COMS20010 — Problem sheet 5

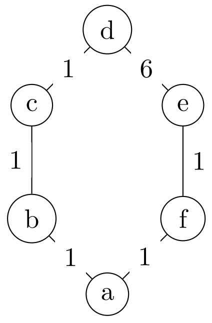
You don't have to finish the problem sheets before the class — focus on understanding the material the problem sheet is based on. If working on the problem sheet is the best way of doing that, for you, then that's what you should do, but don't be afraid to spend the time going back over the quiz and videos instead. (Then come back to the sheet when you need to revise for the exam!) I'll make solutions available shortly after the problem class. As with the Blackboard quizzes, question difficulty is denoted as follows:

- ⋆ You'll need to understand facts from the lecture notes.
- ⋆⋆ You'll need to understand and apply facts and methods from the lecture notes in unfamiliar situations.
- ⋆⋆⋆ You'll need to understand and apply facts and methods from the lecture notes and also move a bit beyond them, e.g. by seeing how to modify an algorithm.
- ⋆⋆⋆⋆ You'll need to understand and apply facts and methods from the lecture notes in a way that requires significant creativity. You should expect to spend at least 5–10 minutes thinking about the question before you see how to answer it, and maybe far longer. At most 10% of marks in the exam will be from questions set at this level.
- ⋆⋆⋆⋆⋆ These questions are harder than anything that will appear on the exam, and are intended for the strongest students to test themselves. It's worth trying them if you're interested in doing an algorithms-based project next year — whether you manage them or not, if you enjoy thinking about them then it would be a good fit.

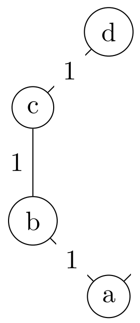
If you think there is an error in the sheet, please post to the unit Team — if you're right then it's much better for everyone to know about the problem, and if you're wrong then there are probably other people confused about the same thing.

This problem sheet covers week 5, focusing on matchings and spanning trees.

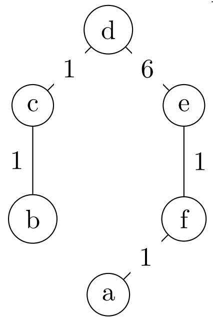
1. (a) [⋆⋆] For each of these graphs give a minimum spanning tree as well as a spanning tree which is not minimum.

i. 

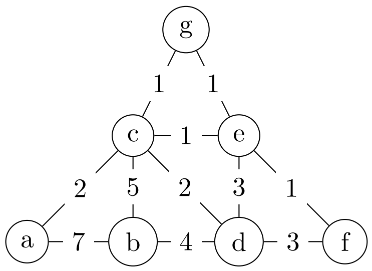
Solution: A minimum spanning tree is



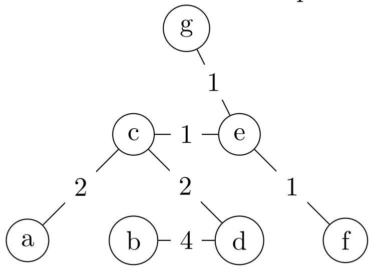
A non-minimum spanning tree is



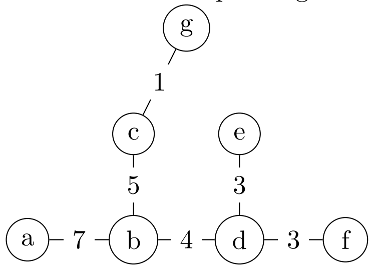
ii.



Solution: A minimum spanning tree is



A non minimum spanning tree is



Note that these are not unique.

(b) [⋆⋆] How is Kruskal's algorithm different to Prim's algorithm?

Solution: Kruskal's algorithm finds a minimum spanning forest by always looking for the next smallest weight edge which can be added without forming a cycle. Prim's algorithms differs by greedily building a **connected** spanning tree. It does this by requiring a starting vertex, and then only considering edges which are adjacent to the existing (growing) spanning tree.

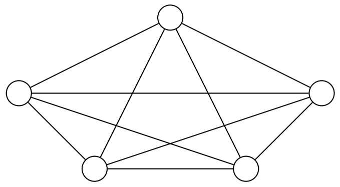
(c) [⋆⋆] Which earlier ideas in the course are these algorithms based on?

