortest common supersequence of three given sequences X, Y and Z.

10 Suppose we need to distribute a message to all the nodes in a rooted tree. Initially, only the root node knows the message. In a single round, any node that knows the message can forward it to at most one of its children. Design an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes in a given tree. See figure below for an example. Assume that the tree is binary (number of children is at most 2).



A message being distributed through a tree in five rounds.

Solved Problems

11 A string w of parentheses ⦅ and ⦆ and brackets ⟦ and ⟧ is ***balanced*** if it is generated by the following context-free grammar:

$$S \to \varepsilon \mid ⦅S⦆ \mid ⟦S⟧ \mid SS$$

For example, the string $w =$ ⦅⟦⦅⦆⟧⟦⟧⦅⦆⦆⟦⦅⦆⦅⦆⟧⦅⦆ is balanced, because $w = xy$, where

$x =$ ⦅ ⟦⦅⦆⟧ ⟦⟧ ⦅⦆ ⦆ and $y =$ ⟦ ⦅⦆ ⦅⦆ ⟧ ⦅⦆.

Describe and analyze an algorithm to compute the length of a longest balanced subsequence of a given string of parentheses and brackets. Your input is an array $A[1 \,..\, n]$, where $A[i] \in \{$⦅, ⦆, ⟦, ⟧$\}$ for every index i.

Solution: Suppose $A[1\,..\,n]$ is the input string. For all indices i and j, we write $\boldsymbol{A[i] \sim A[j]}$ to indicate that $A[i]$ and $A[j]$ are matching delimiters: Either $A[i] =$ ⦅ and $A[j] =$ ⦆ or $A[i] =$ ⟦ and $A[j] =$ ⟧.

For all indices i and j, let $\boldsymbol{LBS(i,j)}$ denote the length of the longest balanced subsequence of the substring $A[i\,..\,j]$. We need to compute $LBS(1,n)$. This function obeys the following recurrence:

$$LBS(i,j) = \begin{cases} 0 & \text{if } i \geq j \\ \max \left\{ \begin{array}{c} 2 + LBS(i+1, j-1) \\ \max_{k=1}^{j-1} \big(LBS(i,k) + LBS(k+1,j)\big) \end{array} \right\} & \text{if } A[i] \sim A[j] \\ \max_{k=1}^{j-1} \big(LBS(i,k) + LBS(k+1,j)\big) & \text{otherwise} \end{cases}$$

We can memoize this function into a two-dimensional array $LBS[1\,..\,n, 1\,..\,n]$. Since every entry $LBS[i,j]$ depends only on entries in later rows or earlier columns (or both), we can evaluate this array row-by-row from bottom up in the outer loop, scanning each row from left to right in the inner loop. The resulting algorithm runs in $\boldsymbol{O(n^3)}$ ***time***.

```
LongestBalancedSubsequence($A[1\,..\,n]$):
for $i \gets n$ down to 1
    $LBS[i,i] \gets 0$
    for $j \gets i+1$ to $n$
        if $A[i] \sim A[j]$
            $LBS[i,j] \gets LBS[i+1, j-1] + 2$
        else
            $LBS[i,j] \gets 0$
        for $k \gets i$ to $j-1$
            $LBS[i,j] \gets \max\{LBS[i,j],\ LBS[i,k] + LBS[k+1,j]\}$
return $LBS[1,n]$
```

Rubric: 10 points, standard dynamic programming rubric

12 Oh, no! You've just been appointed as the new organizer of Giggle, Inc.'s annual mandatory holiday party! The employees at Giggle are organized into a strict hierarchy, that is, a tree with the company president at the root. The all-knowing oracles in Human Resources have assigned a real number to each employee measuring how "fun" the employee is. In order to keep things social, there is one restriction on the guest list: An employee cannot attend the party if their immediate supervisor is also present. On the other hand, the president of the company *must* attend the party, even though she has a negative fun rating; it's her company, after all.

Describe an algorithm that makes a guest list for the party that maximizes the sum of the "fun" ratings of the guests. The input to your algorithm is a rooted tree T describing the company hierarchy, where each node v has a field $v.fun$ storing the "fun" rating of the corresponding employee.

Solution: [two functions] We define two functions over the nodes of T.

- $\mathit{MaxFunYes}(v)$ is the maximum total "fun" of a legal party among the descendants of v, where v is definitely invited.
- $\mathit{MaxFunNo}(v)$ is the maximum total "fun" of a legal party among the descendants of v, where v is definitely not invited.

We need to compute $\mathit{MaxFunYes}(\mathit{root})$. These two functions obey the following mutual recurrences:

$$\begin{aligned}\mathit{MaxFunYes}(v) &= v.\mathit{fun} + \sum_{\text{children } w \text{ of } v} \mathit{MaxFunNo}(w)\\ \mathit{MaxFunNo}(v) &= \sum_{\text{children } w \text{ of } v} \max\{\mathit{MaxFunYes}(w), \mathit{MaxFunNo}(w)\}\end{aligned}$$

(These recurrences do not require separate base cases, because $\sum\varnothing = 0$.) We can memoize these functions by adding two additional fields $v.\mathit{yes}$ and $v.\mathit{no}$ to each node v in the tree. The values at each node depend only on the vales at its children, so we can compute all $2n$ values using a post-order traversal of T.

```
BestParty(T):
    ComputeMaxFun(T.root)
    return T.root.yes
```

```
ComputeMaxFun(v):
    v.yes ← v.fun
    v.no ← 0
    for all children w of v
        ComputeMaxFun(w)
        v.yes ← v.yes + w.no
        v.no ← v.no + max {w.yes, w.no}
```

(Yes, this is still dynamic programming; we're only traversing the tree recursively because that's the most natural way to traverse trees![a]) The algorithm spends $O(1)$ time at each node, and therefore runs in $\boldsymbol{O(n)}$ ***time*** altogether.

