Theoretical Computer Science Round

Instructions

1. Do not look at the test before the proctor starts the round.
2. This test consists of three problems, which require proofs, to be solved within a time frame of 60 minutes. There are 300 points total.
3. No computational aids other than pencil/pen are permitted.
4. Answers should be written and clearly labeled on sheets of blank paper. Each numbered problem should be *on its own sheet*. If you have multiple pages, number them as well (e.g. 1/3, 2/3).
5. Write your team name on the upper-right corner and the problem and page number of the problem whose solution you are writing on the upper-left corner on each page you submit. Papers missing these will not be graded. Problems with more than one submission will not be graded.
6. Write legibly. Illegible handwriting will not be graded.
7. If you believe that the test contains an error, submit your protest to the 2026 CMIMC discord.

Calculators Made In Mellon College

In an effort to raise more money, the CMIMC officers have made a company that buys and sells calculators. However, they ran out of storage space, so they don't actually want to have any calculators in their inventory. Every day, customers come to them with an "order", which indicates that they want to either buy q calculators for p dollars each, or sell q calculators for p dollars each. Then, the CMIMC officers will buy calculators from sellers and sell calculators to buyers, in a way that makes a profit. However, **they cannot take part of a customer order, they must fill the entire quantity**. For example, if two people want to buy a calculator for 10 dollars each, and one person is selling two calculators for 8 dollars each, the CMIMC officers can make 4 dollars from these customers by filling their orders. However, the officers need a way to make sure they make the most amount of money possible each day. Specifically, they need to solve the following problem:

Given a list of n tuples (d_i, q_i, p_i), with $d_i = -1$ or 1, $q_i \leq 1000$, $p_i \leq 200$, find a subset $S \subseteq [n]$ that maximizes

$$\sum_{i \in S} d_i q_i p_i$$

Such that

$$\sum_{i \in S} d_i q_i = 0$$

For example, suppose the company gets the following 8 orders on a given day:

$$(-1, 4, 41)$$

$$(1, 5, 44)$$

$$(1, 3, 45)$$

$$(1, 100, 40)$$

$$(1, 11, 43)$$

$$(-1, 95, 48)$$

$$(-1, 3, 43)$$

$$(-1, 12, 38)$$

Then the maximum profit the company can make is 79.

Part 1: Provide an algorithm below that solves the problem given the input constraints presented in the problem statement. When we ask for an $O(f(n))$ solution, this refers to the algorithm having $O(f(n))$ time complexity. I.e., the number of steps required of the algorithm grows asymptotically to $f(n)$. The algorithm does not need to be written in compileable code, just pseudo-code. The rubric is below:

- **10 point** for a $O(2^n)$ solution.
- **50 points** for a $O(n^2)$ solution.

Part 2: To celebrate the implementation of their new trading algorithm, the CMIMC officers are now looking for new employees to help boost their trading efforts. Their interview consists of a single trading challenge, which is defined as follows:

There are $T = 2026$ rounds. You start with one dollar, and on round i the shares of a calculator firm are at price p_i. You do not know the value of p_i until you reach round i, but you know that the prices are integers such that $1 \le p_i \le P$. Your goal is to buy low and sell high to make as much profit as possible. A trade consists of two rounds (i_b, i_s), where $i_b < i_s$, where you buy as many shares at price p_{i_b} as you can afford, and sell them all at price p_{i_s} (you are allowed to buy fractional shares, i.e. if $p_3 = 10$ and you have 6 dollars, if you choose to buy on round 3 you will spend all 6 dollars for $\frac{6}{10}$ of a share). You don't need to give both of these rounds at once when you initiate a trade, but once you choose to buy at round i_b, you cannot buy again until you sell at some round i_s. I.e., each trade consists of two transactions. If you bought at some round and have not sold by round T, you will be forced to sell at round T even if $p_T < p_{i_b}$. You can only make 3 trades in the entire challenge.

You are tasked with constructing a deterministic algorithm for this challenge which performs as well as possible. Algorithm performance is defined as follows:

- For a trade (i_b, i_s), the "gain" of the trade is defined as the ratio $\frac{p_{i_s}}{p_{i_b}}$. The overall performance of an algorithm on a given price sequence is the product of all of the (at most 3) gains. For an algorithm ALG, and a price sequence $\mathbf{p}_T$, the function $g(\text{ALG}, \mathbf{p}_T)$ returns the product of the gains of the algorithm ran on this price sequence. If an algorithm makes no trades on a given price sequence, the output of this function is just 1.
- Let OPT denote the optimal algorithm that generates the largest gain given prior knowledge of the entire sequence (i.e. best performance if you can see in the future). Given an algorithm ALG, its relative performance on a given price sequence $\mathbf{p}_T$ is defined as
$$\frac{g(\text{OPT}, \mathbf{p}_T)}{g(\text{ALG}, \mathbf{p}_T)}$$
Note that a smaller ratio means better performance as you want to perform as close to optimal as possible.
- Then, the performance of an algorithm is its **competitive ratio**, i.e. the largest relative performance over all possible price sequences. In other words,
$$\max_{\mathbf{p}_T} \frac{g(\text{OPT}, \mathbf{p}_T)}{g(\text{ALG}, \mathbf{p}_T)}$$
Intuitively, this means we measure performance as the worst possible performance compared to the optimal algorithm over all possible universes. Again, a smaller value here means better performance.
- We say an algorithm is $f(P)$-competitive if its competitive ratio is less than or equal to $f(P)$ (note that this is indeed a function of P, the largest possible value of p_i).

Write your algorithm on the following page. The rubric is below:

- **10 point** for a P^3-competitive algorithm.
- **30 point** for a (P^3/c)-competitive algorithm for some constant $c > 1$ that does **not** depend on P.
- **50 point** for a $P^{7/3}$-competitive algorithm.

Weighted Coins

Suppose we have 10^6 **weighted** coin flips, where each coin flip has a 10% chance of landing on heads.

For some integer N, we wish to use use these flips to make a uniform random integer generator on $\{1, \cdots, N\}$. Our generator does not have to be perfectly uniform, but the probability of generating any integer $k \in \{1, \cdots, N\}$ must lie within the interval:

$$\Pr(k) \in \left[\frac{1}{N} - \frac{1}{N^2},\ \frac{1}{N} + \frac{1}{N^2}\right] \ \forall k \in \{1, \cdots, N\}$$

You should design an algorithm which takes in the result of all 10^6 coin flips in an ordered list as the input, and outputs an integer from 1 to N inclusive, where N is a number of your choice. The coin flips you get are the only source of randomness you may have, so given the same ordered list of coin flips as input, your algorithm must always give the same output.

Your problem will be scored based on your choice of N. The higher your value of N, the higher your score, but only if your algorithm satisfies the constraints outlined above. To earn points on this problem, you must actually describe an algorithm, not just prove that *some* algorithm satisfying the given bound must exist. (Note that the algorithm you describe for N doesn't necessarily need to work for all values $n < N$).

You may use the following theorem without proof:

- For any interval of 200 consecutive positive integers less than 10^6, there exists at least one prime.
- Let $X_1, \ldots, X_n$ be independent random variables such that $a_i \leq X_i \leq b_i$. Consider the sum of these random variables,

  $$S_n = X_1 + \cdots + X_n.$$

  Then Hoeffding's theorem states that, for all $t > 0$,

  $$\Pr(S_n - \mathbb{E}[S_n] \geq t) \leq \exp\left(\frac{-2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right),$$

  $$\Pr(|S_n - \mathbb{E}[S_n]| \geq t) \leq 2\exp\left(\frac{-2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

  Here $\mathbb{E}[S_n]$ is the expected value of S_n, and $\exp(x) = e^x$, with $e \approx 2.72$.

Scoring

- **100 points** for $2^{35{,}000}$
- **95 points** for $2^{30{,}000}$
- **85 points** for $2^{25{,}000}$
- **80 points** for $2^{15{,}000}$
- **70 points** for $2^{10{,}000}$
- **65 points** for $2^{3{,}000}$
- **50 points** for 10^{21}
- **45 points** for $2.4 * 10^{11}$
- **40 points** for $9.9 * 10^5$
- **30 points** for 10^5
- **20 points** for 1000
- **10 points** for 10
- **1 point** for 1

