COMS20010 — Problem sheet 2

This problem sheet covers week 2, focusing on greedy algorithms and graphs.

1 Greedy Algorithms

1. Here we will consider the set cover problem. Given a set of elements $U = \{1, 2, \dots, n\}$ (called the *universe*), and a collection of subsets of U, $S = \{A, B, C, \dots\} \in \mathcal{P}(U)$, we want to find a subset of elements of S whose union contains the entirety of U. For example, suppose $U = \{1, 2, 3\}$ and $S = \{\{1, 2\}, \{1\}, \{3\}\}$. Then $\{\{1, 2\}, \{3\}\}$ is the smallest solution to the set cover problem. Note that $\{\{1, 2\}, \{3\}, \{1, 2, 3\}\}$ is also a solution, but is of larger size.

   (a) [⋆⋆] Consider the instance of the set cover problem given by

   $$U = \{1, 2, 3, 4, 5\}, \qquad S = \big\{\{1, 2, 3\}, \{3, 4, 5\}, \{5\}, \{2, 3, 4\}, \{1, 2\}\big\}.$$

   What is the smallest solution to this instance? What is the largest solution?

   (b) [⋆⋆] Consider this greedy algorithm to solve the set cover problem.

   Algorithm: GreedySetCover($U, S \in \mathcal{P}(U)$)

   begin
   - Initialise `Sol` $\leftarrow \emptyset$.
   - **while** *There is an element in U which is not contained in any element of* `Sol` **do**
     - Let I be the element of S containing the most elements of U which are not contained in `Sol`. If there is a tie, choose I arbitrarily.
     - Set `Sol` $\leftarrow$ `Sol` $\cup \{I\}$.
   - Return `Sol`.

   Run this algorithm using U and S from part (a), showing the value of `Sol` at each iteration.

   (c) [⋆⋆⋆] Prove that if a solution exists, then this algorithm will always output a solution (i.e. that this algorithm is correct).

   (d) [⋆⋆] Give an example instance of set cover (i.e. values for U and S) for which no solution exists. How does GreedySetCover go wrong on such an instance? How can it be modified to efficiently deal with this?

   (e) [⋆⋆⋆] Suppose we are instead interested in the size of a *minimum* set cover, i.e. that we wish to ensure `Sol` is as small as possible. Show that GreedySetCover does not always give a minimum set cover, by giving an instance (U, S) on which it gives an answer larger than the minimum. Extend your answer to show that there are arbitrarily large instances on which GreedySetCover fails.

   (f) [⋆⋆⋆⋆] Now show that the set cover returned by GreedySetCover can be $\omega(1)$ times larger than the minimum set cover, i.e. larger by a factor that grows arbitrarily large as the input size increases. (**Hint:** One way of doing this has a minimum set cover with 2 sets, and chooses U with $|U| = 2 \cdot 3^t$ for any $t \geq 1$.)

2. [⋆⋆] You are trying to check whether a log file contains a specific sequence of events, some of which may be duplicates. Since the log file contains records for the whole system, the events may not occur consecutively, but you know they will occur in order. Formally, you are given a *key sequence* of events $a_1, \dots, a_m$ and a *log sequence* of events $b_1, \dots, b_n$ with $n \geq m$, and you are able to check whether two events are equal in $O(1)$ time. Give a greedy algorithm to check whether $b_1, \dots, b_n$ contains a *key subsequence* — indices $i_1 < i_2 < \cdots < i_m \in [n]$ such that $b_{i_j} = a_j$ for all $j \in [m]$ — and to output a key subsequence if one exists. Your algorithm should run in $O(n)$ time; prove it works.

3. [⋆⋆⋆] In lectures we showed that substituting the greedy heuristic in our interval scheduling algorithm GreedySchedule with "add the compatible request with the earliest starting time" or "add the compatible request which takes least total time" breaks the algorithm. Show that the greedy heuristic "add the compatible request which renders fewest other requests incompatible" would also fail.

