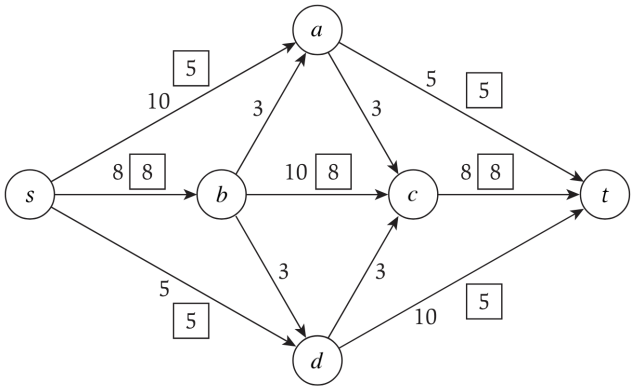
# Section 10: Solutions

## 1. Short answer

These sample questions relate only to material after the midterm, but the real exam will be cumulative.

(a) (KT 7.2) The figure below depicts an instance of maximum flow with the original graph and capacities. The values in the squares denote the amount of flow currently being sent through each edge. Edges with no square currently have no flow being pushed through them.




◯ The flow depicted is a maximum flow.

◯ The flow depicted is not a maximum flow.

Briefly justify your answer.

**Solution:**

Not a maximum flow. 2 more units of flow can be pushed using the augmenting path $s$, $a$, $c$, $b$, $d$, $t$.

(b) (KT 8.1) Recall the Interval Scheduling problem: Given a collection of intervals and an integer $k$, determine if the collection contains at least $k$ nonoverlapping intervals.

(i) Does Interval Scheduling $\leq_p$ Vertex Cover?

◯ Yes

◯ No

◯ Unknown, because the answer would resolve P vs. NP

Briefly justify your answer.

**Solution:**

Yes. Many possible reasons:

- Vertex Cover is NP-complete, in particular NP-hard, and Interval Scheduling is clearly in NP (the certificate is the list of $k$ nonoverlapping intervals). $A \leq_p B$ whenever $B$ is NP-hard and $A$ is in NP.
- Interval Scheduling is in P, as we solved it with a greedy algorithm earlier in this class. $A \leq_p B$ is always true when $A$ is in P.
- We can directly show that Interval Scheduling $\leq_p$ Independent Set (draw an edge between two

intervals if they overlap), and we showed that Independent Set $\leq_p$ Vertex Cover in lecture.

(ii) Does Independent Set $\leq_p$ Interval Scheduling?

◯ Yes

◯ No

◯ Unknown, because the answer would resolve P vs. NP

Briefly justify your answer.

**Solution:**

Unknown. Because Independent Set is NP-complete and Interval Scheduling is in P, Independent Set $\leq_p$ Interval Scheduling would imply that an NP-complete problem is solvable in polynomial time, which is unknown.

(c) Recall the Set Cover problem: Given a collection of sets containing objects, determine the minimum number of sets needed to cover all objects. A greedy attempt for Set Cover is:

```
1: while there exists an uncovered object do
2:     choose a set that covers the most number of still-uncovered objects
```

Suppose you are given an instance of Set Cover in which every set contains exactly 2 elements. Then this algorithm returns a set cover that is at most a factor 2 larger than the minimum set cover.

◯ True

◯ False

Briefly justify your answer.

**Solution:**

True. If there are $n$ objects, the algorithm returns at most $n$ sets because every set chosen contains at least 1 new object (or really, $n-1$ because the first set chosen contains 2 new objects). Since every object must be covered, and every set contains only 2 elements, we require at least $n/2$ sets. Thus the approximation ratio is 2.

## 2. Dynamic programming

*A version of this problem appeared on the Section 5 handout.*

Given two strings, $s = s_1 \dots s_m$ with length $m$ and $t = t_1 \dots t_n$ with length $n$, find the length of their longest common subsequence. (A subsequence may not be contiguous. That is, one finds a subsequence by taking any subset of the indices, and putting together the letters at those indices in their original order.)

Here are a few examples:

- **Input**: $s =$ `backs`, $t =$ `arches`
  **Solution**: The longest common subsequence is `acs`, so the output should be 3.
- **Input**: $s =$ `skaters`, $t =$ `hated`
  **Solution**: The longest common subsequence is `ate`, so the output should be 3.

This problem can be solved with dynamic programming. Give a recurrence, including the base cases, that would be the basis for a dynamic programming algorithm. You should state in what order you will evaluate the subproblems, and briefly explain why your recurrence is correct.

**Solution:**

Let $\text{OPT}(i, j)$ be the longest common subsequence between elements $1..i$ in $s$ and $1..j$ in $t$. The parameter $i$ ranges from 0 to $m$, and the parameter $j$ ranges from 0 to $n$ (so we have our base cases in our memoization table for ease of calculation).

The key idea is that we can always choose to pair $s_i$ and $t_j$ if they are equal. To explain this, consider any optimal solution.