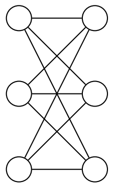
Solution: Both of these algorithms are greedy algorithms. They are also both conceptually similar to the ideas in breadth first search (although we are solving a different problem here).

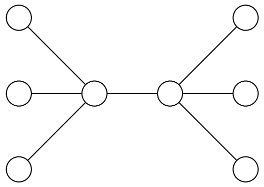
2. [⋆⋆] Let $G = ((V, E), w)$ be a weighted graph, and let T be a minimum spanning tree for G. Give an example to show that the path between two vertices in T may not be a shortest path in G.

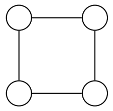
Solution: One example would be as follows. We take G to be a triangle, with $V = [3]$ and $E = \{\{1, 2\}, \{2, 3\}, \{1, 3\}\}$. Then we have $d(x, y) = 1$ for all $x, y \in V$, but any spanning tree must have two non-adjacent vertices.

3. [⋆⋆] Which of these graphs are bipartite?

(a) 

(b) 

(c) 

(d) 

Solution: Graphs B, C, and D are bipartite. Graph A is not.

4. [⋆⋆⋆] Give an algorithm to find a bipartition for any graph with no odd-length cycle in polynomial time. **Hint:** Try using one of the algorithms covered in week 4.

Together with the fact that an odd-length cycle is not bipartite, proved in lectures, this implies a graph is bipartite if and only if it has no odd-length cycle.

Solution: Let $G = (V, E)$ be the input graph, and initialise $A, B \leftarrow \emptyset$. First find a BFS tree rooted at an arbitrary vertex $v \in V$, with layers $L_0, L_1, \dots, L_t$ (where $L_0 = \{v\}$). Update $A \leftarrow A \cup L_0 \cup L_2 \cup \dots$ and $B \leftarrow B \cup L_1 \cup L_3 \dots$, effectively colouring odd layers red and even layers blue. If the BFS tree is not a spanning tree, i.e. G is not connected, then repeat the process until all of G is covered.

This algorithm runs in $O(|E| + |V|)$ time when G is given in adjacency list form. Suppose that the algorithm fails, so there is an edge $\{x, y\}$ internal to A (say) — then we will find an odd cycle in G. Then x and y must lie in the same component of G, so they must be in $V(T)$ for some BFS tree T. Moreover, since T is a BFS tree, it has no edges between non-adjacent layers. Since T's layers alternate between A and B, this implies x and y must be in the same layer, say L_i.

Now, let P_x be the unique path in T from v to x, and let P_y be the unique path in T from y to v. Then $P_x xy P_y$ is a cycle of length $i + 1 + i = 2i + 1$, which is odd.

5. [$\star\star\star$] Show that a bipartite graph G with bipartition (A, B) has at most $|A||B|$ edges. Using this, show that any n-vertex graph with more than $n^2/4$ edges is not bipartite, and hence contains an odd cycle. (You may assume n is even.)

Solution: Since A and B contain no internal edges, every edge must contain one vertex from A and one vertex from B, and there are $|A||B|$ ways of choosing such a pair of vertices.

Now, suppose that G is bipartite with bipartition (A, B). We have $|A| + |B| = n$ and $|A|, |B| \geq 0$, so

$$n^2 = (|A| + |B|)^2 \geq (|A| + |B|)^2 - (|A| - |B|)^2 = 4|A||B|.$$

Rearranging, we obtain $|A||B| \leq n^2/4$, so G has at most $n^2/4$ edges as required.

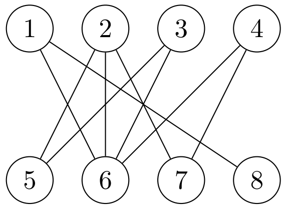
The heart of this proof is the fact that $|A||B| \leq n^2/4$ — the argument above is slick, but quite hard to come up with on the fly. An alternative way would be to write $|A||B| = |A|(n - |A|)$, differentiate with respect to A to obtain

$$\frac{d}{d|A|}|A||B| = n - |A| - |A| = n - 2|A|,$$

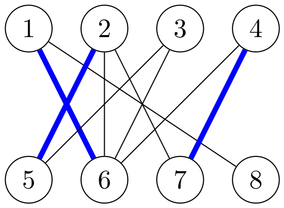
and conclude that $|A||B|$ is maximised when $|A| = n/2$ and hence when $|A||B| = (n/2)(n - n/2) = n^2/4$.

