ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Midterm Exam, Advanced Algorithms 2017-2018

- You are only allowed to have a handwritten A4 page written on both sides.
- Communication, calculators, cell phones, computers, etc... are not allowed.
- Your explanations should be clear enough and in sufficient detail that a fellow student can understand them. In particular, do not only give pseudo-code without explanations. A good guideline is that a description of an algorithm should be such that a fellow student can easily implement the algorithm following the description.
- **You are allowed to refer to material covered in the lecture notes** including theorems without reproving them. You may also refer to statements given in the exercise sheets and the homework sheet. You are however *not* allowed to refer to material from any specific solution to these exercises.
- **Do not touch until the start of the exam.**

Good luck!

Name: ______________________ **N° Sciper:** ______________

Problem 1 / 20 points	Problem 2 / 20 points	Problem 3 / 20 points	Problem 4 / 20 points	Problem 5 / 20 points

Total / 100

1 *(consisting of subproblems **a-b**, 20 pts)* **Basic questions.** This problem consists of two subproblems each worth 10 points.

1a *(10 pts)* Suppose we use the Simplex method to solve the following linear program:

$$\begin{aligned}
\textbf{maximize} \quad & 4x_1 - x_2 - 2x_3 \\
\textbf{subject to} \quad & x_1 - x_3 + s_1 = 1 \\
& x_1 + s_2 = 4 \\
& -3x_2 + 2x_3 + s_3 = 4 \\
& x_1,\ x_2,\ x_3,\ s_1,\ s_2,\ s_3 \geq 0
\end{aligned}$$

At the current step, we have the following Simplex tableau:

$$\begin{aligned}
x_1 &= 1 + x_3 - s_1 \\
s_2 &= 3 - x_3 + s_1 \\
s_3 &= 4 + 3x_2 - 2x_3 \\
\hline
z &= 4 - x_2 + 2x_3 - 4s_1
\end{aligned}$$

Write the tableau obtained by executing one iteration (pivot) of the Simplex method starting from the above tableau.

Solution:

Only x_3 has a positive coefficient in z, we will pivot x_3. We have $\nearrow x_3 \longrightarrow\ x_3 \leq\ \infty\ (1),\ x_3 \leq 3\ (2),\ x_3 \leq 2\ (3)$, Thus we use third equality to pivot x_3. Hence $x_3 = \frac{1}{2}(4 + 3x_2 - s_3)$. And we get

$$\begin{aligned}
x_1 &= 1 + \frac{1}{2}(4 + 3x_2 - s_3) - s_1 \\
s_2 &= 3 - \frac{1}{2}(4 + 3x_2 - s_3) + s_1 \\
x_3 &= \frac{1}{2}(4 + 3x_2 - s_3) \\
\hline
z &= 4 - x_2 + (4 + 3x_2 - s_3) - 4s_1
\end{aligned}$$

That is

$$\begin{aligned}
x_1 &= 3 + \frac{3x_2}{2} - \frac{s_3}{2} - s_1 \\
s_2 &= 1 - \frac{3x_2}{2} + \frac{s_3}{2} + s_1 \\
x_3 &= 2 + \frac{3x_2}{2} - \frac{s_3}{2} \\
\hline
z &= 8 + 2x_2 + -s_3 - 4s_1 \\
x_1 &:= 3\ x_2 := 0\ x_3 := 2\ s_1 := 0\ s_2 := 1\ s_3 := 0
\end{aligned}$$

1b *(10 pts)* Chef Baker Buttersweet just took over his family business - baking tasty cakes! He notices that he has m different ingredients in various quantities. In particular, he has $b_i \geq 0$ kilograms of ingredient i for $i = 1, \dots, m$. His family cookbook has recipes for n types of mouthwatering cakes. A kilogram of cake of type j is worth c_j CHF. For each recipe j, the cookbook says how many kilograms of each of the ingredients are needed to make one kilogram of cake of type j. One kilogram of cake of type j, for $j = 1, \dots, m$, needs precisely a_{ij} kilograms of ingredient i for all $i = 1, \dots, m$.

