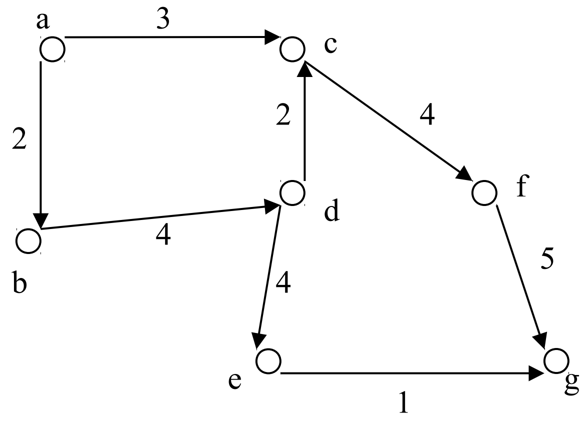
Network Flow

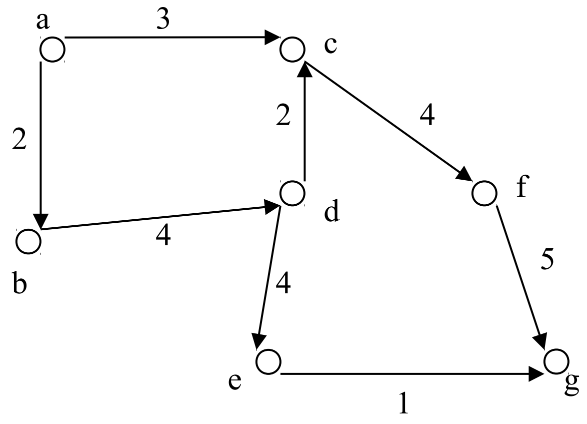
1. [1 pt] You are given an undirected graph with each edge having a capacity of **1 unit** (i.e., a maximum of 1 unit of water can flow in both directions). Suppose you ran the Ford-Fulkerson algorithm between a pair of vertices s and t on this graph and it terminated **with a final flow of 2**. Which of the following is definitely **incorrect**?

    1. The graph is possibly a cycle.
    2. We need to remove at least 2 edges to disconnect the graph such that one piece contains s and the other piece contains t.
    3. There have to be two paths from s to t which do not share edges, though they may share intermediate vertices.
    4. There have to be two paths from s to t that neither share intermediate vertices nor edges.

2. [2 pts] Execute the Ford-Fulkerson algorithm on the following graph to compute the maximum flow from a to g.



3. [2 pts] Execute the Edmond-Karp algorithm on the following graph to compute the maximum flow from a to g.



4. We briefly discussed in the class about how Max flow and Min cut are related. Given a graph, suppose I ran a max-flow algorithm like Edmond-Karp on it. How can you identify the min-cut edges from it?

Bipartite Matching:

1. Suppose you invite n men and n women to a party. Each man knows exactly k women and each woman knows exactly k men. Of course, acquaintance is mutual. As a party organizer, you want to come up with a dancing schedule such that each woman dances with a different man but with the constraint that they both should know each other. Design and analyze an efficient algorithm to solve this problem.

2. Consider the problem of job scheduling. You have a list of jobs and their requirements. For example, a job might require a Windoze machine with video encoding software. Or another might require a Linux machine with Python enabled. You also have a list of diverse machines and their capabilities (such as OS, languages/libraries installed etc). You want to finish as many jobs as possible but each machine can run only one job at any point in time. Design an efficient formulation to solve this problem.

3. (Hard) Consider the problem (2) above. Suppose you want to install additional software into existing machine if it will help run more jobs. For eg, suppose you had a Linux machine m without Python that was not allocated any job and there was some job j that required Linux/Python , then you can assign j to m after installing Python in it. Suppose you are given a set of jobs and their requirements and the current capabilities of machines. You want to alter the systems so that all jobs are allocated. Design an algorithm to identify the smallest set of changes that you can do so that all jobs are allocated.
Hint: For this you might want to lookup Hall's theorem for perfect matching.

