1. Examples

This document is an example of how to use \LaTeX for writing homework solutions. Read the text, commented out by \% signs, to get some explanations.

a) This part includes a theorem with a proof and uses mathematical expressions.

Theorem 1.

\[ \sum_{i=1}^{n} i = \frac{n(n + 1)}{2} \]  

Proof. The proof is by induction.

Base case: Prove that the formula is true when \( n = 1 \). The LHS is \( \sum_{i=1}^{1} i = 1 \), while the RHS is \( \frac{1(1+1)}{2} = 1 \). Hence, the base case holds.

Induction step: For each \( k \geq 1 \), assume that \( (1) \) is true for \( n = k \). We show that it is true for \( n = k + 1 \).

\[
\sum_{i=1}^{k+1} i = \sum_{i=1}^{k} i + (k + 1) = \frac{k(k + 1)}{2} + (k + 1) = \frac{k(k + 1) + 2(k + 1)}{2} = \frac{(k + 1)(k + 2)}{2},
\]

where the second equality follows from induction hypothesis that \( \sum_{i=1}^{k} i = \frac{k(k+1)}{2} \). The formula \( (1) \) is true for \( n = k + 1 \), which proves the theorem.

b) This part has a figure that displays a picture from an external file.

![Figure 1: Comparing two sets of jobs](image)

c) This part has an example of writing algorithm psuedocode.

Assume that there are \( n \) jobs and the \( i^{th} \) job has a start time \( s(i) \) and a finish time \( f(i) \). These jobs are sorted with respect to their finish time. For simplicity, we assume that the sorted jobs are numbered 1,2,..., \( n \) such that \( f(1) \leq f(2) \leq \cdots \leq f(n) \).

A set of jobs is compatible with a job \( j \) if none of the jobs in the set overlaps with \( j \). The algorithm maintains \( A \), a set of selected jobs, which is initially empty. Our intuitive approach is to grow \( A \) by choosing a compatible job with the earliest finish time at each step.

Let \( i_1, \ldots, i_k \) be the set of jobs in \( A \) in the order they were added to \( A \). Similarly, let the set of jobs in \( B \), which selects jobs in some method other than greedy approach, be denoted by \( j_1, \ldots, j_\ell \). One interesting consequence is that the greedy rule stays ahead: \( f(i_m) \leq f(j_m) \) for \( 1 \leq m \leq \min(k, \ell) \).
Algorithm 1: Earliest-Finish-Time($L$).

**input**: a list $L$ of $n$ jobs.

**output**: a maximum set of mutually compatible jobs.

1. Sort jobs by finish times so that $f(1) \leq f(2) \leq \cdots \leq f(n)$.
2. Maintain a set $A$ which is initially empty.
3. for $i = 1$ to $n$ do
   4. If the job $i$ is compatible with $A$, then include $i$ to $A$.
   end
5. Output $A$.

Claim 2. For all indices $m \leq \min(k, \ell)$, $f(i_m) \leq f(j_m)$.

*Proof*. We prove by induction on the index $m$. For $m = 1$, the statement is true because the greedy approach selects the job with the earliest finish time. For $m > 1$, we will assume the statement is true for $m = t - 1$ and prove it for $m = t$. The $t$th job in $B$ must start after $f(j_{t-1})$ since this job is compatible with $B$. It means $f(j_{t-1}) \leq s(j_t)$. By combining the induction hypothesis $f(i_{t-1}) \leq f(j_{t-1})$, it also means $f(i_{t-1}) \leq s(j_t)$. So this job is compatible with $A$ too. As the greedy algorithm selects a job with earliest finish time, $f(i_t)$ is not larger than $f(j_t)$. This completes the induction step; therefore, the statement is true. □

Proof of Correctness Assume for contradiction that the greedy approach returns a non-optimal solution $A$ while an optimal set $O$ has more jobs. Assume that $|A| = k$ and $|O| = \ell$ with $\ell > k$. By Claim 2, we have $f(i_k) \leq f(j_k)$. Let us focus on the $(k+1)$th job $x$ in $O$. The job $x$ starts after the job $j_k$ ends and hence after the job $i_k$ ends. But the greedy algorithm stops with $i_k$ while $x$ is compatible with $A$ – a contradiction.

Implementation Once the input jobs are sorted, an array is enough for the set $A$. When a new job is checked for compatibility with $A$, it is enough to compare its start time with the last added job $x$’s finish time rather than all the jobs’ finish times in $A$ – the resource becomes free after $f(x)$ and the input jobs are sorted.

Time and Space Complexity It takes $\Theta(n \log n)$ time to sort the input jobs of size $n$. Creating an array of size $n$ takes $O(n)$ time. For each job, it takes $O(1)$ time to check whether a job is compatible with the set $A$, and the array can be updated in constant time if we maintain an end-of-the-array pointer. These operations must be repeated for each job, so the For loop takes $O(n)$ time. Hence, the total running time is $O(n \log n)$.

It takes $O(n)$ space to store the input. An in-place sorting takes $O(n)$ space. Finally, the set $A$ can be implemented by an array of size $n$. Thus, the space complexity is $O(n)$. 

2 of 2