[a] A naive recursive implementation would run in $O(\phi^n)$ time in the worst case, where $\phi = (1+\sqrt{5})/2 \approx 1.618$ is the golden ratio. The worst-case tree is a path-every non-leaf node has exactly one child.

Solution: [one function] For each node v in the input tree T, let $\mathit{MaxFun}(v)$ denote the maximum total "fun" of a legal party among the descendants of v, where v may or may not be invited.

The president of the company must be invited, so none of the president's "children" in T can be invited. Thus, the value we need to compute is

$$\mathit{root.fun} + \sum_{\text{grandchildren } w \text{ of } \mathit{root}} \mathit{MaxFun}(w).$$

The function *MaxFun* obeys the following recurrence:

$$\mathit{MaxFun}(v) = \max \left\{ \begin{array}{c} v.\mathit{fun} + \displaystyle\sum_{\text{grandchildren } x \text{ of } v} \mathit{MaxFun}(x) \\ \displaystyle\sum_{\text{children } w \text{ of } v} \mathit{MaxFun}(w) \end{array} \right\}$$

(This recurrence does not require a separate base case, because $\sum \varnothing = 0$.) We can memoize this function by adding an additional field $v.\mathit{maxFun}$ to each node v in the tree. The value at each node depends only on the values at its children and grandchildren, so we can compute all values using a postorder traversal of T.

```
BestParty($T$):
    ComputeMaxFun($T.\mathit{root}$)
    $\mathit{party} \gets T.\mathit{root.fun}$
    for all children $w$ of $T.\mathit{root}$
        for all children $x$ of $w$
            $\mathit{party} \gets \mathit{party} + x.\mathit{maxFun}$
    return $\mathit{party}$
```

```
ComputeMaxFun($v$):
    $\mathit{yes} \gets v.\mathit{fun}$
    $\mathit{no} \gets 0$
    for all children $w$ of $v$
        ComputeMaxFun($w$)
        $\mathit{no} \gets \mathit{no} + w.\mathit{maxFun}$
        for all children $x$ of $w$
            $\mathit{yes} \gets \mathit{yes} + x.\mathit{maxFun}$
    $v.\mathit{maxFun} \gets \max\{\mathit{yes}, \mathit{no}\}$
```

(Yes, this is still dynamic programming; we're only traversing the tree recursively because that's the most natural way to traverse trees![a])

The algorithm spends $O(1)$ time at each node (because each node has exactly one parent and one grandparent) and therefore runs in ***$O(n)$ time*** altogether.

Rubric: 10 points: standard dynamic programming rubric. These are not the only correct solutions.

[a] Like the previous solution, a direct recursive implementation would run in $O(\phi^n)$ time in the worst case, where $\phi = (1+\sqrt{5})/2 \approx 1.618$ is the golden ratio.

13 (100 PTS.) Break into intervals (Fall 2022).

The input is a sequence $\alpha_1, \ldots, \alpha_n$ of n real numbers, and a parameter $k \leq n$. For integers $i \leq j$, their *interval* is the set $[\![i : j]\!] = \{i, i+1, \ldots, j\}$. The *benefit* of such an interval is $\varphi(i,j) = \sum_{t=i}^{j} (\alpha_t + \alpha_i)^2$. Describe an efficient *dynamic programming* algorithm that computes the maximum price solution when one breaks the sequence into (*exactly*) k intervals $I_1, \ldots, I_k$. Specifically, compute the intervals $I_1, \ldots, I_k$, such that:

(a) $\cup_i I_i = [\![1 : n]\!]$,
(b) all the k intervals are disjoint,
(c) all the intervals $I_1, \ldots, I_k$ are non-empty,
(d) and $\sum_i \varphi(I_i)$ is ***maximized***.

In addition, your algorithm also need to output the optimal solution itself (i.e., the k intervals forming the optimal solution).

As usual, your algorithm should be as fast as possible, and try and minimize (within reason) the space used.

(See top of previous homework to see exact instructions of what you have to provide – in particular, you have to provide a recursive formulation of this problem, and a dynamic [non-recursive] program to solve this problem.)

14 (100 PTS.) Convert into graph problems (Fall 22)

14.A. (50 PTS.) For problem 11 (from the previous homework), describe an efficient reduction of the problem to a graph problem. Formally, describe an efficient algorithm that solves it by constructing an appropriate DAG (i.e., directed acyclic graph) G, and then solving the original problem by solving a standard graph problem on G. Assuming that this standard problem can be solved in linear time in the size of the graph (i.e., number of vertices plus the number of edges), what is the running time of your algorithm? (As usual your algorithm should be as fast as possible, but we do not care about the space requirement here.)

14.B. (50 PTS.) Same as previous part, but for problem 13 above (here, you just need to compute the optimal solution value, there is no need to output the solution itself).