Homework 3

Simon Zheng
260744353

November 3$^{\text{rd}}$, 2017

1

First, include all leaves as no 2 leaves can be adjacent in a graph with more than 2 vertices (cases with a graph containing less than 3 vertices are trivial). Then, with the remaining vertices, 2-colour them and keep the biggest set.

Between two adjacent vertices where one is a leaf (single neighbour) and one is an internal vertex (more than 1 neighbour), choosing the leaf allows us to also choose any of the internal vertex's neighbours, while choosing the internal vertex removes the possibility of choosing the leaf or any of it's neighbours. Thus, along with the fact that no two leaves can be adjacent for $n > 2$, choosing the leaf is more advantageous and will allow for a bigger set.

2

For any order of processing chosen, pick 2 successive jobs. The finishing time of each is the processing time of the previous job plus its own. Thus if the processing time of the first one is t_1, both finishing times must be at least as long as t_1. So if $t_1 > t_2$, then both finishing times must be as long as the longest processing time. But if we reverse the job order (so where $t_1 > t_2$ still applies), then both jobs' finishing times must only be as long as the shorter one (t_2) and t_2's finishing time will be shorter than the first case. Therefore, it is always more optimal to put the shorter job first. Now, applies this to every successive pair of jobs from (t_1, t_2), (t_2, t_3), ... , (t_{n-1}, t_n) and we will get the optimal solution, which is exactly putting them in increasing order according to their processing time.

3

?

4

?

5

Graph, edge weight between two points (vertices) is distance, put edges in graph matrix and sort.

6

Hybrid reverse Prim's/Dijkstra's algorithm except don't remove vertices in F.

Start with putting all of the edges in a set. Remove one by one the most costly edge every time but without removing vertices of F.