Algorithm Design and Analysis (Fall 2023)

Assignment 5

Deadline: Jan 2, 2023

Yuqing Tu 522030910152

1. (30 points) **[Menger's Theorem]** Let $G = (V, E)$ be a connected undirected graph. The *edge connectivity* of $s, t \in V$ is the minimum number of edges whose removal disconnects s and t. The *vertex connectivity* of s, t is the minimum number of graph elements (any vertex in $V \setminus \{s, t\}$ or any edge in E) whose removal disconnects s and t. Menger's Theorem states that the edge connectivity of s, t is exactly the number of edge-disjoint paths between s and t and the vertex connectivity of s, t is exactly the number of *internally* vertex-disjoint paths between s and t. (Obviously, two paths between s and t are not vertex-disjoint since they intersect at s and t; *internally* vertex disjoint means they are disjoint except for s and t.)

   Prove that Menger's Theorem is just a consequence of the max-flow-min-cut theorem. Be sure the network you construct is *directed* and *capacitated*.

   Solution:

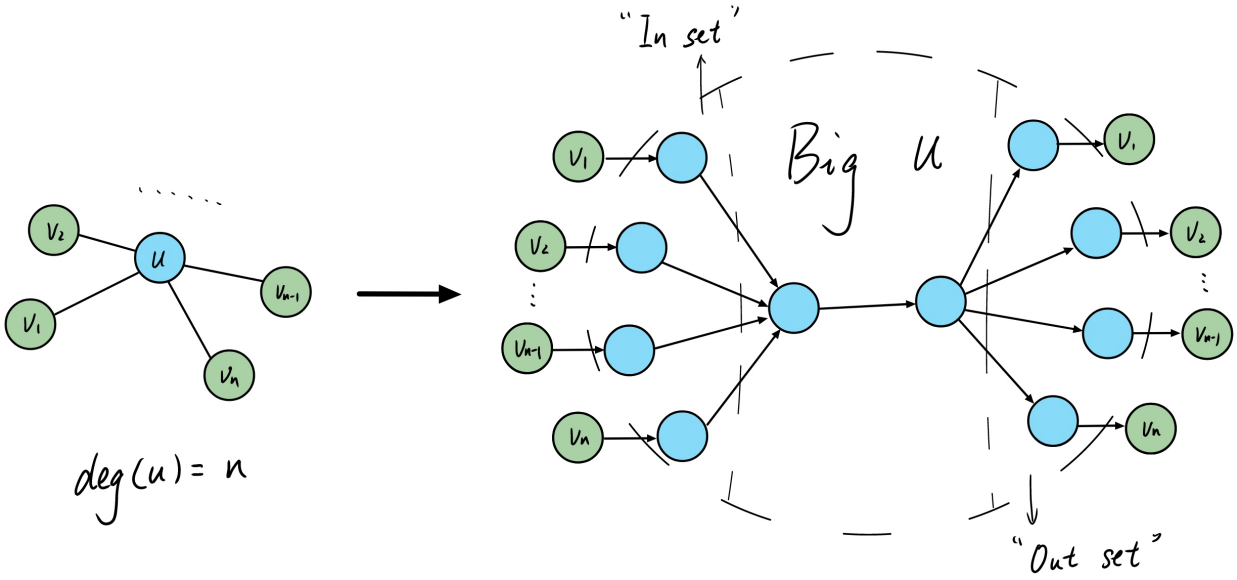
   (a) Prove that the edge connectivity of s, t is exactly the number of edge-disjoint paths between s and t:
   Let $G' = (V, E')$ be a directed graph and $E' = \{\langle u, v\rangle, \langle v, u\rangle | \forall u, v \in V\}$. The capacity of G''s all edges is 1.

   1) First, we prove that the number of edge-disjoint paths n between s and t is equal to the max-flow f in G' from s to t:
   Because each edge's capacity is 1, so there are f paths in G' that contribute 1 to the max-flow. And in these paths, no two paths share the same edge. Otherwise, the capacity of the shared edge will overflow. Therefore, there are f edge-disjoint paths between s and t, which means $n = f$.

