Exercise 0

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

1. You have unlimited objects of different sizes, and you want to completely fill a bag with as few objects as possible. (Greedy: Keep putting in the largest object possible given the space you have left.)

2. You have unlimited objects, all of which are size $3^k$ for different integers $k$, and you want to completely fill a bag with as few objects as possible. (Greedy: Same approach as the previous part.)

3. You have lines that can fit a fixed number of characters. You want to print out a given piece of text while using as few lines as possible. (Greedy: Always fit as many words as you can on the next line.)

Solution 0

1. Greedy does not work! Consider a bag of size 14 and objects of size 10, 7, and 1.

2. Greedy works because the larger sizes are all multiples of the smaller sizes. This is basically how you would write a number in base 3.

   To formally prove this, we would need to inductively establish that each time the greedy algorithm adds an object, it doesn’t rule out an optimal solution. For instance, say we’re adding an object of size $s$. We could argue that either (a) the optimal solution contains enough objects of some smaller size $s'$ to add up to the next-largest size, which can’t actually be an optimal solution, or (b) the remaining blocks in the optimal solution can’t add up to a total size of $s$.

3. Greedy works!

   To formally prove this, we would need to inductively establish that each time the greedy algorithm ends a line, it doesn’t rule out an optimal solution. Intuitively, if we put more words on a given line than the optimal solution, this only frees up more space on subsequent lines so the optimal solution is still viable.

Exercise 1

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 at 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 to achieve that cost?)

Find a greedy algorithm for the minimum cost and prove the correctness of your algorithm.
Solution 1

Our algorithm will always combine the smallest ropes available until we have one single rope.

Observe that we can view a rope-tying procedure as forming a tree, where each time we tie two ropes together they become siblings of a new parent node. The root node represents all the ropes being tied together. The total cost is then the sum over every rope of the length of the rope times its depth in the tree. But this is equivalent to Huffman coding (seen in lecture), with rope length instead of letter frequency! The correctness proof is essentially identical.

Lemma 1: There exists an optimal solution where the two smallest ropes \( x \) and \( y \) are tied together first. As seen in class, consider two siblings \( a \) and \( b \) at the lowest level of the tree. If \( a \notin \{x, y\} \), then clearly swapping \( x \) or \( y \) for \( a \) can only decrease the total cost since \( |a| \leq |x|, |y| \) and \( a \) has the largest depth to start with. Thus we can swap both \( x \) and \( y \) in as lowest-level siblings without increasing the cost. Without loss of generality, we can assume these siblings are joined first.

Lemma 2: It doesn’t really matter whether we’re considering all original ropes, or ropes that may have been formed by joining smaller ropes. Intuitively, the optimal solution is the same whether a rope has length 7 or length 3 + 4.

Inductive hypothesis: After the \( t^{th} \) step there is an optimal solution extending the current ropes.

Base Case: After the 0\( ^{th} \) step any solution extends the current ropes.

Inductive step: Suppose after the \( (t - 1)^{th} \) step there exists an optimal solution extending the current ropes. By lemma 2, we can basically consider these all to be original ropes, and by lemma 1, there exists an optimal solution where the shortest ropes are combined first, which our algorithm will do as the \( t^{th} \) step.

Conclusion: After the last step, there is an optimal solution extending our single rope, i.e., we have chosen an optimal solution.

Exercise 2

Minimum graph coloring is an open NP-hard problem for finding the minimum number of colors needed to color all the nodes in a graph such that no nodes of the same color share an edge.

1. Although the problem is NP-hard, we can use greedy algorithms to obtain a pretty good solution. Describe a greedy algorithm that never uses more than \( d + 1 \) colors, where \( d \) is the maximum degree of a vertex in the given graph. Your algorithm should run in \( O(n^2) \) where \( n \) is the number of nodes.

2. Prove by counterexample that your greedy algorithm does not always return the correct minimum coloring. Your solution should include a graph, the correct minimum coloring, and the coloring returned by the greedy algorithm.

3. Prove that your greedy algorithm will return a coloring that uses at most \( d + 1 \) colors. (Note: You may use proof by induction, but you do not need to for this problem.)
Solution 2

1. Keep a list of colors used so far (initially empty). Loop over the vertices. For each vertex \( v \), assign the first color in the list which hasn’t already been used on a neighbor of \( v \); if all the colors have been used on \( v \)’s neighbors, add a new color to the list.

   The algorithm loops over the nodes and looks at every neighbor, of which there are at most \( n \). Thus, the runtime is \( O(n^2) \).

2. In the following counterexample, the greedy solution loops over the nodes in numerical order. (There always exists some ordering which would cause the greedy algorithm to return an optimal coloring, e.g., ordering the nodes by their color in an optimal coloring!)

   ![Counter-Example](image)

3. Since \( d \) is the maximum degree, a vertex cannot be attached to more than \( d \) vertices. When we color a vertex, at most \( d \) colors could have already been used by its adjacent vertices. To color this vertex, we need to pick the first color that is not used by the adjacent vertices. This is clearly possible when there are \( d + 1 \) colors even if every adjacent vertex has a distinct color.

   ![Correct Solution: 3 colors](image) ![Greedy Solution: 4 colors](image)
Exercise 3

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 3

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 Prim’s algorithm for example. Then, add nodes in $U$ to this tree by taking the lightest edge from a node $u \in U$ to any node $t \in T$.

This gives us a minimum-weight solution because any such solution having $U$ as leaves must have an MST on $T$ as a sub-graph—otherwise there would be a lower-weight solution to the original problem.

Exercise 4

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.

Solution 4

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 $i$. 

4