CS217 - Algorithm Design and Analyis
Homework Assignment 4

Group Name: Static
Group Members: Zhenjia Xu, JiaCheng Yang
Zhuolin Yang, Wenda Qiu

16$^{\text{th}}$ Nov, 2016

4 Network Flows, Matchings, and Paths

Exercise 4.1. The problem can be easily proved by Max-flow min-cut theorem.

Proof. If there is a flow from s to r of value k, then the minimum cut between s and r must be greater or equal to k(the maximum flow must be greater or equal to k). Similarly, the minimum cut between r to t must be greater or equal to k. Assume we found a cut between s and t which were less than k, then after we removed the edges in the cut, s and r would still be connected(otherwise we get a cut whose value is less than k). Similary, r and t is still connected. Hence s and t is connected, which contradicts to the assumption that there were a cut from s to t whose value is less than k. Thus the maximum flow from s to t is greater or equal to k, which means there is a flow from s to t of value k. □

Exercise 4.2. First of all, we have a basic equivalence: $|V_1| \times d_1 = |V_2| \times d_2$.

For convenience, we can assume $|V_1| \leq |V_2|$, what we need to do is proving there exists a matching of size $|V_1|$.

Consider solving it by using Hall's theorem: After choosing i vertexes from V_1 arbitrary, there are k vertexes from V_2 which are connected directly, we need to show $k \geq i$.

The size of the rest part of V_2 equals to $|V_2| - k$, and each of them has d_2 degrees which have not been used. thus, we have:

$$d_2 \times (|V_2| - k) + i \times d_1 \leq |V_1| \times d_1$$

Transforming this equivalence by using $|V_1| \times d_1 = |V_2| \times d_2$:

$$i \times d_1 \leq k \times d_2$$

thus, we have $k \geq i \times \frac{d_1}{d_2}$.

Because we have $|V_1| \leq |V_2|$, according to basic equivalence: $|V_1| \times d_1 = |V_2| \times d_2$, we can get $d_1 \geq d_2$, which equals to $\frac{d_1}{d_2} \geq 1$, so we finally have :

$$k \geq i \times \frac{d_1}{d_2} \geq i$$

By Hall's theorem, there's a matching of size $|V_1| = min(|V_1|, |V_2|)$.

Exercise 4.3. In $H_n[L_i \bigcup L_{i+1}]$, the degree of every vertex in L_i is $n - i$ (transforming a zero in Hamming code to one), and the degree of every vertex in L_{i+1} is $i + 1$ (transforming an one in Hamming code to zero).

Using the conclusion what we have proved in Exercise 2, there exists a matching of size equals to $min(\binom{n}{i}, \binom{n}{i+1})$. when $i < n/2$, the size equals to $\binom{n}{i}$.

Exercise 4.4.

(1) Transfer vertex capacities into edge capacities:

- Divide each vertex $u \in V$ into two vertexes, denoted by u_{in} and u_{out}. Connect an edge from u_{in} to u_{out} with a capacity of c
- For each edge $\langle u, v \rangle \in E$, connect an edge from u_{out} to v_{in} with a capacity of ∞
- Calculate the maximum flow from S_{in} to T_{out}

Transfer edge capacities into vertex capacities:

