Maximum Flow Algorithms and Applications III

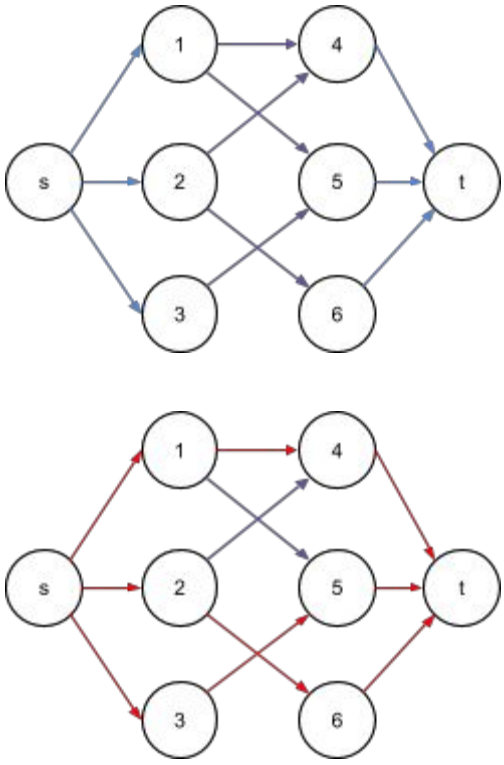
Alexander Shieh
INFOR, Taipei Chien Kuo High School

After learning the two algorithms in previous reports, I started to look for problems that can be solved elegantly using maximum flow concepts. Here are two common categories of problems:

Maximum Bipartite Matching

Bipartite Graph: A graph that could be splitted into two set of vertices, and there exists no edges between two vertices in the same set.

Matching: A set of non-adjacent edges (edges that do not share endpoints).



We can use the maximum flow model to calculate the amount of edges in the largest matching set: First we divide the two set of vertices as set L and R, and for every edge between L and R in the original bipartite graph we add an edge $L \rightarrow R$ and a capacity of 1 (purple edges). For every vertex $v \in L$ we add an edge $s \rightarrow v$ with capacity 1, and for every vertex $v \in R$ we add an edge $v \rightarrow t$ (blue edges). We can ensure that for every flow path $s \rightarrow t$ only involve two other vertices $(u \in L,\ v \in R)$, and the edge between them is a matched edge because by flow conservation, no more flow will flow out from u and no more flow will flow into t. So eventually the maximum flow of this network equals to maximum matched edges.

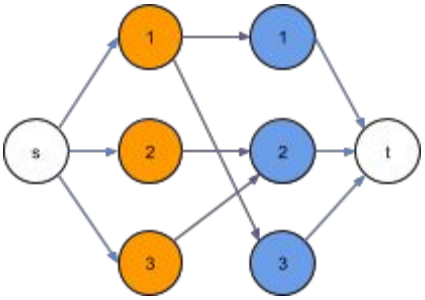
Right Top: The flow model for a bipartite graph.
RIght Bottom: The maximum flow / bipartite matching of the graph.

Here's a classic bipartite matching problem (NTUJ0423):

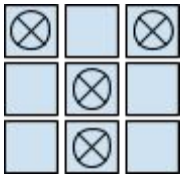
Given a NxN grid and the location of K asteroids, if we fire a weapon from a column or a row, then the asteroids on that column or row are all eliminated. Find out the minimum shots required to take out all asteroids.

We can infer that for all asteroids, the shot that wipes it out comes from either the column or the row it resides in. We can build a bipartite graph of row as L and column as R and every asteroid represents an edge $(u \in L,\ v \in R)$. Then we want to find the minimum set of vertices that every edge in the graph has at least one endpoint in the set (called **minimum vertex cover**).

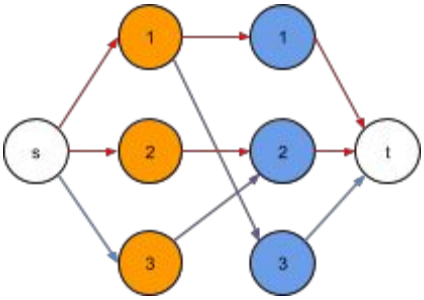
The König's theorem state that the maximum matching of a bipartite graph equals to the minimum vertex cover, so we just need to compute the maximum bipartite matching of the above graph.



Left: The bipartite model for a 3x3 grid

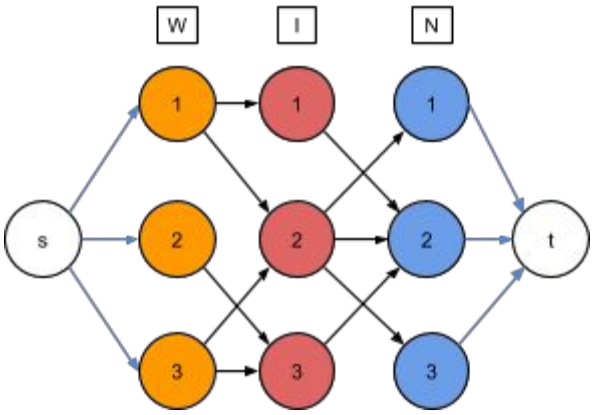


Right: matching result.