In fact, any graph with more than $n^2/4$ edges contains not just an odd cycle but a triangle — this is Mantel's theorem (1907), one of the foundational results in extremal graph theory.

6. (a) [$\star\star$] For the following graph, run the naive greedy algorithm from the start of video 5-2 to compute a matching which you can't add any more edges to (which might or might not be maximum). Start with the vertices of lowest index, and break ties by choosing edges which connect to vertices of lower index.

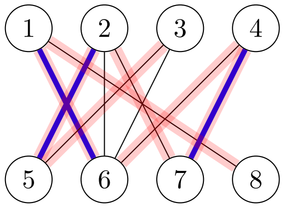


Solution: The matching found by the greedy algorithm is:



(b) [⋆⋆] Find an augmenting path for this matching.

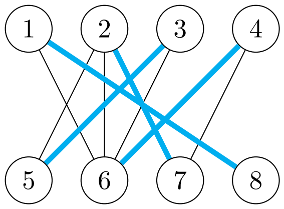
Solution: An augmenting path is



i.e. $\{3,5\}, \{5,2\}, \{2,7\}, \{7,4\}, \{4,6\}, \{6,1\}, \{1,8\}$.

(c) [⋆⋆] Use the augmenting path you have found to turn the matching into a maximum matching.

Solution: We can swap our previous matching with a new one constructed by the augmenting path to get:



i.e. $\{\{1,8\}, \{2,7\}, \{3,5\}, \{4,6\}\}$.

7. [⋆⋆⋆] In lectures, we discussed a purely greedy algorithm for finding a maximum matching — forming a matching by iterating over all edges and adding as many as possible — and noted that it doesn't always work. Show that if a maximum matching has size k, then this greedy algorithm returns a matching of size at least $k/2$. Why might this be useful?

Solution: Let M be a maximum matching, and let M' be the output of our greedy algorithm. For each edge e of M, when our algorithm looked at e, either we didn't add it to M' (in which case it must intersect some edge of M') or we did (in which case it must *also* intersect some edge of M', namely itself). We conclude that every edge of M intersects some edge of M'. But since M and M' are matchings, each edge of M' can intersect at most two edges in M, so $|M| \leq 2|M'|$. Rearranging, we obtain $|M'| \geq |M|/2$. (This is an example of a double-counting argument, applied to the set $\{(e,f)\colon e \in M, f \in M', e \cap f \neq \emptyset\}$ which has size at least $|M|$ and at most $2|M'|$.)

This could be useful if our input graph is very large and we are willing to accept an approximate answer, since the greedy algorithm runs in time $O(|E|)$ compared to $O(\sqrt{|V|}|E|)$ and $O(|V|^{2.37})$ for the best-known exact algorithms.

8. [$\star\star\star\star\star$] Suppose you are trying to design a clustering algorithm. You are given a set P of n photos of k objects, with $n > k$, and an algorithm which is capable of evaluating a "distance" $d(x, y)$ between any two photos x and y. You may assume that $d(x, y) > 0$ whenever x and y are distinct, and that $d(x, y) = d(y, x)$ for all $x, y \in P$; however, you may not assume that d satisfies the triangle inequality. You expect that if $d(x, y)$ is low, then x and y are likely to be photos of the same object, and vice versa. Your objective is to use this information to categorise the photos according to which object they depict — that is, to partition P into non-empty subsets $P_1, \dots, P_k$ such that if x and y lie in two different sets P_i, then $d(x, y)$ is large. Specifically, you are trying to maximise the *spacing*:

$$\min\{d(x, y)\colon x \text{ and } y \text{ are not in the same } P_i\}.$$

Find an algorithm which finds $P_1, \dots, P_k$ in $O(|P|^2 \log |P|)$ time, given P and d, and prove it works. **Hint:** Try to find a way of applying Kruskal's algorithm.

