CSE 101 Homework 5

Brian Masse, Kreshiv Chawla, Taira Sakamoto, Emily Xie, Annabelle Coles

October 31, 2025

1. **Cart horses**

   Consider the following problem: We have n horses, $Horse_1, ..Horse_n$, each with a cart-pulling speed s_i. We need to pair the horses up into teams to pull carts.The faster horse will need to slow down for the slower horse,so if we pair $Horse_i$ and $Horse_j$, the cart will be pulled at speed $min(s_i, s_j)$. Each horse can only be in one pair. We want to maximize the sum of all the cart speeds.

   Give a greedy strategy for this problem, and prove that it is correct using any of the proof templates from class.

   (a) Greedy Algorithm: When picking the next pair of horses, pick the 2 fastest horses from the remaining horses.

   (b) Proof of correctness (via exchange):
   Let G be the first greedy choice. Let OS be an optimal solution
   Find the 2 pairs with the 2 fastest horses, H_1, H_2: $(H_1, H_i), (H_2, H_j)$. These horses will be separate, otherwise OS already contains G. Exchange H_i and H_2
   Note:

$$\begin{aligned} & H_1 \geq H_2, H_2 \geq H_i, H_2 \geq H_j \\ & min(H_1, H_i) = H_i, min(H_2, H_j) = H_j \\ \implies \quad & min(H_1, H_2) \geq min(H_1, H_i) \end{aligned}$$

   For the second new pairing, (H_i, H_j) there are 2 possibilities

   i. $H_i \geq H_j \implies min(H_2, H_j) = min(H_i, H_j) = H_j$:
   ii. $H_i < H_j$:

$$\begin{aligned} & min(H_1, H_i) + min(H_2, H_j) - min(H_1, H_2) - min(H_i, H_j) \\ = \quad & H_i + H_j - H_2 - H_i = H_j - H_2 \leq 0 \\ \implies \quad & \text{New solution is faster} \end{aligned}$$

   Thus, the net speed of the exchanged solution is always as good as the original and meets the constraints of the problem

   Now, proving correctness via induction on number of horses. This proof assumes there will always be an even number of horses to make the pairings.

   i. Base Case: $n = 0, 2$: Trivially, the greedy solution (and all solutions) are optimal
   ii. Induction Step. Let OS be any solution for the group of horses $I = \{H_1, ...H_k\}, k > 2$
   By the exchange argument above, $\exists OS'$ such that $|OS| \leq |OS'|$ and $G \in OS$
   Let I' be the set of pairs of horses that have neither H_1 or H_2 in them.

$$\begin{aligned} & OS' = \{G\} \cup I' = \{(H_1, H_2), (H_i, H_j)\} \cup I' \\ & \text{Because } ||I'|| = k - 4 \leq k, \text{by the induction hypothesis } |S(I')| \leq |GS(I')| \\ \implies \quad & |OS| \leq |OS'| = |\{G\} \cup OS(I')| \leq |\{G\} \cup GS(I')| = |GS(I)| \end{aligned}$$

Thus, for every set of horses, the greedy solution is as good as the optimal solution.

(c) Time Analysis. This algorithm only needs an input of horses, sorted by their speed. Thus, the algorithm takes $O(n \cdot log(n))$

- $log(n)$ to sort the horses
- n to scan through the sorted list of horses and create the pairings

2. **Sensor maximization**

Suppose you are placing sensors on a one-dimensional road. You have identified n possible locations for sensors, at distances $0 \leq d_1 \leq d_2 \leq ... \leq d_n$ from the start of the road. Each sensor must be at most M distance from the previous one (so they can communicate reliably). The first one must be at 0 and the last at d_n. Given that, you want to minimize the number of sensors placed.

The following greedy algorithm, which places each sensor as far as possible from the previous one, will return a list $d_{i_1} \leq d_{i_2} \leq ...d_{i_k}$ of locations where sensors can be placed.

GreedySensorMin $[d_1...d_n, M]$

(a) Initialize a list to (0).

(b) Initialize $I = 1$, $PreviousSensor = 0$.

(c) While $I \leq n$ do:

(d) While $I \leq n$ and $d_I < PreviousSensor + M$ do:$I++$.

(e) If $I \leq n$ THEN append d_{I-1} to list; $PreviousSensor = d_I$;$I++$.

(f) Append d_n to list.

