Assignment 5

Kailing Wang 521030910356

December 5.2022

Problem 1. (25 points) [Fair Division Revisited] Suppose we want to allocate m indivisible items to n agents. A $n \times m$ matrix $V = [v_{ij}]_{i=1,\ldots,n;j=1,\ldots,m}$ describes agents' preferences over the items, where v_{ij} is the value of item j according to agent i 's preference. We assume agents' valuations are binary, i.e., $v_{ij} \in \{0, 1\}$ for any i and j. Given a subset S of items, agent i 's value on S is naturally defined by $v_i(S) = \sum_{j \in S} v_{ij}$.
An allocation $(A_1, \ldots, A_n)$ is a partition of $[m] := \{1, \ldots, m\}$, where $A_i \subseteq [m]$ is the set of the items allocated to agent i. An allocation is proportional if $v_i(A_i) \geq \lfloor \frac{1}{n} v_i([m]) \rfloor$ holds for any agent i. That is, in a proportional allocation, each agent receives his/her average integer value, rounded down, of the entire item set. For example, if $v_{ij} = 1$ for all i and j, the average value for each agent is m/n, and each agent needs to get at least $\lfloor \frac{m}{n} \rfloor$ items to guarantee proportionality. In this question, we are going to prove the existence of proportional allocations and design an algorithm to find one.

(a) (5 points) Let us first assume, for this moment, that items are divisible. That is, we are allow to fractionally allocate a single item such that an agent gets some portion of it. Show that there exists a fractional allocation such that each agent i receives a value of at least $\frac{1}{n} v_i([m])$.

(b) (5 points) Consider the flow network with $m + n + 2$ vertices $s, u_1, \ldots, u_n, w_1, \ldots, w_m, t$ described as follows. There is a directed edge from s to each u_i with capacity $\frac{1}{n} v_i([m])$. There is a directed edge from u_i to w_j with capacity ∞ if $v_{ij} = 1$. There is a directed edge from each w_j to t with capacity 1 . According to Part (a), what is the maximum flow in this network?

(c) (5 points) Now, we modify the network by reducing the capacity of each edge (s, u_i) from $\frac{1}{n} v_i([m])$ to $\lfloor \frac{1}{n} v_i([m]) \rfloor$. What is the maximum flow in this network?

(d) (5 points) Show that the maximum flow in Part (c) can be made integral. Use this result to show that the proportional allocation always exists even when items are indivisible.

(e) (5 points) Design an algorithm to compute a proportional allocation.

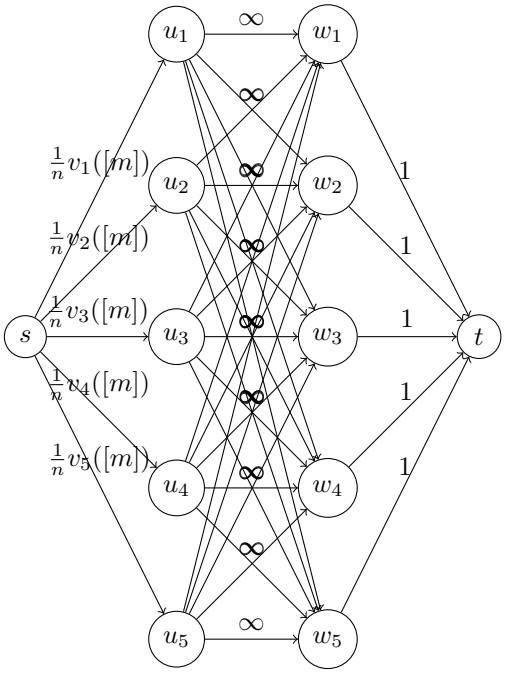
Answer 1.

(a) If the items are fractional, we can simply divide each item to n pieces and give one piece to each agent regardless of whether he/she wants it or not. This way, the value each agent receives is exactly $\frac{1}{n} v_i([m])$ because he/she actually receives $\frac{1}{n}$ portion of each item.

In fact, we can also throw away the $\frac{1}{n}$ of item j if agent i does not prefer item j. This does not change the value he/she gets.

(b) First we prove that this flow network depicts the problem of Part (a). Draw the network as Plot 0. The capacity from s to u_i is the required value we expect that each agent receive.

