**Applications of Network Flow**

**Obvious applications of network flow involve physical situations, such as a set of pipes moving water, or traffic in a network. For these situations, the translation of the input data into an appropriate graph is fairly intuitive.**

**However, a vast majority of the applications of network flow pertain to problems that don't seem to involve the physical movement of items through networks.**

**While we don't have time to look at all the types of applications of network flow, we will analyze one specific problem, bipartite matching, that can be solved using network flow, along with a few extra sample problems that can be solved with network flow.**

**Bipartite Matching**

**The bipartite matching problem is as follows:**

**Input: two mutually exclusive sets of equal size U and V, along with a list of ordered pairs of the form (u, v) where u U and v V, indicating pairs of members, one of each set that can be "paired" together.**

**Output: True, if there exists a way to pair up each item in U with an item in V such that each item in both sets appears in exactly one pairing, and false otherwise.**

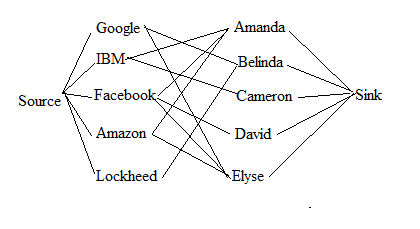
**Solution: Let n = the size of each input set. Set up a graph with 2n+2 vertices. Create one vertex for each item in each set and add source and sink vertices. Add an edge from the source to each item in set U with capacity 1. Add an edge from each item in set V to the sink with capacity 1. Add an edge between each item in set U and set V that are in the set of ordered pairs with capacity 1. Calculate the maximal flow of this network. If the answer is n, then a complete matching exists, otherwise a complete matching doesn't exist. If you want the matching, keep track of each "edge" added during each iteration of the algorithm. (Note: some edges change as well.)**

**Instance of Bipartite Matching**

**Set of companies:{Google,Microsoft, Facebook, Amazon, Lockheed}**

**Set of students: {Amanda, Belinda, Cameron, David, Elyse}**

**Set of offers: {(Google, Belinda), (Google, Elyse), (Microsoft, Amanda), (Microsoft, Cameron), (Facebook, Amanda), (Facebook, David), (Facebook, Elyse), (Amazon, Amanda), (Amazon, Elyse), (Lockheed, Belinda) }**

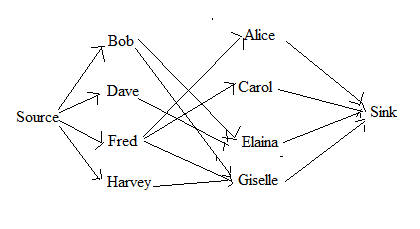
****

**Second Instance of Bipartite Matching**

**Set of boys: {Bob, Dave, Fred, Harvey}**

**Set of girls: {Alice, Carol, Elaina, Giselle}**

**Set of ordered pairs: { (Bob, Elaina), (Bob, Giselle), (Dave, Elaina), (Fred, Alice), (Fred, Carol), (Fred, Giselle), (Harvey, Giselle) }**

****

**Grand Dinner Problem (from ACMUVA)**

**N teams attend a dinner. Team i has ti members. There are M tables at the dinner, with M ≥ N. Table i can has si chairs. We wish to seat all teams such that no two team members are at the same table, so that we maximum students getting to meet members of other teams. Can we do so?**

**Solution using Network Flow**

**Create a flow network with N + M + 2 vertices. Create one vertex for each team and one for each table. Create extra source and sink vertices. Create edges from the source to each team with a capacity of ti. Create edges from each table vertex to the sink vertex with capacity si. Finally, add edges from each team to each table, with capacity 1, since each team can provide at most one person per table. Run the network flow algorithm. If the maximal flow equals the sum of the number of team members, the seating can be done. Otherwise, it can not be.**

**Two small examples:**

**team sizes: 4, 5, 3 , 5**

**table sizes: 3, 5, 2, 6, 4**

**team sizes: 4, 5, 3, 5**

**table sizes: 3, 5, 2, 6, 3**

**Museum Guard Problem: 2009 South East Regional**

**A museum employs guards that work in 30 minute shifts: 12am - 12:30am, 12:30am-1am, ..., 11:30pm-12 am. Each guard has a list of times he/she can NOT work. (For example, a particular guard might not be able to work from 3:30 am to 7:30am and from 4:29pm to 8:01pm. In this case, the guard could work the 3am-3:30am and 7:30am-8am shifts, but not the 4pm - 4:30pm shift or 8pm-8:30pm shift.) Furthermore, each guard has a maximum number of hours they can work in a single day. Determine the maximum number of guards we can have scheduled to cover each shift without violating any of the constraints.**

**Solution using a flow network**

**As usual, we create an extra source and sink.**

**Each guard will be a vertex in the flow network. From the source, we connect an edge to each guard with an integer representing the maximum number of shifts that guard can work.**

**Each shift (there are 48) will become a node in the flow network. Add an edge with capacity 1 from each guard to each shift where the guard is able to work that shift.**

**Finally, if we are to set the capacities from all the shifts to the sink to 1, and we run the network flow algorithm at obtain 48, we know that we can post 1 guard for each slot. We can re-run the algorithm with all of these capacities set to 2 and see if the resulting flow is 96 or not. If so, we move onto 3, and so forth. Since there are at most 50 guards, our answer will never exceed 50 and this will run in time.**

**A more clever approach involves a binary search. Try capacities at 25 for each of these edges. If this doesn't work, go to 12, If it does, go to 37, and so on. Basically, rather than checking if 1 works, then 2, then 3, etc. a binary search will hone in on the answer a bit more quickly, especially in the cases that the answer is closer to 50.**

**Cow Steeplechase Problem (from USACO)**

**Given a list of horizontal line segments (none of which intersect each other) and vertical line segments (none of which intersect each other), calculate the minimum number of line segments that must be removed, so that no two lines intersect each other, or alteratively, the most number of line segments that mutually don't intersect one another.**

**Solution**

**We can create a bipartite matching solution. Our goal is to match horizontal line segments to vertical line segments in such a way that each pair intersects. We know that if we have a set of these intersecting pairs, at the very least, one item in each pair must be removed. Thus, what we really want is the maximum matching. Once we have this (say there are 7 matching pairs in our maximal matching), then we have proof that 7 of the segments must be removed. No other matching forces us to remove more. Thus, that how many we are forced to remove to create no intersecting line segments. Alternatively, we can calculate the maximum set of segments we can have without any two intersecting by taking the total number and subtracting out this maximum matching.**