CS 161 (Stanford, Fall 2025) Section 7

1 Warm-up: Greedy or Not?

Sometimes it can be tricky to tell when a greedy algorithm applies. For each problem, say whether or not the greedy solution would work for the problem. If it wouldn't work, give a counter example.

1. You have unlimited objects of different sizes, and you want to completely fill a box with as few objects as possible, or output that it is impossible. (Greedy: Keep putting the largest object possible in for the space you have left)
2. You have unlimited objects of size 3^k for every k, and you want to completely fill a box with as few objects as possible. (Greedy: same approach as the previous problem)
3. You have lines that can fit a fixed number of characters. You want to print out a series of words in a given order while using as few lines as possible. (Greedy: Fit as many words as you can on a given line)
4. There are n hotels in a line, each distance 1 apart and hotel i costing h_i dollars to stay at. You can travel at most distance k every day. Find the minimum total cost of hotels you need to stop at to go from hotel 1 to hotel n. (Greedy: Go as far as you can before stopping at a hotel)

Solution

1. Greedy does not work! Consider a box of size 14 and objects of size 10, 7, and 1.
2. Greedy works! This is basically how you would write a number in base 3.
3. Greedy works!
4. Greedy does not work! Consider hotel costs [10, 20, 100, 10], where each hotel is 1 apart and k = 2.

2 Cutting Ropes

Suppose we are given n ropes of different lengths, and we want to tie these ropes into a single rope. The cost to connect two ropes is equal to sum of their lengths. We want to connect all the ropes with the minimum cost.

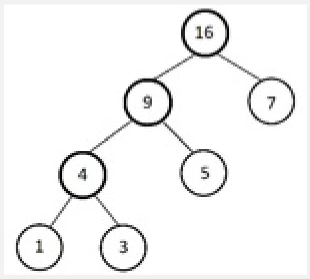
For example, suppose we have 4 ropes of lengths $7, 3, 5$, and 1. One (not optimal!) solution would be to combine the 7 and 3 rope for a rope of size 10, then combine this new size 10 rope with the size 5 rope for a rope of size 15, then combine the rope of size 15 with the rope of size 1 for a final rope of size 16. The total cost would be $10 + 15 + 16 = 41$. (Note: the optimal cost for this problem is 29. How might you combine the ropes for that cost?)

Find a greedy algorithm for the minimum cost and prove the correctness of your algorithm.

Solution

Solution: Always combine the smallest ropes available to you until you have one single rope. Using data structures such as Red-Black Trees, this can be implemented in $O(n \log n)$ time.

Justification: We can write the strings as a graph, where the leaves are the original ropes and every node with two children is a sum of two ropes.



(from the example)

From here we can use the idea from our Huffman analysis- that is, we can prove that if x and y are ropes with the shortest length, there is an optimal tree where they are siblings. Likewise, we can prove that if we treat the nodes at a given level as leaves, we can still apply the above argument.

Inductive hypothesis. By combining the two smallest ropes available to us at any given point, there is a minimal solution that extends the current solution.

Base case. When we haven't combined any of the ropes, there is clearly a minimal solution that extends the current (empty) solution.

Inductive step. Suppose that we have combined ropes k times (meaning there are $n - k$ ropes remaining). We know that we can treat previously combined ropes the same as ropes that haven't been combined. Since there is an optimal solution where the shortest length ropes are 'siblings' to a parent node that's the sum of them, it follows that there is an optimal solution where the smallest ropes available are tied together.

Conclusion. By the n^{th} step, we have not ruled out the optimal solution. Therefore, the solution we chose is optimal.

3 Mice to Holes

There are n mice and n holes along a line. Each hole can accommodate only 1 mouse. A mouse can stay at his position, move one step right from x to $x+1$, or move one step left from x to $x-1$. Any of these moves consumes 1 minute. Mice can move simultaneously. Assign mice to holes such that the time it takes for the last mouse to get to a hole is minimized, and return the amount of time it takes for that last mouse to get to its hole.

Example:

Mice positions: $[4, -4, 2]$

Hole positions: $[4, 0, 5]$

Best case: the last mouse gets to its hole in 4 minutes. $\{4 \rightarrow 4, -4 \rightarrow 0, 2 \rightarrow 5\}$ and $\{4 \rightarrow 5, -4 \rightarrow 0, 2 \rightarrow 4\}$ are both possible solutions.

Solution

Solution: Sort the mice locations and the hole locations. For $1 \leq i \leq n$, have the i^{th} mouse go to the i^{th} hole. The maximum distance will be the max distance between each mouse and its corresponding hole. The total runtime is $O(n \log n)$ due to sorting the locations.

Justification:
Lemma: For $i_1 < i_2$, $j_1 < j_2$ and $dist(x, y) = |x - y|$,

$$\max(dist(i_1, j_1), dist(i_2, j_2)) \leq \max(dist(i_1, j_2), dist(i_2, j_1))$$

Proof. Without loss of generality, let's say $i_1 \leq j_1$. Our three cases are then that $i_1 \leq i_2 \leq j_1 \leq j_2$, $i_1 \leq j_1 \leq i_2 \leq j_2$, or $i_1 \leq j_1 \leq j_2 \leq i_2$. In any of these cases, the lemma holds (you should verify this!).

Inductive hypothesis. By sending the j^{th} (sorted) mouse to the j^{th} (sorted) hole for every $j \in \{1, \ldots, i\}$, there is a minimal solution that extends the current solution.

Base case. If we haven't sent any mice to any holes, we haven't eliminated the ideal solution.

Inductive step. Suppose that we have sent the first $k-1$ sorted mice to the first $k-1$ sorted holes. Now suppose there is an optimal solution where the k^{th} mouse is sent to the p_0^{th} hole, where $k < p_0 < n$, the p_0^{th} mouse is sent to the p_1^{th} hole, and so on until the p_d^{th} (for some d) mouse is sent to the k^{th} hole. We could then swap the p_0^{th} hole with the k^{th} hole; we know by our lemma that the result will not be worse than the optimal solution. Therefore, by sending the k^{th} mouse to the k^{th} hole, we have not eliminated an optimal solution.

Conclusion. By the n^{th} step, we have not ruled out the optimal solution. Therefore, the solution we chose is optimal.

4 MST With Leaf Requirements

We are given an undirected weighted graph $G = (V, E)$ and a set $U \subseteq V$. Describe an algorithm to find a minimum spanning tree such that all nodes in U are leaf nodes. (The result may not be an MST of the original graph G.)

Solution

Let $T = V \setminus U$ be the set of nodes we don't require to be leaves, and let $D \subseteq E$ be the edges between nodes in T. Create an MST on (T, D) using, say, Prim's algorithm. Then add nodes in U to this tree by taking the lightest edge from a node $i \in U$ to any node $t \in T$. The total runtime is $O(n \log n + m)$ by using a Fibonacci Heap to implement the Prim's algorithm.

We show that any solution must have an MST on T as a sub-graph. Let S' be some optimal MST satisfying the leaf requirements. Since each leaf by definition has degree 1, removing the leaf nodes of U does not affect the connectedness of the remaining nodes. Once we remove from S' all the leaf nodes of U, we are left with a spanning tree over the nodes of T. Now, suppose for the sake of contradiction that this spanning tree was not a MST over T. Then taking the MST over T in conjunction with the leaf edges we just removed would yield a lower cost spanning tree than S', which is a contradiction. Therefore, some MST on T must be contained in S'.

Therefore, our solution gives us a minimum-weight solution because otherwise there would be a lower-weight solution to the graph (T, D).