- If $s_i$ and $t_j$ were both unpaired, that solution would not be optimal because we could just pair them.
- If $s_i$ is unpaired and $t_j$ is paired with $s_k$, where $k < i$, another common subsequence with the same length would be to leave $s_k$ unpaired and pair $s_i$ with $t_j$, which is what we do.
- If $t_j$ is unpaired but $s_i$ is paired, the case is similar.

Thus we pair $s_i$ and $t_j$ if they are equal, resulting in the recurrence (for $i, j \geq 1$):

$$\text{OPT}(i,j) = \begin{cases} 1 + \text{OPT}(i-1, j-1) & \text{if } s_i = t_j \\ \max(\text{OPT}(i-1,j), \text{OPT}(i,j-1)) & \text{if } s_i \neq t_j \end{cases}$$

The base cases are $\text{OPT}(i, 0) = \text{OPT}(0, j) = 0$ for all $i$ and $j$.

To evaluate, first do all the base cases, then either an outer loop for $i$ and inner loop for $j$, or vice versa.

## 3. Network flows

A group of traders are leaving Switzerland, and need to convert their cash in Francs (the local currency) into various international currencies. Meanwhile, the central bank is updating the security of its cash bills, and needs to collect as much cash in Francs as possible in order to efficiently retire the old bills.

There are $n$ traders and $m$ currencies. Trader $i$ has $T_i$ Francs to convert. The bank has $C_j$ of currency $j$, and the exchange rate is $R_j$ of currency $j$ for every 1 Franc. Trader $i$ is traveling to multiple countries and is willing to trade anywhere from $L_{ij}$ to $H_{ij}$ of their Francs for currency $j$. For example, a trader with 1000 Francs might be willing to convert between 300 and 700 of their Francs for US dollars, between 200 and 500 of their Francs for Euros, and exactly 0 Francs for Japanese yen. It would be valid to have them trade 400 Francs for US dollars and 400 Francs for Euros. Assume that $\sum_j L_{ij} \leq T_i$.

All traders give their requests to the bank at the same time, and the bank is deciding how to fulfill them. Describe an efficient algorithm that determines whether or not the bank can satisfy all requests, and if so, a method of satisfying the requests to maximize the amount of Francs it collects. Briefly explain all the choices made in your algorithm.

**Solution:**

First, attempt to give every trader their minimum requests $L_{ij}$ by checking whether or not $C_j/R_j \geq \sum_i L_{ij}$ for all $j$ (if the bank has at least as much of currency $j$, after exchange rate, as the total amount demanded). If not possible, return "not possible".

Then, set up a flow network with a source $s$, sink $t$, a row of vertices $t_1, ..., t_n$ represents the traders, and a row of vertices $b_1, ..., b_m$ represents the currency held by the bank. Assign capacities to edges as follows:

- $(s, t_i)$ has capacity $T_i - \sum_j L_{ij}$, the total amount trader $i$ still wants to change after giving their minimum request.
- $(t_i, b_j)$ has capacity $H_{ij} - L_{ij}$, the maximum number of Francs trader $i$ still wants to trade into currency $j$ after giving their minimum request.
- $(b_j, t)$ has capacity $C_j/R_j - \sum_i L_{ij}$, the number of Franc equivalents the bank has of currency $j$ after giving away all minimum requests.

The maximum flow in this graph, plus the amount given by satisfying the minimum requests, is the maximum number of Francs the bank can receive.

# 4. Linear programming

*A version of this problem appeared on the Section 8 handout.*

You are a politician running for local office, and you want to appeal to a wide voter base. There are $k$ groups of voters, let $m_i$ be the number of voters in the $i$th group, and you want at least half of each group to vote for you. Without any campaigning, $a_i$ voters from group $i$ will vote for you ($0 \leq a_i \leq m_i$).

Your campaign staff have determined that there are $n$ issues that voters care about, and they will react differently depending on their group. In particular, for every \$1000 you spend on advertising for issue $j$, $d_{ij}$ is the number of additional voters in group $i$ who will now vote for you. (If $d_{ij}$ is negative, it means you lost voters in group $i$.)

Write a linear program in standard form to determine the minimum advertising cost so that at least half of each group votes for you, if possible at all. Briefly explain all choices made in your program.

**Solution:**

Let $x_j$ be the amount of money in thousands spent on advertising issue $j$. The linear program (not in standard form) is to:

$$\begin{aligned} \textbf{minimize} \quad & x_1 + \cdots + x_n \\ \textbf{subject to} \quad & d_{11}x_1 + \cdots + d_{1n}x_n + a_1 \geq \frac{m_1}{2} \\ & \vdots \\ & d_{k1}x_1 + \cdots + d_{kn}x_n + a_k \geq \frac{m_k}{2} \\ & x \geq 0 \end{aligned}$$

This expresses that we want to minimize spending, and ensure that every group has at least $n_i/2$ voters voting for you, after starting with $a_i$ support and gaining $d_{ij}x_j$ support for spending $x_j$ money on issue $j$.

