CSE 101 Homework 3

Winter 2023

This homework is due on gradescope Friday February 10th at 11:59pm on gradescope. Remember to justify your work even if the problem does not explicitly say so. Writing your solutions in LaTeXis recommend though not required.

Question 1 (Road Trip on a Budget, 30 points)**.** *Jane is planning a road trip from city s to city t. She has a map of the relevant country given as a directed graph G with s and t as vertices. Each edge of the graph has an associated monetary cost which must be paid to cross the edge, however, by working as a part time courier, some of these costs are negative (though you may assume that G has no negative weight cycles). Jane wants to ensure that she has at least m dollars on her when she reaches t and she knows that she can never allow her bank balance to go negative during the trip. Give an algorithm that given G, m, s, t and the edge weights, computes the least amount of money that Jane would need to have at the start of her trip in order to make this happen. For full credit, your algorithm should run in time $O(|V||E|)$ or better.*

The high-level idea is to have a Bellman-Ford-type algorithm. However, instead of calculating "the shortest path with at most k edges", we calculate "the least amount of money Jane would need to get to city t with path of at most k edges". Let $n_k(v)$ denote the least amount of money Jane should bring in order to get to city t with path of at most k edges starting at vertex v.

Our pseudocode is as follows:

```
1. $n_0(v) \leftarrow \infty$ for all $v$
2. $n_0(t) \leftarrow m$
3. For $k = 1$ to $|V| - 1$:
    • For $w \in V$:
        – $n_k(w) \leftarrow min_{(u,e)\in E}(max(n_{k-1}(u) + l(w,u), 0))$
```

Proof of Correctness:

First, we wish to prove that if $k \geq |V|-1$, then $n_k(v)$ is the least money Jane needs to bring in order to ensure that she has at least m dollars when reaching t and her balance never goes under 0. This is the same as proving that the path allowing Jane to bring least amount of money has less or equal to $|V|-1$ edges. Suppose towards contradiction that the "shortest path" has at least $|V|$ edges. Then, following the proof in class, there must been a loop somewhere. Because there is no negative cycle in this graph, the loop would have a net cost of at least 0. Removing the loop gives as a path that is at least as good as before. Thus, this contradicts the assumption that the "shortest path" has at least $|V|$ edges.

Next, we can prove by induction on the step k m that $n_k(v)$ contains the correct least amount of money Jane has to bring for all k.

Base case: When $k = 0$, the claim is trivially true: Jane has to have m dollars at t. All other vertices have value ∞ because there is no way to get to t with 0 edge.

Inductive hypothesis: Assume the values are correct for all values under or equal to k.

Inductive step: Assume the inductive hypothesis. Consider $n_{k+1}(w)$ for some $w \in V$. We know that we can only get to t from w by it's outgoing edges and $n_k(u)$ is already calculated correctly for all outgoing edges (w, u). If for some u_0 such that $(w, u_0) \in E$, $n_k(u_0) = a$. Then, we know we need at least a dollars at u_0 to get to t with at most k steps. Then, given the edge weight $l(w, u)$, we need at least $a + l(w, u)$ to get to t with at most $k + 1$ edges and the first stop being u_0. We also need to make sure if $a + l(w, u) < 0$, then we still need at least 0 at w. Because every path w can get to t has to go through one of its outgoing edges, then $n_{k+1}(w)$ is correctly calculated as $min_{u \in V}(max(n_k(u) + l(w, u), 0)$.

Combining the two parts together, the correctness is proved.

Runtime Analysis: The outer loop runs $O(|V|)$ times and the inner loop runs at most $O(|E|)$ time. Thus, the resulting runtime is $O(|V||E|)$ similar to the Bellman-Ford algorithm.

Question 2 (Inversion Counting, 30 points)**.** *Given an array A of real numbers an* inversion *is a pair of indices $i < j$ where $A[i] > A[j]$. Give an algorithm that computes the number of inversions in such an array A. For full credit, your algorithm should run in time $O(n \log(n))$ or better where n is the number of elements in A.*

The high-level idea is to divide into halves, count each half and then have to count the pairs crossing the middle. In other words, for each element in B we need to add the number of elements in A that are bigger than it.