The ∞ capacity ensures that all the received value should cost the same amount from w_i. If agent i does not prefer item j, we don't allocate any of item j to agent i. The 1 capacity shows that there is only 1 item for each kind. When dividing the items, we can set the flow on each line at any value so that the items should be fractional, which is the same as what we suppose in Part (a).



Plot 0: Here I use TIKZ to draw

The flow network describes the same problem as problem a. To solve the max flow problem, we can set the flow from u_i to w_j to $\frac{1}{n}$ if v_{ij} is 1 and otherwise 0. This way, we have

$$\sum_{i=1}^{n} f(s \to u_i) = \sum_{i=1}^{n} c(s \to u_i)$$

And these concerned edges form a cut of the network. According to the Maximum Flow-Minimum Cut Theorem, this allocation forms a max flow for this problem. The max flow is $\sum_{i=1}^{n} \frac{1}{n} v_i([m]) = \frac{m}{n}$.

(c) As what I've explained in Part (b), if we reduce the capacity of edge (s, u_i), we have to reduce the flow on the edges (u_i, w_j) and that of w_j, t. Making this reduction is always legal. Since the flow is reduced from the one in Part (b), the flow on edge (s, u_i) is $\lfloor \frac{1}{n} v_i([m]) \rfloor$. We get a flow with total value $\lfloor \frac{1}{n} \sum_{i=1}^{n} v_i([m]) \rfloor$, which equals the cut formed by all the (s, u_i) edges. According to the theorem, the flow is maximum.

(d) In this case, we have to modify the flow network by limiting the flow on edge (u_i, w_j) to be either 0 or 1. We can simply set all these capacities 1. Note that now all the capacities are integers. Under this condition, the result of Edmonds-Karp algorithm will give an integer flow so the flow on (u_i, w_j) is indeed either 0 or 1. The actual modification is that we set the capacity on edge (u_i, w_j) 1. However, in the max flow given by previous problems, the flow on these edges must be smaller than one because (w_j, t) has capacity 1. The max flow for the modified network is still $\lfloor \frac{1}{n} \sum_{i=1}^{n} v_i([m]) \rfloor$. Also we know Edmonds-Karp algorithm can find an integral solution with the max flow.

Algorithm 1: Fair Division Revisited

Data: a $n \times m$ matrix V

Result: A proportional allocation

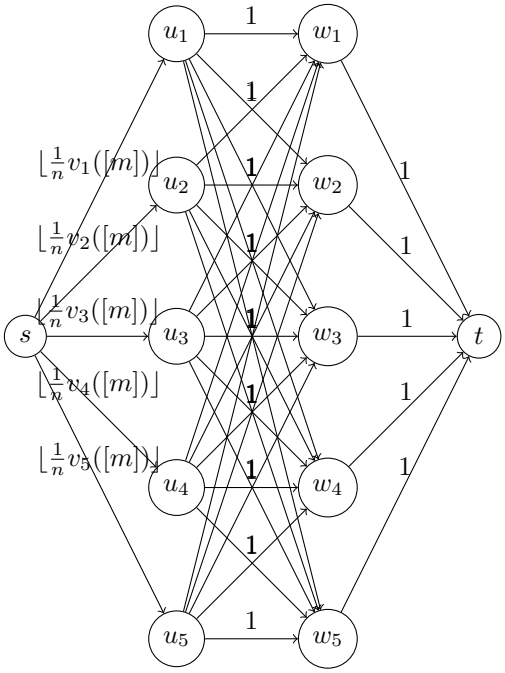
1 Construct a flow network:

(a) Set each agent as a vertex u_i, item w_i
Set super vertices s and t

(b) Set the capacity of (s, u_i) : $\lfloor \frac{1}{n} v_i([m]) \rfloor$
(u_i, w_j) : 1
(w_j, t) : 1

Draw the network as Plot 1;
Apply the Edmonds-Karp algorithm, the result flow is the proportional allocation;

(e) See the pseudo-code Algorithm 1 and Plot 1.



Plot 1: solution network

The time complexity is the one of Edmonds-Karp, and the correctness is proved in the previous parts.

Problem 2. (25 points) [Hall Marriage Theorem] Given a bipartite graph $G = (A, B, E)$ with $|A| = |B| = n$, Hall Marriage Theorem states that G contains a perfect matching (i.e., a matching with size n) if and only if $|N(S)| \geq |S|$ for any $S \subseteq A$, where $N(S) = \{b \in B \mid \exists a \in S \subseteq A : (a, b) \in E\}$ is the set of all the neighbors of the vertices in S. In this question, we are going to prove Hall Marriage Theorem.

