CSE Template

Rohan Mukherjee

May 8, 2024

P1.

Our algorithm is as follows: We prove the following lemma.

Algorithm 1 Independent Set Approximation

```
procedure INDP(G)
    $v \leftarrow$ a vertex in $G$
    $S \leftarrow \{v\}$
    for $u$ a neighbor of $v$ do
        $G \leftarrow G - u$
    end for
    return $\{v\} \cup$ INDP($G$)
end procedure
```

Lemma 1. *If G is a graph with $\Delta \leq k$, $\text{OPT}(G)$ is the optimal independent set of G, $v \in G$ is a vertex, and S is the set of neighbors of v, then*

$$|\text{OPT}(G)| \leq |\text{OPT}(G \setminus \{v\} \cup S)| + k.$$

Proof. Either v is in the optimum or not. If so, none of the neighbors of v are in the optimum by the property of being an independent set, and we get that $\text{OPT}(G) = \text{OPT}(G \setminus \{v\} \cup S) \cup \{v\}$, which shows that

$$|\text{OPT}(G)| = |\text{OPT}(G \setminus \{v\} \cup S)| + 1 \leq |\text{OPT}(G \setminus \{v\} \cup S)| + k.$$

The set $\text{OPT}(G) \setminus S$ is an independent set in $G \setminus \{v\} \cup S$, and thus we must have $|\text{OPT}(G) \setminus S| \leq |\text{OPT}(G \setminus \{v\} \cup S)|$. Now, if $S \subset \text{OPT}(G)$ then $|\text{OPT}(G) \setminus S| = |\text{OPT}(G)| - |S|$, otherwise we remove fewer than $|S|$ elements from OPT(G). In any case,

$$\begin{aligned}
&|\mathrm{OPT}(G)| - |S| \leq |\mathrm{OPT}(G \setminus \{v\} \cup S)| \\
\implies &|\mathrm{OPT}(G)| \leq |\mathrm{OPT}(G \setminus \{v\} \cup S)| + |S| \\
\implies &|\mathrm{OPT}(G)| \leq |\mathrm{OPT}(G \setminus \{v\} \cup S)| + k.
\end{aligned}$$

Since by hypothesis $|S| \leq k$. □

We now prove the correctness of the above algorithm with the following theorem.

Theorem 1. *Let G be a graph with n vertices and $\Delta \leq k$. Then,*

$$|\mathrm{OPT}(G)| \leq k \cdot |\textsc{Indp}(G)|.$$

Proof. We prove the claim by induction. Clearly the claim is true for all graphs of size 1, since the only nonempty independent set is the graph itself and our algorithm clearly finds that one. Suppose the claim is true for all graphs G sastisfying $\Delta \leq k$ with $< n$ vertices. Let v be the vertex chosen by the algorithm, and let S be v's neighbors. Then by the inductive hypothesis we know that

$$|\mathrm{OPT}(G \setminus \{v\} \cup S)| \leq k \cdot |\textsc{Indp}(G \setminus \{v\} \cup S)|$$

Also, by construction our algorithm has $\textsc{Indp}(G) = \{v\} \cup \textsc{Indp}(G \setminus \{v\} \cup S)$, and thus we have $|\textsc{Indp}(G)| = 1 + |\textsc{Indp}(G \setminus \{v\} \cup S)|$. By the lemma we have that

$$|\mathrm{OPT}(G)| \leq |\mathrm{OPT}(G \setminus \{v\} \cup S)| + k \leq k \cdot |\textsc{Indp}(G \setminus \{v\} \cup S)| + k = k \cdot |\textsc{Indp}(G)|.$$

Which completes the proof. □

Recall that it takes $O(n + m)$ time to remove a vertex from a graph G using the adjacency list. Thus, the middle loop of our algorithm takes $\leq O(k(n + m))$ time, since v has at most k neighbors. Thus our algorithm satisfies:

$$T(n) \leq T(|G \setminus \{v\} \cup S|) + O(k(n + m)) \leq T(n - 1) + O(k(n + m)).$$

Which shows that $T(n) = O(k(n^2 + nm))$.

P2.

