CS 161 **Problem Set 7**

Spring 2023 **Due:** Wednesday, May 31 at 1pm on Gradescope.

Style guide and expectations: Please see the "Homework" part of the "Resources" section on the webpage for guidance on what we are looking for in homework solutions. We will grade according to these standards.

Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

Collaboration policy: You may do the HW in groups of size up to three. Please submit one HW for your whole group on Gradescope. (Note that there is an option to submit as a group). See the "Policies" section of the course website for more on the collaboration policy.

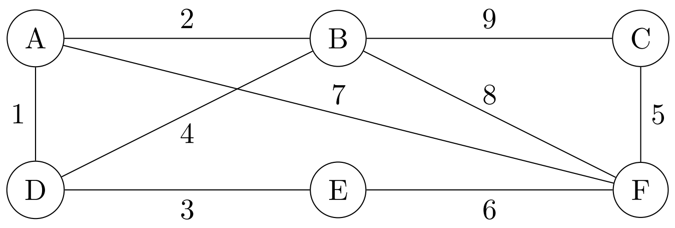
Please list all members of your homework group and all other students in the class you have collaborated with on this homework, in accordance with the Collaboration Policy, at the beginning of each homework submission.

Exercises

We recommend you do the exercises on your own before collaborating with your group.

---

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



   (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. That is, in class, we looked at a version where you had to take an integer number of each item; in this exercise you can take fractional amounts of each item. For example, suppose that one of the "items" is 3.6 ounces of brightly colored sand; in this version of the problem you can choose to add any amount of sand (up to 3.6 oz) to your knapsack; e.g., you could take 2.5235 if you wanted to.

   More formally, suppose that we have a knapsack with a capacity of Q ounces and suppose there are n items. 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 value that you can derive from x ounces of item i is $x \cdot v_i$.

   Your goal is to specify the amount of each item i to take, $x_i \geq 0$, in order to maximize the total value $\sum_i x_i v_i$ of the knapsack, while satisfying:

   (1) you don't overfill the knapsack (that is, $\sum_i x_i \leq Q$), 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 $\sum_i q_i \geq Q$, so there always is some way to fill the knapsack.

   (a) **(3 pt.)** Design a greedy algorithm that achieves the goals outlined above. Your algorithm should take as input the capacity Q, along with a list of tuples (i, v_i, q_i) for each item $i = 0, \ldots, n-1$. It should output a list of tuples (i, x_i) so that (1) and (2) above both hold, and so that $\sum_i x_i v_i$ 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.*
   - *A justification of the running time.*

   ]

   (b) **(3 pt.)** In this part, you will prove that your algorithm from part (b) is correct. Following the examples that we saw in class, the structure of the proof is shown below: we have filled in the inductive hypothesis, base case, and conclusion. Fill in the inductive step to complete the proof.

   - **Inductive hypothesis:** After making the t'th greedy choice, there is an optimal solution that extends the partial 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 $\sum_i x_i = Q$. Thus, there is no solution extending S^* other than S^* itself. Thus, the inductive hypothesis implies that S^* is optimal.

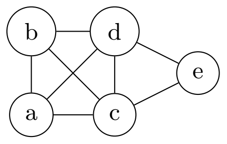
   [**We are expecting:** *A proof of the inductive step: assuming the inductive hypothesis holds for $t-1$, prove that it holds for t.*]

Problems

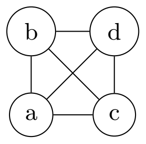
3. **(6 pt.)** [**k-well-connected graphs.**] Let $G = (V, E)$ be an undirected, unweighted graph with n vertices and m edges. For a subset $S \subseteq V$, define the **subgraph induced by** S to be the graph $G' = (S, E')$, where $E' \subseteq E$, and an edge $\{u, v\} \in E$ is included in E' if and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

   $G = (V, E)$

   $G' = (S, E')$, for $S = \{a, b, c, d\}$

   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:

   - `degree(v)`: return the degree of a vertex in time $O(1)$
   - `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. **(6 pt.)** Vonix the Vole is back, with $n-1$ of their friends! There are n voles in total. You'd like to get a message to all n of the voles (perhaps you'd like to tell them your solutions to HW4!). However, each of the voles pops up out of the ground for only one interval of time. Say that vole i is above ground in the interval $[a_i, b_i]$. Your plan is to shout your message repeatedly, at certain times $t_1, \ldots, t_m$. Any vole who is above ground when you shout your message will hear it, while any vole who is below ground will not. Assume that shouting the message is instantaneous.

   You'd like to ensure that each vole hears your message at least once (it's okay if a vole hears it multiple times), while making m as small as possible (so that you don't have to shout too much).

   (a) **(6 pt.)** Design a greedy algorithm which takes as input the list of intervals $[a_i, b_i]$ each vole is above ground and outputs a list of times $t_1, \ldots, t_m$ so that every vole hears your message at least once and m is as small as possible. Your algorithm should run in time $O(n \log(n))$. If it helps, you may assume that all the a_i, b_j are distinct.

   You do not need to prove that your algorithm is correct (although you may do so for a bonus point in part (b)). However, we strongly suggest that you at least prove it to yourself, to make sure that your algorithm is correct!

   [**We are expecting:** *Pseudocode and an English description of the main idea of your algorithm, as well as a short justification of the running time.*]

   (b) **(1 BONUS pt.)** Prove that your algorithm from part (a) is correct.

