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. (Greedy: Keep putting the largest object possible in for the space you have left)

2. You have unlimited objects, all of which are size $3^k$ for some integer 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 fixed series of words while using as few lines as possible. (Greedy: Fit as many words as you can on a given line)

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!

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: Always combine the smallest ropes available to you until you have one single rope.

Justification:

Lemmas 1 and 2: 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 here we can use the lemma 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 (lemma 1). Likewise, we can prove that if we treat the nodes at a given level as leaves, we can still apply lemma 1 (lemma 2).

**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). Lemma 2 tells us that we can basically treat previously combined ropes the same as ropes that haven’t been combined, and Lemma 1 tells us that there’s an optimal solution where the shortest length ropes are ‘siblings’ to a parent node that’s the sum of them — in other words, there’s 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.

---

**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:** Sort the mice locations and the hole locations. For $0 <= i < n$, 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.

**Justification:**

**Lemma:** For $i_1 < i_2, j_1 < j_2$ and $\text{dist}(x, y) = |x - y|$.

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

Without loss of generality, let’s say $i_1 <= j_1$. Our two 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.

**Inductive hypothesis.** By sending the $i^{th}$ (sorted) mouse to the $i^{th}$ (sorted) hole, 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.

**Roads and Airports**

Given a set of \( n \) cities, we would like to build a transportation system such that there is some path from any city \( i \) to any other city \( j \). There are two ways to travel: by driving or by flying. Initially all of the cities are disconnected. It costs \( r_{ij} \) to build a road between city \( i \) and city \( j \). It costs \( a_i \) to build an airport in city \( i \). For any two cities \( i \) and \( j \), we can fly directly from \( i \) to \( j \) if there is an airport in both cities. Give an efficient algorithm for determining which roads and airports to build to minimize the cost of connecting the cities.

To find the roads and airports to build, we first note that there are two cases: either we do not build any airports or we build at least one airport.

To consider the case where we do not build any airports, we construct an undirected graph where the cities are the nodes and the roads are the edges with weights corresponding to the cost of building that road. We then construct the MST of this graph. This gives us the minimum construction cost using no airports. (If we constructed a non-tree connected graph, we could always remove a road to decrease cost without disconnecting the graph, so the optimal solution must be a tree.)

Then we consider the case where we choose to build at least one airport. To model this, we construct a slightly different graph. We start with the same graph from the previous case: an undirected graph where the cities are the nodes and the roads are the edges with weights corresponding to the cost of building that road. We then add another node to the graph, representing the air. Call this node \( a \). We add an undirected edge between every city \( i \) and \( a \) with weight \( a_i \). We then construct the MST of this graph.

To find the overall minimum cost set of roads and airports to build, we use either the MST from the first case or the MST from the second case, whichever has lower total cost. For every edge in the MST between two cities \( i \) and \( j \), we build a road between \( i \) and \( j \), and for every edge between a city \( i \) and \( a \) the airport node, we build an airport in city \( i \).