Part I

Homework

[26.1.1]

We will show that each flow in G' is mapping to a flow of same value in G and vice versa. Thus the maximum flow in G' has the same value as a maximum flow in G.

For a flow in G, consider the flow f, in which there exist antiparallel edge $v_i -> v_j$ and $v_j -> v_i$. Suppose the flow on $v_i -> v_j$ is F and F' in another direction. The flow f' in G' has same flow in each edge except that $v_i -> v_j$ is replaced by $v_i -> v'$ and $v' -> v_j$ and the flow is F. It is obvious that Capacity constraint and Flow conservation meets. The value of f' is

$$|f'| = \sum_{v \in V'} f'(s, v) - \sum v \in V' f'(v, s)$$

. If there is no antiparallel edges incident with s, the value is same. Otherwise, $f(s, v) = f'(s, v')$ the value is still the same.

[26.1.2]

Deleting the source and the sink, we get a unique multiple-source, multiple-sink flow network.

For the other side, for a multiple-source, multiple-sink flow network, we add two kinds of edges $s \to s_i$ and $t \to t_i$. We will prove the flow on each kind of each is unique.

For the first kind, by flow conservation , $f(s, s_i) = \sum f(s_i, v)$, where v is vertices incident with s_i except s. Thus the value is unique. The same proof applied for the second kind.

Thus we have proved the problem.

[26.1.6]

Build the graph in which corners are vertices and edges (u,v) and (v,u) are added if there is a road connecting them. Constrain the edge to have capacity 1, and the problem is reverted to find a maximum flow and check if it can be 2 and with integer flow value in each edge.

However, this leads to a problem. The two may take a road in different direction, namely one pass (u,v) and the other (v,u). A direct algorithm is to exhaust all possible direction of each edge and run the above procedure, namely for one road connecting u, v, adding edge (u,v) in one time and (v,u) in another. However, this costs a lot of time, I wonder if there is a better way.

[26.1.7]

Build network G' in the following way: for a vertex $v \in G$ with vertex constrant C,

add two vertices v_1, v_2 into G', all edge connected from u to v will lead to edge $u \to v_1$ in G' and all edge connected from v to u will lead to edge $v_2 \to u$ in G'. Two edges $(v_1, v_2), (v_2, v_1)$ with capacity C is also added, which has flow $\sum_u f(u, v) - \sum_u f(v, u)$. In this way, the constraint of edge between v_1 and v_2 serves as constraint of v.

[26.2.2]

The flow is $11 + 1 - 4 + 7 + 4 = 19$, the capacity of this cut is $16 + 4 + 7 + 4 = 31$

[26.2.6]

We construct a new flow network by adding a super source s with edge (s, s_i) of capacity p_i and a super sink t with edge (t_i, t) of capacity q_i. We will prove the original problem is to find the maximum flow in the new network and check whether it can be sum_{p_i}

First, by conservation of flow in s_i, we have $\sum_{v \in V} f(s_i, v) = f(s, s_i) = p_i$. The same proof applies for each t_i. Sum over all flow in and out of $V - \{s, \}$, we get $\sum_i f(s, s_i) = \sum_i f_{t_i,j}$. In the case we find the maximum flow in the new network and check whether it can be sum_{p_i}, we can have $\sum_i p_i = \sum_j q_j$. So the convertion is right.

[26.2.8]

To prove this problem, we will prove that in the while loop, if we find the path p from s to t will never contain edge (v, s). Otherwise, since the path begins at s and reaches s again, thus the path is not a simple path which contradict the definition of augment path.

Since (v, s) will not appear in any augment path, the path finded each time by Ford-Fulkerson method remains the same. Thus this procedure is not influenced and can get the right result.

[26.2.10]

Suppose we already find a maximum flow f. Make a new network where the capacity of each edge equals the flow in the original network. We only need to prove by $|E|$ times of finding augment path we can find the maximum flow of the new network. Note that the maximum flow of the new network is full on every edge.

