Problem 1 Chapter 7-3

(a) The value of the flow: 10
Maximum flow: 11

(b) Min cut from s, a, b, c to t, d.
Capacity of the min cut is 11.

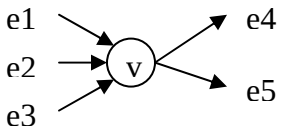
Problem 2 Chapter 7 -12

Idea: Run max-flow algorithm on G, and find a min cut of G, which separates the vertices set V into two sets S and T. Remove any k edges from S to T. if Max-flow(G) > k, the maximum s-t flow on G' will be Max-flow(G) – k; otherwise the new max flow will be 0.

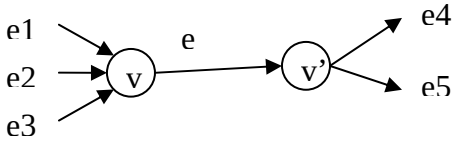
Problem 3 Chapter 7-13

We are given a graph G with node capacity, and we can build a new graph G' = (V', E'), which has only edge capacities as follows:

For each node v in G, we separate it into two nodes v and v', then define the capacity of the new added edge e(v, v') = capacity(v). And assign all the edges in G with capacity infinity.



v in G



v, v'in G'

Clearly, G and G' could hold the same amount of maximum flow, where G with node capacity but G' with edge capacity. Now we reduced the problem of finding the maximum flow in Graph with node capacity to the maximum flow problem with edge capacity we talked in class. We can simply run the Max-flow algorithm on the new graph G', to find an s-t max flow.

How to write an algorithm for such problem?

Algorithm Max-Flow-Node-Capacity

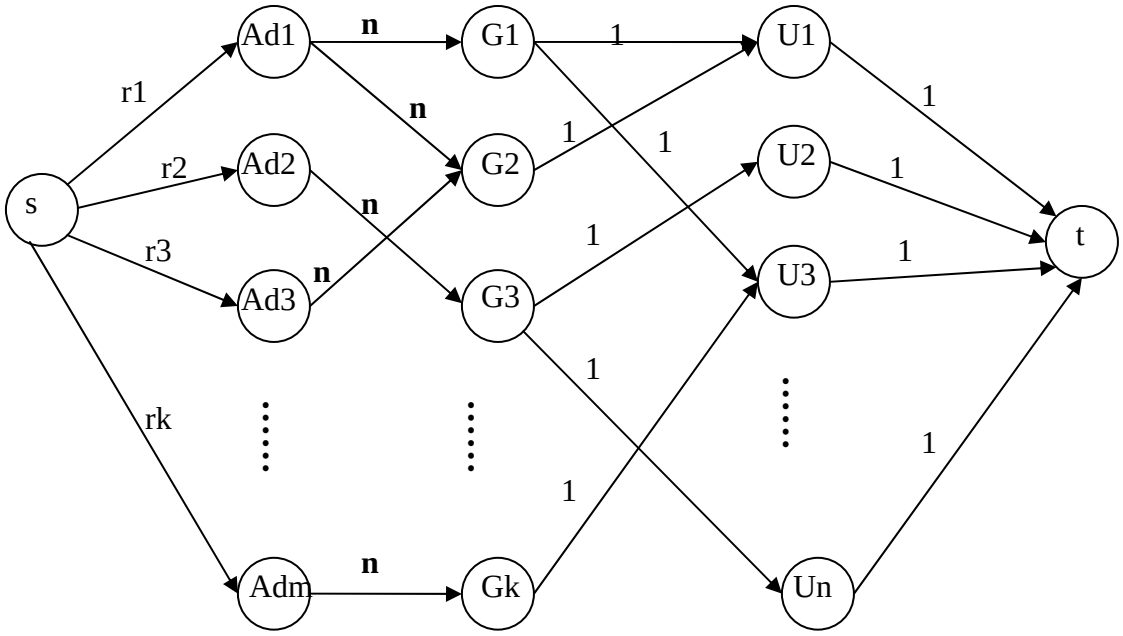
Input: A graph G= (V, E), and each node v with capacity c(v)
Output: The maximum flow value
Step 1: Construct the new graph G' following the instructions above;
Step 2: Run Max-flow algorithm on G', get the max-flow F;
Step 3: Output the max-flow F.
End of the algorithm.