(g) Return list

Questions:

(a) What constraints do solutions $d_{i_1}, ..d_{i_k}$ need to meet for this problem?

- For a solution $d_{i_1}, ...d_{i_k}$, minimize k
- $d_{i_1} < d_{i_2} < \cdots < d_{i_k}$
- $d_{i_j} - d_{i_{j-1}} < M \implies d_{i_j} < d_{i_{j-1}} + M$
- $d_{i_1} = 0, d_{i_k} = d_n$

(b) What is the value of the objective function for a solution, of the form $d_0, d_{i_1}, ..d_{i_k} = d_n$, ? The value of the objective function of such a solution is k. The goal is to minimize k

(c) In using the "greedy stays ahead" proof technique to show that this is optimal, we would compare the greedy solution $d_{g_1}, ..d_{g_k}$ to another solution, $d_{j_1}, ..d_{j_{k'}}$. In what sense is the greedy solution "staying ahead" of the other solution at each step $1 \leq t \leq k'$?

The i^{th} choice to place a marker in the greed solution must always be farther or equally along the road compared to the i^{th} choice in the normal solution. $(d_{g_i} \geq d_{j_i})$

(d) Prove the claim you wrote above using induction on the step t.

i. Base case: $t = 0 \implies d_{g_0} = d_0 = 0 = d_{j_0}$

ii. Induction Step: Assume $d_{g_{t-1}} \geq d_{j_{t-1}}$, show $d_{g_t} \geq d_{j_t}$ for $t > 0$

- Let I be the set of valid choices for the greedy algorithm at step t. It contains markers between $d_{g_{t-1}}, d_{g_{t-1}} + M$

- Let J be the set of valid choices for the regular optimal solution. It contains markers between $d_{j_{t-1}}, d_{j_{t-1}} + M$

By the inductive hypothesis, $d_{g_{t-1}} \geq d_{j_{t-1}} \implies d_{g_{t-1}} + M \geq d_{j_{t-1}} + M$. Thus, any valid solution in the regular set, $j \in J$, is either less than all solutions in I, or contained within I. Because the Greedy algorithm picks the largest marker in I:

- If the regular solution picks $d_{j_t} \notin I \implies d_{j_t} < d_{g_t}$
- If the regular solution picks $d_{j_t} \in I \implies d_{j_t} \leq d_{g_t}$

Thus, the i^{th} greedy choice, d_{g_i}, is at least as far along the as the i^{th} normal choice d_{j_i}

(e) In O notation, how much time does the algorithm as written take?

The above algorithm runs in $O(n)$ time. Each one of the n markers is considered ($O(1)$) exactly once.

3. **Quests**

In a role-playing game, your character will complete a series of k quests. There is a list of $n > k$ possible quests Q_i, each with a first time it can be attempted $k \geq f_i \geq 1$ and the amount of gold earned on the quest, $g_i > 0$. You cannot complete the same quest twice, and you cannot put quest i in a position before f_i in your list. You want to maximize your total gold at the end of all your quests.

(a) Clearly state a greedy strategy that gives the most possible total gold
At choice i pick the quest with the most gold and that satisfies $f_i \leq i$

(b) Prove that this strategy is correct. Hint: use a modify-the-solution proof with two cases, based on whether the next quest the greedy algorithm performs is performed at some point in OS.

Proof by exchange:

- Let G be the first greedy solution (Qg_o)
- Let OS be an optimal solution

There are 2 possibilities:

i. OS never picks G:
In this case, create OS' by exchanging the first non greed choice, Q_{j_0} with G
   - OS', by definition, still satisfies the ordering constraint (f_{g_0} must $= 1$ to be chosen first)
   - G yields the highest gold at choice 1: $g_{g_0} \geq g_{j_0} \implies |OS'| \geq |OS|$

ii. OS Picks G as the i^{th} quest.
In this case, exchange G with the first pick, Q_{j_0}
   - Still satisfies constraints: both f_{j_0} and $f_{g_0} \leq 1$. Thus both can serve as the first and i^{th} choice without breaking a constraint
   - $|OS'| = |OS|$ since all the quests are the same

Thus, you can always create a $|OS'|$ from an OS, such that OS' meets the constraints of the problem and is equally as good.

Now, proving correctness via induction on number of quests.

i. Base case: n = 1: Greedy (and all) solutions are trivially correct

ii. Induction Step:
   - Let I be a set of quests $\{Q_1, ...Q_n\}$. We can create OS' from any $OS(I)$ such that $|OS'| \geq |OS|$ and $G \in OS'$
   - Let I' be the set of quests without G
   - Let I'' be the set of quests where G is not the first choice.