(a) (5 points) Prove the necessity part: if G contains a perfect matching, then $|N(S)| \geq |S|$ for any $S \subseteq A$

(b) (10 points) Construct a directed weighted graph with vertex set $V = A \cup B \cup \{s\} \cup \{t\}$. The edge set is defined as follows. For each $a \in A$, construct a directed edge (s, a) with weight

1 ; for each $a \in A$ and $b \in B$, construct a directed edge (a, b) with weight ∞ if (a, b) is an edge in the original graph G; for each $b \in B$, construct a directed edge (b, t) with weight 1 . Prove that, if $|N(S)| \geq |S|$, the minimum s - t cut has value n.

(c) (10 points) Prove the sufficiency part: if $|N(S)| \geq |S|$ holds for all $S \subseteq A$, then G contains a perfect matching.

Answer 2.

(a) If G contains a perfect match, each vertex in A must attach at least one vertex in B. So, for a subset S of A, in the perfect match there are $|S|$ vertices in B attached to S, which implies $|N(S)| \geq |S|$. Let's prove by contradiction: if $|N(S)| < |S|$, there mustn't be a perfect match.

(b) In this problem, we draw the network like Plot 0 and 1. There are two significant positions: (a_i, a_{i+1}), (b_j, b_{j+1}) between which the cut line go through. Ignore the cut including only $(s, a_i), (b_j, t)$ or (a_i, b_j). These three conditions have flow at least n. According to symmetry, we only consider the condition the line start from upper-right. That is: the cut set include $(s, a_1), (s, a_2) \cdots (s, a_i)$, $(b_{j+1}, t), (b_{j+2}, t) \cdots (b_n, t)$ and other edges. When $i \geq j$, the cut of (s, a_p) and (b_q, t) is already larger than n. When $i < j$, the flow include three parts: (s, a_p) with flow i, (b_q, t) with flow $n - j$ and the edges from set $S_A = \{a_{i+1}, a_{i+2} \cdots a_n\}$ to $S_B = \{b_1, b_2 \cdots b_j\}$. Here we have $n - i > n - j$. According to the claim we made in Part (a), $N(S_A) \geq n - i > n - j$, and so there must be at least one edge from S_A to S_B, causing the cut value to be over n. Now we prove the min s-t cut is n.

(c) Like what we've done in Problem 1., we can replace the ∞ capacities with 1, which will not influence the flow. Construct the network like Part (b). Since we know the min cut equals the max flow, we can use Edmonds-Karp to find an integral flow with value n. In this flow, each a_i is linked to a unique b_j through an edge with flow 1 on it. This is a required perfect matching.

Problem 3. (25 points) [LP-Duality and Total Unimodularity] In this question, we will prove König-Egerváry Theorem, which states that, in any bipartite graph, the size of the maximum matching equals to the size of the minimum vertex cover. Let $G = (V, E)$ be a bipartite graph.

(a) (4 points) Explain that the following linear program describes the fractional version of the maximum matching problem (i.e., replacing $x_e \geq 0$ to $x_e \in \{0, 1\}$ gives you the exact description of the maximum matching problem).

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{e : e = (u,v)} x_e \leq 1 \quad (\forall v \in V) \\ & x_e \geq 0 \qquad\qquad (\forall e \in E) \end{aligned}$$

(b) (4 points) Write down the dual of the above linear program, and justify that the dual program describes the fractional version of the minimum vertex cover problem.

(c) (7 points) Show by induction that the incident matrix of a bipartite graph is totally unimodular. (Given an undirected graph $G = (V, E)$, the incident matrix A is a $|V| \times |E|$ zero-one matrix where $a_{ij} = 1$ if and only if the i-th vertex and the j-th edge are incident.)

(d) (6 points) Use results in (a), (b) and (c) to prove König-Egerváry Theorem.

(e) (4 points) Give a counterexample to show that the claim in König-Egerváry Theorem fails if the graph is not bipartite.

Answer 3.

(a) The description of the problem is ambiguous. I'm not sure what exactly I need to explain.