   2) Then, we prove that the edge connectivity of s, t is equal to the min-cut between s and t:
   Because each edge's capacity is 1, so the min-cut is equal to the number of the edges in the min-cut. According to the definition, it is obviously that the edge connectivity of s, t is equal to the min-cut between s and t.

   So we have proved that a part of Menger's Theorem is just a consequence of the max-flow-min-cut theorem.

   (b) Prove that the vertex connectivity of s, t is exactly the number of internally vertex-disjoint paths between s and t:
   Let $G'' = (V, E')'$ be a directed graph. In G'', make each vertex $u \in V$ into "big vertex" as shown below.



And between $u, v \in V$, u's outgoing edges will reach to the vertex in "In set" of "big v" and u's incoming edges will reach to the vertex in "Out set" of "big v". The capacity of G'''s all edges is 1. So the maximum capacity of each "big vertex" is 1.

1) First, we prove that the number of internally vertex-disjoint paths n between s and t is equal to the max-flow f in G' from s to t:
   Because each edge's capacity is 1, so there are f paths in G' that contribute 1 to the max-flow. And in these paths, no two paths share the same vertex. Otherwise, the capacity of the shared vertex will overflow. Therefore, there are f vertex-disjoint paths between s and t, which means $n = f$.
2) Then, we prove that the vertex connectivity of s, t is equal to the min-cut between s and t:
   Because each edge's capacity is 1 and the maximum capacity of each "big vertex" is 1, so the min-cut is equal to the number of the edges and "big vertices" in the min-cut. According to the definition, it is obviously that the vertex connectivity of s, t is equal to the min-cut between s and t.

So the other part Menger's Theorem is just a consequence of the max-flow-min-cut theorem.

To sum up, Menger's Theorem is just a consequence of the max-flow-min-cut theorem.

2. (30 points) **[Perfect Matching on Bipartite Graph]** A graph is *regular* if every vertex has the same degree. Let $G = (A, B, E)$ be a regular bipartite graph with $|E| > 0$.

   (a) (10 points) Prove that $|A| = |B|$.

   (b) (15 points) Let $n = |A| = |B|$. Can we conclude that G must contain a matching of size n? If so, prove it; if not, provide a counterexample.

   Proof:

   (a) In a bipartite graph, we have $\sum_{u \in A} deg(u) = \sum_{v \in B} deg(v)$. It is because that the end vertices u, v of every edge e in G can't both in A or B. So each edge will contribute 1 to both A's sum of degree and B's sum of degree. And because every vertex has the same degree k, so

$$\frac{\sum_{u \in A} deg(u)}{k} = \frac{\sum_{v \in B} deg(v)}{k}$$

$$|A| = |B|$$

   (b) Yes, we can!
   Let

$$G' = (V, E')$$

$$V = A + B + \{s, t\}$$

$$E' = \{\langle s, u\rangle | \forall u \in A\} + \{\langle v, t\rangle | \forall v \in B\} + \{\langle u, v\rangle | \forall u \in A, \forall v \in B\}$$

   The capacity of $e_1 \in \{\langle s, u\rangle | \forall u \in A\} + \{\langle v, t\rangle | \forall v \in B\}$ is 1 and the capacity of $e_2 \in \{\langle u, v\rangle | \forall u \in A, \forall v \in B\}$ is ∞.
   To prove it, we just need to prove that the max-flow from s to t of G' is n.
   For every vertex $u \in A$, it gets a input flow of size 1 from s. And we divide it into k output flows of size $\frac{1}{k}$. Each output flow f_i will be the input of $v_i \in B$. So for every vertex $v \in B$, it gets k input flows of size $\frac{1}{k}$, then there will be a output flow of size 1 from v to t. Finally, t will get n input flows of size 1, so the flow from s to t is n. And we know that the flow's upper bound from s to t is n, so the max-flow from s to t of G' is n. Then the proposition is proved.

3. (40 points) **[König-Egerváry Theorem]** In the class, we have seen that the maximum matching problem can be formulated by the following linear program

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{e:e=(u,v)} x_e \leq 1 && (\forall v \in V) \\ & x_e \geq 0 && (\forall e \in E) \end{aligned}$$

and the minimum vertex cover problem can be formulated by the following linear program

$$\begin{aligned} \text{minimize} \quad & \sum_{u \in V} x_u \\ \text{subject to} \quad & x_u + x_v \geq 1 && (\forall (u,v) \in E) \\ & x_u \geq 0 && (\forall u \in V) \end{aligned}$$

We have also seen that the second linear program is the dual program of the first.

