CS/ECE 374 ✧ Spring 2021

Homework 6

Due Thursday, March 25, 2021 at 10am

Groups of up to three people can submit joint solutions. Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

The following unnumbered problems are not for submission or grading. No solutions for them will be provided but you can discuss them on Piazza.

- 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.

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

1. 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 cost c_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. McKing wants to open k restaurants based on their budget and profitablity considerations. Describe an algorithm that McKing can use to figure out the minimum cost it needs to incur to open k restaurants while satisfying the city's zoning law. Your algorithm should output ∞ if k restaurants cannot be opened due to the zoning law. For full credit your algorithm should use only $O(n)$ space; note that you cannot assume that k is a fixed constant.

2. 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. You may want to first solve the two sequence problem to get you started.

3. In lecture we saw an efficient algorithm to compute a maximum weight independent set in a given tree $T = (V, E)$.

- Given a tree $T = (V, E)$ describe an efficient algorithm to *count* the number of distinct independent sets in T. Two independent sets S_1 and S_2 are distinct if they are not identical as sets of vertices. Note that the empty set counts as a valid independent set.
- What is the exact number of independent sets if the tree is a star with a center and $n-1$ leaves? Would the answer for a star with $n = 500$ fit in a 64-bit integer word? Briefly justify your answer.
- How would you implement your counting algorithm from part (a), more carefully, to run on a 64 bit machine? Accounting for this more careful implementation, what is the running time of your algorithm?

Solved Problems

4. A string w of parentheses **(** and **)** and brackets **[** and **]** is ***balanced*** if it is generated by the following context-free grammar:

$$S \to \varepsilon \mid \mathbf{(}S\mathbf{)} \mid \mathbf{[}S\mathbf{]} \mid SS$$

For example, the string $w = \mathbf{([()][]())[()()]()}$ is balanced, because $w = xy$, where

$$x = \mathbf{(\ [()]\ []\ ()\ )} \qquad \text{and} \qquad y = \mathbf{[\ ()\ ()\ ]\ ()}.$$

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 $A[i] \in \{\mathbf{(}, \mathbf{)}, \mathbf{[}, \mathbf{]}\}$ 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 $A[i] = \mathbf{(}$ and $A[j] = \mathbf{)}$ or $A[i] = \mathbf{[}$ and $A[j] = \mathbf{]}$.

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:

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

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 ← n down to 1
      LBS[i,i] ← 0
      for j ← i+1 to n
          if A[i] ~ A[j]
              LBS[i,j] ← LBS[i+1, j−1] + 2
          else
              LBS[i,j] ← 0
          for k ← i to j−1
              LBS[i,j] ← max {LBS[i,j], LBS[i,k] + LBS[k+1,j]}
  return LBS[1,n]
```

■

Rubric: 10 points, standard dynamic programming rubric

5. 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.

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

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

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

   (These recurrences do not require separate base cases, because $\sum\varnothing = 0$.) We can memoize these functions by adding two additional fields $v.yes$ and $v.no$ 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 postorder 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![1]) The algorithm spends $O(1)$ time at each node, and therefore runs in $\boldsymbol{O(n)}$ ***time*** altogether. ■

[1]A naïve 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 $MaxFun(v)$ 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

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

The function $MaxFun$ obeys the following recurrence:

$$MaxFun(v) = \max \begin{Bmatrix} v.fun + \displaystyle\sum_{\text{grandchildren } x \text{ of } v} MaxFun(x) \\ \displaystyle\sum_{\text{children } w \text{ of } v} MaxFun(w) \end{Bmatrix}$$

(This recurrence does not require a separate base case, because $\sum \varnothing = 0$.) We can memoize this function by adding an additional field $v.maxFun$ 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.root)
    party ← T.root.fun
    for all children w of T.root
        for all children x of w
            party ← party + x.maxFun
    return party
```

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

(Yes, this is still dynamic programming; we're only traversing the tree recursively because that's the most natural way to traverse trees![2])

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

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

[2]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.