This is the table for the DP knapsack algorithm I got, where the (i, j) position is the maximum value that can be obtained using the first i items and a knapsack of size j:

Table 1: Knapsack Data

n/wgt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	1	1	2	3	3	3	3	3	3	3	3	3	3	3
3	0	1	1	2	3	4	5	5	6	7	7	7	7	7	7
4	0	1	1	2	3	4	5	5	6	7	7	8	9	10	10
5	0	1	1	2	3	4	5	5	6	7	8	8	9	10	11

P3.

Our algorithm is as follows:

Algorithm 2 Make Change Procedure

```
procedure MINIMUM-COINS($v_1, \dots, v_n, k$)
    $M \leftarrow$ an $n \times k$ matrix of $\infty$
    for $c = 1$ to $k/v_1$ do
        $M[1, v_1 \cdot c] \leftarrow c$
    end for
    for $i = 2$ to $n$ do
        for $j = 1$ to $k$ do
            $M[i, j] \leftarrow \min_{0 \le \ell \le j/v_i} M[i-1, j - \ell \cdot v_i] + \ell$
        end for
    end for
    return $M[n, k]$
end procedure
```

We start with the running time. The first for loop takes $O(k)$ time. The inner for loop takes $O(k^2)$ time, since $j/v_i \le j \le k$ and we run this loop k times. Thus the outer for loop takes $O(nk^2)$ time, so the total running time of the algorithm is $O(nk^2)$.

We now prove the correctness of the above algorithm. We claim the matrix $M[i, j]$ represents the minimum number of coins needed to make change for j using only the first i coins, taking a value of ∞ if no way exists. We prove this by induction. The first coin will only be able to make change for j if j is a multiple of v_1, and in that case we will need j/v_1 coins. This is precisely what the first for loop accomplishes for each $j \le k$. Now suppose that M takes on the correct values for the first $n-1$ rows and k columns of those rows. I claim that the minimum number of coins needed to make change for k using n coins is equal to

$$\min_{0 \le \ell \le k/v_n} M[n-1, k - \ell \cdot v_n] + \ell.$$

This is because we are simply partitioning over the number of times we use the nth coin. If we use the nth coin ℓ times, then we need to make change for $k - \ell \cdot v_n$ using the first $n-1$ coins, and the total number of coins will be the minimum number of coins needed for that plus the ℓ copies of the nth coin. Since the minimum will have between 0 and k/v_n copies of the nth coin, we are taking the minimum over all possibilities and thus the claim is proven, which shows the algorithm is correct.

P4.

Our algorithm is as follows, where we use the convention that if $i < 0$ then $C[i] = 0$ for every list C:

Algorithm 3 No Long Consecutive Subsequences Procedure

```
procedure NLCS($n, k_A, k_B$)
    $EA, EB \leftarrow [1], [1]$
    for $i = 2$ to $n$ do
        $EA[i] \leftarrow \sum_{\ell=1}^{k_A} EB[n-\ell]$
        $EB[i] \leftarrow \sum_{\ell=1}^{k_B} EA[n-\ell]$
    end for
    return $EA[n] + EB[n]$
end procedure
```

First, we can see that the sum inside the for loop will take $O(k_A + k_B)$ time, and the loop runs n times thus our algorithm runs in $O(n(k_A + k_B))$ time.

We now prove the correctness of this algorithm. We claim that $EA[n]$ and $EB[n]$ represent the number of valid strings with n letters that end in A and B respectively. Clearly the claim is true for $n = 1$ beacuse $k_A, k_B \geq 1$ and there is precisely 1 string that ends in A, B respectively. Suppose the claim is true for $< n$. If S is a valid string with n letters ending in A, then the consecutive subsequence containing the last A has length $1 \leq \ell \leq k_A$. The final observation is that the number of valid strings on n letters which has a length ℓ consecutive subsequence as 0s on the right is the same as the number of strings on $n - \ell$ letters ending in B, since we have fixed the last ℓ letters to be A and we need this sequence to stop at letter $n - l$–i.e., it the $n - l$th letter needs to be B. Applying symmetric logic to B and then using the inductive hypothesis shows that EA, EB hold the right values. Finally, every valid subsequence on n letters either ends in A or B, which proves the correctness of the algorithm.

P5.