Chef wants to make $x_j \leq 1$ kilograms of cake of type j. Having studied linear programming, he knows that the maximum revenue he can get is given by the following linear program, where $A \in \mathbb{R}_{+}^{m \times n}$, $b \in \mathbb{R}_{+}^{m}$ and $c \in \mathbb{R}_{+}^{n}$.

$$\begin{array}{ll} \textbf{Maximize} & \displaystyle\sum_{j=1}^{n} c_j x_j \\ \textbf{subject to} & Ax \leq b \\ & 1 \geq x_j \geq 0 \quad \forall j. \end{array}$$

Chef realizes that he can use Hedge algorithm to solve this linear program (approximately) but he is struggling with how to set the costs $m_i^{(t)}$ at each iteration. Explain how to set these costs properly.

(In this problem you are asked to define the costs $m_i^{(t)}$. You do ***not*** *need to explain how to solve the reduced linear program that has a single constraint. Recall that you are allowed to refer to material covered in the lecture notes.)*

Solution: Here we give a detailed explanation of how to set the costs. Your solution does not need to contain such a detailed explanation.

The idea of using the Hedge method for linear programming is to associate an expert with each constraint of the LP. In other words, the Hedge method will maintain a weight distribution over the set of constraints of a linear problem to solve, and to iteratively update those weights in a multiplicative manner based on the cost function at each step.

Initially, the Hedge method will give a weight $w_i^{(1)} = 1$ for every constraint/expert $i = 1, \dots, m$ (the number m of constraints now equals the number of experts). And at each step t, it will maintain a convex combination $\vec{p}^{(t)}$ of the constraints (that is defined in terms of the weights). Using such a convex combination $\vec{p}$, a natural easier LP with a single constraint is obtained by summing up all the constraints according to $\vec{p}$. Any optimal solution of the original LP is also a solution of this reduced problem, so the new problem will have at least the same cost as the previous one. We define an oracle for solving this reduced problem:

Definition 1 *An oracle that, given $\vec{p} = (p_1, \dots, p_m) \geq \mathbf{0}$ such that $\sum_{i=1}^{m} p_i = 1$, outputs an optimal solution x^* to the following reduced linear problem:*

$$\begin{array}{ll} \textbf{\textit{Maximize}} & \displaystyle\sum_{j=1}^{n} c_j x_j \\ \textbf{\textit{subject to}} & \displaystyle\left(\sum_{i=1}^{m} p_i A_i\right) \cdot x \leq \sum_{i=1}^{m} p_i b_i \\ & x \geq 0 \end{array}$$

As explained, we associate an expert to each constraint of the covering LP. In addition, we wish to increase the weight of unsatisfied constraints and decrease the weight of satisfied constraints (in a smooth manner depending on the size of the violation or the slack). The Hedge algorithm for covering LPs thus becomes:

- Assign each constraint i a weight $w_i^{(1)}$ initialized to 1.

At each time t:

- Pick the distribution $p_i^{(t)} = w_i^{(t)}/\Phi^{(t)}$ where $\Phi^{(t)} = \sum_{i \in [N]} w_i^{(t)}$.
- Now *we define the cost vector instead of the adversary* as follows:
  - Let $x^{(t)}$ be the solution returned by the oracle on the LP obtained by using the convex combination $\vec{p}^{(t)}$ of constraints. Notice that cost of $x^{(t)}$, i.e., $c^\top x^{(t)}$, is at least the cost of an optimal solution to the original LP.
  - Define the cost of constraint i as

    $$m_i^{(t)} = b_i - \sum_{j=1}^{n} A_{ij} x_j = b_i - A_i x.$$

    Notice that we have a positive cost if the constraint is satisfied (so the weight will be decreased by Hedge) and a negative cost if it is violated (so the weight will be increased by Hedge).
- After observing the cost vector, set $w_i^{(t+1)} = w_i^{(t)} \cdot e^{-\varepsilon \cdot m_i^{(t)}}$.

Output: the average $\bar{x} = \frac{1}{T} \sum_{t=1}^{T} x^{(t)}$ of the constructed solutions.

(Primal) LP Relaxation

$$\begin{aligned}
\textbf{minimize} \quad & \sum_{S \in \mathcal{T}} c(S) x_S \\
\textbf{subject to} \quad & \sum_{S \in \mathcal{T} : e \in S} x_S \geq 1 \quad \text{for } e \in \mathcal{U} \\
& x_S \geq 0 \quad \text{for } S \in \mathcal{T}
\end{aligned}$$

(Dual)

$$\begin{aligned}
\textbf{maximize} \quad & \sum_{e \in \mathcal{U}} y_e \\
\textbf{subject to} \quad & \sum_{e \in S} y_e \leq c(S) \quad \text{for } S \in \mathcal{T} \\
& y_e \geq 0 \quad \text{for } e \in \mathcal{U}
\end{aligned}$$

Figure 1. The standard LP relaxation of set cover and its dual.