Run Ford-Fulkerson algorithm on the new network with one modification that each time when we find the minimal capacity of n argument path, instead of update the corresponding edges in residual network, we just delete (u,v). This will not influence the augment path finded afterwards for it is has the same value of the final flow. Otherwise, there is an augment path contain (v,u), then another augment path containing (u,v) must exist. Denote the first as $s \to v \to u \to t$ and the other $s \to u \to v \to t$, we can construct

two augment path $s \to t$ without traversing (u,v).

Since each time an edge is deleted, at most $|E|$ augment path can be finded. In this way, we can get at most $|E|$ augment path to get a maximum flow.

[26.2.12]

Since there is an edge (v,s) that f(v,s)=1, we hope to find a circle from s to s that there is flow on each edge. Since the capacity is all integer, if $f(v,u) > 0$ the n$f(v,u) \geqslant 1$. Thus by deleting 1 from flow on the edges of the circle, we get a flow f' with the same value of f. In a flow, each vertex is connected with s, so is v. Thus there exists circle $s \sim v \to s$.

To get f', we use a deep-first search on f starting at s. When we find this circle, we can perform above operations to get the desired flow f'.

[26.2.13]

Add $\frac{1}{2|E|}$ to each edge to form a network G'. To prove the minimal cut in G' is the minimal cut in G with the minimal edges, we will prove that the minimal cut in G' has corresponding minimal value of cut in G and in all minimal value cut it contains the least edges.

For the first thing, for all the cut $f(S,T)$ in G, the corresponding cut in G' is $f(S,T)+\frac{1}{2|E|}E(S,T) < f(S,T)+1$. Since the flow is integer, two flow with different value differ at least one. Thus the minimal one remains minimal.

For the second thing, the minimal cut in G' has minimal value $f(S,T)+\frac{1}{2|E|}E(S,T)$. Since $f(S,T)$ is the same for minimal cut in G, thus E(S,T) is minimal, namely it contains minimal edges in G.

[26.3.3]

The augment path is a simple path. Since the graph is a partite graph, the path must go like: $s \to l_1 \to r_1 \cdots \to t$. Thus the over bounding of the augment path is $2\min(V(L),V(R))+1$.

[26-1]

(b) Construct the flow network as follows: add super source s and edges connecting s to all initial points. If two node in the grid can be reached from each other, then add edges in the flow. Finally, add edges from boundary point to super sink t. All edges and vertices have unit capacity. Then the problem converted to determine whether the maximum flow can be m.

With the relabeled-to-front algorithm, the running time is $O(V^3) = O(n^6)$.

[26-2]

(a). Construct the network G' as suggested and let all edges have capacity 1. This network solves a matching problem and we have the partite graph X, Y. We can construct a path cover from a flow as follows:

- select a node not in path unless all nodes are in the path.
- if there is no $f(x_i, y_j) > 0$, put it into path cover as a path of length.
- else put x_i, y_j into path, and select x_j back to step two.

In this way, we construct a path for each flow. Actually, each edge in the path cover is an edge in the matching of the partite network. Thus we get $|V| = \sum_{p \in P} v(p) = \sum_{p \in P} 1 + e(p) = P + \sum_p e(p) = |P| + M = |P| + |f|$(Note the graph is DAG). When $|f|$ is maximal, we will get the minimal path cover.

(b) If the graph contains circle, the deduction $\sum_{p \in P} v(p) = \sum_{p \in P} 1 + e(p)$ will not hold.

An counter-example is graph with $E = \{(1,2),(2,3),(3,4),(4,1)\}$. With our algorithm, (x_1, y_2) and (x_4, y_1) can be either chosen. If the latter one is chosed, then a false answer of 2 is generated. Thus in this case we cannot guarantee the algorithm to get the right answer.

0.0.1 中文测试

这是一段中文测试。