We can see the problems of finding the max-flow with node capacity and finding the max-flow with edge capacity are equivalent. So, the Max-flow and Min-cut Theorem also hold for the node capacity case.

Problem 4 Chapter 7-16

To solve this problem, we show it can be reduced to the max-flow problem. The technique used is very similar to the reduction from Bipartite Matching to Max-flow.

From the problem given we can construct a graph with source s and sink t. The graph is constructed as follows: G = (V, E):
V is made up of four parts: {s, t}; the vertices which represent the m advertisers {Ad1, Ad2, …, Adm}; the vertices which represent the k groups {G1, G2, …., Gk} and the vertices which stand for the n users {U1, U2, … Un}. Then we add edges with capacity to G:
(1) For each vertex Adi, we add an edge from s to Adi with capacity ri (i = 1.. m).
(2) For each pair (Adi, Gj), we add an edge with capacity **n** if advertiser i wants to show its add to group Gj. i = 1.. m and j = 1..k .
(3) For each pair (Gi, Uj), we add an edge with capacity 1 if Uj is a member of Gi (j = 1.. n; i = 1..k).
(4) For each vertex Ui, add an edge from Ui to t with capacity 1. i = 1.. n.
Then we run the max flow algorithm on the graph G.
If there is a flow with size $\sum_{i=1..k} r_i$, then we can find a way to show the ads that could satisfy the advisors. In the max flow solution, if there is a flow from Adi to Uj then show ad i to user j; otherwise, there is no flow going through Uj, simply show any ad i to Uj where there is a path from ad i to Uj in G.



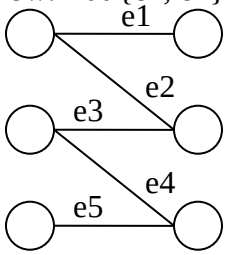
Another solution (more related to the survey problem solved in class):
Construct the graph following the steps given above. Set the capacity from s to Adi (i=1..m) with capacity (ri, n) (ri is the lower bound of the flow). All the other edges remain to have the same capacity as shown in the above.
Find a max flow on the graph which requires lower bound on some edges, using the algorithm introduced in class. If we can find a maximum flow with size n and satisfies the lower bound requirement; then show User j the Ad i, if there is a flow from Adi to Uj. Otherwise, there is no solution for this problem.

Problem 5 Chapter 7 - 18

(a) I will solve (b) and (c) first, then give the answer to (a).

(b) An example is given below:

M = {e2, e4}, k = 1. It is very easy to check there is a 3-matching M' which cover all the vertices below. But {e2, e4} is not a subset of {e1, e3, e5}.



(c)

We denote the nodes covered by M in Y as Y'.

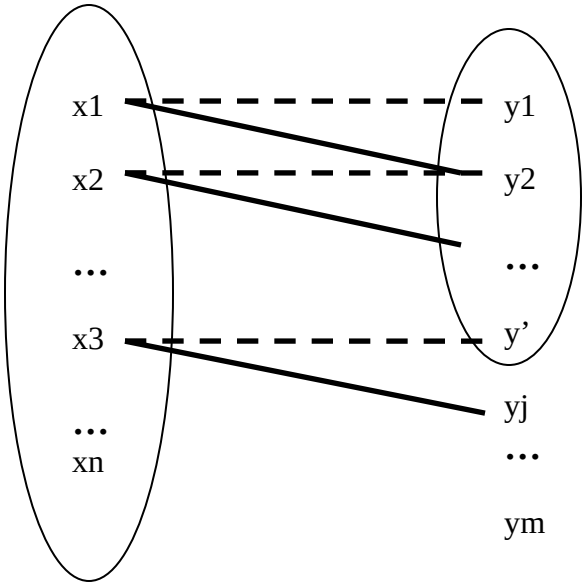
M': |M'| = K1, and M' is a largest matching cover every node in Y'.

M'': |M''| = K2, and M'' is a maximum matching in G with maximum cover property.