2 *(20 pts)* Recall that a set cover instance is specified by a universe $\mathcal{U} = \{e_1, \ldots, e_n\}$, a family of subsets $\mathcal{T}$, and a cost function $c : \mathcal{T} \to \mathbb{R}_+$. The task is to find a collection $C \subseteq \mathcal{T}$ of subsets of minimum total cost that covers all elements. The natural LP relaxation and its dual (as seen in class) are given in Figure 1.

In the homework, we analyzed a primal-dual algorithm for vertex cover. In this problem you should design and analyze a primal-dual algorithm for set cover that has an approximation guarantee of f, where f is the maximum number of sets any element belongs to: $f = \max_{e \in \mathcal{U}} |\{S \in \mathcal{T} : e \in S\}|$. In other words, you should prove that your primal-dual algorithm returns a set cover of weight at most f times the weight of an optimal solution.

(In this problem you are asked to (i) design the primal-dual algorithm and (ii) show that it has an approximation guarantee of f. We remark that an answer that solves and rounds the LP relaxation is rewarded 0 *points. Recall that you are allowed to refer to material covered in the lecture notes.)*

Solution: We present the following primal-dual algorithm which maintains a feasible dual solution, and eventually constructs a feasible primal solution.

```
1. Initialize the dual solution $y$ to be $y_e = 0$ for every $e \in \mathcal{U}$.
2. While $C = \{S \in \mathcal{T} : \sum_{e \in S} y_e = c(S)\}$ is not a set cover (i.e., $\cup_{S \in C} S \neq \mathcal{U}$):
     • Select an element $e \in \mathcal{U}$ that is not covered by any set in $C$ (i.e., choose
       an element $e \in \mathcal{U} \setminus (\cup_{S \in C} S)$.
     • Increase $y_e$ until one of the dual constraints becomes tight (i.e., $\sum_{e \in S} y_e =$
       $c(S)$ for some $S \notin C$).
3. Return $C = \{S \in \mathcal{T} : \sum_{e \in S} y_e = c(S)\}$.
```

This algorithm terminates because, in each iteration of the loop, a new element $e \in \mathcal{U}$ becomes covered. Therefore there may be at most n iterations.

Note that, by construction, the output C is a set cover and the computed dual solution is feasible. Let y be the dual solution when the primal-dual algorithm terminates. We have the following:

$$\begin{aligned}
\sum_{S\in C} c(S) &= \sum_{S\in C}\sum_{e\in S} y_e && \text{(by the definition of set } C\text{)}\\
&\le \sum_{S\in \mathcal{T}}\sum_{e\in S} y_e && \text{(because } y_e\text{'s are non-negative and } C\subseteq \mathcal{T}\text{)}\\
&= \sum_{e\in \mathcal{U}}\sum_{S\ni e} y_e && \text{(by rearranging the summations)}\\
&= \sum_{e\in \mathcal{U}} |\{S\in\mathcal{T} : e\in S\}|\cdot y_e \\
&\le f\cdot \sum_{e\in\mathcal{U}} y_e && (f=\max_{e\in\mathcal{U}} |\{S\in\mathcal{T} : e\in S\}|)\\
&\le f\cdot LP_{OPT} && \text{(by the weak-duality theorem)}\\
&\le f\cdot OPT. && \text{(because the LP is a relaxation)}
\end{aligned}$$

3 *(20 pts)* **LP-based algorithm for packing knapsacks.** Homer, Marge, and Lisa Simpson have decided to go for a hike in the beautiful Swiss Alps. Homer has greatly surpassed Marge's expectations and carefully prepared to bring n items whose total size equals the capacity of his and his wife Marge's two knapsacks. Lisa does not carry a knapsack due to her young age.

More formally, Homer and Marge each have a knapsack of capacity C, there are n items where item $i = 1, 2, \ldots, n$ has size $s_i > 0$, and we have $\sum_{i=1}^n s_i = 2 \cdot C$ due to Homer's meticulous preparation. However, being Homer after all, Homer has missed one thing: although the items fit perfectly in the two knapsacks fractionally, it might be impossible to pack them because items must be assigned integrally!

