Lab05-Path & Flow

CS2308-Algorithm and Complexity, Xiaofeng Gao, Spring 2023.

∗ Contact TA Jiasen Li for any questions. Include both your .pdf and .tex files in the uploaded .rar or .zip file.

∗ Name:张祥东 Time Spent on this Lab:3-4 h

1. (Johnson's Algorithm & Bellman-Ford) Johnson's Algorithm designs a potential function on each vertex to convert each edge weight non-negative. A choice of the potential function of vertex v is the distance from a special vertex s to v derived by Bellman-Ford. Discuss whether s can be a vertex from the graph or has to be an extra virtual vertex. Give the pseudo-code of Johnson's Algorithm.

Solution. s can be a vertex from the graph. Here comes the proof.

Take the potential function as

$$h : V \to R$$

The reweighted edge's weight

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$$

We need $\hat{w}(u, v) \geq 0$,namely

$$w(u, v) + h(u) - h(v) \geq 0$$
$$h(v) - h(u) \leq w(u, v)$$
$$h(v) \leq h(u) + w(u, v)$$

Because after figuring out the $distance(s, u)$ value for every $u \in V$,we've already known

$$distance(s, v) \leq w(u, v) + distance(s, u)$$

Then the value of $distance(s, u)$ for each $u \in V$ can be used as the value set for h function.Thus,the choice of the potential function of vertex v can be distance(s,v) and the s can be a vertex from the graph and it doesn't have to be an extra virtual vertex.

Here comes the pseudo-code of Johnson's Algorithm.

Algorithm 1: Johnson's Algorithm

Input: G(V,E) with weight,n = —V—
Output: shortest path and its length between u,v for each u,v ∈ V if u != v

```
h ← size n and initiated with value ∞;
h[1] ← 0;
run Bellman-Ford with source index 1 with distance record array being h;
foreach edge(u, v) ∈ E do
    w(u, v) ← w(u, v) + h(u) − h(v);// reweight the edges
dist ← n x n and initiated with value ∞;
paths ← empty;//used to record the path between u and v
foreach node u ∈ V do
    dist[u][u]← 0;
    run Dijkstra algorithm with sorce index u with distance record array being dist[u] and
     simutaneously record the shortest path from u to all other nodes;
foreach path p ∈ paths do
    u ← start node of path p;
    v ← end node of path p;
    dist[u][v] ← dist[u][v] + h[u] - h[v];
return dist, path;
```

The time complexity for this algorithm is

$$O(mn + n^2 + n^2 \log n)$$

Namely

$$O(mn + n^2 \log n)$$

□

2. (Ford-Fulkerson) Given a grid graph G as shown in Figure 3. The capacities of each edge are shown as integers in the graph. Please solve the following problems regarding network flow, where s is the top left corner, and t is the bottom right corner.

   (a) Calculate the maximum s-t flow using Ford-Fulkerson algorithm. Output the flow function of each edge and also output the total flow value. You can draw the flow function on the figure directly.

   (b) Calculate the minimum cut. Give the cut value and the cut set. You can draw the cut on the figure. E.g., the value of two cut examples, illustrated as the red dash lines in Figure 3, are 18 and 27 respectively.

(a)

Solution. The total flow value is 18.The flow function of each edge is shown in the figure 1.

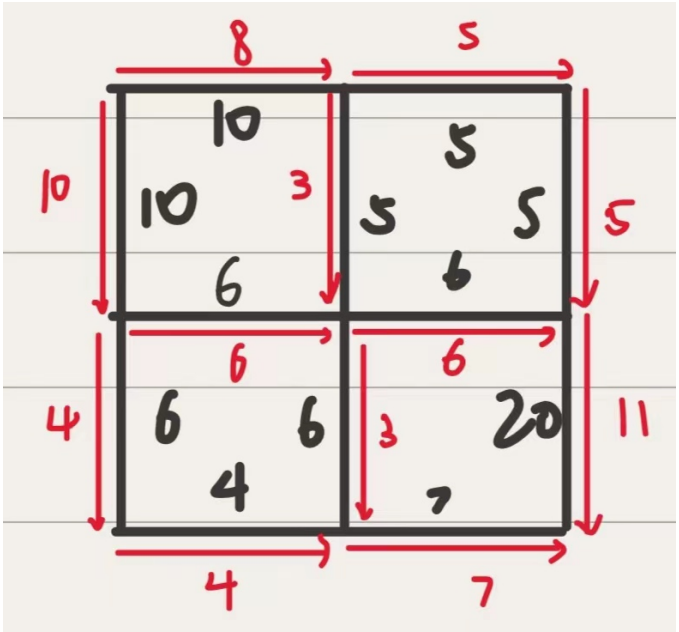

图 1: flow function

□

(b)

Solution. For the minimum cur, the cut value is 18. The cut set (A,B):

$$A = left\ top\ part\ of\ the\ following\ figure$$

$$B = right\ bottom\ part\ of\ the\ following\ figure$$

They're shown in the following figure.