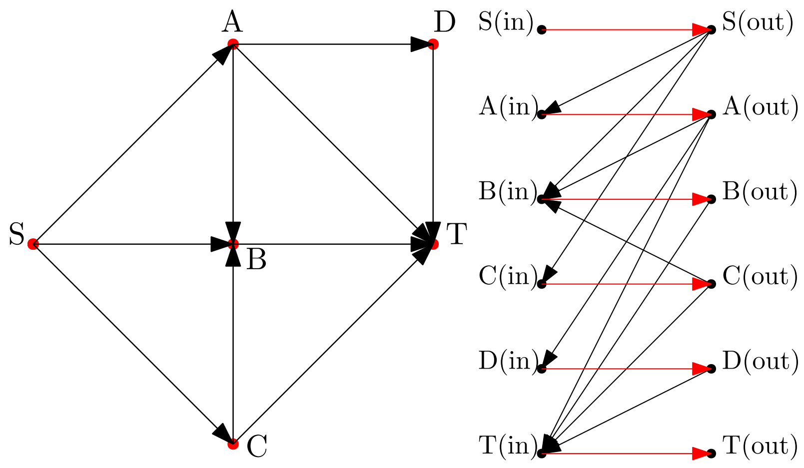
- For each vertex $u \in V$, still construct a vertex v in V' with a capacity of ∞
- For each edge $e\langle u, v, c \rangle \in E$, construct a vertex p_e with a capacity of c, and connect edges from u to p_e and from p_e to v
- Calculate the maximum flow from S to T

(2) Illustration

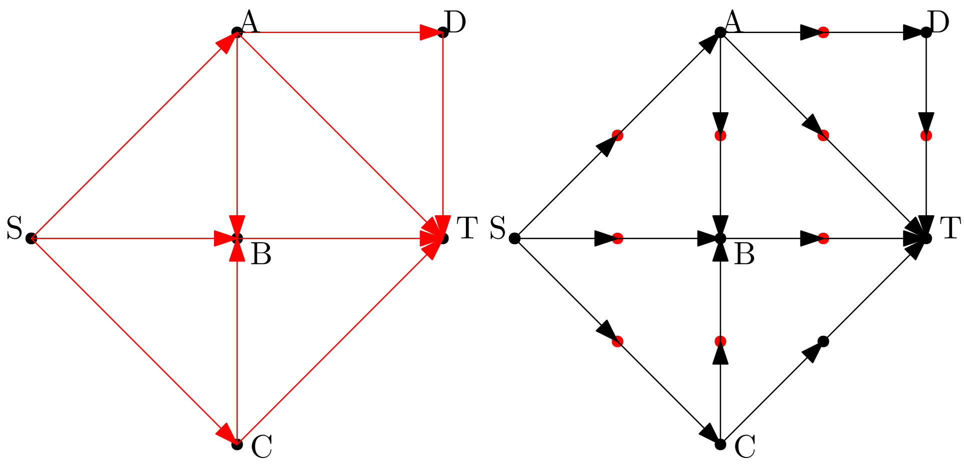
The black edge's capacity is ∞, the red edge's capacity is c

The black vertex's capacity is ∞, the red vertex's capacity is c

These two pictures show Transfering vertex capacities into edge capacities. These two pictures show Transfering edge capacities into vertex capacities.



(3) Construct a vertex capacities network $G = (V, E, c)$. The c of s and t is ∞, and for other vertics, c is 1. Calculate the maximum flow from s to t. Each flow from s to t represent a path from s to t. Because the vertex capacities are 1 (except for s, t), these paths are internally vertex disjoint. There k paths from s to t, such that the paths are internally vertex disjoint if and only if the maximum flow if no less than k.



Exercise 4.5. We will construct a network graph and use the Max-flow min-cut theorem to show the (i) and (ii).

Proof. Considering a network graph:

1. Connect every nodes in L_k to $L_{k+1}(k+1 \leq n-i)$ with capacity ∞.
2. Connect source to every node in L_i with capacity ∞.
3. Connect every nodes in L_{n-i} to sink with capacity ∞.

Then we empow each node except the source and the sink with vertex capacity 1. We will show that the minimum cut of this network graph must be of value $|L_i| = |L_{n-i}|$.

