Homework 6

Date: 2023-07-25 06:50
type: #Homework
Category: 📌cs 311 MOC

Greedy Algorithms

Question 1

> Let $m_1, m_2, \dots m_n$ be distinct numbes on the number line, in the increasing order. Your goal is to color all of them green. You have magical green pents with the following property: When you place the pen at coordinate x, all the points in the range $[x-5, x+5]$ turn green. A pen can be used only once. Given an algorithm to color all the points using as few pens as possible. Prove the correctness of the algorithm and derive the runtime.

```
colorPoints(int[] points) {
 pensUsed = 0
 i = 0

 while i < len(points) {
   currentPoint = points[i]
   j = i + 1

   while j < len(points) and points[j] - currentPoint <= 10 {
     j += 1
   }

   pensUsed += 1
   i = j
 }

 return pensUsed
}
```

1. Start with the leftmost point, m_1.
2. Scan to the right until you find a point m_j that is more than 10 units away from m_1.
3. Color the point m_{j-1}, which ensures all points from m_1 to m_{j-1} are colored.
4. Repeat the process starting from m_j until all points are colored.

Proof of correctness:

We need to prove that this algorithm uses the minimum number of pens. The greedy choice is to color as many points as possible with each pen. We ensure this by selecting the point as far right as possible within the 10-unit range of our first selected point. This approach guarantees that we cover the maximum possible number of points with one pen.

Let's suppose there's an optimal solution that uses fewer pens. This means that there's a pen in the optimal solution that colors more points than one pen in our greedy solution. But this contradicts our greedy approach, because in our method we always color the maximum possible points with a pen, therefore, our solution is optimal.

Runtime analysis:

The algorithm visits each point exactly once, so the time complexity is $O(n)$, where n is the number of points.

Question 2

> A bakery gets order for many different kinds of cakes with fancy decorations. They have plenty of guys who can decorate cakes (at least n of them), plenty of supplies for decorations, but there is only one oven, and it can only hold one cake at a time. Each cake C_i required B_i minutes of baking time by itself in the oven, plus D_i minutes of decoration time, which can occur in parallel with other cakes being decorated or baked. Note that any given cake cannot be decoared unless it has already been baked. Write an algorithm that, given a set of n cakes, determines a schedule $C_i, C_2, \ldots, C_n$ such that all the cakes are completely finished in the shortest overall time. Derive the runtime and prove the correctness of your algorithm.

```
scheduleCakes(cakes: list of (bakeTime, decorateTime)) {
 cakes.sort(decreasing order)

 totalTime = 0
 bakeTime = 0
 for cake in cakes {
   bakeTime += cake[0]
   // We try to finish decoration of the current cake as the next cake finishes baking
   totalTime = max(totalTime, bakeTime) + cake[1]
 }

 return totalTime
}
```

The idea here is to sort the cakes in decreasing order of their decoration time and then bake the cakes in this order. This way, we can decorate a cake while another one is being baked, aiming to have the decoration of the current cake finish as close as possible to when the next cake finishes baking, thereby minimizing idle time.

1. Sort the cakes in descending order of their decoration time.
2. Schedule the cakes in this order for baking.
3. As soon as a cake is done baking, start decorating it.

Proof of correctness:

Let's consider an optimal schedule and the first cake that differs from our schedule. This cake has a smaller decoration time than the cake at the same position in our schedule. If we swap

these two cakes, the time when we finish decorating the cake from the optimal schedule will not change because this cake is decorated while a cake with at least as long bake time is being baked in the oven. However, the time when we finish decorating the swapped cake might decrease because we're now decorating it while a cake with longer baking time is in the oven. Therefore, the new schedule is at least as good as the optimal schedule, which implies that our schedule is indeed optimal.

Runtime analysis:

Sorting the cakes takes $O(n \log n)$ time, where n is the number of cakes. The for-loop takes $O(n)$ time because we're iterating over all cakes once. Therefore, the total time complexity of the algorithm is $O(n \log n)$.

Question 3

> You are given n jobs numbered $1, 2, \ldots, n$ to complete and each job i comes with a difficuly d_i. Each job takes exactly one week to complete irrespective of its difficult. If you complete job i during week $j (\leq n)$, then you ear a prodit of $d_i \cdot (n - j)$. Your objective is to maximize the sum of of the profit. Consider a greedy strategy that completes the jobs in the decreasing order of difficuly. Prove/disprove that such a strategy produces an optimal solution.

The proposed greedy strategy sorts the jobs in decreasing order of difficulty and completes them in that order. According to this strategy, the most difficult job is done in the first week, the second most difficult job in the second week, and so on.

Proof:

The profit from job i done in week j is $d_i \cdot (n - j)$. So, the profit depends on the difficulty of the job and the week it is done, with later weeks leading to less profit.

Suppose there is an optimal solution that disagrees with the greedy solution. Let job i be the first job where the two solutions disagree, with $d_i > d_k$ in the optimal solution and job k being done in the i^{th} week instead in the greedy solution. The profit from job i in the optimal solution will be $d_i \cdot (n - i)$ and for job k will be $d_k \cdot (n - k)$.

We can swap job i and job k in the optimal solution. The profit from job i after swapping will be $d_i \cdot (n - k)$ and for job k will be $d_k \cdot (n - i)$. Since $d_i > d_k$ and $n - k > n - i$ (since $i < k$), it can be seen that the profit after swapping is more than or equal to the profit in the original optimal solution. Hence, the greedy solution gives an optimal solution.

Therefore, the greedy strategy that completes the jobs in the decreasing order of difficulty indeed produces an optimal solution to maximize the sum of the profit.

The runtime of this algorithm is $O(n \log n)$, which is the time taken to sort the jobs based on their difficulties.