Luckily Lisa has studied linear programming and she saves the family holiday by proposing the following solution:

- Take *any* extreme point x^* of the linear program:

$$\begin{aligned}
x_{iH} + x_{iM} &\leq 1 && \text{for all items } i = 1, 2, \ldots, n \\
\sum_{i=1}^n s_i x_{iH} &= C \\
\sum_{i=1}^n s_i x_{iM} &= C \\
0 \leq x_{ij} &\leq 1 && \text{for all items } i = 1, 2, \ldots, n \text{ and } j \in \{H, M\}.
\end{aligned}$$

- Divide the items as follows:
  - Homer and Marge will carry the items $\{i : x^*_{iH} = 1\}$ and $\{i : x^*_{iM} = 1\}$, respectively.
  - Lisa will carry any remaining items.

Prove that Lisa needs to carry at most one item.

(In this problem you are asked to give a formal proof of the statement that Lisa needs to carry at most one item. You are not allowed to change Lisa's solution for dividing the items among the family members. Recall that you are allowed to refer to material covered in the lecture notes.)

Solution:

Note that, if x is a feasible solution, $x_{iH} + x_{iM} = 1$ for all $i = 1, \ldots, n$. Otherwise, if $x_{jH} + x_{jM} < 1$, we would have that (since $s_i > 0$ for every item i)

$$2 \cdot C = \sum_{i=1}^n s_i x_{iH} + \sum_{i=1}^n s_i x_{iM} = s_j \underbrace{(x_{jH} + x_{jM})}_{<1} + \sum_{i=1, i \neq j}^n s_i \underbrace{(x_{iH} + x_{iM})}_{\leq 1} < \sum_{i=1}^n s_i = 2 \cdot C,$$

which is a contradiction.

Now suppose that x^* is an extreme-point solution. We claim that $0 < x^*_{iH} < 1$ for at most one index i. Suppose that $0 < x^*_{iH} < 1$ is true for more than one index i. If so, we show that x^* can be written as a convex combination of two other feasible solutions, and hence x^* is not an extreme point, contradicting our choice of x^*. Assume $0 < x^*_{iH} < 1$ and $0 < x^*_{jH} < 1$ for $i \neq j$. Since $x_{iH} + x_{iM} = 1$ and $x_{jH} + x_{jM} = 1$, this also implies that $0 < x^*_{iM} < 1$ and $0 < x^*_{jM} < 1$. Now consider the solutions

$$x^{(1)} = \left(x^*_{1H}, x^*_{1M}, \ldots, x^*_{iH} + \epsilon, x^*_{iM} - \epsilon, \ldots, x^*_{jH} - \epsilon \left(\frac{s_i}{s_j}\right), x^*_{jM} + \epsilon \left(\frac{s_i}{s_j}\right), \ldots, x^*_{nH}, x^*_{nM}\right) \text{ and}$$
$$x^{(2)} = \left(x^*_{1H}, x^*_{1M}, \ldots, x^*_{iH} - \epsilon, x^*_{iM} + \epsilon, \ldots, x^*_{jH} + \epsilon \left(\frac{s_i}{s_j}\right), x^*_{jM} - \epsilon \left(\frac{s_i}{s_j}\right), \ldots, x^*_{nH}, x^*_{nM}\right).$$

We select $\epsilon > 0$ to be small enough so that all the values $x^*_{iH} \pm \epsilon, x^*_{iM} \pm \epsilon, x^*_{jH} \pm \epsilon\left(\frac{s_i}{s_j}\right), x^*_{jM} \pm \epsilon\left(\frac{s_i}{s_j}\right)$ stay in the range $[0,1]$ (note that, since $s_i > 0$ and $s_j > 0$, $0 < \frac{s_i}{s_j} < \infty$). As shown below, we can verify that the solutions $x^{(1)}$ and $x^{(2)}$ both satisfy the LP constraints, and hence are feasible solutions. For $x^{(1)}$, we have that

$$\sum_{i=1}^{n} s_i x^{(1)}_{iH} = \sum_{i=1}^{n} s_i x^*_{iH} - s_i\epsilon + s_j\epsilon\left(\frac{s_i}{s_j}\right) = \sum_{i=1}^{n} s_i x^*_{iH} = C$$

and

$$\sum_{i=1}^{n} s_i x^{(1)}_{iM} = \sum_{i=1}^{n} s_i x^*_{iM} + s_i\epsilon - s_j\epsilon\left(\frac{s_i}{s_j}\right) = \sum_{i=1}^{n} s_i x^*_{iM} = C.$$