Note that by the above proof, we can trivially exchange any solution $OS(I'')$ into a solution on I starting with G with changing the optimality of the solution. Thus,

$$\begin{aligned} OS' &= \{G\} \cup I' \cup I'' \\ |OS| \leq |OS'| &= |\{G\} \cup OS(I') \cup OS(I'')| \\ &= |\{G\} \cup OS(I')| \text{ (By exchanging solutions from } I'') \\ &\leq |\{G\} \cup GS(I')| \text{ (By Induction Hypothesis)} \\ &= |GS(I)| \end{aligned}$$

Thus, the greedy solution provides an optimal solution on any set of quests.

(c) Give an efficient algorithm carrying out this strategy and give a time analysis of your algorithm.

i. Explanation of algorithm:

- Order all quests by g_i
- Initialize an empty max heap
- Loop through 1...k. At the i^{th} step, insert all quests with $f_i \leq i$ into the max heap. Pop the max as the i^{th} choice.

ii. Runtime analysis:

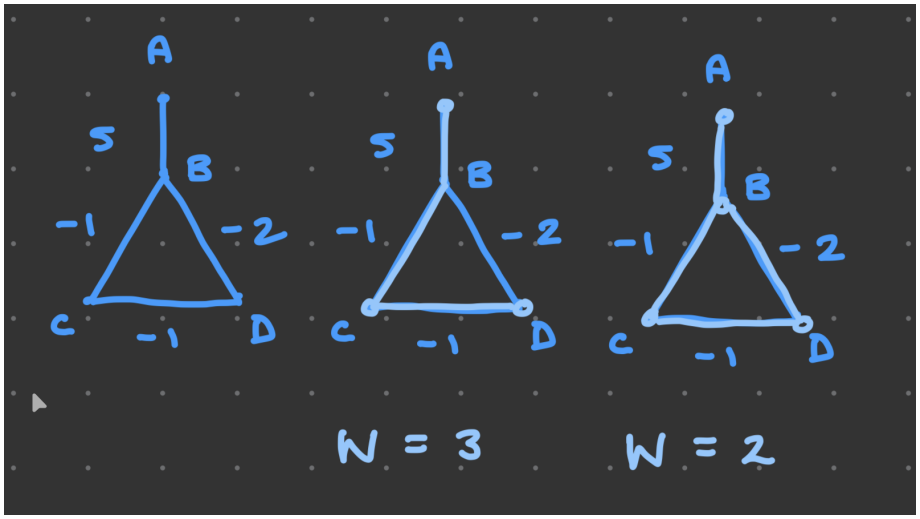
- Sorting elements takes $O(n \cdot log(n))$
- Inserting and removing from the max-heap takes $O(logn)$
- Each quest is inserted into the heap

Thus the total runtime is $O(n \cdot log(n))$

4. **Minimal spanning subgraph with negative edge weights**

Say we allow negative edge weights into the spanning tree problem. We are given a connected undirected graph $G = (V, E)$ with possibly negative edge weights. We wish to find the set $E' \subset E$ so that $G' = (V, E')$ is connected that minimizes $\sum_{e \in E'} w(e)$.

(a) Give an example of a graph with some negative edge weights where the minimum cost connected subgraph is not a tree.



The above graph ($G = \{(A, B, 5), (B, C, -1), (B, D, -2), (C, D, -1)\}$) has some negative and some positive weighted edges. The minimum cost connected subgraph, however, is not a tree because of the (B, C, D) cycle. A regular spanning tree necessarily has a higher weight.

(b) Which edges are always in the minimum cost subgraph?

All negative weighted edges. $E' = \{e \in E | w(e) < 0\}$

Adding any negative edge will always reduce the overall cost of the subgraph, and thus must be included in minimum connected subgraph.

(c) Once the edges above are added, what is an equivalent problem?

After adding all edges with negative weights, it's possible to represent nodes connected with negative edges as strongly connected components, inter-connected with the remaining positive edges. In this state the problem can be interpreted as a standard MST problem.

(d) Describe how to use the answers to the above two questions to reduce this problem to minimum spanning tree. Give a time analysis for the resulting algorithm.