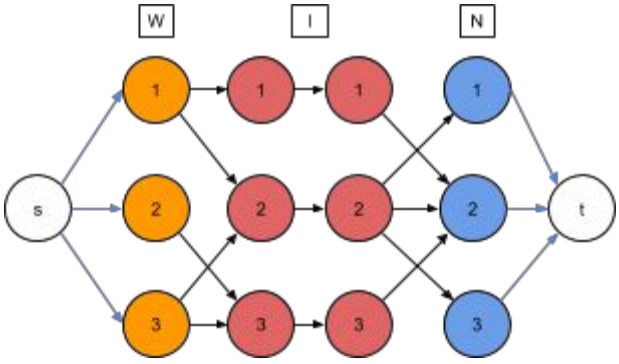
Now, let's look at a similar problem (NTUJ2277):

Given a NxM tile, with W, I or N on each square. Find the maximum number of $W \rightarrow I \rightarrow N$ *(Could be three in a line or L-shaped) that can be cut of from the given tile.*



This looks like a "tripartite" graph matching problem, so we build a flow model like this and set the capacity of each edge to one. But a flaw of this model emerges: If there's a vertex $y \in I$ that was connected to more than one vertex $x \in W$ and more than one vertex $z \in N$ (like red 2 in the middle), than y could be used in more than one matching path which violates our requirements.

One way to solve this is by splitting a vertex $y \in I$ into y_{in} and y_{out}, then connect them with an edge with capacity of 1, then it looks like the following graph. It is guaranteed that the above situation would not occur in our new model because no more than a unit flow can flow through y_{in} and y_{out}.



Maximum Flow Minimum Cut Theorem

Cut: A set of edges in a flow network that there exist no path $s \rightarrow t$ once removed (separates the graph into two parts). The weight of a cut equals to the sum of the capacity of every edge in the set.

The theorem states that the maximum flow equals to the minimum cut (the minimum cut can be deemed as the bottleneck of the maximum flow).

Here are some problems we can solve using the maximum flow minimum cut theorem(NTUJ0618 and POJ3469):

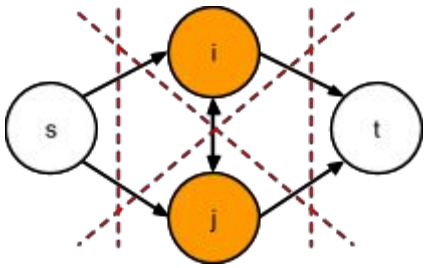
There are n experiments, completing ith experiment brings you Pi dollars of reward, though every experiments needs some apparatus which buying the ith apparatus costs Ci dollars. Given P1~Pn, C1~Cm and n lists of apparatus for experiment 1~n, find out the maximum gain by conducting some experiments.

The decision we made is either choose to conduct an experiment or not. Thus, we can use another viewpoint, which assume we finished all experiments and earn $\sum_{i=1}^{n} Pi$ and if we cancel an experiment i then we lose P_i, and if we approve one experiments then we loose the cost of buying apparatus.

After the analysis, we can construct a bipartite graph with experiments as L and apparatus as R, then we connect each experiment to the corresponding apparatus. Now we add a source s and a sink t then build edge $(s,\ u \in L)$ with capacity P_u and $(v \in R,\ t)$ with capacity C_v, then set every edge between experiments and apparatus to infinity. We can prove that every cut on this graph represents a legal choice of conducting some experiments while canceling the others, because if we don't carry out an experiment u then we have to cut the flow $(s,\ u)$ and if we are using an apparatus v then there will be a flow $(v,\ t)$. As a result, the weight of the minimum cut is our minimum total cost.

You have a computer with two CPUs A and B, and you have to run n programs. Running the ith program costs Ai when executed on A and costs Bi on B. There are m tasks which program i and j have to interact, if i and j are conducted on different CPUs, it costs W(i, j). Find the minimum cost of running all programs and tasks.

It is another decision problem: On which CPU should I run this program? So we can apply the thoughts we just use to



build a flow network with programs as vertices, then connect $(s,\ i)$ with capacity equal to Ai and connect $(i,\ t)$ with the capacity of Bi. Again, a cut of this graph means the designation of each program, because in order to cut the graph, we must cut $(s,\ i)$ or $(i,\ t)$, which is equivalent of choosing between CPU A and CPU B. How about a program j that interacts with i?

We can build a bidirectional edge with the capacity of $W_{(i,j)}$. The reason is that we choose to cut (s, i) and (j, t) while there is still an edge between (i, j) then edge (i, j) must be cut as well to prevent a flow $s \rightarrow j \rightarrow i \rightarrow t$ and that means we must add the cost $W_{(i,j)}$ when we choose to select Ai and Bj. *Left: Four possible cuts.*

There are lots of topics relating to maximum flow problems, this report only covers a tiny portion of them. Though sometimes figuring out the correct model to solve a problem is troublesome, but the answers are astonishing. Learning algorithms can be a breathtaking voyage through the heart of computer science and at the same time, enhance our approach to problems.

Albert Einstein once said "If you can't explain it simply you don't understand it well enough", I wish my report stated the concepts clearly and may be valuable for those who are new to these topics.

Reference:

NTU Online Judge

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms*.
2. 秋葉 拓哉, 岩田 陽一, 北川 宜稔. *プログラミングコンテストチャレンジブック*