If we replace $x_e \geq 0$ to $x_e \in \{0, 1\}$, the new constriction means each edge has only two states: selected or not selected. Each vertex can only be connected to one selected edge. The both ending vertices of a selected edge is a pair, and the result of maximizing the number of selected edges is the max match.

The fractional version of the problem is: a vertex can be divided into fractional parts and then be matched to different vertices. There restrictions are: the matched part should be in [0,1], represented by the value on an edge x_e and the sum of the part is 1. This liner program describes this problem.

(b) The dual program is

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} y_v \\ \text{subject to} \quad & \sum_{(v_1, v_2) \in E} y_{v_i} \geq 1 \quad (\forall e \in E) \\ & y_v \geq 0 \qquad\qquad (\forall v \in V) \end{aligned}$$

We have to explain this is a fractional version of vertex cover. However, I can't see what is a fractional vertex cover. So I guess we have to replace $y_v \geq 0$ with $y_v \in \{0, 1\}$. Here variable y_v represents whether the vertex is in the vertex cover. $\sum_{(v_1, v_2) \in E} y_{v_i} \geq 1$ means for each edge, at least one end is in the vertex cover. The result of the minimum vertex cover is exactly the minimized y_v.

(c) Show by induction:

1. For case $|V| = 1$, the incident matrix is obviously totally unimodular.

2. For other cases, if the incident matrix for bipartite graph with $k - 1$ vertices is totally unimodular, discuss the case for k.

The k graph G is obtained by adding a kth vertex onto a $k - 1$ graph G'. According to the induction, the incident matrix A' for G' is unimodular.

According to the definition of the incident matrix, the incident matrix of G can be written as

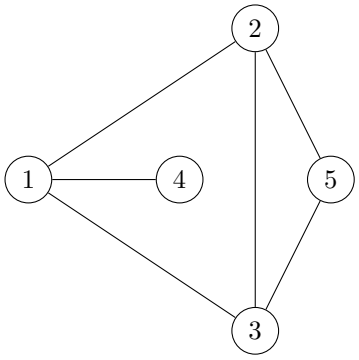
$$A = \begin{pmatrix} A' & E_k \\ \mathbf{0} & \mathbf{1} \end{pmatrix}$$

E_k is a matrix representing the additional edges based on A'. The square sub-matrices of A' have determinate 0 or ± 1. Adding a row with all 0 does not change unimodularity, so the left part of A: $\begin{pmatrix} A' \\ \mathbf{0} \end{pmatrix}$ is unimodular. For matrix $\begin{pmatrix} E_k \\ \mathbf{1} \end{pmatrix}$, the last row is all 1. In E_k, there is one and only one 1 in each column. Any sub-matrix of E_k has determinate 0 or $\pm$ 1 because at most one component is non-zero in the determinant calculation formula expanded by column. Since all the element is 0 and ± 1, the determinate is 0 or ± 1.

Adding a row with all 1, the right part of A is still unimodular. For sub-matrices across the two part: consider the square matrices with size $|V|$ composed of some columns in $\begin{pmatrix} A' \\ \mathbf{0} \end{pmatrix}$ and some columns in $\begin{pmatrix} E_k \\ \mathbf{1} \end{pmatrix}$. For sub-matrices in the upper k-1 rows, since E_k has only one 1 in each column and A' is composed of 0 and 1s, the determinate is 0 or ± 1. For sub-matrices including the last row, expand the determinate by the last column to get 0 and a determinate of a matrix in the upper k-1 row. Thus A is totally unimodular.

(d) According to the duality of linear programming, the linear program in (a) and (b) share the same solution. The linear program in (a) represent the max matching problem and (b) the min vertex cover problem. Therefore, we have transformed the maximum matching problem into the minimum vertex cover problem, and the minimum vertex cover problem can also be transformed into the maximum matching problem. Thus, we have $|M| = |C|$, which is the König-Egerváry Theorem.

(e) Plot 2 shows the example with min vertex cover size 3 but max match 2.



Plot 2: example

Problem 4. (25 points) [LP and Vertex Cover] In this question, we are going to design a 2-approximation algorithm for the minimum vertex cover problem that is different from the one learned in the class. Let $G = (V, E)$ be an undirected graph for which we want to find a minimum vertex cover.