NP-Completeness

1. Answer true or false to the following questions, and give proofs:
   - A problem X is NP-Complete if for all problems Y in NP, X $\leq$P Y
   - If problem X is polynomially reducible to Y and Y is polynomially reducible to Z, then X is polynomially reducible to Z
   - 1-SAT is in P
   - 2-SAT is in NP

2. Given an algorithm A for the decision version of the Maximum Independent Set problem, design an algorithm B to compute the size of the maximum independent set that uses A as a subroutine. If A takes time T(n, m) where the graph has n vertices and m edges, what is the running time of B?

3. Given two graphs G and G', the Subgraph Isomorphism problem seeks to determine whether G' is a subgraph of G if we are allowed to relabel the vertices of G' but make no other changes to its structure. Prove that the Subgraph Isomorphism problem is NP-complete.

4. Show that the Maximum Independent Set problem is NP-Complete by reducing from Vertex Cover. Likewise, prove in the reverse direction, i.e. that Vertex Cover is NP- Complete by a reduction from Maximum Independent Set.

5. Solve problem 35.2-3 from the book.

6. Solve problem 35.2-5 from the book.

7. Prove that Maximum Independent Set problem is NP-Complete by reducing it to Max-Clique problem. Now do the vice versa.

8. Prove that Hamiltonian cycle problem in NP-C using Traveling Salesman problem.

9. Prove Hamiltonian cycle is NP-C from Hamiltonian Path (look it up).

10. Prove that Partition problem is NP-C using Subset Sum.

11. Consider three popular graph problems – minimal vertex cover, maximum independent set and maximum clique. For each of the special graphs given below, list if the problem is still in NP-C or not. For example, if you know the input is a complete graph, does vertex cover still in NP? (resp. MIS and CLIQUE). Come up with an efficient algorithm if they are not NP-C. If you do not know what the graph, see at the end of the document for a link to Wikipedia.

Graph	**VC**	**MIS**	**CLIQUE**
Complete graph			
Completely disconnected graph (no edges at all)			
Bipartite graph			
Complete bipartite graph			
Trees (hint: trees are special type of bipartite graphs – why?)			
Petersen graph			
Star graph			
Cycle graph			
Friendship graph			
Wheel graph			
Grid graph			
Path graph			
(harder) Rook graph			
(harder) Perfect graphs			
(harder) DAGs			

Approximation and Randomized Algorithms:

1. The vertex-cover problem and the clique problem are complimentary in the sense that the complement of an optimal vertex cover corresponds to a maximum clique in the complement graph. We also know that the vertex cover problem has an approximation algorithm with a constant approximation ratio. Does that mean that there is an approximation algorithm for the clique problem with a constant approximation ratio? Justify your answer.
2. What if the problems were MIS and CLIQUE?

Useful Links:

Petersen Graph: http://en.wikipedia.org/wiki/Petersen_graph

Complete Graph: http://en.wikipedia.org/wiki/Complete_graph

Complete Bipartite graph: http://en.wikipedia.org/wiki/Complete_bipartite_graph

Bipartite Graph: http://en.wikipedia.org/wiki/Complete_bipartite_graph

Tree: http://en.wikipedia.org/wiki/Tree_(graph_theory)

Cycle graph: http://en.wikipedia.org/wiki/Cycle_graph

Friendship graph: http://en.wikipedia.org/wiki/Friendship_graph

Star graph: http://en.wikipedia.org/wiki/Star_(graph_theory)

Wheel graph: http://en.wikipedia.org/wiki/Wheel_graph

Perfect graphs: http://en.wikipedia.org/wiki/Perfect_graph

Rook graph: http://en.wikipedia.org/wiki/Rook%27s_graph

Grid graph: http://mathworld.wolfram.com/GridGraph.html

Path graph: http://mathworld.wolfram.com/PathGraph.html