Assume the number of paths from the source to the sink is p. Then if we remove a node in L_k, according to the symmetry of the graph(each node in L_i plays equal roles in the graph), the number of paths from the source to the sink will decrease by $p/\binom{n}{k}$. If we cut the nodes $v_1, v_2, v_3, ..., v_s$ and $v_j \in L_{b_j}$, then we can decrease the paths by

$$\min\left\{p, \sum_{j=1}^{s} \frac{p}{|L_{b_j}|}\right\} \leq \min\left\{p, \sum_{j=1}^{s} \frac{p}{|L_i|}\right\} = \min\left\{p, \frac{sp}{|L_i|}\right\}$$

In other words, if we cut s nodes, we can at most decrease the paths by $\min\{p, sp/|L_i|\}$. And if $v_j \in L_i$ for every $j \in \{1, 2, ..., s\}$, then the equal sign established. Considering the cut of this graph can be exactly vertex, we can easily know that the minimum cut of this graph is minimum s satisfying

$$\frac{sp}{|L_i|} \geq p$$

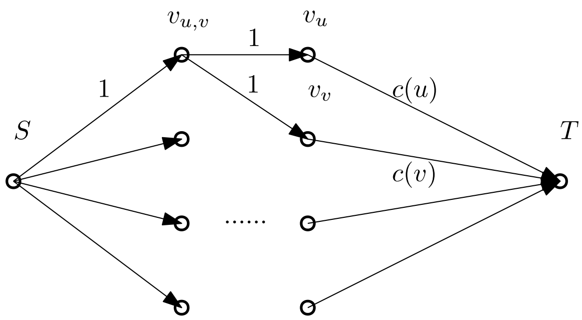
Hence the minimum cut of this graph is exactly $|L_i|$.

Hence, we know that the maximum flow of this graph is $|L_i|$, which means that there are $|L_i|$ disjoint paths(the vertex capacity being one meets the the disjoint properties). $\square$

Exercise 4.6.

Max flow

We can make nodes $v_{u,v}, \{u,v\} \in E$, $v_i, i \in V$, a source S and a sink T. Then connect edges from S to $v_{u,v}$ with capacity of 1, from $v_{u,v}$ to v_u and v_v with capacity of 1 and from v_i to T with capacity of $c(i)$. If this graph has a maximum flow of $|E|$, then the original graph has a feasible orientation.

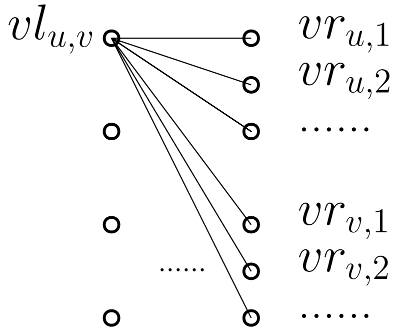


For a maximum flow S, since the flow is full and nodes $v_{u,v}$ has exactly one unit flow input, exactly one of the edges $< v_{u,v}, v_u >$ and $< v_{u,v}, v_v >$ will has a flow. We can simply orientate edge $\{u,v\}$ to $u \to v$ if $< u_{u,v}, v_v >$ has a flow, or $v \to u$ if $< u_{u,v}, v_u >$ has a flow. Thus, the in-dgree of i is just the flow through v_i, which is less than or equal to the capacity to sink, $c(i)$. So, we get a feasible orientation.

For a feasible orientation, if $\{u,v\}$ is converted to $u \to v$, we can add the flow on path $S \to v_{u,v} \to v_v \to T$ by one unit, or $S \to v_{u,v} \to v_u \to T$ when converted to $v \to u$. Since the in-degree of a node i doesn't exceed $c(i)$, so the flow from v_i to T is less than or equal to the capacity $c(i)$.

Maximum matching

We can make nodes $vl_{u,v}, \{u,v\} \in E$ as X and $vr_{i,j}, i \in V, 1 \le j \le c(i)$ as Y. Then connect $c(u)$ edges between $vl_{u,v}$ and $vr_{u,j}, 1 \le j \le c(u)$, $c(v)$ edges between $vl_{u,v}$ and $vr_{v,k}, 1 \le k \le c(v)$. If this bipartite graph has a maximum matching of size $|E|$, then the original graph has a feasible orientation.