Converting to standard form, we get

$$\begin{aligned} \textbf{maximize} \quad & -x_1 - \cdots - x_n \\ \textbf{subject to} \quad & -d_{11}x_1 - \cdots - d_{1n}x_n \leq a_1 - \frac{m_1}{2} \\ & \vdots \\ & -d_{k1}x_1 - \cdots - d_{kn}x_n \leq a_k - \frac{m_k}{2} \\ & x \geq 0 \end{aligned}$$

# 5. Reduction

Consider the following problems:

HAMILTONIANPATH
**Input**: A directed graph $G$
**Output**: Determine if there is a Hamiltonian path in $G$ (a path that visits each vertex exactly once).

HAMILTONIANCYCLE
**Input**: A directed graph $G$
**Output**: Determine if there is a Hamiltonian cycle in $G$ (a cycle that visits each vertex exactly once).

Suppose that HAMILTONIANPATH is NP-hard. Use that fact to show HAMILTONIANCYCLE is NP-hard.

**Solution:**

We will show that HAMILTONIANPATH $\leq_P$ HAMILTONIANCYCLE, that is, we will convert an instance of HAMILTONIANPATH into an equivalent instance of HAMILTONIANCYCLE (so that it could be solved with a library function for HAMILTONIANCYCLE).

**Algorithm**:
Let $G$ be an input for the HAMILTONIANPATH problem. Create the graph $H$, an instance for the HAMILTONIANCYCLE problem, as follows: Starting from $G$, add a completely new vertex $u$. For every vertex $v$ in $G$, add the edges $(v, u)$ and $(u, v)$.

**Correctness**:
We will convert certificates between the two instances (the actual Hamiltonian path or cycle).

Suppose that $G$ has a Hamiltonian path $v_1, v_2, \ldots, v_n$. Then in $H$, note that $v_1, v_2, \ldots, v_n, u, v_1$ is a cycle that visits every vertex in $H$ (since we copied $H$ and then added, among other edges, the edges $(v_n, u)$ and $(u, v_1)$). Thus $H$ has a Hamiltonian cycle.

Conversely, suppose that $H$ has a Hamiltonian cycle. The cycle must include $u$. Thus, for some labeling $v_1, v_2, \ldots, v_n$ of the remaining vertices, the cycle can be written as $u, v_1, v_2, \ldots, v_n, u$. Since all edges in $H$ but not in $G$ have $u$ as an endpoint, the edges $v_i, v_{i+1}$ are from $G$ for all $i$, and we have that $v_1, \ldots, v_n$ is a Hamiltonian path in $G$.

**Running time**:
Copying the graph, adding two vertices and $2n + 1$ edges can be done in polynomial time.

## 6. Bonus: Dynamic programming, again

*The inclusion of this problem is simply because the problem and solution are existing from previous iterations of this course, and should not be interpreted as indicating stronger emphasis on dynamic programming on the final exam.*

The problem is to determine, in a fictional country with two-party elections and an electoral college, the smallest number of votes needed to win the election. Let the number of states in this country be $n$.

Let $p_i$ be the total number of voters participating in the election from state $i$, and let $v_i$ be the number of electoral votes for state $i$. Each state holds a statewide election between the two candidates, and assume that there are no state-level ties, so candidates must win $\lfloor p_i/2 \rfloor + 1$ votes in state $i$ to win it. All electoral votes of a state go to the candidate winning that state. If a candidate receives at least $V$ electoral votes in total, where $V = \lfloor (\sum_i v_i)/2 \rfloor + 1$, they win the overall election.

Determine the minimum percent of the total popular vote that a candidate must obtain in order to receive at least $V$ electoral votes and win the overall election.

**Solution:**

Note that an optimal solution obtains 0 votes from states where the candidate loses, and $\lfloor p_i/2 \rfloor + 1$ votes from states $i$ where the candidate wins.

Let $\text{OPT}(i, v)$ denote the minimum number of popular votes from states $1, 2, \ldots, i$ in order to obtain at least $v$ electoral votes. This is defined for $i$ from 0 to $n$, and $v$ from the negative of the largest $v_i$ to $V$ (for easier base cases). Then,

$$\text{OPT}(i, v) = \min\left(\text{OPT}(i-1, v), \text{OPT}(i-1, v-v_i) + \left\lfloor \frac{p_i}{2} \right\rfloor + 1\right)$$

The first term corresponds to the candidate losing state $i$, in which case the candidate needs to obtain at least $v$ electoral votes from states $1, \ldots, i-1$, and the second term corresponds to the candidate winning state $i$, where the candidate wins $v_i$ electoral votes with the requisite popular votes, and needs states $1, \ldots, i-1$ to cover the rest.

The base cases are $\text{OPT}(i, v) = 0$ for all $i$ and all $v \leq 0$, and $\text{OPT}(0, v) = \infty$ for $v \geq 1$.

The algorithm outputs the desired percentage as $100\,\text{OPT}(n, V)/\sum_i p_i$.