MergeAndCount(A,B)

```
• global count
• $C \leftarrow$ list of length $Len(A) + Len(B)$
• $a \leftarrow 1, b \leftarrow 1$
• For c = 1 to Len(C)
    – If $(b > Len(B))$
        ∗ $C[c] \leftarrow A[a], a++$
    – Else if $(a > Len(A))$
        ∗ $C[c] \leftarrow B[b], b++$
    – Else if $A[a] \leq B[b]$
        ∗ $C[c] \leftarrow A[a], a++$
    – Else
        ∗ $C[c] \leftarrow B[b], b++$
        ∗ $count \leftarrow count + Len(A) - a + 1$
• Return C
```

MergeSortAndCount(L)

```
• If $Len(L) = 1$
    – Return L
• Split $L$ into equal $L_1$ and $L_2$
• $A \leftarrow MergeSortAndCount(L_1)$
• $B \leftarrow MergeSortAndCount(L_2)$
• return MergeAndCount(A,B)
```

Main(L)

```
• count $\leftarrow$ 0
• MergeSortAndCount(L)
```

Proof of correctness:

We will prove by induction as well. The **base case** is obviously correct since if there are only one element, there is guaranteed to be no inversion. For the inductive step, we assume that the inversion count is correct so far. We know that any inversion falls into one of three cases:

1. i, j both on the left half;
2. i, j both on the right half;
3. i on the left half and j on the right half

By our inductive hypothesis, case 1 and case 2 are correctly handled and added to the count. None of case 3 has been counted. For each $l \in L_2$, the number of inversion pairs containing l in case 2 is equal to the number of i such that $i > l$ and $i \in L_1$. Because L_1 and L_2 has been sorted before merged together, the number of $i \in L_1$ such that $i > l$ for $l \in L_2$ is equal to the number of elements left in L_1 when l is put into the merged list. Therefore, the algorithm correctly count case 3 as well. Adding the count of case 3 to the orginal count, we have the correct count of inversions.

Runtime Analysis: Similar to MergeSort covered in class, we split the list into two equal halves and merge them takes $O(n)$ complexity. We have $T(n) = 2T(n/2) + O(n)$. By the Master theorem, we have $T(n) = O(n\ \log(n))$

Question 3 (Other Runtime Recurrences, 40 points)**.** *The Master Theorem tells us that if a divide and conquer algorithm on an input of size n needs to make finitely many recursive calls on inputs whose sizes are a constant factor smaller than n on top of a polynomial amount of overhead, then the runtime of the full algorithm is polynomial. Here we see what happens if one has other kinds of recurrences. In all of the following, we assume a base case of $T(1) = 1$. Note that here $T(n)$ is polynomial if it is $O(n^k)$ for some k, and super-polynomial if this is not the case for any k.*

(a) If $T(n) = \sqrt{n}T(\lfloor n/2 \rfloor)$ for $n > 1$, show that $T(n)$ is superpolynomial. [10 points]

(b) If $T(n) = T(n - \lfloor \sqrt{n} \rfloor) + T(\lfloor n/2 \rfloor)$ for $n > 1$, show that $T(n)$ is superpolynomial. [10 points]

(c) If $T(n) = n^9 T(\lfloor n^{0.9} \rfloor)$ for $n > 1$, show that $T(n)$ is polynomial. [10 points]

(d) If $T(n) = T(\lfloor n^{0.9} \rfloor) + T(n - \lfloor n^{0.9} \rfloor)$ for $n > 1$, show that $T(n)$ is polynomial. [10 points]

(d) (Corrected) If $T(n) = T(n-1) + T(\lfloor n^{0.9} \rfloor)$ for $n > 1$, show that $T(n)$ is polynomial. [10 points]

If you are proving any of this using induction be sure to be careful not to fall into the trap discussed in Homework 0 problem 4.

(a)

$T(n) = \sqrt{n}\, T(\lfloor n/2 \rfloor)$, Unravel to reach the base case

$T(n) = \sqrt{n}\, \sqrt{\lfloor n/2 \rfloor}\, T(\lfloor n/4 \rfloor)$

$T(n) = \sqrt{n}\, \sqrt{\lfloor n/2 \rfloor}\, \sqrt{\lfloor n/4 \rfloor}\, T(\lfloor n/8 \rfloor)$

.

.

$T(n) = \sqrt{n}\, \sqrt{\lfloor n/2 \rfloor}\, \sqrt{\lfloor n/4 \rfloor} \,..\, \sqrt{\lfloor n/2^{k-1} \rfloor}\, T(\lfloor n/2^k \rfloor)$

$T(n) \geq \sqrt{n/2}\, \sqrt{n/4}\, \sqrt{n/8} \,..\, \sqrt{\lfloor n/2^k \rfloor}\, T(\lfloor n/2^k \rfloor)$ Using $\sqrt{\lfloor (n/2^k) \rfloor} \geq \sqrt{n/2^{k+1}}$

$T(n) \geq \sqrt{n^k/2^{1+2..k}}\, T(\lfloor n/2^k \rfloor)$

$T(n) \geq \sqrt{n^k/2^{(k+1)k/2}}\, T(\lfloor n/2^k \rfloor)$

To get the closed form we need to reach the base case i.e T(1). So equate $\lfloor n/2^k \rfloor$ to 1.

