Worth: 10% **Due:** By 9:59pm on Tuesday 8 December

Remember to write the *full name* and *student number* of *every group member* prominently on your submission.

---

*Please read and understand the policy on Collaboration given on the Course Information Sheet. Then, to protect yourself, list on the front of your submission **every** source of information you used to complete this homework (other than your own lecture and tutorial notes). For example, indicate clearly the **name** of every student from another group with whom you had discussions, the **title and sections** of every textbook you consulted (including the course textbook), the **source** of every web document you used (including documents from the course webpage), etc.*

*For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks **will** be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions.*

---

1. (a) [Worth 20%]

   Suppose we want to compute the shortest path from node s to node t in a *directed* graph $G = (V,E)$ with edge lengths $\ell_e > 0$ for each $e \in E$.

   Show that this is equivalent to finding a *pseudo-flow* f from s to t in G such that $|f| = 1$ and $\sum_{e\in E} \ell_e f(e)$ is minimized. There are no capacity constraints.

   Part of this problem requires you to **define** precisely what we mean by "pseudo-flow" in a general, directed graph. This is a natural extension of the notion of flow in a network.

   (b) [Worth 10%]

   Write the shortest path problem as a linear or integer program **<u>where your objective function is *minimized*, based on your answer to the previous part</u>**. Give a detailed justification that your solution is correct.

2. (a) [Worth 25%]

   Consider the following DISJOINTHAMILTONIANPATHS decision problem ("DHP" for short).

   - *Input:* Graph $G = (V,E)$—G may be directed or undirected.
   - *Output:* Does G contain at least two edge-disjoint Hamiltonian paths?

   (Two paths are said to be *edge-disjoint* if there is no edge that belongs to both paths.)

   Write a *detailed* proof that DISJOINTHAMILTONIANPATHS is NP-complete. State what you are doing at each step of your solution: it will be graded on its structure as much as on its content.

   (b) [Worth 15%]

   Give a precise definition for the Disjoint Hamiltonian Paths *Search Problem*. Then, write a *detailed* argument that this problem is polynomial-time self-reducible. Once again, your solution will be graded on its structure as well as its content, so make sure to state what you are doing at each step.

   <u>You do NOT have to "separate" the two paths—your algorithm should simply output the collection of edges that makes up the two disjoint paths. You MAY get some bonus marks if you correctly separate the paths but note that this is difficult to do correctly!</u>

3. You are using a multi-processor system to process a set of jobs coming in to the system each day. For each job $i = 1, 2, \dots, n$, you know v_i, the time it takes one processor to complete the job.

Each job can be assigned to one and only one of m processors—no parallel processing here! The processors are labelled $\{A_1, A_2, \ldots, A_m\}$. Your job as a computer scientist is to assign jobs to processors so that each processor processes a set of jobs with a reasonably large total running time. Thus given an assignment of each job to one processor, we can define the *spread* of this assignment to be the minimum over $j = 1, 2, \ldots, m$ of the total running time of the jobs on processor A_j.

Example: Suppose there are 6 jobs with running times $3, 4, 12, 2, 4, 6$, and there are $m = 3$ processors. Then, in this instance, one could achieve a spread of 9 by assigning jobs $\{1, 2, 4\}$ to the first processor (A_1), job 3 to the second processor (A_2), and jobs 5 and 6 to the third processor (A_3).

The ultimate goal is find an assignment of jobs to processors that **maximizes** the spread. Unfortunately, this optimization problem is NP-Hard (you do not need to prove this). So instead, we will try to approximate it.

(a) [Worth 15%]

Give a polynomial-time algorithm that approximates the maximum spread to within a factor of 2. That is, if the maximum spread is s, then your algorithm should produce a selection of processors for each job that has spread at least $s/2$. In your algorithm you may assume that no single job has a running time that is significantly above the average; specifically, if $V = \sum_{i=1}^{n} v_i$ denotes the total running time of all jobs, then you may assume that no single job has a running time exceeding $\frac{V}{2m}$.

Show that your algorithm achieves the required approximation ratio.

(b) [Worth 15%]

Give an example of an instance on which the algorithm you designed in part (3a) does not find an optimal solution (that is, one of maximum spread). Say what the optimal solution is in your instance, and what your algorithm finds.