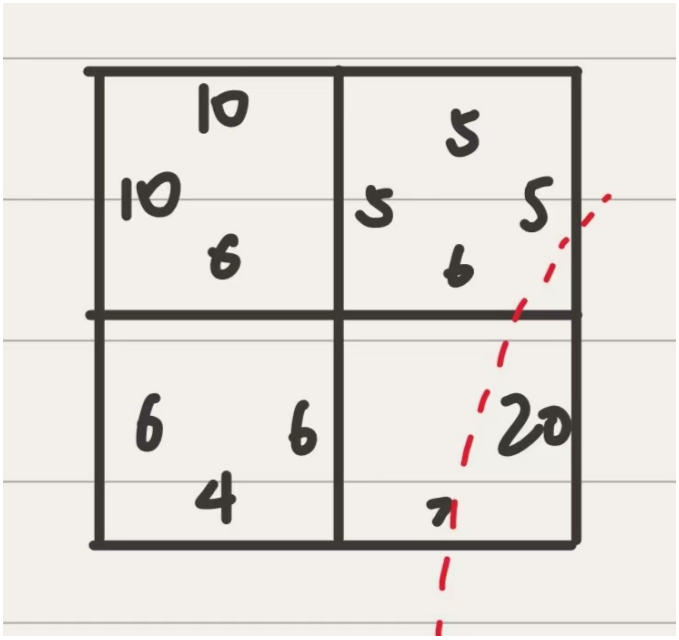

图 2: flow function

□

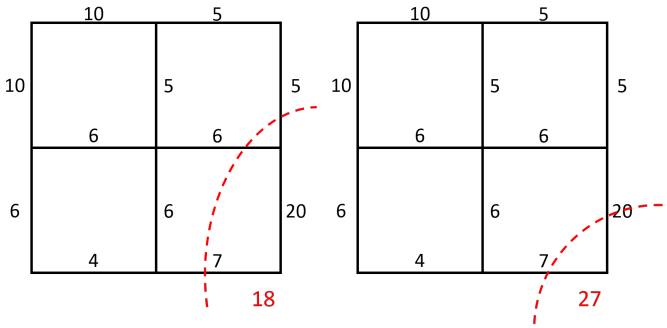

图 3: Edge capacity with two cut examples.

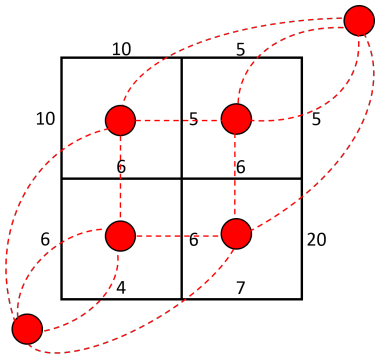

图 4: Dual graph G'

3. (Network Flow vs Shortest Path) We can also solve Question 2 by shortest path algorithm. First, construct a dual graph as shown in Figure 4. For each internal face in the primal graph G, set a vertex in the dual graph G'. Next, set a source and a destination point for the external face. The edge weights in the dual graph are the weights of edges in the primal graph where the dual edges pass through. In this way, the length of the shortest path from the left-bottom vertex to the right-top vertex in G' is the maximum flow from the left-top corner to right-bottom corner in G.

   Based on the construction above, define the maximum flow problem from the top left corner to the bottom right corner for an $n \times n$ grid. Design a solution based on Dijkstra algorithm and analyze the time complexity.

Definition 1. *define the maximum flow problem's flow value:*

after constructing a dual graph,the $v(f)$ in the previous graph = the shortest path's length between two external vertices.

By constructing a dual graph of the nxn grid, we call two external vertices s and t and it's obvious that every path from s to t can form a $s-t$ cut in the nxn grid.And all cut in the nxn grid can be found by enumerate the path in the dual graph.And the capacity of this cut equals to the length of path due to the construction way of the dual graph.

So to find the minimum cut capacity,we only to find the shortest path between s and t.From the Max-flow Min-cut Therorem,we know when f is a max flow,the $v(f) = min\ cap(A, B)$.

We run Dijkstra algorithm in the dual graph and get the min cut capacity being 18,so the flow value $v(f)$ is 18. And the s-t cut set is shown in the figure 2.

Here comes the pseudo code that gives the max flow value and the cut set.