Let $d(x, y) = |x - y|$ be the usual Euclidean metric over $\mathbb{R}$. Our algorithm is as follows:

Algorithm 4 Optimal Village Cost Procedure

```
procedure OPTIMAL-VILLAGE-COST($x_1, \ldots, x_n, K$)
    Sort $x_1, \ldots, x_n$
    $A \leftarrow$ an $n \times n \times K$ tensor
    $Closer \leftarrow$ an $n \times n \times n$ tensor
    $D \leftarrow$ an $n \times n \times n$ tensor
    for $1 \le i \le n$ do
        for $1 \le l < u \le n$ do
            $D[i, l, u] \leftarrow \sum_{\alpha=l}^{u} d(x_\alpha, x_i)$
        end for
    end for
    for $1 \le i < j \le n$ do
        $Closer[i, j] \leftarrow \min\left\{\ell : d(x_\ell, x_i) \le d(x_\ell, x_j)\right\}$
    end for
    for $1 \le i < j \le n$ do
        $A[i, j, 1] \leftarrow D[j, 1, i]$
    end for
    for $k = 2$ to $K$ do
        for $k \le i, j \le n$ do
            $A[i, j, k] \leftarrow \min_{k-1 \le \ell \le j} A[Closer[j, \ell] - 1, \ell, k - 1] + D[j, Closer[j, \ell], i]$
        end for
    end for
    return $\min_{K \le j \le n} A[n, j, K]$
end procedure
```

Let $OPT(i, j, k)$ for $k \le j \le i$ be the optimal sum of the distances from each village to its nearest post office, where we use only the first i villages, the rightmost post office is at position j, and we are allocated k post offices. It is easy to see that if $k = 1$, then $OPT(i, j, 1)$ is just the sum of the distances from the first i villages to the post office at position j, because there is only one post office and it is at position j.

We shall now prove the recursive formula in the algorithm. The main idea is to guess where the second rightmost post office is. Let $k \le j \le i$ be fixed. Since we have used the kth post office by putting it at position j, we see that the second rightmost post office must be at position $\ge k - 1$ otherwise there could not possibly be $k - 1$ post offices placed to the left and including it. So we guess the position of the second rightmost post office to be at $k - 1 \le \ell < j$. Now we want to find the distance from the first i villages to their nearest post office. Since j is the rightmost post office, we see obviously that for $k > j$ the kth village is certainly closest to j. However, between ℓ and j could be incredibly complicated and intertwined.

We use the following trick: let $Closer[j, \ell]$ be the index of the leftmost post office that is closer to j than to ℓ. This is certainly less than j because j is closer to j itself than to ℓ. Since every other post office is to the left of ℓ, the post offices that are closest to j are precisely those with index between $Closer[j, \ell]$ and i. Now, to find the optimal placement of the villages with rightmost post office j and second rightmost post office ℓ, we simply need to look for the optimal position of the villages that are closer to ℓ than to j (which is just those with index strictly less than $Closer[j, \ell]$ by construction), with ℓ as the rightmost post office, and then finally add the distances from every village that is closer to j than to ℓ (also by construction just those that come after and including $Closer[j, \ell]$). Since the second rightmost post office is certainly somewhere left of j (since the villages are in sorted order), taking the minimum over all (feasible) ℓ indeed finds the right value. Putting all this together we get the following formula:

$$OPT(i, j, k) = \min_{k-1 \leq \ell < j} OPT(Closer[j, \ell] - 1, \ell, k - 1) + \sum_{\alpha = Closer[j,\ell]}^{i} d(x_\alpha, x_j).$$

The last step in the algorithm is to simply take a minimum over all possible rightmost post office locations, which shows correctness.

The sorting procedure clearly take $O(n \log n)$ time as per the usual. The first for loop takes $O(n^4)$ time, since the innermost step takes $O(n)$ time, which is being repeated $O(n^2)$ time for the $i < j$ part, which is being repeated again $O(n)$ times. The second for loop runs over all $i < j$ and takes $O(n)$ time per iteration, which gives us a total of $O(n^3)$ time. The second for loop follows similarly to be $O(n^3)$. The signle line in the triple nested for loop can be computed in $O(n)$ time, since ℓ runs through $\leq n$ values. This single line is then repeated $O(n^2 \cdot K)$ times, yielding an incredible final running time of $O(n^3 K) \leq O(n^4)$. Clearly the final line can be computed in $O(n)$ time, which shows our algorithm runs in polynomial time.