The following algorithm implements the procedure described above:

- Identify all negative edges $E' = \{e \in E | w(e) < 0\}$
- Identify connected components from those edges via DFS. Label a component that contains any vertex u as $C(u)$
- let H be a subgraph of G: $H = \{V, (u, v) \in E \backslash E' | C(u) \neq C(v)\}$

- Now the graph H is a standard, positive edged connected graph, which can be a candidate of any MST algorithm.

The overall time analysis is $O(|E| \cdot log(|V|))$ or $O(|V|^2)$ depending on the implementation used for the MST

- Constructing $E' \in O(|E|)$
- Decomposition algorithm to get SCCs $\in O(|E| + |V|)$
- Constructing $H \in O(|E|)$
- Running MST via Prims algorithm $\in O(|V|^2)$ (Array implementation of Priority Queue) or $\in O(|E| \cdot log(|V|))$ (min-heap implementation of priority Queue)

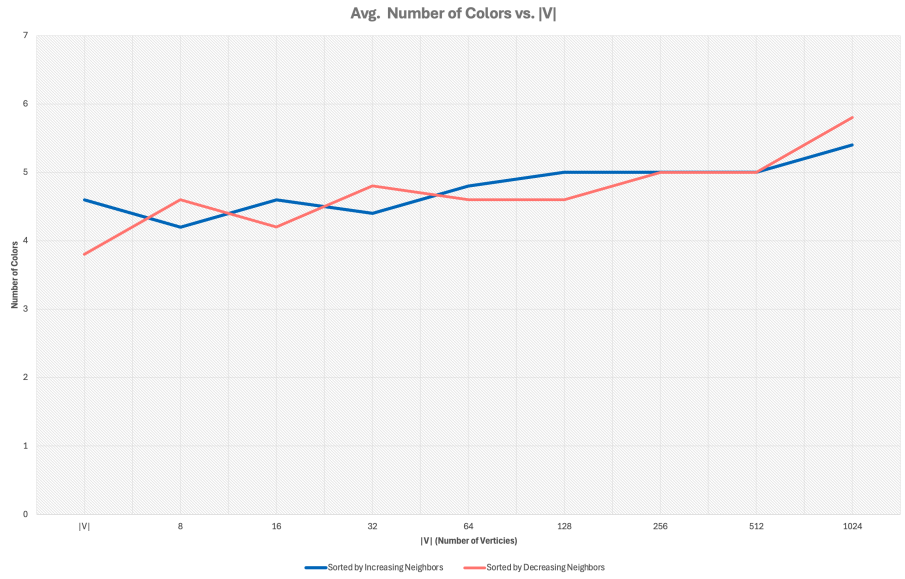
Thus, the dominant time is the MST, and total time complexity $= O(|E|) + O(|E| + |V|) + O(|E|) + O(|V|^2)|O(|E| \cdot log(|V|)) \in O(|V|^2)|O(|E| \cdot log(|V|))$

5. **Graph coloring heuristics**

The graph coloring problem is, given an undirected graph, give each vertex a color, so that neighboring vertices have different colors. You want to minimize the number of colors. This is an NP-complete problem, so we might want to use a greedy heuristic.

One greedy heuristic would be to order the vertices, then assign each the smallest color that is different from its neighbors.

Implement this heuristic on random graphs where we start with $|V|$ vertices and add $5|V|$ random edges. For a variety of sizes (e.g., $|V| = 2^k$ for $k = 3, ...12$), and several graphs of each size, try the heuristic ordering the vertices by increasing degree (number of neighbors) and decreasing degree. Plot the two numbers of colors used. Do these increase with $|V|$? How consistent are the results with different graphs of the same size?



The above graph represents the number of colors needed for a given graph plotted against the number of verticices in that graph. Specifically,

- x-axis: number of verticies, $|V|$, in the graph. The experiment was run on inputs of $2^k, k \in \{3, ...12\}$
- y-axis: number of colors needed to properly solve the graph coloring problem. For each input, 5 graphs were generated. Above is the average of the number of colors needed for each.
- Color: The color denotes how the verticies were ordered before applying the greedy solution to the problem. Red ordered verticies with the most neighbors first, blue did the opposite.

Questions:

(a) Generally the number of colors needed to color the graphs increases with $|V|$, however the increase is only marginal. The correlation is roughly 0.1

(b) Different graphs of the same size generally had similar number of colors. The largest ever recorded difference was ± 1 (for 1 input, 1 graph needed 4 colors, another 5, and another 6.)