Exercises

Please do the exercises on your own.

1. (2 pt.) Consider the graph $G$ below.

   ![](image)

   (a) (1 pt.) In what order does Prim’s algorithm add edges to an MST when started from vertex $C$?

   (b) (1 pt.) In what order does Kruskal’s algorithm add edges to an MST?

   [We are expecting: For both, just a list of edges. You do not need to draw the MST, and no justification is required.]
2. (6 pt.) In this exercise we’ll look at a continuous variant of the knapsack problem that we saw in class. You have a knapsack with a capacity of $Q$ ounces and there are $n$ items; the difference between this exercise and the version that we saw in class is that you can take a fractional amount of each item. For example, perhaps one item is 3.6 ounces of brightly colored sand; you can choose to take 2.5235 ounces of sand for your knapsack if that’s how much you want.

Each item $i$ has a value per ounce $v_i > 0$ (measured in units of dollars per ounce) and a quantity $q_i > 0$ (measured in ounces). There are $q_i$ ounces of item $i$ available to you, and for any real number $x \in [0, q_i]$, the total value that you derive from $x$ ounces of item $i$ is $x \cdot v_i$.

Your goal is to choose an amount $x_i \geq 0$ to take for each item $i$ in order to maximize the value $\sum x_i v_i$ that you receive while satisfying:

1. you don’t overfill the knapsack (that is, $\sum x_i \leq Q$), and
2. you don’t take more of an item than is available (that is, $0 \leq x_i \leq q_i$ for all $i$).

Assume that $\sum q_i \geq Q$, so there always is some way to fill the knapsack.

(a) (0 pt.) Suppose that you already have partially filled your knapsack, and there is some amount of each item left. What item should you take next, and how much?

[We are expecting: Nothing, this part is worth zero points, but it’s a good thing to think about before you go on to the next part.]

(b) (3 pt.) Design a greedy algorithm which takes as input $Q$, along with the tuples $(i, v_i, q_i)$ for $i = 0, \ldots, n - 1$, and outputs tuples $(i, x_i)$ so that (1) and (2) hold and $\sum x_i v_i$ is as large as possible. Your algorithm should take time $O(n \log(n))$.

[We are expecting:
- Pseudocode AND an English explanation of what it is doing.
- An informal justification of the running time.
]

(c) (3 pt.) Fill in the inductive step below to prove that your algorithm is correct.

- **Inductive hypothesis:** After making the $t$'th greedy choice, there is an optimal solution that extends the solution that the algorithm has constructed so far.

- **Base case:** Any optimal solution extends the empty solution, so the inductive hypothesis holds for $t = 0$.

- **Inductive step:** (you fill in)

- **Conclusion:** At the end of the algorithm, the algorithm returns a set $S^*$ of tuples $(i, x_i)$ so that $\sum x_i = Q$. Thus, there is no solution extending $S^*$ other than $S^*$ itself. Thus, the inductive hypothesis implies that $S^*$ is optimal.

[We are expecting: A proof of the inductive step: assuming the inductive hypothesis holds for $t - 1$, prove that it holds for $t$.]
Problems

You may talk with your fellow CS161-ers about the problems. However:

- Try the problems on your own before collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

3. (6 pt.) [k-well-connected graphs.] Let $G = (V, E)$ be an undirected, unweighted graph with $n$ vertices and $m$ edges. For a subset $S \subseteq V$, define the subgraph induced by $S$ to be the graph $G' = (S, E')$, where $E' \subseteq E$, and an edge $\{u, v\} \in E$ is included in $E'$ if and only if $u \in S$ and $v \in S$.

For any $k < n$, say that a graph $G$ is $k$-well-connected if every vertex has degree at least $k$.

For example, in the graph $G$ below, the subgraph $G'$ induced by $S = \{a, b, c, d\}$ is shown on the right. $G'$ is 3-well-connected, since every vertex in $G'$ has degree at least 3. However, $G$ is not 3-well-connected since vertex $E$ has degree 2.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$G = (V, E)$</td>
<td>$G' = (S, E')$, for $S = {a, b, c, d}$</td>
</tr>
</tbody>
</table>

Design a greedy algorithm to find a maximal set $S \subseteq V$ so that the subgraph $G' = (S, E')$ induced by $S$ is $k$-well-connected. In the example above, if $k = 3$, your algorithm should return $\{a, b, c, d\}$, and if $k = 4$ your algorithm should return the empty set.

You may assume that your representation of a graph supports the following operations:

- $\text{degree}(v)$: return the degree of a vertex in time $O(1)$
- $\text{remove}(v)$: remove a vertex and all edges connected to that vertex from the graph, in time $O(\text{degree}(v))$.

Your algorithm should run in time $O(n^2)$.

You do not need to prove that your algorithm works, but you should give an informal (few sentence) justification.

