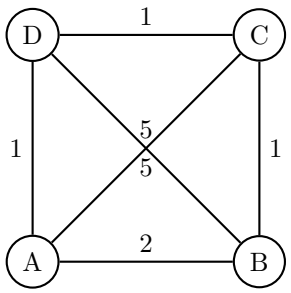
Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Traveling Salesman Problem

In the lecture, we learned an approximation algorithm for the Traveling Salesman Problem based on computing an MST and a depth first traversal. Suppose we run this approximation algorithm on the following graph:



The algorithm will return different tours based on the choices it makes during its depth first traversal.

1. Which DFS traversal leads to the best possible output tour?

2. Which DFS traversal leads to the worst possible output tour?

3. What is the approximation ratio given by the algorithm in the worst case for the above instance? Why is it worse than 2? (*Hint*: Consider the triangle inequality on the graph).

Solution:

1. **A-D-C-B**, **D-C-B-A**, **B-C-D-A**, or **C-D-A-B**
   Explanation:
   The MST is {(A,D),(D,C),(C,B)}. The optimal tour uses all of these edges.
   For example, if we start traversing the tree at A, then the only traversal would be to follow the tree and go through the vertices in the order D, C, B
   Another potential traversal would be to start at D, move to C, then B, and backtrack, before going to A.
   Notice that if we started at D and moved to A first, then we would have to visit C next, and this would not lead to the best possible output tour (as per part (b)).

2. **C-B-D-A** or **D-A-C-B**
   See above explanation.
   Notice that the worst possible tour overall, **D-C-A-B**, is not possible to be output by this algorithm, regardless of the DFS traversal used.

3. $\frac{12}{5}$
   From previous parts, the optimal tour length is 5 while the worst possible is 12. This is worse than 2 because our graph does not satisfy the triangle inequality, specifically $\ell(A, C) > \ell(A, B) + \ell(B, C)$ and $\ell(B, D) > \ell(A, B) + \ell(D, A)$, which is the necessary condition for our algorithm to guarantee an approximation ratio of 2.

2 Independent Set Approximation

In the Max Independent Set problem, we are given a graph $G = (V, E)$ and asked to find the largest set $V' \subseteq V$ such that no two vertices in V' share an edge in E.
Given an undirected graph $G = (V, E)$ in which each node has degree $\leq d$, give an efficient algorithm that finds an independent set whose size is at least $1/(d+1)$ times that of the largest independent set. Only the main idea and the proof that the size is at least $1/(d+1)$ times the largest solution's size are needed.
Solution: Initially, let G be the original graph and $I = \emptyset$. Repeat the process below until $G = \emptyset$:

1. Pick an arbitrary node v in G and let $I = I \cup \{v\}$.
2. Delete v and all its neighbors from the graph.
3. Let G be the new graph.

Notice that I is an independent set by construction. At each step, I grows by one vertex and we delete at most $d+1$ vertices from the graph (since v has at most d neighbors). Hence there are at least $|V|/(d+1)$ iterations. Let K be the size of the maximum independent set. Since $K \leq |V|$, we can use the previous argument to get:

$$|I| \geq \frac{|V|}{d+1} \geq \frac{K}{d+1}$$

3 Local Search for Max Cut

Sometimes, local search algorithms can give good approximations to NP-hard problems. In the Max-Cut problem, we have a graph $G(V, E)$ and we want to find a cut (S, T) with as many edges crossing as possible. One local search algorithm is as follows: Start with any cut, and while there is some vertex $v \in S$ such that more edges cross $(S - v, T + v)$ (or some $v \in T$ such that more edges cross $(S + v, T - v)$), move v to the other side of the cut. Note that when we move v from S to T, v must have more neighbors in S than T.

(a) Give an upper bound on the number of iterations this algorithm can run for (i.e. the total number of times we move a vertex).

(b) Show that when this algorithm terminates, it finds a cut where at least half the edges in the graph cross the cut.

Solution:

(a) $|E|$ iterations. Each iteration increases the number of edges crossing the cut by at least 1. The number of edges crossing the cut is between 0 and $|E|$, so there must be at most $|E|$ iterations.

(b) $\delta_{in}(v)$ be the number of edges from v to other vertices on the same side of the cut, and $\delta_{out}(v)$ be the number of edges from v to vertices on the opposite side of the cut. The total number of edges crossing the cut the algorithm finds is $\frac{1}{2}\sum_{v \in V} \delta_{out}(v)$, and the total number of edges in the graph is $\frac{1}{2}\sum_{v \in V}(\delta_{in}(v) + \delta_{out}(v))$. We know that $\delta_{out}(v) \geq \delta_{in}(v)$ for all vertices when the algorithm terminates (otherwise, the algorithm would move v across the cut), so the former is at least half as large as the latter.