Testing the Freidkin Gate

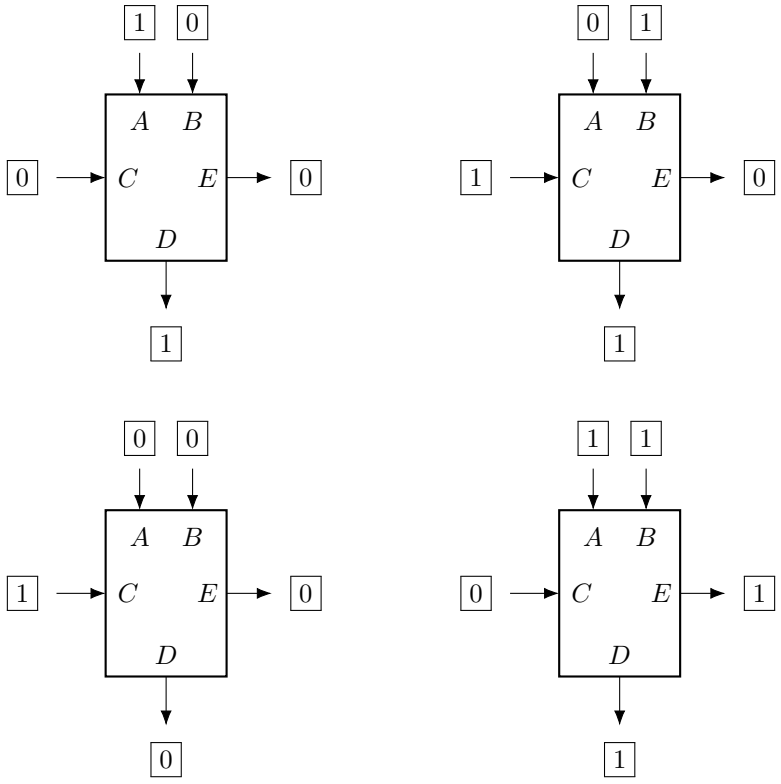
In Boolean logic, a "gate" takes in some number of binary (either 0 or 1) inputs and produces some number of binary outputs. The Fredkin gate is a Boolean logic gate with inputs $A, B, C \in \{0, 1\}$ and outputs $D, E \in \{0, 1\}$ defined as follows:

$$\text{If } C = 0, \text{ then } D = A \text{ and } E = B; \qquad \text{if } C = 1, \text{ then } D = B \text{ and } E = A.$$

We can write this more compactly as follows:

$$F(A, B; C) = (D, E) = \begin{cases} (A, B) & \text{if} \quad C = 0 \\ (B, A) & \text{if} \quad C = 1 \end{cases}$$

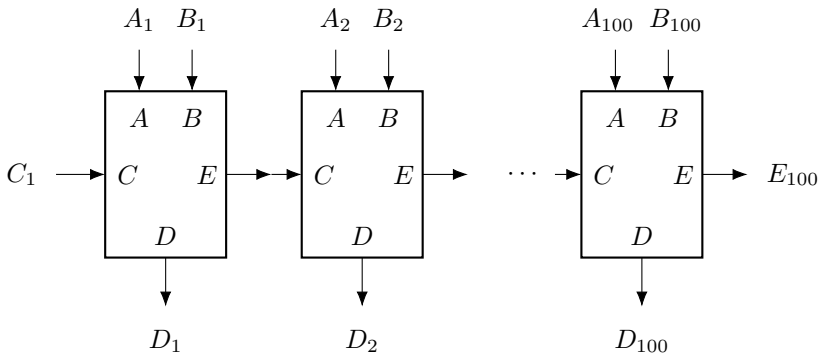
A few example inputs and outputs for a Fredkin gate are shown in the following diagram:



The Carnegie Mellon Nanofabrication Lab is manufacturing Fredkin gates. A gate is called "defective" when it does not match the expected behavior of a Fredkin gate: in other words, there is some input (A, B, C) where the output (D, E) does not match the definition above. We assume that all gates, whether they are defective or non-defective, will always produce the same output if they are given the same input (they are deterministic). Notice that for each fixed input (A, B, C), there are exactly three possible incorrect output pairs (D, E) and one correct output pair.

For a single gate, a "test" consists of choosing some binary input (A, B, C), applying it to the circuit, and measuring the outputs (D, E) of the gate. Notice that testing all $2^3 = 8$ possible input triples (A, B, C) and checking whether the measured outputs match the expected outputs is sufficient to detect whether a single gate is defective or non-defective.

Now consider a setup where $N = 100$ gates are connected to each other in a chain as follows:



Notice that the output value E from one gate becomes the input value C for the next gate. **IMPORTANT: Assume that at most one gate in this chain is defective**.

A "test" for this chain of gates consists of choosing a 201-bit input vector $(C_1, A_1, B_1, \cdots, A_{100}, B_{100})$, applying it to the circuit, and measuring the 101-bit output vector $(D_1, \cdots, D_{100}, E_{100})$.

Problem. Prove that there is a testing strategy that can always correctly detect whether one of the gates in the chain is defective using N total tests of the chain, where N is a number of your choice. Your solution will be scored based on your value of N; smaller values of N will receive more points.

Scoring

- $100 - 20 \cdot \frac{N-\textbf{OPTIMAL}}{100-\textbf{OPTIMAL}}$ **points** for $N \leq 100$, where **OPTIMAL** denotes the optimal N solution.
- **60 points** for 800
- **45 points** for 2400
- **25 points** for 2^{100}
- **15 points** for 2^{201}