This is an oral presentation assignment. Again, there are three regular problems (#1-#3) and one programming problem (#4). You should work in groups of three. The sign-up sheet will be online soon (details on Piazza) and your group should sign up for a 1-hour slot. Each person in the group must be able to present every regular problem. The TA/Professor will select who presents which regular problem. You are not required to hand anything in at your presentation, but you may if you choose.

The programming problem will be submitted to autolab, due Mar 10th, 11:59pm. You will not have to present anything orally for this. You can discuss the problem with your group-mates, but must write the program by yourself. Please do not copy.

(25 pts) 1. **Eliminating a Subsequence.**

You’re given a string $S$ of length $n$, and a string $F$ of length $m$. $F$ is called the forbidden word. Both of these strings are from some finite alphabet $\Sigma$.

In the following parts you are to give an algorithm to compute the minimum number of letters that must be deleted from $S$ to form a new string $S'$ such that there is no subsequence of $S'$ that equals $F$. In other words $|\text{LCS}(S', F)| < |F|$. 

(a) Assume that $F$ contains no repeated letter. For full points give an algorithm that runs in $O(n + m)$ time. (A weaker time bound of $O(nm)$ will get you partial credit.)

(b) Removing the assumption of part (a), give an algorithm that runs in time $O(nm)$.

(25 pts) 2. **Locating stores in DEN.** Having graduated, you work for a company that is charged with locating $s \geq 1$ food shops in Denver airport. This terminal is narrow and long, and you model it as a straight line. There are $n \geq s$ gates, which are located at positions $0 = a_1 < a_2 < \ldots < a_n$ on the line, with distances on the line modeling walking distances between the points. You are allowed to locate the shops anywhere on the line. If you decide to locate the $s$ shops at positions $b_1, b_2, \ldots, b_s$, the “pain” of this solution is the distance from an average gate to its closest shop, i.e.,

$$\frac{1}{n} \sum_{i=1}^{n} \left( \min_{j=1}^{s} |a_i - b_j| \right).$$

Give a dynamic programming algorithm to find the solution (i.e., the location of shops) that incurs the least pain, and that runs in time $O(n^3)$. It should not depend on the values of $a_i$.

Hint: consider subproblems that correspond to a prefix $\{a_1, \ldots, a_i\}$ of the gates and a number of shops $s' \leq s$. One more useful fact is that we need only consider solutions that place shops at a subset of the gate locations $\{a_i\}$ (i.e., $\{b_j\} \subseteq \{a_i\}$); this can be seen most easily in the case of $s = 1$ where if the shop is between $a_i$ and $a_{i+1}$, we can simply move it in whichever direction has more gates, or to either one if there is a tie. You may use this fact without proof, but convince yourself that it is true.
3. (The 61C Bus.) The eternal dilemma is: wait for the bus, or just walk to school? And if indeed you’re to wait, how long before you give up? This is what we’ll solve.

We model this process as a zero-sum game. Walking to work takes $A$ units of time. The bus takes $B$ units of time. The bus arrives at time 1 or 2 or 3, etc. If the bus arrives at time $t \geq 1$ and you take it, you reach school at time $t + B$. However, if you have waited $w \geq 0$ units of time, and if the bus has not yet arrived, you may decide to walk, which takes $A$ units of time, so you reach at time $w + A$. (Once you decide to start walking you cannot catch the bus any more.)

Your actions are “wait $w$ time; if no bus yet, walk”: one action for every integer $w \geq 0$. (So $w = 0$ means you don’t wait at all.) The adversary’s actions are: “bus arrives at time $t$”, one for every positive integer $t \geq 1$.

For instance if $w = 2$ and Will $t \leq 2$ then you catch the bus and arrive at time $t + B$, but if $w = 2, t = 3$ then you wait 2 time steps, (leave just before the bus arrives) and reaching at time $2 + A$.

Given the pair of actions $(t, w)$, your pain (aka. the payoff to the adversary) is the ratio: (the time it took you) divided by (the optimum time it would take you if you knew when the bus were to arrive).

(a) Let $R$ be the matrix of payoffs to the adversary whose rows $t$ are the adversary’s actions and columns $w$ are your actions, then for any $t \geq 1$ and $w \geq 0$, write down the mathematical expression for the payoff to the adversary.

For the rest of the parts, assume that $B = 1$ and $A = 3$.

(b) This game has an infinite number of columns and an infinite number of rows. Let’s add one more row, called “row $\infty$” for the scenario that the bus never arrives. Argue that without loss of generality, we can assume the adversary chooses only rows $t = 1$ or $t = \infty$. Formally, argue that for any $t \geq 2$ we have $R_{tw} \leq R_{\infty w}$ for all $t$. This means that your strategy achieving expected payoff $V$ in a world with only those two scenarios possible (bus arrives at time $t = 1$, or it never arrives) will achieve also payoff $V$ over the whole range of scenarios.

(c) Now that the game has just two rows, argue that the minimax-optimal strategy can safely put probability 0 on all columns except for $w \in \{0, 1\}$. Formally, argue that for any $w > 1$ we have $R_{tw} \geq R_{t1}$ for $t \in \{1, \infty\}$.

(d) Now write down the 2-by-2 matrix that results. Write down the LP for the minimax-optimal strategy for you, the column player, and solve it to find the value of the game.

(e) Finally, suppose there is an bus driver (row player) who controls the arrival time of the bus, and is trying to cause you the most pain (get the highest payoff for herself). Solve for the row player’s strategy.
(25 pts) 4. **Travelling Salesperson Problem**

In this problem you will implement an algorithm that can solve the Travelling Salesperson Problem (TSP). We discussed this problem in Lecture #11. The input to your program will be a connected graph $G$ with $n$ vertices and $m$ edges. Each edge $\{u, v\}$ has a non-negative length in each direction, and $\text{len}(u, v)$ may be different from $\text{len}(v, u)$. Your program will compute the length of the shortest TSP tour for the given graph, and also output the tour. The time limit is 10 seconds. For Python we’ll give 1 minute.

**Input**

First line consists of two positive integers $n$ and $m$. $n \leq 20$ and $n-1 \leq m \leq n(n-1)/2$.

The following $m$ lines each contain four integers that represent an edge. These numbers are $i \ j \ d_{i,j} \ d_{j,i}$. This means that there is an edge $\{i, j\}$ in the graph and the directed length from $i$ to $j$ is $d_{i,j}$ and the directed length from $j$ to $i$ is $d_{j,i}$. Each edge will occur at most once in the list in either direction. There are no self loops, and the graph is connected. The vertex numbers are in $[0, n-1]$, and the lengths of the edges are in $[0, 10^6]$.

**Output**

The first line of output contains the cost of the shortest travelling salesperson tour for the given graph.

The second line contains list of the cities in the order visited, starting at city 0 and ending at city 0. Each pair of neighboring cities on this list must be an edge in $G$. Any tour which is of minimum cost is acceptable.

See the samples below for formatting details.

**Samples**

For example, if the input is:

```
5 5
0 1 1 1
1 2 1 1
2 3 1 2
3 4 1 1
4 0 1 1
```

Then the output (which is unique in this case) will be:

```
optimal tour cost = 5
 tour: 0 1 2 3 4 0
```
Samples

For example, if the input is:

```
6 6
0 1 1 1
1 2 1 1
2 3 1 1
3 1 1 2
2 4 1 1
3 5 1 1
```

Then the output might be:

```
optimal tour cost = 9
tour: 0 1 2 4 2 3 5 3 1 0
```