Algorithm 2: Dijkstra algorithm figures out max flow value and s-t cut

Input: nxn grid graph G
Output: the max flow value and the s-t cut (A,B)

$G^{'} \leftarrow$ building a dual graph of nxn grid graph G using the strategy of the description in the question state.
$s, t \leftarrow$ two external vertices' number in dual graph.
$flowValue \leftarrow$ using to record v(f);
run Dijkstra algorithm in $G^{'}$ record the shortest path length in $flowValue$ and use a array $cameFrom$ recording previous in the shortest path using the s as the start node.
$SET \leftarrow$ empty edge set.
$SET \leftarrow SET \cap t$ **while** *cameFrom[t] is assigned* **do**
    $SET \leftarrow SET \cap t$;
    $t \leftarrow cameFrom[t]$;
using the SET find the edges in the shortest path and find the previous edges in the nxn graph and store them in a new variable **edgeSet**.
$G \leftarrow$ G(V,E\edgeSet);
$A \leftarrow$ the nodes that can be reached from s in the nxn grid.
$B \leftarrow$ the nodes that can be reached from t in the nxn grid.
return flowValue,(A,B);

Using the above algorithm,we can figure out the maximum flow value and the minimum s-t cut.

As for the time complexity,the transfer process is $O(n^2)$ and the dual graph has $O(n^2)$ points and $O(n^2)$ edges,the time complexity for Dijkstra algorithm is $O(n^2 \log(n^2))$.The tracing back process is $O(n)$.

The overall time complexity is

$$O(n^2 \log n)$$

4. (Network Flow Application) Given a DAG $G(V, E)$ with $n+1$ vertices $v_0, v_1, \cdots, v_n$. Assume G has been sorted topologically such that each edge (v_i, v_j) of G satisfies $i < j$. You should give several paths starting from v_0 to cover $v_1, v_2, \cdots, v_n$ exactly once. Convert this problem into a network flow problem and prove its correctness.

   (Hint: split each vertex into two and form a bipartite graph)

Solution. we can split each vertice $v \in \{v_1, v_2, \ldots, v_n\}$ into to vertices,one being in-node,the other being out-node.We call in-node $v_i^{'}$and call out-node v_i.We create k v_0.k is the out degree of v_0.Call them $v_0^1, \ldots, v_0^k$. And we create a source node s and a destination node t.Then here comes the edge connection stage.We link s with $v_0, v_1, \ldots, v_n, v_0^1, \ldots, v_0^k$ separately with one edge whose capacity is 1.We link the $v_1^{'}, v_2^{'}, \ldots, v_n^{'}$ to the t separately with one edge whose capacity is 1. And if there is an edge (v_i, v_j) in the previous graph, we link the $v_i, v_j^{'}$ with one edge whose capacity is 1.

To solve this question, we only need to run maximum flow algorithm in this graph.**If the maximum flow is n,the question has a solution.**

To find these path,we need to start from v_0^i,checking whether the out-edge's flow is 1.If it is,we can find a corresponding node $v_m^{'}$ which is a in-node,then we can visit the corresponding out-node v_m to find any edge whose flow is 1,and keep this finding process utill there is no edge whose flow is 1.Record this sequence,and it is exactly a path that we need.

To find all path,we only need to visit every v_0^i and do the above operation.Deleting the corresponding edges of the found path in the graph is needed. Then the several paths we get from the above methods are the answer that we want.

If the maximum flow is not n,simply output *no solution*.

Here comes the proof of my algorithm's correctness.

At first,we prove that if the maximum flow is n,then every node in $\{v_1, \ldots, v_n\}$ is covered exactly once.Exactly,in this question we solve a maximum graph match problem by using network flow algorithm.When we get the maximum flow is n,it means every in-node has a out-node and only one out-node.So each of them is only in one path,namely being covered exactly once.

Next,we prove that every path in this graph starts from v_0^i.We prove it by contradiction.

Assume there exist a path doesn't start from v_0^i,but starts from *index* m.Then,there is no flow to in-node v_m.Because if so,the previous node can be found and m is not a start node.However,we know that for every in-node,there is a out-node flows 1 to it.So,it's a contradiction.

So every path in this graph found by us should start from v_0^i.

And from the previous contradiction proof process,it's obvious if the maximum flow is n,every in-node has a out-node which flows to it.So by recursively doing this,every in-node can be contained in some path starting from v_0^i.

Besides,if the maximum flow is not n,then there exist a node that doesn't has out-node which flows to it.So it cannot be covered by paths which start from v_0^i.We need to output no solution in this case.

Above all,we only need to find paths starting from v_0^i and these paths can cover all nodes $v_1, v_2, \ldots, v_n$ exactly once.

Thus,the proof is completed.

□