Maximum Cover Property: Among all the maximum matching of G, M'' covers the largest subset of Y'.

Now we assume K1 < K2. Later, we will show there is a contraction, which implies K1 cannot be smaller than K2.

If K1 < K2, then M'' cannot cover every node y in Y'. We find the first node in Y' that is not covered by M'', noting it as y1. Due to the definition of M', we know y1 is covered by M' and we assume (x1, y1) in M' (we call x1 is the partner of y1 in M'). In M'', x1 must be matched with some other node y2; otherwise, we can add (x1, y1) to M'', which contradicts the fact that M'' is a maximum matching. If y2 is not in Y', then we modify M'' by M'' = M'' – e{x1, y2} + e(x1, y1), which contradicts with the maximum cover property of M''. So, we know y2 must be in Y'; then we go on with this process. But, since Y' is finite, and our process will end finally. So, we can always find a way to switch the edges to let M'' cover more vertices in Y'. Take the below graph as an example, we can change M'' by removing the solid lines and adding all the dashed lines to let M'' cover one more node y1 in Y'. So, we get a contraction, and we showed that K1 cannot be smaller than K2. => K1 = K2.



(a) From (c) we know, there must be an maximum matching which contains the covered set Y'. To see if there is a coverage expansion with matching size |M'| +k.
Algorithm Coverage Expansion
Step 1: Run the maximum flow algorithm and find a maximum matching M'' with size K1.
Step 2: If K1< |M'|+k, then reports no such solution;
Else { While (M'' doesn't cover all the vertices in Y')
Find a switching path (use the method given above, start with y1 and the path ends up with a node yj which is not in Y', but covered in M'')
End of the Algorithm
This algorithm takes O(nm+n^2) time

Another method: Reduce this matching problem to a maximum flow problem. It is very similar to the normal reduction from bipartite matching to max flow problem, except that you have to add a lower bound 1 on the edge from any node y in Y' to t(sink).

Problem 6 Chapter 11 – 1
(a) A set of weights: 1, 1, 2, 2; K is 3. The minimum number of trucks needed is 2 but by the greedy algorithm, 3 trucks is assigned.
(b) First, we give a lower bound for the optimal loading solution

OPT number of trucks $\geq \left\lceil (\sum_{i=1..n} w_i) / K \right\rceil = \lceil W / K \rceil$.

For the greedy algorithm, we assume there are total m trucks are needed. $w_1, ..., w_{i1}$ are assigned to the first truck; and $w_{i1+1}, ..., w_{i2}$ assigned to the second truck; ……; $w_{i(m-1)+1}, ..., w_n$ assigned to the mth truck. From the procedure of the greedy algorithm we

know that $w_1 + ... + w_{i1} + w_{i1+1} > K$; $w_{i1+1} + ... + w_{i2} + w_{i2+1} > K$ …… and

$w_{i(m-2)+1} + ... + w_{i(m-1)} + w_{i(m-1)+1} > K$. If we add all the equations together, we get

=> $\sum_{i=1..n} w_i + \sum_{x=1,...,(m-1)} w_{ix+1} > (m-1)K$

=> Rewrite the above formula, we have $m < (\sum_{i=1..n} w_i + \sum_{x=1,...,(m-1)} w_{ix+1}) / K + 1 < 2W / K + 1$

=> So, we know the number of trucks $m \le \lceil 2W / K \rceil = 2 \cdot$ (OPT number of trucks)

Problem 7 Chapter 11 - 3

(a) A = {1, 2, 2, 7} , B = 11; The feasible sum is 11, but the algorithm gives result 5.

(b)

Algorithm

Step 1:

In set A find a subset A' = {ai | ai is in A; and the value of B>= ai >= B/2}.

Step 2:

If A' is not empty, then return any number in A'.

Else Use the algorithm given in (a): insert as many numbers as possible to T. return T.

The running time of the algorithm is O(n).

Now we prove the total sum is at lease half as large as the maximum total sum.

Case 1: A' is not empty, then the value returned is at least B/2.

Case 2: All the items in A are smaller than B/2. Then the sum of T must be bigger than B/2; otherwise a new number of A will be added to T.