   [**Note:** *We felt this HW already had enough proofs on it, so we made this one bonus. However, we recommend that you give this bonus question a try for extra practice, in addition to the bonus point!*]

   [**We are expecting:** *Nothing, this part is not required. To get the bonus point, we are looking for a formal proof by induction. Be sure to clearly state your inductive hypothesis, base case, inductive step, and conclusion.*]

5. **(7 pt.)** [**CS161 HW Groups**] In CS161, we let you pick your own homework groups (since that's what you voted for at the beginning of the quarter!) Here's another thing we could have done instead, if we were going to assign students to work in pairs.

   - Each student would tell us which other students they would be okay working with. We'd use this information to create a graph $G = (V, E)$, where the vertices V are students, and there is an edge between student a and student b if each would be okay working with the other.
   - We would run an algorithm (see below) to try to pair up as many students as possible, subject to the constraints (1) that everyone who was paired up would need to be okay with their HW partner; and (2) that each student is in at most one pair. Notice that the algorithm would output a subset of the edges of E in the graph that we defined above.
   - Students who were not paired up would work alone (or else work with someone they hadn't originally identified as wanting to work with).

   Here is a greedy algorithm that we could have used to pair up students, which takes as input our graph $G = (V, E)$.

```
function PairUpStudents($G = (V, E)$):
  • While there is a pair of students $a, b \in V$ so that $\{a, b\} \in E$:
      ◦ Pair up the students $a$ and $b$, and remove the vertices $a$ and $b$, and all the edges
        touching them, from $G$.
```

   That is, we just greedily take pairs of students who would be willing to work together and pair them up, until we can't anymore.

   (a) **(3 pt.)** Call a matching *optimal* if as many students as possible are matched with partners, subject to constraints (1) and (2) above. More precisely, in the graph G described above, a *matching* is a set of edges in E so that the edges are disjoint (no two share a vertex). An *optimal matching* is a matching that contains as many edges as possible. Suppose that the optimal matching for our class would result in M edges. Show that **PairUpStudents** returns a matching with at least $M/2$ edges.

   [**Hint:** *While you* can *use induction for this if you like, there is a short proof that does not involve induction. We recommend you try to find that one!*]

   [**Hint:** *Here's a hint for a non-induction proof. Suppose that $S^* \subseteq E$ is an optimal matching, and suppose that S is the matching returned by our greedy algorithm. For each edge $e = \{a, b\} \in S$, let S^*_e be the set of edges in S^* that touch either a or b. How big can $|S^*_e|$ be? And are there are any edges in $\bigcup_{e \in S} S^*_e$ that* aren't *in S^*?*]

   [**We are expecting:** *A formal proof.*]

   (b) **(4 pt.)** **(EthiCS)** There are several reasons that we did not use this algorithm in our class (and not just because it's within a factor of two of optimal, rather than exactly optimal). Using the language that we've developed in previous EthiCS problems (e.g., *idealization*, *abstraction*, and *incommensurability*), explain some of the real-world issues with using this algorithm to pair up students to work together. **Optional:** Do you think that letting everyone picking their own partners is the best method? If not, what would you suggest?

[**We are expecting:** *About a paragraph explaining some real-world issues with using this algorithm, explicitly mentioning at least two of* {*idealization, abstraction, incommensurability*}*. The **optional** questions at the end are indeed optional, but we'd be really curious about what you think; this may affect how we do HW partners in future offerings!*]

6. **[Please Go Fill Out This Group-Work Survey] (1 pt.)**

   In collaboration with Professor Shima Salehi of the Stanford Ed School and her team, we have created a short google form on your experience with group-work in this class, which is linked below. SUNet IDs will be collected, but we will only use hashes of IDs for the study, so the responses will be effectively anonymous. The form is very short, and will take ≈ 1 minute to fill out: `https://forms.gle/1deqQVcHsxvorgP89`.

   We really appreciate all your feedback and help with making CS161 the best possible learning experience it can be! If you would like to give more general feedback on the class as a whole, please check out the many possible options highlighted in the linked Ed post[1].

   Have you submitted your answers to the google form above?

   [**We are expecting:** *Yes*]

[1] `https://edstem.org/us/courses/38246/discussion/2970522`

7. **CS 161 Art! (1 BONUS pt.)**

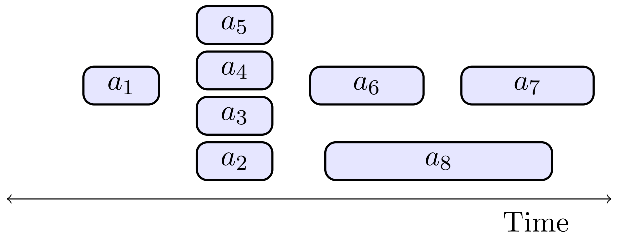
   Reflect on the class. Take one of the themes or concepts and write or present something creative about it! Some of our favorite submissions may be highlighted in class and/or on Ed :) [If you'd prefer your submission not be highlighted, or be highlighted anonymously, please mention that.]

   Some standout past submissions: an algorithms rap music video, Plucky cosplay, a red-black Christmas tree.

   [**We are expecting:** *Nothing, this question is optional. To get the bonus point, we are expecting...whatever you can come up with! Feel free to share your creative output with us in whatever way makes the most sense: a written piece, an image, a link to a video, etc.*]

8. **(1 BONUS pt.) [Another activity selection algorithm?]** Recall the activity selection problem from class. Here is an alternative greedy algorithm for this problem: The idea is 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:



then the number of conflicts to begin with are:

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
0	3	3	3	3	1	1	2

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:** *Nothing, this question is optional. 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).*

]