Solution: Let $G = ((P, E), d)$ be a complete weighted graph on P, so that the weight of $\{u, v\}$ is equal to $d(u, v)$ for all distinct $u, v \in P$. Let T be a minimum spanning tree, found using Kruskal's algorithm, and form T^- by removing the $k - 1$ most expensive edges from T. By the fundamental lemma of trees, deleting an edge from a tree separates it into two components, so this results in a k-component forest. We then take $P_1, \dots, P_k$ to be the vertex sets of the components of T^-. Since $|E| = \Theta(|P|^2)$, this takes $O(|P|^2 \log |P|)$ time as required.

To see that the algorithm works, let D be the spacing of $P_1, \dots, P_k$, so that $d(x, y) \geq D$ for all objects x, y in distinct clusters. This means that the edges removed in forming T^- all have weight at least D, and at least one of them has weight exactly D; since Kruskal's algorithm chooses edges in increasing order of weight, this implies that all the edges remaining in T^- have weight at most D.

We use this fact to prove that any clustering $Q_1, \dots, Q_k$ with *maximum* possible spacing must have spacing at most D, so that $P_1, \dots, P_k$ is optimal. Suppose $(Q_1, \dots, Q_k)$ is not equal to $(P_1, \dots, P_k)$ (perhaps in a different order); then there is at least one set P_i which is split across multiple sets Q_j. Without loss of generality, say there is a vertex $v \in P_1 \cap Q_1$ and a vertex $w \in P_1 \cap Q_j$ for some $j \neq 1$. Then there must exist a vertex $w \in P_1 \setminus Q_1$ for some $j \neq 1$. Moreover, v and w are joined by a path in T^-, so one of the edges in this path must leave Q_1. Since all edges in T^- have weight at most D, it follows that there is an edge $\{a, b\}$ out of Q_1 with $d(a, b) \leq D$; thus the spacing of $Q_1, \dots, Q_k$ is at most D, as required.

This problem is known as *single-linkage clustering*.

9. [$\star\star\star$ and a half] You are teaching a class of schoolchildren $C_1, \dots, C_n$ in the pre-COVID era. The end of term is approaching, and you have bought them a large tub of chocolates — you plan to give them four chocolates each, and eat the rest yourself. However, there are many types $T_1, \dots, T_k$ of chocolate in the tub, and not every child likes every type of chocolate — a given child C_i is willing to accept only chocolates from a set $S_i \subseteq \{T_1, \dots, T_k\}$. The tub is also not bottomless — for each $i \in [k]$, there are only $t_i > 0$ chocolates of type T_i. Give an algorithm which, given $T_1, \dots, T_k$, $t_1, \dots, t_k$ and $S_1, \dots, S_n$, assigns four chocolates to every child in polynomial time if possible and returns `Impossible` otherwise.

Solution: We construct a bipartite graph as follows. We assign each child C_i four vertices $c_{i,1}, \dots, c_{i,4}$, each one representing a choice of chocolate. We assign each type T_i of chocolates t_i vertices $x_{i,1}, \dots, x_{i,t_i}$, each one representing an available chocolate. We join two vertices $c_{i,j}$ and $x_{\ell,m}$ if and only if $T_\ell \in S_i$ — that is, if and only if child C_i is willing to eat chocolates of type T_ℓ. A matching of size $4n$ in this graph corresponds to giving four chocolates to every child:

- Each edge $\{c_{i,j}, x_{\ell,m}\}$ in the matching corresponds to giving a chocolate of type T_ℓ to child C_i;

- The requirement that matching edges are disjoint in vertices $c_{i,j}$ corresponds to the requirement that each child receives at most four chocolates;
- The requirement that there are $4n$ edges in the matching corresponds to the requirement that each child receives at least four chocolates (since this can only occur if every vertex $c_{i,j}$ is matched);
- The requirement that matching edges are disjoint in vertices $x_{i,j}$ corresponds to the requirement that no more than t_i chocolates of each type are given out.

We can therefore simply find the maximum matching in the graph with the algorithm from lectures, and return either the corresponding assignment of chocolates (if it has size $4a$) or `Impossible` (otherwise).