(a) (20 points) Prove that both linear programs have integral optimal solutions if the graph is bipartite.

(b) (10 points) Using the result in the first part, prove König-Egerváry Theorem, which states that the size of the maximum matching equals to the size of the minimum vertex cover in a bipartite graph.

(c) (10 points) Provide a counterexample showing that the claim fails for non-bipartite graphs.

Solution:

(a) The primal linear program can be expressed as

$$\begin{aligned} \text{maximize} \quad & c^T x_e \\ \text{subject to} \quad & A x_e \leq b \\ & x_e \succcurlyeq 0 \end{aligned}$$

and the dual program can be expressed as

$$\begin{aligned} \text{maximize} \quad & b^T x_v \\ \text{subject to} \quad & A^T x_v \leq c \\ & x_v \succcurlyeq 0 \end{aligned}$$

Where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1|E|} \\ a_{21} & a_{22} & \cdots & a_{2|E|} \\ \cdots & \cdots & \cdots & \cdots \\ a_{|V|1} & a_{|V|2} & \cdots & a_{|V||E|} \end{bmatrix}$$

(if the vertex i is one of the end of edge j, $a_{ij} = 1$, else $a_{ij} = 0$), $b = [1\ 1\ \cdots\ 1]$ has columns $|v|$, $c = [1\ 1\ \cdots\ 1]$ has columns $|E|$, $x_e = [x_{e1}\ x_{e1}\ \cdots\ x_{e|E|}]$ and $x_v = [x_{v1}\ x_{v2}\ \cdots\ x_{v|V|}]$.

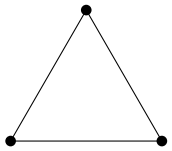
To prove both linear programs have integral optimal solutions, we just need to prove A is a totally unimodular matrix.

Prove by induction:

- Base Step: Each cell of a belongs to $\{0, 1\}$
- Inductive Step: Suppose every $k \times k$ submatrix of A has determinant belongs to $\{0, -1, 1\}$. Consider any $(k+1) \times (k+1)$ submatrix A'
- Case 1: If a column of A' is all-zero, then $det(Z') = 0$
- Case 2: If a column of A' contains only one 1 entry, then $det(A') = \pm 1$ times the determinant of a $k \times k$ submatrix. $det(A') \in \{0, -1, 1\}$ by induction hypothesis
- Case 3: If every column of A' has two 1 entries, then $k+1$ must be an even number. It is because if the subgraph of this bipartite graph is a cycle, it must an even cycle. Assume $c_1, c_2, \cdots, c_{k+1}$ is all the columns of A' that their order is the order in which they are traversed through the even cycle(each column corresponds to an edge). It is obviously that $\sum_{i=1}^{k+1}(-1)^i c_i$ is a zero vector. So $det(A') = 0$
- So the determinant of any submatrix of A belongs to $\{0, -1, 1\}$, which means A is a totally unimodular matrix.

So both linear programs have integral optimal solutions.

(b) Because both linear programs have integral optimal solutions if the graph is bipartite, so we can solve linear programs in order to get the size of the maximum matching and the size of the minimum vertex cover in a bipartite graph. Then by Strong Duality Theorem, $Primal\ OPT = Dual\ OPT$, so the size of the maximum matching equals to the size of the minimum vertex cover in a bipartite graph.

(c) 

4. (Bonus 60 points) **[Dinic's Algorithm]** In this question, we are going to work out *Dinic's algorithm* for computing a maximum flow. Similar to Edmonds-Karp algorithm, in each iteration of Dinic's algorithm, we update the flow f by increasing its value and then update the residual network G^f. However, in Dinic's algorithm, we push flow along *multiple* s-t paths in the residual network instead of a *single* s-t path as it is in Edmonds-Karp algorithm.

   In each iteration of the algorithm, we find the *level graph* of the residual network G^f. Given a graph G with a source vertex s, its level graph $\overline{G}$ is defined by removing edges from G such that only edges pointing from level i to level $i+1$ are kept, where vertices at level i are those vertices at distance i from the source s. An example of level graph is shown in Fig. 1.