[HINT: Think about greedily removing vertices.]

[We are expecting:

- Pseudocode AND an English description of what your algorithm is doing.
- An informal justification of the running time.
- An informal justification that the algorithm is correct.]
4. (10 pt.) [Fish Stops.] Plucky the Pedantic Penguin is walking $t$ miles across Antarctica. He needs to eat along the way, but he can only eat when there’s a fishing hole for him to catch fish. He can walk at most $m$ miles between meals, and he knows how $n$ fishing holes are laid out along his route.

Plucky is given an array $F$ so that $F[i]$ gives the distance from the start of his journey to the $i$’th fishing hole. There are $n$ fishing holes along the way, including at the beginning and the end: $F[0] = 0, F[n - 1] = t$. For example, the array $F = [0, 3, 4, 6, 10, 12]$, with $t = 12$ corresponds to the setup below:

Plucky wants to stop as few times as possible, given that he can walk at most $m$ miles without eating. (It is okay if he walks exactly $m$ miles between meals). He starts out hungry, so he will always fish at 0 miles; he will also always fish at his destination (at $t$ miles), whether or not he’s hungry.

In the example above, if $m = 4$, then Plucky should stop 5 times (including his stops at the beginning and the end), for example at 0, 4, 6, 10, 12 miles.

(a) (4 pt.) Design a greedy algorithm for Plucky to use. The algorithm should have the following properties:

- Your algorithm should take as input the array $F$, as well as the parameters $m$ and $t$. You may assume that $F$ is sorted.
- Your algorithm should output a list fishStops which contains a shortest list of places that Plucky could stop for fish. In the example above, the algorithm could output [0, 4, 6, 10, 12]. If Plucky cannot make it to his destination $t$ miles away, then your algorithm should return Stay Home.
- Your algorithm should run in time $O(n)$.

[We are expecting: Pseudocode AND an English description of what it is doing. You do not need to justify the running time.]

(b) (6 pt.) Prove by induction that your algorithm is correct. You may assume that there is a way for Plucky to make it $t$ miles (aka, the algorithm won’t return Stay Home) if it’s easier.

[We are expecting: A formal proof by induction. Be sure to clearly state your inductive hypothesis, base case, inductive step, and conclusion.]
5. [Minimum-maximum spanning trees.] (6 pt.) Let $G$ be a connected weighted undirected graph. In class, we defined a minimum spanning tree of $G$ as a spanning tree $T$ of $G$ which minimizes the quantity

$$X = \sum_{e \in T} w_e,$$

where the sum is over all the edges in $T$, and $w_e$ is the weight of edge $e$. Define a “minimum-maximum spanning tree” to be a spanning tree that minimizes the quantity

$$Y = \max_{e \in T} w_e.$$

That is, a minimum-maximum spanning tree has the smallest maximum edge weight out of all possible spanning trees.

(a) (2 pt.) Give an example of a graph $G$ which has a minimum-maximum spanning tree $T$ so that $T$ is not a minimum spanning tree.

[We are expecting: An example, with an informal explanation of why it is an example.]

(b) (4 pt.) Prove that a minimum spanning tree in a connected weighted undirected graph $G$ is always a minimum-maximum spanning tree for $G$.

[HINT: Suppose toward a contradiction that $T$ is an MST but not a minimum-maximum spanning tree, and say that $T'$ is a minimum-maximum spanning tree. How can you use $T'$ to modify $T$, to come up with a cheaper MST than $T$ (and hence a contradiction)? (Sub-hint: consider the heaviest edge in $T$).]

[We are expecting: A formal proof.]
6. (NOT REQUIRED, WORTH ONE BONUS pt.) [Another activity selection algorithm?]

In class, we considered an alternative greedy algorithm for activity selection. The idea was that at each step, we greedily add a valid activity with the fewest conflicts with other valid activities. (An activity is valid if it doesn’t conflict with an already selected activity).

For example, if the activities looked like:

```
  a1  a2  a3  a4  a5  a6  a7  a8
Time       0   3   3   3   3   1   1   2
```

then the number of conflicts to begin with are:

The algorithm (breaking ties arbitrarily) could choose \( a_1 \), then \( a_6 \), then \( a_7 \), then \( a_2 \).

Is this algorithm correct?

[We are expecting: To get the bonus point, give either a counterexample or a formal proof of correctness.

- If you give a proof by induction, make sure to clearly state your inductive hypothesis, base case, inductive step and conclusion. (Note, in this case you should show that the algorithm is correct no matter how it breaks ties).

- If you give a counterexample, it should be a drawing like the one above; you can either draw it by hand or use your favorite software. You should also explain what this algorithm does on your counter-example and why it is not optimal. (Note, in this case it is okay to give an example where there is some way of breaking ties so that the algorithm messes up).]