10. [$\star\star\star\star$] Suppose you're trying to schedule meetings across a company, with multiple rooms (from a set R) available. For simplicity, suppose that all meetings are an hour long and occur on the hour, and write T for the set of available times. A set P of people are trying to book rooms for meetings, and each person $p \in P$ has a set T_p of available times and a set R_p of rooms that will meet their needs. (For example, some people may need a projector, some people may need large seating capacity, some people may need a nicely-furnished room to impress a visitor, and so on.) Describe a polynomial-time algorithm to assign as many rooms and times as possible while satisfying everyone's preferences and avoiding double-booking. In other words, it should return a set $\mathcal{S}$ of k triples $(p, r, t) \in P \times R \times T$ such that:

    (i) each person p appears in at most one triple in $\mathcal{S}$;

    (ii) each pair $(r, t) \in R \times T$ appears in at most one triple in $\mathcal{S}$;

    (iii) for all $(p, r, t) \in \mathcal{S}$, we have $r \in R_p$ and $t \in T_p$; and

    (iv) k is as large as possible subject to the remaining constraints.

    Hint: Try to reduce this problem to finding a maximum matching in an auxiliary graph.

Solution: The trick is to look at **pairs** (r, t) as **single** vertices in your graph. So form an auxiliary graph $G = (V, E)$, where

$$\begin{aligned} V &= P \cup (R \times T), \\ E &= \{\{p, (r, t)\} : r \in R_p,\ t \in T_p\}. \end{aligned}$$

Then G is bipartite, with bipartition $(P, R\times T)$, and a matching M in G corresponds exactly to a set $\mathcal{S}$ satisfying (i)–(iii). Indeed, (i) corresponds to the requirement that edges in M don't intersect in P, (ii) corresponds to the requirement that edges in M don't intersect in $R\times T$, and (iii) corresponds to the requirement that they *are* edges. Thus a maximum matching will also satisfy (iv), so we can find a valid set $\mathcal{S}$ by running MaxMatching on G. We have $|V| = |P| + |R||T|$ and $|E| \leq |P||R||T|$, so the overall running time is $O(|E||V|) = O(|P|^2|R|^2|T|^2)$ which is polynomial in the size of the input. (Although it's quite a large polynomial — this might be an instance where we'd be better off using an approximation algorithm with a faster running time.)

11. Suppose you are given coordinates for n cities, and you wish to connect their electrical networks as cheaply as possible. The local terrain may not be flat, so cost may not scale linearly with distance. However, it is still true that all costs are non-negative, and that it costs at least as much to build a cable from u to v and then a cable from v to w as it does to build a cable from u to w directly.

(a) [$\star$] Explain how to use Prim's algorithm to solve the problem, under the assumption that power cables can only connect to each other inside cities.

> **Solution:** Let V be the set of all cities, let $E = \{\{u, v\} : u, v \in V, u \neq v\}$ be the set of all pairs of cities, and let $w(u, v)$ be the cost of building a cable from u to v. Then an optimal junctioned network is precisely a minimum spanning tree of the weighted complete graph $G = ((V, E), w)$, so we can apply Prim's algorithm to this graph. This is the example application of minimum spanning trees discussed in lectures.

(b) [$\star\star\star\star\star$] Prove that the additional assumption in part (a) increases the cost by at most a factor of 2. **Hint**: Consider an optimum network without the assumption, and try to replace it with one that only branches in cities. You may use the fact that depth-first search on a tree generates a walk that crosses every edge twice.