Furthermore, for $x^{(1)}_{iH} + x^{(1)}_{iM} = x^*_{iH} + x^*_{iM} + \epsilon - \epsilon = 1$, $x^{(1)}_{jH} + x^{(1)}_{jM} = x^*_{jH} + x^*_{jM} - \epsilon\left(\frac{s_i}{s_j}\right) + \epsilon\left(\frac{s_i}{s_j}\right) = 1$, and for $k \neq i$ and $k \neq j$, $x^{(1)}_{kH} + x^{(1)}_{kM} = x^*_{kH} + x^*_{kM} = 1$. By a similar argument, we can show that $x^{(2)}$ is also feasible.

It is easy to see that $x^* = \frac{1}{2}x^{(1)} + \frac{1}{2}x^{(2)}$ and $x^{(1)} \neq x^*$, and hence, x^* is not an extreme point. $\square$

4 *(20 pts)* **Balancing degrees.** A beautiful result by the Swiss mathematician Leonhard Euler (1707 - 1783) can be stated as follows:

> Let $G = (V, E)$ be an undirected graph. If every vertex has an even degree, then we can orient the edges in E to obtain a directed graph where the in-degree of each vertex equals its out-degree.

In this problem, we address the problem of correcting an imperfect orientation A to a perfect one A' by flipping the orientation of the fewest possible edges. The formal problem statement is as follows:

Input: An undirected graph $G = (V, E)$ where every vertex has an even degree and an orientation A of E. That is, for every $\{u, v\} \in E$, A either contains the directed edge (u, v) that is oriented towards v or the directed edge (v, u) that is oriented towards u.

Output: An orientation A' of E such that $|A' \setminus A|$ is minimized and

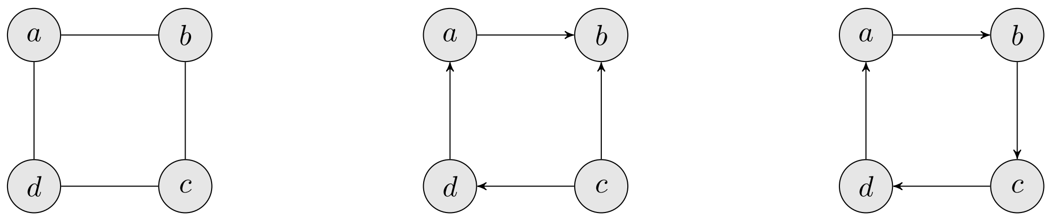
$$\underbrace{|\{u \in V : (u,v) \in A'\}|}_{\text{in-degree}} = \underbrace{|\{u \in V : (v,u) \in A'\}|}_{\text{out-degree}} \qquad \text{for every } v \in V.$$

Design and analyze a polynomial-time algorithm for the above problem.

(In this problem you are asked to (i) design the algorithm, (ii) analyze its running time, and (iii) show that it returns a correct solution. Recall that you are allowed to refer to material covered in the lecture notes.)

An example is as follows:

G $\qquad$ $A = \{(a,b), (c,b), (c,d), (d,a)\}$ $\qquad$ $A' = \{(a,b), (b,c), (c,d), (d,a)\}$



The solution A' has value $|A' \setminus A| = 1$ (the number of edges for which the orientation was flipped).

Solution: Consider the directed graph $G' = (V, E')$ obtained from G by replacing every edge $\{u, v\} \in E$ by the two arcs $e_1 = (u, v)$ and $e_2 = (v, u)$. If $e \in A'$, we assign weight $w_e = n^2 + 1$ to it, otherwise we set $w_e = n^2$.

Let $\delta^+(v) = \{u \in V : (v, u) \in E'\}$ denote the set of outgoing edges of v in G' and $\delta^-(v) = \{u \in V : (u, v) \in E'\}$ be the set of incoming edges of v in G'. With the arc set E' as ground set we define two partition matroids $\mathcal{M}_1$ and $\mathcal{M}_2$:

- To be independent in $\mathcal{M}_1$ one can take at most one of $\{(u, v), (v, u)\}$ for every $\{u, v\} \in E$, i.e.,

  $$\mathcal{I}_1 = \{F \subseteq E' : |F \cap \{(u, v), (v, u)\}| \leq 1 \text{ for all } \{u, v\} \in E\}.$$

  This matroid enforces the constraint that each edge should be oriented in one direction.

