CSE 6140 Assignment 1 Solutions

September 25, 2017

1 Algo and complexity

1. The complexity in $O(\log(n))$ is a hint: One should use a binary search. Indeed, if we have $k \geq \lceil \log(n) \rceil$, we know the result for the floors whose indices range from i to j by dropping a box from the m-th floor where $m = \lfloor \frac{i+j}{2} \rfloor$ and then by iterating with floors i to $m-1$ if the box broke, and by iterating with floors m to j otherwise. The principle of the binary search guarantees that we will obtain the desired result (when $i = j$) and in at most $\lceil \log(n) \rceil$ steps, and, thus, after having broken at most $\lceil \log(n) \rceil$ boxes.

2. As we have only $k < \lceil \log(n) \rceil$ boxes, we cannot directly apply a binary search. We, however, will solve this problem in a simple way. We apply the binary search using $k-1$ boxes in order to narrow as much as possible the search interval around the desired floor. We then use the last box to scan the remaining interval floor by floor, from the lowest to the highest. After throwing $k-1$ boxes, if the target floor has not been identified, there are at most $n/2^{k-1}$ floors in the search interval, hence a worst-case complexity of $O(k + n/2^{k-1})$.

3. When $k = 2$ we do not want to have to test each floor one after the other, thereby ending up with a linear complexity. We, therefore, will adapt the idea of narrowing the search interval. We partition the set of floors into "slices" of $\sqrt{n}$ floors (assume that n is a square without loss of generality, or use ceil functions). Then we throw the first box from the first floor of each slice until it breaks, starting with the lowest slice. Then, we return to the last tested floor, m, from which the box was dropped but did not break. We then test one by one the floors using the second box. We start with floor $m+1$ and, in the worst case, we go up to the floor from which the first box broke. There are, by construction, $\sqrt{n}$ floors in that slice. Therefore, we have two series of tests, with $O(\sqrt{n})$ tests in each of them. Hence, the overall complexity is $O(\sqrt{n})$.

2 Greedy 1

2.1 Algorithm

Let us visualize the leaks as points $l_1, l_2 ... l_n$ lying along a straight line, where l_n is the location of the n^{th} leak. Without loss of generality suppose the strips are given in sorted order; i.e., $\forall i < j$, $l_i < l_j$. Let the length of the strip be s. The greedy algorithm is as follows.

1. Start with $i = 1$
2. Lay the first strip at point l_i. This will cover all leaks $l_i ... l_j$ in the interval $[l_i, l_i + s]$ where $l_j \leq l_i + s$
3. Update $i = j + 1$
4. Repeat steps 2 and 3 until $i > n$ (all leaks have been covered)

The algorithm has a runtime complexity of $\mathcal{O}(n)$. Note: if you assume that the strips are not sorted and your algorithm sorts them initially, the total complexity is then $\mathcal{O}(n \log n)$.

Let the solution obtained by the greedy algorithm be represented as $G = \{G_1, G_2, \cdots, G_t\}$, where G_i is a point on the line/strip representing the location of the starting point of the strip covering the interval on the pipe from $[G_i, G_i + s]$. Without loss of generality, we say that for $\forall i < j$, we have $G_i < G_j$, i.e., that the solution is given by strips placed in order from left to right. Using a similar notation, the optimal solution will be represented as $O = \{O_1, O_2, \cdots, O_r\}$.

2.2 Greedy Choice & Optimal Substructure Proof

2.2.1 Greedy Choice Property

Now O_1 lies from point $[O_1, O_1 + s]$ and must cover the first leak, l_1 (in order to be a feasible solution, it must cover all the leaks). By definition of the concept of a greedy choice, $G_1 = l_1$, since the greedy solution places the first strip as far right as possible while still covering l_1, it follows that $O_1 \leq G_1$.

Now let O' be the solution obtained by replacing O_1 with G_1 in O. Since l_1 is the first and leftmost leak, we know that there is no other leak lying in the space from $[O_1, G_1)$. Consequently, we see that strip at G_1 covers all the leaks that strip at O_1 covers. Hence, $O' = \{G_1, O_2, \cdots O_r\}$ is also an optimal solution.