> **Solution:** Consider an optimal network N which allows power cables to branch in arbitrary locations. Let V be the set of all cities, and let B be the set of all branch locations. Let $G = ((V \cup B, E), w)$ be the weighted complete graph on $V \cup B$ with Euclidean weights, defined as in part (i). Then N corresponds to a spanning tree T of G.
>
> The idea is as follows: Since T is a tree, we know there's a unique path $P_{u,v}$ in T between any pair of cities u and v. Since we have assumed that edge weights satisfy the triangle inequality, i.e. $w(x, y) + w(y, z) \geq w(x, z)$ for all $x, y, z \in V$, we also know that the total weight of $P_{u,v}$ is at least as big as $w(u, v)$. So we would like to remove $P_{u,v}$ and replace it with the edge $\{u, v\}$, which will remove dependence on B but which won't increase the weight of T. The tricky part is that we might disconnect T in the process, so we need to somehow do it for all pairs of cities at the same time.
>
> To solve this, consider a walk W on T as generated by depth-first search, greedily exploring along the power network, traversing each cable twice and visiting each city before returning to its start point. We can split W into subwalks according to when it enters a city, writing $W = W_1 \dots W_n$ where W_i is from city v_i to city v_{i+1} and visits only vertices in B in between. (Here $v_1 = v_n$ is the starting point of the walk.) Consider the graph T' on V with edges $\{\{v_i, v_{i+1}\} : i \in [n-2]\}$. This graph is certainly connected, since W visits each city at least once, but it may contain cycles; we therefore take an arbitrary spanning tree T'' of T'.
>
> We now bound the weight of T'' in terms of the weight of T. Since W traverses each cable exactly twice, it has total weight $2w(T)$. Since our edge weights satisfy the triangle inequality, we have $w(v_i, v_{i+1}) \leq w(W_i)$ for all i, and hence $w(T') \leq 2w(T)$. Finally, since all our weights are non-negative and T'' is contained in T', it follows that $w(T'') \leq 2w(T)$. Thus we have constructed a spanning tree T'' of $G[V]$ with total weight at most twice that of an optimal network, as required.
>
> (This bound can actually be improved from a factor of 2 to a factor of $2 - 2/x$, where x is the number of leaves of T. This drops out if you discard the highest-weight walk W_i before forming T'.)

12. [$\star\star\star\star$] Let G be a **3-regular** graph (also known as a **cubic** graph). Show that if a maximum matching in G has size k, then the purely greedy algorithm of Question 7 returns a matching of size at least $3k/5$. (**Hint:** How many edges are in the matching returned by the greedy algorithm?)

> **Solution:** Writing $n = |V|$, we prove that the greedy algorithm returns a matching of size at least $3n/10$ — since a maximum matching has size at most $n/2$, this will imply the result.
>
> Let M be the output of the greedy algorithm. Each edge $e \in E(G)$ is either present in M or intersects M, so every edge in G intersects some edge of M. Since G is 3-regular, every edge of M intersects

exactly 5 edges of G (including itself), so $|M| \geq |E(G)|/5$. By the handshaking lemma proved in Lecture 4, G has exactly $\frac{1}{2}\sum_{v \in V} d(v)$ edges. Since G is 3-regular, it follows that $|E(G)| = 3n/2$. Since we already proved $|M| \geq |E(G)|/5$, it follows that $|M| \geq 3n/10$ as claimed.

13. [$\star\star\star$] Let G be an n-vertex bipartite graph, and let M be a matching in G. Let k be the size of a maximum matching in G. In lectures, we proved Berge's lemma: $|M| = k$ if and only if M has no augmenting paths. Prove that for all odd integers $\ell \geq 3$, if M has no augmenting paths of length less than ℓ, then $|M| \geq \frac{\ell-1}{\ell+1}k$. **Hint:** Consider the proof of Berge's lemma.

Solution: As in the proof of Berge's lemma, let M' be a maximum matching, and consider the symmetric difference $S = M \triangle M'$. Recall that every component of S is an even cycle or a path, and that their edges alternate between M and M'. So in more detail, for some $n_1, \dots, n_4$, S consists of:

- n_1 even cycles with the same number of M-edges and M'-edges;
- n_2 even paths with the same number of M-edges and M'-edges;
- n_3 odd paths with one more M-edge than M'-edges;
- and n_4 odd paths with one more M'-edge than M-edges (i.e. augmenting paths).

Thus we have $|M'| = |M| - n_3 + n_4 \leq |M| + n_4$.

By hypothesis, M has no augmenting paths of length less than ℓ, so each augmenting path contains at least $(\ell - 1)/2$ edges of M. Thus $|M| \geq n_4(\ell - 1)/2$, which implies $n_4 \leq 2|M|/(\ell - 1)$. Plugging this back into $|M'| \leq |M| + n_4$ yields

$$|M'| \leq |M| + \frac{2|M|}{\ell - 1} \Leftrightarrow |M'| \leq \frac{\ell + 1}{\ell - 1}|M|.$$

Since $|M'| = k$, this implies the result.

With some more effort, this can be adapted into an algorithm to find a matching which is *almost* maximum, but which works in $O(|E|)$ time — much faster than MaxMatching.