(a) (5 points) Explain that the following linear program describes the fractional version of the vertex cover problem (i.e., replacing $x_u \geq 0$ to $x_u \in \{0, 1\}$ gives you the exact description of the minimum vertex cover problem).

$$\begin{aligned} &\text{minimize} \sum_{u \in V} x_u \\ &\text{subject to } x_u + x_v \geq 1 \quad (\forall (u, v) \in E) \\ &x_u \geq 0 \quad (\forall u \in V) \end{aligned}$$

(b) (5 points) Let $\{x_u^*\}_{u \in V}$ be an optimal solution to the linear program in Part (a). Prove that $\sum_{u \in V} x_u^*$ is a lower bound to the size of any vertex cover.

(c) (10 points) Consider the following algorithm.

- Solve the linear program in Part (a) and obtain an optimal solution $\{x_u^*\}_{u \in V}$;
- Return $S = \{u \in V \mid x_u^* \geq 0.5\}$.

Prove that the algorithm returns a valid vertex cover, and it is a 2-approximation algorithm.

(d) (5 points) Show that there exists an integral optimal solution to the linear program in Part (a) if G is a bipartite graph. Notice that an integral optimal solution to this linear program directly gives you a minimum vertex cover. (Hint: This almost straightforwardly follows from one of the previous questions you have solved. Can you see it?)

Answer 4.

(a) I guess this liner program is the dual program of the last problem, which I explained in (b) in the last problem.

(b) Replace $x_u \geq 0$ with $x_u \in \{0, 1\}$. This is a stronger restriction and is included in the original restriction. Suppose the smallest vertex cover has size s^*, $\sum_{u \in V} x_u^*$ should be larger than s^* because in the last problem we explained the modified program describes the vertex cover problem and the case s^* is a solution for the modified problem. Since the original program has weaker restriction, the solution should be no larger, and thus is the lower bound.

(c) First prove the solution is valid. Since the program ensures $x_u + x_v \geq 1$, at least one end will be over 0.5 and selected. This satisfies the definition of vertex cover. Then prove the algorithm is 2-approximate. Suppose the optimal size is S_{opt}. $\sum_{u \in S} x_u^* \geq 0.5|S|$. Since $\sum_{u \in S} x_u^*$ is the lower bound of any vertex cover: $\sum_{u \in V} x_u^* \leq |S_{opt}|$. So $|S| \leq 2 \cdot |S_{opt}|$

Problem 5. (Bonus 60 points) [Dinic's Algorithm] In this question, we are going to work out Dinic's algorithm for computing a maximum flow. Similar to Edmonds-Karp algorithm, in each iteration of Dinic's algorithm, we update the flow f by increasing its value and then update the residual network G^f. However, in Dinic's algorithm, we push flow along multiple s - t paths in the residual network instead of a single s - t path as it is in Edmonds-Karp algorithm.

In each iteration of the algorithm, we find the level graph of the residual network G^f. Given a graph G with a source vertex s, its level graph $\bar{G}$ is defined by removing edges from G such that only edges pointing from level i to level $i + 1$ are kept, where vertices at level i are those vertices at distance i from the source s. An example of level graph is shown in Fig. 1.

Next, the algorithm finds a blocking flow on the level graph $\bar{G}^f$ of the residual network G^f. A blocking flow f in G is a flow such that each s - t path contains at least one critical edge. Recall that an edge e is critical if the amount of flow on it reaches its capacity: $f(e) = c(e)$. Fig. 2 gives examples for blocking flow.

Dinic's algorithm is then described as follows.

1. Initialize f to be the empty flow, and $G^f = G$.
2. Do the following until there is no $s - t$ path in G^f :
   - construct the level graph $\bar{G}^f$ of G^f.
   - find a blocking flow on $\bar{G}^f$.
   - Update f by adding the blocking flow to it, and update G^f.

Complete the analysis of Dinic's algorithm by solving the following questions.

(a) (15 points) Prove that, after each iteration of Dinic's algorithm, the distance from s to t in G^f is increased by at least 1 .

(b) (15 points) Design an $O(|V| \cdot |E|)$ time algorithm to compute a blocking flow on a level graph.

(c) (10 points) Show that the overall time complexity for Dinic's algorithm is $O(|V|^2 \cdot |E|)$

(d) (20 points) (challenging) We have seen in the class that the problem of finding a maximum matching on a bipartite graph can be converted to the maximum flow problem. Show that Dinic's algorithm applied to finding a maximum matching on a bipartite graph only requires time complexity $O(|E| \cdot \sqrt{|V|})$.