- To be independent in M_2, one can take at most $\frac{1}{2}\text{deg}(v)$ arcs among the set $\delta^+(v)$ of outgoing arcs for every v:

$$\mathcal{I}_2 = \{F \subseteq E' : |F \cap \delta^+(v)| \leq \frac{1}{2}\text{deg}(v), \text{ for all } v \in V\}\,.$$

Let solution S be the maximum weight independent set in the intersection of the two matroids $\mathcal{M}_1$, and $\mathcal{M}_2$. Now we prove that a solution S is feasible if and only if it is independent in both $\mathcal{I}_1$ and $\mathcal{I}_2$.

First observe that any solution with maximum weight, also has the maximum cardinality. Every solution of size k has weight at most $k \cdot (n^2 + 1)$, whereas any solution of size $k + 1$ has weight at least $(k + 1)n^2$ which is larger than any solution of size at most k. Thus the maximum weighted solution has maximum size i.e. $|A'|$.

Now we prove that any solution (with maximum cardinality) that is independent in $\mathcal{I}_2$, satisfies both indegree and outdegree constraints. Suppose $F \subseteq \mathcal{I}_2$ and $|F| = |A'|$. Thus we have

$$|A'| = \sum_{v \in V} |F \cap \delta^+(v)| \leq \sum_{v \in V} \frac{1}{2}\text{deg}(v) = |A'|.$$

Thus for all $v \in V$, we have $|F \cap \delta^+(v)| = \frac{1}{2}\text{deg}(v)$, so that $|F \cap \delta^-(v)| = \frac{1}{2}\text{deg}(v)$. Thus F is a feasible solution to the problem.

Recall that solution S has the maximum weight among all feasible solutions. Thus S has maximum cardinality, and among all the feasible solutions with the same cardinality, S maximizes $|E' \cap A'|$. By Edmonds, Lawler's theorem, there is a polynomial-time algorithm for finding a maximum weight independent set in the intersection of two matroids $\mathcal{M}_1$, and $\mathcal{M}_2$.

5 *(20 pts)* **Comparing algorithms with little communication.** Two excellent students, Alice from EPFL and Bob from MIT, have both built their own spam filters. A spam filter is an algorithm that takes as input an email and outputs 1 if the email is spam and 0 otherwise. Alice and Bob now want to compare their two spam filters.

To perform the comparison, they both download the same huge data set consisting of n emails out of which some are spam. Alice then runs her spam filter on the data set to obtain $a_1, a_2, \ldots, a_n$ where $a_i \in \{0,1\}$ is the output of her spam filter on the i:th email in the data set. Similarly, Bob runs his spam filter on the data set to obtain $b_1, b_2, \ldots, b_n$ where $b_i \in \{0,1\}$ is the output of his spam filter on the i:th email in the data set. Their goal is then to determine whether their outputs are the same.

An issue that they face is that $a_1, a_2, \ldots, a_n$ are stored on Alice's computer and $b_1, b_2, \ldots, b_n$ are stored on Bob's computer. They thus need to transfer (or communicate) information to solve the problem. A trivial solution is for Alice to transfer all her outputs $a_1, a_2, \ldots, a_n$ to Bob who then performs the comparison. However, this requires Alice to send n bits of information to Bob; an operation that is very costly for a huge data set. In the following, we use randomization to achieve a huge improvement on the number of bits transfered between Alice and Bob.

Specifically, motivated by something called pseudo-random generators, we assume that Alice and Bob have access to the same randomness (called shared randomness). That is, Alice and Bob have access to the same infinite stream of random bits $r_1, r_2, \ldots$.

Your task is now to use this shared randomness to devise a randomized protocol of the following type:

- As a function of $a_1, a_2, \ldots, a_n$ and the random bits $r_1, r_2, \ldots$, Alice computes a message m that consists of only 2 bits. She then transmits this 2-bit message m to Bob.
- Bob then, as a function of $b_1, b_2, \ldots, b_n$, the message m, and the random bits $r_1, r_2, \ldots$, outputs EQUAL or NOT EQUAL.