$\lfloor n/2^k \rfloor = 1$

$k = \lfloor \log_2 n \rfloor$

Substitute the value of k in the equation to get the closed form for $T(n)$.

$T(n) \geq \sqrt{n^{\lfloor \log_2 n \rfloor}/2^{((\lfloor \log_2 n \rfloor + 1)\lfloor \log_2 n \rfloor)/2}}\, T(1)$

Simplifying above equation we get $\Omega(n^{((\lfloor \log_2 n \rfloor)/4 - 1/4)})$. Thus, T(n) is superpolynomial.

(b) It is easy to show that subtracting $\lfloor\sqrt{n}\rfloor$ gives result that is at most the result of subtracting $\lfloor\sqrt{m}\rfloor$ where m is the current input to T. Thus, unraveling the first term in the recurrence relation $\lceil\sqrt{n}/2\rceil$ times, we have

$$\begin{aligned}
T(n) &\geq T(n-2\lfloor\sqrt{n}\rfloor)+T(\lfloor(n-\lfloor\sqrt{n}\rfloor)/2\rfloor)+T(\lfloor n/2\rfloor)\\
&\geq \sum_{k=0}^{\lceil\sqrt{n}/2\rceil} T(\lfloor(n-k\lfloor\sqrt{n}\rfloor)/2\rfloor)\\
&\geq \sum_{k=0}^{\lceil\sqrt{n}/2\rceil} T(\lfloor(n-\lceil\sqrt{n}/2\rceil\cdot\lfloor\sqrt{n}\rfloor)/2\rfloor)\\
&\geq \sum_{k=0}^{\lceil\sqrt{n}/2\rceil} T(\lfloor n/8\rfloor) \quad \text{easy to show that } n-\lceil\sqrt{n}/2\rceil\cdot\lfloor\sqrt{n}\rfloor \geq \lfloor n/8\rfloor\\
&\geq \frac{\sqrt{n}}{2}T(\lfloor n/8\rfloor)\\
&= \sqrt{\frac{n}{4}}T(\lfloor n/8\rfloor)
\end{aligned}$$

By the same logic in part a, we can show that this is superpolynomial.

(c) **claim:** $T(n)\leq n^{100}$ We will prove by induction.

Base case: When $n=2$, $T(2)=2^9\cdot T(1)=2^9\leq 2^{100}$

Inductive Hypothesis: Assume $T(m)=m^{100}$ for all $m<n$.

Inductive Step: Let's consider $T(n)$. $T(n)=n^9T(\lfloor n^{0.9}\rfloor)\leq n^9(\lfloor n^{0.9}\rfloor)^{100}\leq n^9(n^{0.9})^{100}=n^9n^{90}=n^{99}\leq n^{100}$

Conclusion: We have proved by strong induction that $T(n)\leq n^{100}$. Hence, $T(n)=O(n^{100})$, showing that $T(n)$ has a polynomial growth rate.

(d) **claim**: $T(n)=n$.

We will prove by induction.

Base case: When $n=2$, $T(n)=T(\lfloor 2^{0.9}\rfloor)+T(n-\lfloor n^{0.9}\rfloor)=T(1)+T(1)=2=n$

Inductive Hypothesis: Assume $T(m)=m$ for all $m<n$.

Inductive Step: Let $a=\lfloor n^{0.9}\rfloor, b=n-\lfloor n^{0.9}\rfloor$. Then, $T(n)=T(a)+T(b)$. We know that a and b are both at most $n-1$. Thus, by the inductive hypothesis, $T(n)=a+b=n$

Conclusion: We have proved by strong induction that $T(n)=n$. Hence, $T(n)=O(n)$, showing that $T(n)$ has a polynomial growth rate.

(d*) This problem is meant to be $T(n)=T(n-1)+T(\lfloor n^{0.9}\rfloor)$

Unravel the first term, we have

$$\begin{aligned}
T(n) &= T(n-2)+T(\lfloor(n-1)^{0.9}\rfloor)+T(\lfloor n^{0.9}\rfloor)\\
&= T(1)+T(\lfloor(n-(n-2))^{0.9}\rfloor)+T(\lfloor(n-(n-3))^{0.9}\rfloor)+\ldots+T(\lfloor n^{0.9}\rfloor)\\
&= T(1)+\sum_{k=0}^{n-2}T(\lfloor(n-k)^{0.9}\rfloor)\\
&\leq nT(\lfloor n^{0.9}\rfloor)
\end{aligned}$$

This could be solved similar to part (c).