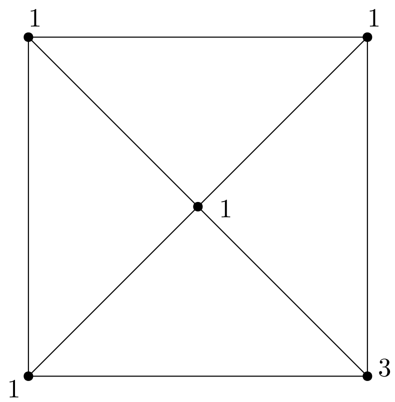


For a maximum matching S, since the size of matching is $|E|$, every node $vl_{u,v}$ has a paired node in Y, then we convert $\{u,v\}$ to $u \to v$ if the paired node is $vr_{v,i}$, or to $v \to u$ if that node is $vr_{u,i}$. Since nodes $vr_{i,j}$ in Y is no more than $c(i)$, so the in-degree of i is less or equal to $c(i)$. Then we get a feasible orientation.

For a feasible orientaion, we processe edges one by one in an arbitrary order. If $\{u,v\}$ is converted to $u \to v$, we can match $vl_{u,v}$ with $vr_{v,i}$, or $vr_{u,j}$ when converted to $v \to u$, where i and j is the smallest one that $vr_{v,i}$, $vr_{u,j}$ has not matched before. Since the in-degree of a node i doesn't exceed $c(i)$, so there is always an avalible i or j.

Exercise 4.7.

Example



The picture shows an graph without a feasible orientation. If we ignore the node in the right-bottom corner, the subgraph will has 4 nodes, 5 edges and every node has a $c(i) = 1$. According to the pigeonhole principle, there will be a node has an in-dgree greater than 1, thus the feasible orientation doesn't exsist.

Witness

If a graph doesn't has a feasible orientation, then there exists a subgraph(witness), $e \subseteq E, v = \{u|\{u,x\} \in e$ or $\{x,u\} \in e, x, u \in V\}$, where $|E| > \sum_{u \in v} c(u)$.

Prove

We have already reduced the orientation problem into a maximum mathcing in Ex.6. So we can translate the miximum mathcing witness into feasible orientation one: some set $X \subseteq U$ is an edge set $e \subseteq E$, $\tau(X)$ is the sum of $c(i)$, where node i is involved in e: $\{i,x\} \in e$ or $\{x,i\} \in e$. So the witness we described above is also the witness of maximum matching witness in the reduced problem.

Exercise 4.8. First, we can make Team 1 to win all $\sum_{i=2}^{n} m_{1,i}$ matches and recalculating the scores. what we still have to do is determining the results of other competitions and making Team 1 become the unique winner.

We use flow algorithm to solve this problem, first we create two extra vertexes S and T.

Because all matches Team 1 attends was **over**, we do not need to consider Team 1. For each pair of two other teams i and j, if there are $m_{i,j}$ matches between them, we build a edge from S to (i,j) *(pair of i, j)*, whose capacity is $m_{i,j}$. then we build two edges which from (i,j) to i and j, whose capacity is ∞. These edges are built by considering each match will increase the score of i or j by 1.

Then, each teams except Team 1, links a edge to T. The capacity of edge $i \to T$ is $K - 1 - s_i$, which K is the maximum possible score of Team 1, s_i is the current score of Team i. In case of $s_i \geq K$, Team 1 will never be the unique winner.

After working flow algorithm on this graph, we can get the maximum flow F. if F equals to the total number of the rest matches, we can construct a plan to judge every undetermined matches by observing the flow from (i, j) to i and j, otherwise there must exist a team which can get the score $s_i \geq K$, Team 1 will never be the unique winner.

Since there are many polynomial-time algorithm for dealing with Maximum-flow Problem, the solution is acceptable.