   Next, the algorithm finds a *blocking flow* on the level graph $\overline{G}^f$ of the residual network G^f. A blocking flow f in G is a flow such that each s-t path contains at least one *critical edge*. Recall that an edge e is *critical* if the amount of flow on it reaches its capacity: $f(e) = c(e)$. Fig. 2 gives examples for blocking flow.

   Dinic's algorithm is then described as follows.

   1. Initialize f to be the empty flow, and $G^f = G$.
   2. Do the following until there is no s-t path in G^f:
      - construct the level graph $\overline{G}^f$ of G^f.
      - find a blocking flow on $\overline{G}^f$.
      - Update f by adding the blocking flow to it, and update G^f.

   Complete the analysis of Dinic's algorithm by solving the following questions.

   (a) (15 points) Prove that, after each iteration of Dinic's algorithm, the distance from s to t in G^f is increased by at least 1.

   (b) (15 points) Design an $O(|V| \cdot |E|)$ time algorithm to compute a blocking flow on a level graph.

   (c) (10 points) Show that the overall time complexity for Dinic's algorithm is $O(|V|^2 \cdot |E|)$.

   (d) (20 points) **(challenging)** We have seen in the class that the problem of finding a maximum matching on a bipartite graph can be converted to the maximum flow problem. Show that Dinic's algorithm applied to finding a maximum matching on a bipartite graph only requires time complexity $O(|E| \cdot \sqrt{|V|})$.

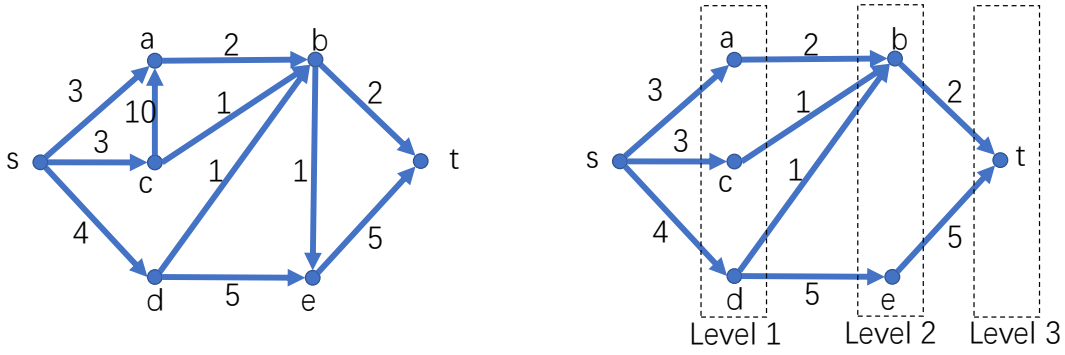


Figure 1: The graph shown on the right-hand side is the level graph of the graph on the left-hand side. Only edges pointing to the next levels are kept. For example, the edges (c, a) and (b, e) are removed, as they point at vertices at the same level. If there were edges pointing at previous levels, they should also be removed.

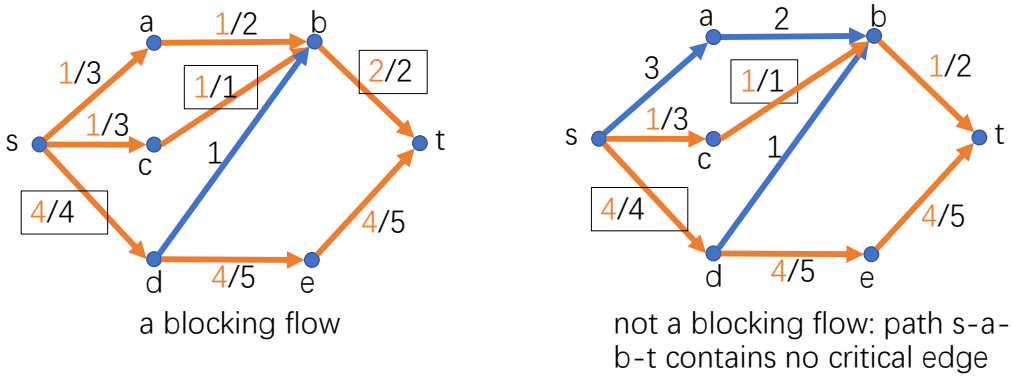


Figure 2: Blocking flow examples

Solution:

(a) Assume that in this iteration, the distance from s to t is n. Because in each iteration, we find a blocking flow on $\overline{G}^f$, which means that each s-t path contains at least one critical edge. So when we update G^f, these s-t path will no longer exist. In next iteration, if

1. there is no s-t path on $\overline{G}^f$, then the distance from s to t is ∞, obviously increasing more than 1.
2. there is some new s-t path on $\overline{G}^f$, then these paths' distance mustn't be n because all s-t paths with n distance were eliminated in the last update. These paths' distance are also not smaller than n because in the last update, each new edge is added from high level to low level, so these new edges won't do any help to reduce the distance from s to t. To sum up, the the distance from s to t must be bigger than n and it is an integer, so the distance from s to t is increased by at least 1.

In conclusion, the distance from s to t in G^f is increased by at least 1.

(b) **Algorithm 1** compute_blocking_flow($\overline{G}^f$)

```
1: f ← 0
2: while s has at least one edge do
3:    do DFS from s
4:    if reach to t then
5:       eliminate the critical edge whose capacity is c_min
6:       f ← f + c_min
7:    else
8:       eliminate all the edges of the end vertex
9:    end if
10: end while
11: return f
```

Time complexity:
In each iteration, if we can reach to t, then we will do $c_i \leftarrow c_{min}$ for all capacities c_i, and at least one edge will be eliminated. Otherwise we will eliminate all the edges of the end vertex. So we will eliminate at least one edge, the number of iteration is at most $|E|$. And in each iteration, we at most spend $O(|V|)$ because we don't need to turn back. To sum up, the total time complexity is $O(|V| \cdot |E|)$.

(c) Because the distance from s to t in G^f is increased by at least 1 after each iteration and the longest distance from s to t is $|V| - 1$, so the number of iteration is at most

$|V| - 1$. And the time complexity of computing a blocking flow is $O(|V| \cdot |E|)$, so the total time complexity of Dinic's algorithm is $O(|V|^2 \cdot |E|)$.

(d) Let s connect with all vertices in L and each new edge is from s to $l_i \in L$. Let t connect with all vertices in R and each new edge is from $r_i \in R$ to t. Let all edges between L and R from $l_i \in L$ to $r_j \in R$. Then let the capacity of all edges be 1. So we can do a special Dinic's algorithm to find a maximum matching in $O(|E| \cdot \sqrt{|V|})$. The special part is finding a blocking flow:

- If we reach to t, we eliminate all the edges in this path because every edge is critical.
- If we reach to an end but not t, we eliminate the last edge we travelled, then we turn back and continue do DFS. It is because we don't travel the same edge twice.

So we only travel each side once, the time complexity of each finding a blocking flow is $O(|E|)$. Then we need to find the number of iteration of this special Dinic's algorithm.

- If the number of iteration is at most $\sqrt{|V|}$, then we have the total time complexity is $O(|E| \cdot \sqrt{|V|})$.
- Else, the number of iteration is more than $\sqrt{|V|}$, so each path in the maximum flow has length at least $\sqrt{|V|}$. It is because in each iteration, the distance between s and t is increased at lease 1 and we have more than $\sqrt{|V|}$ iterations. Then there are at most $\frac{|V|}{\sqrt{|V|}} = \sqrt{|V|}$ paths in the maximum flow. So this algorithm will end within at most another $\sqrt{|V|}$ iterations, the total number of iterations is least than $2\sqrt{|V|}$. So the time complexity is still $O(|E| \cdot \sqrt{|V|})$.

The correctness is because the maximum flow can be transformed into the maximum matching.

Reference:
Mr. Tao's AI2615 slide of previous years.

5. How long does it take you to finish the assignment (including thinking and discussion)? Give a score (1,2,3,4,5) to the difficulty. Do you have any collaborators? Please write down their names here.

   It takes me about 7-8 hours to finish it. I will give 4 to the difficulty. My collaborator is Zhou Zheli, his student ID is 522030910150.