2.2.2 Optimal Substructure Property

Let the optimal solution to the entire problem P be O. After placing the first strip at O_1, we are left with the task of solving the problem P', which is to cover all leaks to the right of the point $O_1 + s$. Let the optimal solution to P' be O'. Since, cost(P) = cost(P') + 1 (strip), the globally optimum solution O, contains the solution O'.

2.3 Exchange Argument Proof

- Suppose greedy is not optimal. Take the optimal solution that agrees with the greedy one for the longest time: Suppose there is some k such that $\forall i < k$, $G_i = O_i$. That is, $G_k \neq O_k$. Now let us examine the optimal solution that agrees with greedy for the maximum value of k.
- Examine the k^{th} element in both solutions, so $G_k \neq O_k$. We know that $O_k < G_k$, that is, the starting point of the optimal solution's k^{th} strip is before the starting point of the greedy solution's k^{th} strip. This is because:

    - The greedy algorithm, by definition, places strips left aligned with the location of a leak. Therefore, G_k is representing the strip covering the next unconvered leak in order on the pipe that strips $1 \cdots k-1$ are not covering. Call this leak l'.
    - If our optimal solution places the k^{th} strip after the greedy's strip, then it would not cover the leak l'. Therefore, it must begin before the greedy's strip to be a feasible solution and we thus have $O_k < G_k$.

- Suppose we swap the two elements in question to create a new solution $O' = \{O_1, O_2, \cdots, O_{k-1}, G_k, O_{k+1}, \cdots\}$. We wish to show that this new solution is still both feasible and optimal.
    - Since $O_k < G_k$, the strip represented by G_k covers at least all the leaks covered by the strip represented by O_k. Therefore, we are still covering all the leaks from the original optimal solution O and our new solution O' is still feasible.
    - We have not increased the number of strips used in the optimal solution, we have merely changed the location of the k^{th} strip; hence the solution is still optimal (uses the same number of strips as the original optimal solution).
- We started with the assumption that we had the optimal solution that agreed with greedy for maximum k. However, we have shown that we can increase this value of k with the previous argument while preserving feasibility and optimality. Therefore, we have obtained a contradiction to our original assumption and thus greedy is optimal.
- Alternatively, we can continue swapping elements as done before until we completely transform the original optimal solution into our greedy one, without increasing the number of strips.

3 Greedy 2

3.1 Algorithm

- Compute the value density d_i of each item, v_i/w_i.
- Sort items by decreasing density.
- Take the items with the highest value density until your bag is full, possibly taking a fracional amount of the last item.

3.2 Proof (Exchange Argument)

Consider an optimal selection of objects and amounts O, and the selection produced by the greedy algorithm G. We will assume that there is enough weight to fill the bag. Otherwise the greedy algorithm and any optimal solution will take everything and will therefore be equal.

Consider the case where the two are not equal. Then G and O disagree on the value of some set of items I. Let $i \in I$ be the item of highest density for which O and G have differing amounts. As the greedy algorithm always takes as much as it can of the next highest density item, G must have more of i than O. Say G has amount g_i and O has o_i. We can take the amount $g_i - o_i$ from items with less value density than i and add this amount of item i to O to create O'. Note that this transformation increases the value by $(g_i - o_i) * d_i - (g_i - o_i)d'$ with $d' \leq d_i$ the density of items removed from O to make space. As this value is greater than or equal to 0, we have not worsened the solution. This transformation is always possible as we assume both bags are full. (Specifically, the sum of the weight of all items with value density less than or equal to i is the same in both G and O.)

If we continue to apply this transformation, we will eventually turn O into G without worsening the solution. Since O is optimal, G is also optimal.

4 Programming

4.1 Static Computation

Either an implementation of Prim's or Kruskal's as gone over in class would suffice.

4.2 Dynamic Recomputation

Consider the addition of edge $e = (u, v)$ into the graph G. Adding this to the MST creates a cycle. By the cycle property, we know that the MST will not contain the maximum edge of any cycle in the graph, so we find the cycle created and delete the maximum edge. We thus have our new MST.