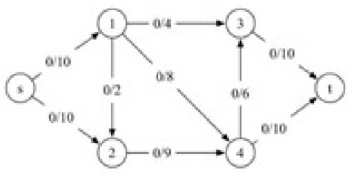
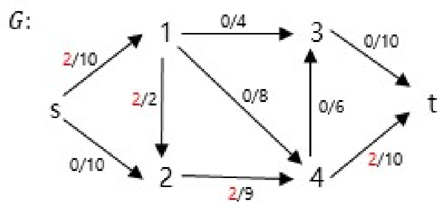
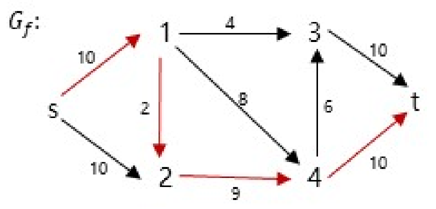
CS240 Homework #2

Zhiqiang Xie 77892769

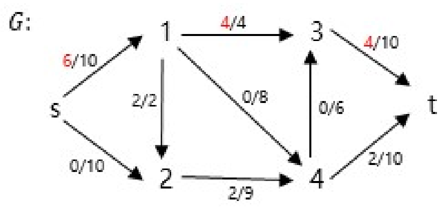
Problem 1: Ford-Fulkerson algorithm

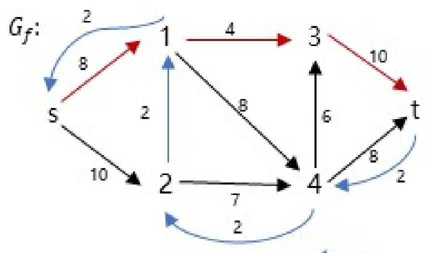




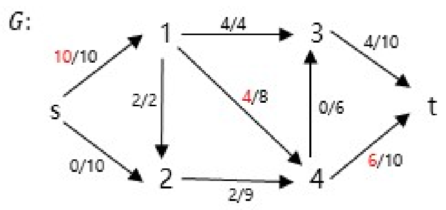


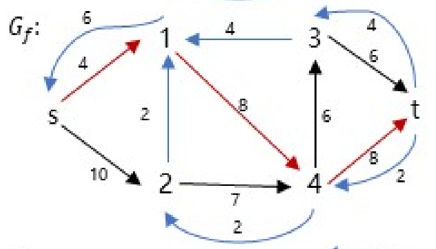
Flow value = 2



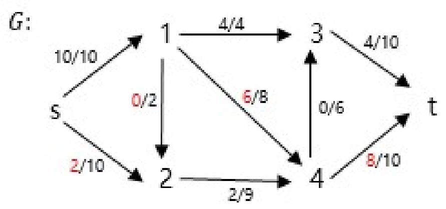


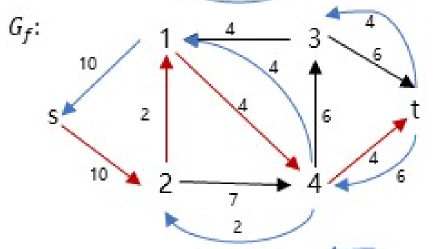
Flow value = 6



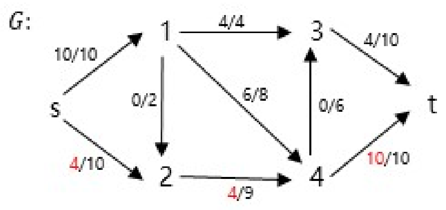


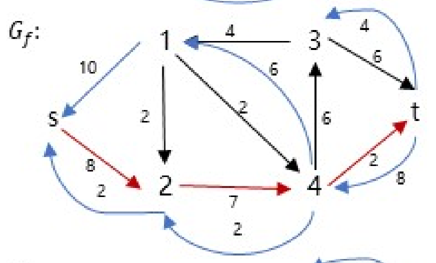
Flow value = 10



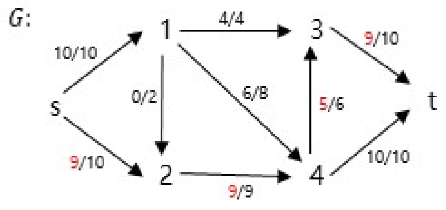


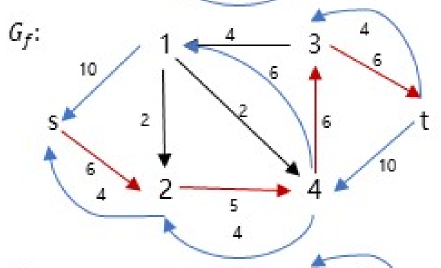
Flow value = 12





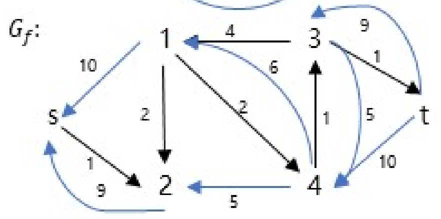
Flow value = 14





Flow value = 19

No augmenting path,
Here we terminate.



Max flow value = 19

Problem 2: Selecting staffs

Here we can construct a flow network like this:

- Source node `s` has four output conduits with capacity m_1, m_2, m_3, m_4 to node `A`, `B`, `C`, `D`, respectively. Where `A`, `B`, `C` and `D` represent the corresponding classes.
- Every class node (`A`, `B`, `C`, `D`) has m output conduit with capacity 1, where m is the number of staffs in the corresponding class. Every node next to the class layer stands for a specific staff in that class.
- Next to the staff layer, we have a department layer with k nodes which reprensents the corresponding department. Every staff node in the staff layer points to a department node with capacity 1.
- Finally, every department node has a conduit with capacity 1 to the sink node `t`.

Now we can find the max-flow of the flow network by Ford-Fulkerson algorithm or any other efficient methods.

If the $v(f) == k$ is satisfied, we can check the staff layer to get the selected staffs. Otherwise, the constrains are not satisfiable.

Problem 3: Maintain the maximum flow