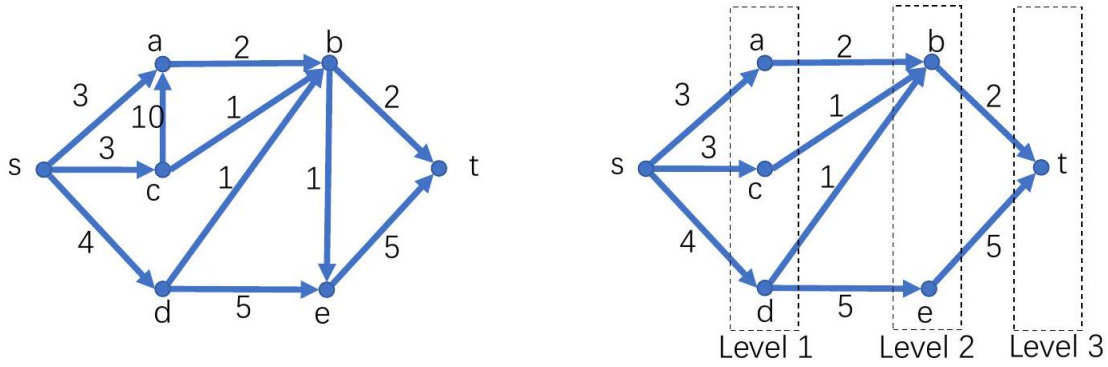


Figure 1: The graph shown on the right-hand side is the level graph of the graph on the left-hand side. Only edges pointing to the next levels are kept. For example, the edges (c, a) and (b, e) are removed, as they point at vertices at the same level. If there were edges pointing at previous levels, they should also be removed.

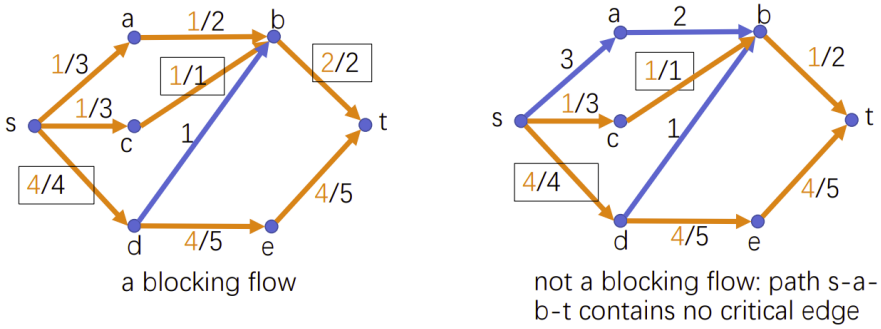


Figure 2: Blocking flow examples

It's a pity that we vegetables are too busy by the end of the term. I will finish it as short.

(a) The level is defined by the distance from s to a vertex i. If the s-t distance k does not increase after an iteration, it means there is still a s-t path with k. Since before the update the vertex t is k-level, this s-t path is also in $\bar{G^f}$: each vertex on the path is different level. However, during the iteration, the update rule requires us to find the critical path on this s-t path, which means after the update, this s-t path shouldn't exist.

(b) We can use DFS to find all the route and record the min capacity. The min capacity is updated both when going to successors and returning to predecessors. When DFS, you can

only go to and return back from each vertex once, and so there are at most $O(n)$ moves forward and backward. Each DFS will cause at least one pair of vertices to disconnect, and there are $O(|E|)$ DFSs. The time complexity is $O(|V| \cdot |E|)$ and it ensures in the $\bar{G^F}$ there are no more s-t path.

(c) There are $|V|$ vertices so there are initially at most $|V|$ levels. Each update is $O(|V| \cdot |E|)$ and the level will reduce by 1. The total complexity is $O(|V|^2 \cdot |E|)$

(d) I will skip it for now. If I have time in the future.

Problem 6. How long does it take you to finish the assignment (including thinking and discussing)? Give a score $(1, 2, 3, 4, 5)$ to the difficulty. Do you have any collaborators? Write down their names here.

Answer 5.

Difficult score 3. My collaborator is chatGPT. However, I only use it to confirm my thoughts.