4. [⋆⋆⋆⋆] You are consulting for a trucking company that does a large amount of business shipping packages from New York to Boston. The volume is high enough that they have to send a number of trucks each day between the two locations. Trucks have a fixed limit W on the maximum weight they are allowed to carry. Boxes arrive at the New York station one by one, and each package i has a weight $w_i \leq W$. The trucking station is quite small, so at most one truck can be at the station at any time. Company policy requires that boxes are shipped in the order they arrive; otherwise, a customer might get upset upon seeing a box that arrived after his make it to Boston faster. At the moment, the company is using a simple greedy algorithm for packing: they pack boxes in the order they arrive, and whenever the next box does not fit, they send the truck on its way.

   But they wonder if they might be using too many trucks, and they want your opinion on whether the situation can be improved. Perhaps one could decrease the number of trucks needed by sometimes sending off a truck early, allowing the next few trucks to be better packed?

   Prove that this is not the case — that for any given number k of packages, the greedy algorithm they are currently using minimises the number of trucks they need subject to their other constraints. (**Hint:** Use an argument like the one we used for GreedySchedule...)

5. Consider a variant of the interval scheduling problem where we have multiple "satellites" available, and wish to satisfy **all** our requests while using as few of them as possible. Formally: writing our input as $\mathcal{R} = [(s_1, f_1), \ldots, (s_n, f_n)]$, instead of finding a maximum compatible set of requests, we must partition $\mathcal{R}$ into as few disjoint compatible sets as possible.

   (a) [⋆⋆⋆⋆⋆] Prove that the following greedy algorithm returns the correct answer. (**Hint:** Rather than proving optimality directly, try to find a nice lower bound on the size of a minimum partition, and show the algorithm produces something which matches it.)

   Algorithm: GreedyPartition

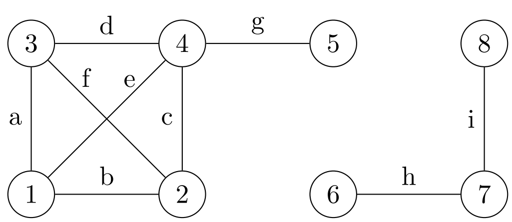
   1 **begin**
   2 Sort $\mathcal{R}$ according to start time, so that $s_1 \leq \cdots \leq s_n$.
   3 Initialise $A_1, \ldots, A_n = [\,]$.
   4 **for** $i \in \{1, \ldots, n\}$ **do**
   5 Find the least j such that (s_i, f_i) is compatible with A_j.
   6 Append (s_i, f_i) to A_j.
   7 Return the collection of non-empty lists A_j.

   (b) Is the sorting step in line 2 necessary?

2 Graph Theory

Hint: It can be extremely useful to draw a graph with the property you are trying to consider.

6. In this question we will consider the following graph G.



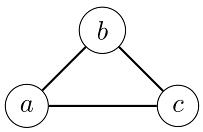
(a) $[\star\star]$ How many components does G have?
(b) $[\star\star]$ Is $\{6, 7, 8\}$ a component of G?
(c) $[\star\star]$ What is the degree of vertex 3?
(d) $[\star\star]$ How many paths are there from vertex 1 to vertex 4? List them.
(e) $[\star\star]$ How many walks are there from vertex 1 to vertex 4?
(f) $[\star\star]$ Draw the graph $G[\{1, 2, 4\}]$ induced by the set of vertices $\{1, 2, 4\}$.
(g) $[\star\star]$ Is $G[\{1, 2, 4\}]$ isomorphic to $G[\{6, 7, 8\}]$?
(h) $[\star\star]$ Is $G[\{3, 4, 5\}]$ isomorphic to $G[\{6, 7, 8\}]$?
(i) $[\star\star]$ Does $G[\{1, 2, 3, 4\}]$ contain an Euler walk?

