**Due 30th June 2022 at 4pm Sydney time**

In this assignment we apply the greedy paradigm and related graph algorithms. There are *four problems* each worth 20 marks, for a total of 80 marks.

Your solutions must be typed, machine readable PDF files. *All submissions will be checked for plagiarism!*

For each question requiring you to design an algorithm, you *must* justify the correctness of your algorithm. If a time bound is specified in the question, you also *must* argue that your algorithm meets this time bound.

Partial credit will be awarded for progress towards a solution.

The clarifications may be updated further in the Assignment 2 FAQ thread on the Ed forum: https://edstem.org/au/courses/8646/discussion/925161.

*General clarifications:*

- **How do I justify the correctness of a greedy algorithm?**

  The nature of greedy algorithms is that we discard some alternatives before considering their later implications, so the standard of justification is higher than for other paradigms of algorithm design. Therefore, you often have to provide a proof. This does not always have to be a formal proof involving mathematical notation; see the solutions to problem set 3 for examples.

- **What methods of proof are allowed?**

  Any logically 'watertight' proof is acceptable. In lectures, we suggest two templates for these proofs, namely *exchange argument* and *greedy stays ahead.*

- **Is it necessary to prove the correctness of graph algorithms presented in this course?**

  No. If the implementation requires a non-standard data structure (e.g. augmented heap or union-find) to achieve the desired time complexity, you must make specific reference to that data structure.

  However, you do need to make clear (possibly with further reasoning) why your chosen graph algorithm solves the problem at hand.

- **Can data structures not presented in this course be used with only a citation?**

  No. In particular, this applies to the versions of Dijkstra's algorithm and Prim's algorithm which use the Fibonacci heap data structure to achieve worst case time complexity of $O(E + V \log V)$. We will ensure that any problems requiring either of these algorithms can be solved using the $O(E \log V)$ version presented in lectures.

## Question 1

You have $n$ identical flowers, which you would like to place into $k$ vases, according to the following rules:

- Each flower must be placed in a vase.
- The $i$th vase (where $1 \leq i \leq k$) must contain at least 1 and at most $f_i$ flowers.
- No two vases are allowed to contain the same number of flowers.

Your goal is to assign a number of flowers to each vase following the above rules, or to determine that no such assignment exists.

**1.1** [**10 marks**] Suppose that $f_1 = \ldots = f_k = n$. Design an algorithm which runs in $O(k)$ time and achieves the goal.

**1.2** [**10 marks**] Now we return to the original problem, in which the $f_i$ can take any positive integer values. Design an algorithm which runs in $O(k \log k)$ time and achieves the goal.

*Clarifications:*

- **How are the $f_i$ values given?**

  The $f_i$ are given in an array.

- **Are the $f_i$ all equal?**

  Not necessarily.

- **Is it guaranteed that a satisfactory assignment of flowers to vases exists?**

  No, as indicated by the phrase 'determine that no such assignment exists.

## Question 2

There are $n$ towns in a line. You are a traveller starting at town 1 and wanting to reach town $n$. It takes a week to travel from any town to the next.

In each town, there is a market where you can buy rations. In the market of town $i$, you can buy a week's worth of rations for $c_i$ dollars. Furthermore, you can buy several weeks' rations for later consumption.

Your goal is to calculate the minimum cost to travel from town 1 to town $n$.

**2.1** [**3 marks**] Which town's rations do you want to consume as you travel from town $n-1$ to town $n$? Provide reasoning to support your answer.

**2.2** [**3 marks**] When is the last time (if any) that you should change using from one town's rations to another? Provide reasoning to support your answer.

**2.3** [**14 marks**] Design an algorithm which runs in $O(n)$ time and achieves the goal.

*Clarifications:*

- **How are the $c_i$ values given? Are all $c_i$ known when you begin the journey.**

  The $c_i$ are given in an array, and are they are known to you from the start.

- **Can I buy several weeks' rations from a single market?**

  Yes.

- **Can I travel through the towns in any sequence?**

  No. You must travel from town 1 to town 2, then to town 3, and so on.

- **Do I start with any rations?**

  No.

## Question 3

You are given a simple directed graph with $n$ vertices and $m$ edges. Each vertex $v$ has an associated 'starting cost' $s_v$ and 'finishing cost' $f_v$, and each edge $e$ has a corresponding weight $w_e$. All values $w_e$, $s_v$ and $f_v$ are positive integers.

For this problem, we define the *score* of a path from vertex $a$ to vertex $b$ as the sum of its starting cost $s_a$, all edge weights on the path, and its finishing cost $f_b$. Your goal is to determine the smallest score of any path in the graph.

A path may consist of zero or more edges. The endpoints of a path do not need to be distinct.

**3.1** **[8 marks]** Design an algorithm which runs in $O((n+m)\log(n+m))$ time and achieves the goal.

An algorithm running in $\Theta((n+m)^2\log(n+m))$ time will be eligible for up to 6 marks.

How can you transform this to a *single source* shortest path problem?

**3.2** **[6 marks]** Now consider only paths consisting of *at least one edge*.

Design an algorithm which runs in $O((n+m)\log(n+m))$ time and achieves the goal.

**3.3** **[6 marks]** Now consider only paths consisting of *at least three edges*.

Design an algorithm which runs in $O((n+m)\log(n+m))$ time and achieves the goal.

*Clarifications:*

- **How is the graph represented?**

  The original graph is provided as a list of vertices and edges, but you may use any reasonable representation without specifying the details of the conversion. Please note however that you should use a graph representation that does not cause you to violate the time complexity requirement.

## Question 4

You are given an array $A$, which contains each integer from 1 to $n$ exactly once.

On each move, you can swap the value at index $i$ with the value at index $j$, for a cost of $|i - j|$ dollars. Your goal is to sort the array, spending as few dollars as possible.

Let $S = \sum_{i=1}^{n} \frac{|A[i] - i|}{2}$.

**4.1** **[2 marks]** Show that $S$ is an integer.

We are interested in the parity (even/odd) of $|A[1] - 1| + |A[2] - 2| + \ldots + |A[n] - n|$. What is the effect of the absolute values in this expression?

**4.2** **[2 marks]** Show that any sequence of swaps which sorts the array will cost at least $S$ dollars.

**4.3** **[8 marks]** Show that the array can be sorted for a total cost of $S$ dollars.

**4.4** **[8 marks]** Design an algorithm which runs in $O(n + S)$ time and finds a list of swaps (each described as a pair of indices) which sorts the array for the minimum possible total cost.

First aim for a time complexity of $O(n^2)$, then reason as to why the bound can be improved to $O(n + S)$.

*Clarifications:* Nil.