Bob's output is correct if he outputs EQUAL when $a_i = b_i$ for all $i \in \{1, \ldots, n\}$ and NOT EQUAL otherwise. Your protocol should ensure that Bob outputs the correct answer with probability at least $2/3$, where the probability is over the random bits $r_1, r_2, \ldots$.

(In this problem you are asked to (i) explain how Alice computes the message m of 2 bits (ii) explain how Bob calculates his output, and (iii) prove that Bob's output is correct with probability at least 2/3. A correct solution where Alice sends a message m of $O(\log n)$ bits is rewarded 12 points. Recall that you are allowed to refer to material covered in the lecture notes.)

An interesting fact (but unrelated to the exam) is that any correct deterministic strategy would require Alice and Bob to send n bits of information.

Solution: Let $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_n)$. Alice generates two independent random vectors $\mathbf{r}^1, \mathbf{r}^2 \sim \text{Uniform}(\{0,1\}^n)$ using the shared random bits. Note that this is equivalent to choosing each element of $\mathbf{r}^1$ and $\mathbf{r}^2$ independently and uniformly at random from $\{0,1\}$. Alice then computes $x_1 = \langle \mathbf{a}, \mathbf{r}^1 \rangle \bmod 2$ and $x_2 = \langle \mathbf{a}, \mathbf{r}^2 \rangle \bmod 2$, and transmits (x_1, x_2) to Bob. Bob uses the shared random bits to generate the same vectors $\mathbf{r}^1$ and $\mathbf{r}^2$, and computes $y_1 = \langle \mathbf{b}, \mathbf{r}^1 \rangle \bmod 2$ and $y_2 = \langle \mathbf{b}, \mathbf{r}^2 \rangle \bmod 2$. If $x_1 = y_1$ and $x_2 = y_2$, Bob outputs EQUAL. Otherwise, Bob outputs NOT EQUAL.

We prove that the above protocol succeeds with probability at least 2/3. Clearly, it succeeds whenever $\mathbf{a} = \mathbf{b}$. Thus, we only have to show that it succeeds with probability at least 2/3 when

$\mathbf{a} \neq \mathbf{b}$. We first show that $\Pr[x_1 = y_1 | \mathbf{a} \neq \mathbf{b}] = 1/2$. Notice that

$$\begin{aligned}\Pr[x_1 = y_1 | \mathbf{a} \neq \mathbf{b}] &= \Pr[\langle \mathbf{a}, \mathbf{r}^1 \rangle \bmod 2 = \langle \mathbf{b}, \mathbf{r}^1 \rangle \bmod 2 | \mathbf{a} \neq \mathbf{b}] \\ &= \Pr[\langle \mathbf{a} - \mathbf{b}, \mathbf{r}^1 \rangle \bmod 2 = 0 | \mathbf{a} \neq \mathbf{b}].\end{aligned}$$

Let $\mathbf{c} = \mathbf{a} - \mathbf{b}$. Since $\mathbf{a} \neq \mathbf{b}$, we have that $\mathbf{c} \neq \mathbf{0}$. This means that for at least one index j, $c_j = \pm 1$. Now fix such j, and suppose that we have chosen all elements r^1_i for $i \neq j$ independently and uniformly at random from $\{0, 1\}$. Then, there will be only one choice for r^1_j that would make $\langle \mathbf{c}, \mathbf{r}^1 \rangle = 0$. Thus, using the principle of deferred decisions, we have that

$$\Pr[\langle \mathbf{c}, \mathbf{r}^1 \rangle \bmod 2 = 0 | \mathbf{a} \neq \mathbf{b}] = \Pr[\langle \mathbf{a} - \mathbf{b}, \mathbf{r}^1 \rangle \bmod 2 = 0 | \mathbf{a} \neq \mathbf{b}] = 1/2.$$

As a result, $\Pr[x_1 = y_1 | \mathbf{a} \neq \mathbf{b}] = 1/2$, and similarly, $\Pr[x_2 = y_2 | \mathbf{a} \neq \mathbf{b}] = 1/2$. Since $\mathbf{r}^1$ and $\mathbf{r}^2$ are independent from each other, $\Pr[(x_1, x_2) = (y_1, y_2) | \mathbf{a} \neq \mathbf{b}] = (1/2) \cdot (1/2) = 1/4$, which implies that $\Pr[(x_1, x_2) \neq (y_1, y_2) | \mathbf{a} \neq \mathbf{b}] = 1 - 1/4 \geq 2/3$ as required. □