We assume that $f(v, u) = c(v, u) = 0$ if $(v, u) \notin E$ and $c_f(v, u) = 0$ if $(v, u) \notin E_f$ for simplification.

1. - Let $G_f = (V, E_f)$ be the residual graph of G. The residual graph can be composed from the original maximum in G. This step costs $O(V + E)$ time.
   - $c(v, u) = c(v, u) + 1$
   - $c_f(v, u) = c_f(v, u) + 1$

   Now we can apply BFS or DFS to search an augmenting path. Once we found it, we can update the maximum flow by it, otherwise, the maximum flow stay unchanged. This step costs $O(V + E)$ time.

   Since here is at most one augmenting path existed, we just need to run this procedure once.

   Therefore, the time complexity $T(V, E) = O(V + E) + O(V + E) \in O(V + E)$.

2. - Let $G_f = (V, E_f)$ be the residual graph of G. The residual graph can be composed from the original maximum in G. This step costs $O(V + E)$ time.
   - If $f(v, u) < c(v, u)$ in G, we set $c(v, u) = c(v, u) - 1$ and the maximum flow stay unchanged. Terminated.

   Otherwise we have $f(v, u) == c(v, u)$:

   - Apply BFS or DFS to find a path p_1 in G_f from v to s. This step costs $O(V + E)$ time.
   - Apply BFS or DFS to find a path p_2 in G_f from t to u. This step costs $O(V + E)$ time.
   - Now we get a path p' through the edge (u, v) from t to s in G_f.
   - For every edge e in p', we decrease $c_f(e)$ by 1.
   - For every edge (a, b) in p' except (u, v), we increase $c_f(b, a)$ by 1.
   - Now we set $c(v, u) = c(v, u) - 1$ and for every edge (a, b) in p', we decrease $f(b, a)$ by 1. So the value of the maximum flow is deducted by 1.
   - Finally we perform a search in G_f for an augmenting path. This step costs $O(V + E)$ time.
   - Once we found an augmenting path, we can update the maximum flow by it, otherwise, return the deducted maximum flow above.

   Therefore, the time complexity $T(V, E) = O(V + E) + O(V + E) \in O(V + E)$.