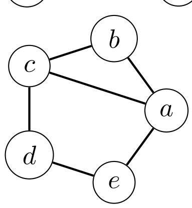
7. (a) $[\star\star]$ Let G be a connected graph containing vertices u and v. Show that any walk from u to v contains a path from u to v. (**Hint:** Try induction based on the length of the walk.)

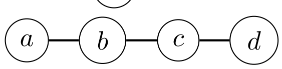
   (b) $[\star\star\star]$ Let G be a connected graph. Show that any two longest paths in a connected graph must have at least one vertex in common. (Here "two longest paths" means that both paths have the same length, and no other path in the graph is longer.)

8. Let G be a graph. To construct the *line graph of* G, $L(G)$, we define the vertices to be the edges of G, and say that two vertices of $L(G)$ (i.e. two edges of G) are connected by the edge in $L(G)$ if they have a non-empty intersection. For example, if G has edges $\{1, 2\}$ and $\{3, 4\}$, then the vertex set of $L(G)$ will be $\{\{1, 2\}, \{3, 4\}\}$ and the edge set of $L(G)$ will be empty.

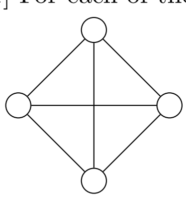
   (a) $[\star\star]$ For each of these graphs G, draw its line graph $L(G)$.

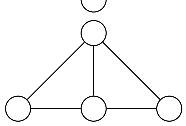
   i. 

   ii. 

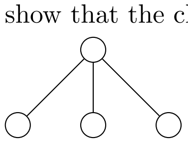
   iii. 

   (b) $[\star\star\star]$ For each of the following graphs H, give a graph G such that $L(G)$ is isomorphic to H.

   i. 

   ii. 

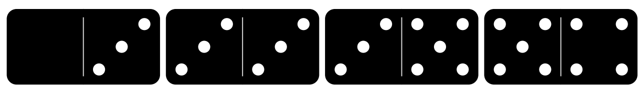
   (c) $[\star\star\star]$ The claw graph is shown below. Show that if a graph H is a line graph, i.e. if $H = L(G)$ for some graph G, then H doesn't contain any induced subgraph isomorphic to the claw. (**Hint:** First show that the claw itself is not isomorphic to any line graph.)



9. [$\star\star\star$] A *closed* Euler walk is an Euler walk from a vertex to itself. Suppose G is a graph with exactly two connected components C_1 and C_2, each of which has more than one vertex. Suppose the graphs induced by C_1 and C_2 each have a closed Euler walk. What is the least number of edges we can add to G to give it a **closed** Euler walk?

10. The *complement* of a graph $G = (V, E)$ is the graph $G^c = (V, \overline{E})$, where $\overline{E} = \{\{u, v\} \colon u, v \in V, u \neq v\} \setminus E$. A graph is *self-complementary* if it is isomorphic to its complement. Show that:
    (a) [$\star$] a four-vertex path and a five-vertex cycle are both self-complementary;
    (b) [$\star\star\star$] every self-complementary graph is connected;
    (c) [$\star\star\star\star$] if G is self-complementary, then $|V(G)| \equiv 0$ or $1 \bmod 4$;
    (d) [$\star\star\star\star$] every self-complementary graph on $4k + 1$ vertices has a vertex of degree $2k$.

11. [$\star\star\star$ and a half] A *numbered domino* is a rectangle divided into two halves, with a number on each half. A standard "double six" set of numbered dominoes contains one domino with each possible pair of numbers from zero to six, for a total of 28. Is it possible to lay them all out in a line so that each adjacent pair of dominoes agrees, as shown below for four dominoes? What about a "double k" set, which contains one domino with each possible pair of numbers from zero to $k \in \mathbb{N}$? (**Hint:** I will never ask a question like this unless there's a way to solve it quickly with pencil and paper.)



12. [$\star\star\star\star\star$] Give an example of a self-complementary graph (see Question 3) with infinitely many vertices.