Problem 4: Determine the matrix

Here we can construct a flow network like this:

- Source node `s` has m output conduits with capacity $c_1, c_2, \ldots, c_m$ to node $C_1, C_2, \ldots, C_m$, respectively.
- Every node C_i in column layer has n output conduits with capacity 1 to node $R_1, R_2, \ldots, R_n$, respectively.
- Every node R_i in row layer has one output conduits with capacity $r_1, r_2, \ldots, r_n$ respectively to the sink node `t`.
- Since each element in the matrix is either 0 or 1, all of the capacity and value of flow are **integral**.

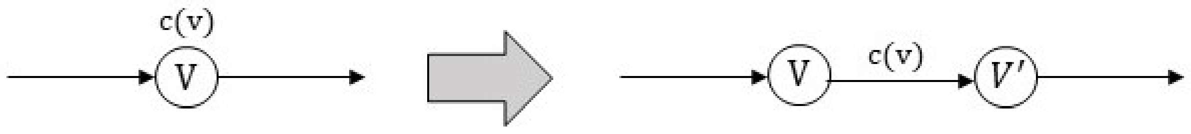
Now we can find the max-flow of the flow network by Ford-Fulkerson (polynomial-time variants) algorithm or any other efficient polynomial-time methods.

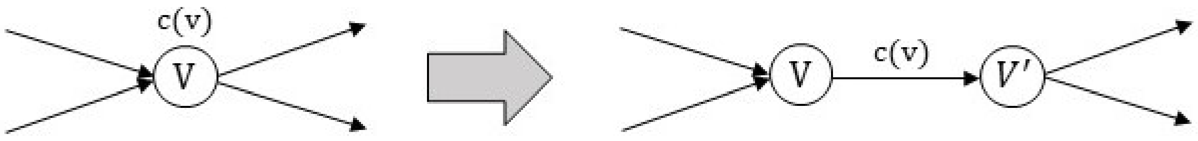
Since we guarantee that $\sum_i r_i = \sum_j c_j$, if the $v(f) == \sum_i r_i$ is satisfied, we can check the connections between column layer and row layer to get every entry of the matrix. Otherwise, such a matrix doesn't exist.

Problem 5: Vertex capacity

Just cascade one more dummy vertex `v'` behind or ahead of v with vertex capacity $c(v)$. And then set the capacity of the primary road ($(v', v) or (v, v')$, depends on the direction) between v and the dummy node to be $c(v)$.

In other word, regard subgraph $v + (v, v') + v'$ to be a super-vertex which connects to other vertices just like the original v.





Now the question is totally a standard maximum flow problem. We achieve the vertex capacity by the constraints from an extra neighbor conduit. So the algorithm for this problem is just Ford-Fulkerson algorithm or any other efficient methods.

Problem 6: Unique min-cut

Firstly, we find a minimum $s - t$ cut C, and denote its capacity by $|C|$.

For all edges e_i in C, we try to increase the capacity of e_i by 1 and find a minimum cut C_i in that new graph. If $|C| = |C_i|$ for some i, then clearly C_i is also a minimum cut in the original graph and $C \neq C_i$, so the minimum cut is not unique.

For any $|C| \neq |C_i|$, there must be at least one edge in C that is not in C_i, and we just increase capacity of edges one by one. Thus, the graph has unique minimum cut **iff**$|C| < |C_i|$ for all i.

Since the number of edges in C is less than $|E|$ and the algorithm for finding the minimum cut is polynomial, so this algorithm runs in polynomial time.

However, the algorithm method runs finding the min-cut multiple times, which is expensive even though it's polynomial. I'll present another fast method.

Run Ford-Fulkerson algorithm and get the residual graph G_f.

- In G_f, from the source node `s` we run BFS and then we get a min-cut (A, B), where all nodes in A are reachable from `s`.
- Reverse all edges in G_f, from the sink node `t` we run BFS and then we get a min-cut (A^c, B^c), where all nodes in B^c are reachable from `t`.

Therefore, we could see that (A, B) is the leftmost min-cut and (A^c, B^c) is the rightmost. If $V(A) == V(A^c)$ (the $V(A)$ means that the vertex set in subgraph A), we say that the minimum $s - t$ cut is unique. Otherwise, here exists at least two different minimum $s - t$ cut.

And it's obviously running in polynomial time.