SUN Yilin
520030910361

Homework 2

Algorithms
March 27, 2022

1

1.1

By the hint, for our guessed ratio r, we can construct another graph by the following method. We construct a directed graph G' which has exacty the same vertices and edges as G. The weight of edges in G' is given by $w_{uv} = rc_{uv} - p_v$ where r is the ratio we guessed and c_{uv}, p_{uv} is what the problem has defined. Then we run Bellman-Ford Algorithm on G' to check if there is a negative cycle. We claim that if there is a negative cycle in G' then $r < r^*$. If not then $r > r^*$.
The reason why our claim is true is that if there is a negative cycle, denoted by C', in G', then

$$\sum_{(u,v)\in C'} w_{uv} = \sum_{(u,v)\in C'} rc_{uv} - p_v < 0$$

or equavilently

$$\frac{\sum_{(u,v)\in C'} p_v}{\sum_{(u,v)\in C'} c_{uv}} > r$$

This means that there exists a cycle C which has exacty the same edges and vertices as C' in G. The ratio of C is larger than r. So $r < r^*$ must hold. Similarly, if there is not a negative cycle, then our r must be larger than any ratio of cycles in G.

1.2

By definition of R we can see that the optimal ratio r^* is at most R. So we can do a binary search in the real interval $(0, R)$ until we find an $r(C)$ with desired accuracy.

Algorithm 1 Find Optimal Ratio

```
1: procedure FIND OPTIMAL RATIO(G, ϵ, min, max):
2:     mid ← (min + max)/2
3:     Compute G'                                   ▷ G' is described in part 1.1
4:     C ← Bellman-Ford(G')          ▷ BF algorithm to calculate a negative cycle in G'
5:     if C is empty then return Find Optimal Ratio(G, ϵ, min, mid)  ▷ No negative cycle
6:     Calculate r(C) in G
7:     if r(C) ≥ r* − ϵ then return C              ▷ C is the good-enough cycle we want
8:     return Find Optimal Ratio(G, ϵ, mid, max)
```

We run FindOptimalRatio$(G, \epsilon, 0, R)$ and this will give us a good-enough cycle. The correctness lies in that by our analysis above, our algorithm cut the interval where we search the

result in half each time. So after finite times of search we guarantee that this algorithm will provide us a good-enough answer.
As for running time, since we are searching in an intercal with length R and using Bellman-Form Algorithm to find a negative cycle, we can say

$$T(R) = T(\frac{R}{2}) + O(|V||E|) \tag{1}$$
$$\leq T(\frac{R}{2}) + c|V||E| \tag{2}$$
$$\leq T(\frac{R}{4}) + 2c|V||E| \tag{3}$$
$$\leq \dots \tag{4}$$
$$\leq T(1) + c|V||E| \log_2 \frac{R}{\epsilon} \tag{5}$$

So the time complexity of our algorithm is $O(|V||E| \log_2 \frac{R}{\epsilon})$. If $|E|$ not provided, we can loose the complexity to $O(|V|^3 \log_2 \frac{R}{\epsilon})$.

2

2.1

Sufficiency:
We prove by showing that how an Eulerian circuit is constructed.
We pick a vertex u randomly. Since our graph is strongly connected, there is a cycle starting from and ending with u. Denote the cycle by C_0 and delete all the edges and the vertices with zero degree to get a new graph $G_1 = (V_1, E_1)$. Its easy to see that in G_1 the condition that the in-degree and out-degree of all vertices are equal still holds. Now if $\deg(u) \neq 0$, we can find another cycle denoted by C_1 by the same process. We carry out the provedure until vertex u is deleted. Now we pick another vertex and carry out the same process again, finally $\cup_{i=0}^{n} C_i$ is en Eulerian circuit.
Necessity:
We prove by contradiction.
If there is a vertex denoted by u where the in and out degree are not equal, it is clear that there exists at least one edge from or to vertex u that is not in any cycle of graph G. So there will not be an Eulerian circuit in the graph. So the in and out degree of any vertex must be the same.
The **sufficient and necessary condition** is that exactly one vertex u has $\deg_{in}(u) = \deg_{out}(u) + 1$ and exactly one vertex v has $\deg_{in}(v) = \deg_{out}(v) - 1$. And all the other vertices must have the same in and out degree.

2.2

We first describe a tool funcion modified from DFS which can find a cycle in a strongly connected directed graph starting from any edge. And this DFS can also mark the first vertex with deg larger than 2. We then use this function to implement an algorithm.

Algorithm 2 Eulerian Circuit

```
1: procedure EULERIAN CIRCUIT(G):
2:     Pick a vertex u randomly
3:     Initialize C, C is empty                    ▷ C is the circuit we want
4:     while G is not empty do
5:         Pick an edge (u, v) randomly
6:         C_1 ← DFS((u, v)), update u meanwhile by the process
7:         Update G by deleting all edges in C_1
8:         C ← C ∪ C_1
9:     return C
```

Since our algorithm visits every edge exactly once, it is clear that the time complexity is $O(|E|)$.

3

3.1

Consider a simple case where $V = \{1, 2\}$, $E = \{(1, 2)\}$ in the undirected connected G. It is clear that G' contains only one edge and hence is not strongly connected.

3.2

Prove by contradiction. If there is a bridge in G denoted by (u, v), we now show that u and v are not mutually reachable in G'. If u and v are mutually reachable, then there is a cycle in G' containing u and v. Since (u, v) is a bridge in G, it must be in this cycle. Thus removing (u, v) in G will not make G unconnected. So there is no bridge in G.

3.3

We prove by showing that any two vertices u, v in G' are mutually reachable.
We are going to use a lemma from course discrete maths. This lemma says that for any

vertices u, v, w, if u is an ancestor of v and there is an edge between v, w in the corresponding DFS search tree, then either u is w's ancestor or w is u's ancestor or $u = w$.
What this lemma tells us is that any back edge will not be connecting a vertex to another branch in the DFS search tree. Namely, all the back edges in G' are connecting vertices to one of their ancestors.
Now we consider two cases.
If u, v are ancestor and descendant of each other, without loss of generality, always assume u is v's ancestor. Then v is reachable from u. Now we consider v and all of v's descendants. There exists at least one edge from v or its descendants to w. If $w = u$ or w is u's ancestor then u is reachable from v and we are done. If not, i.e. none of v or its descendants has a back edge to u or u's ancestors, then it must follows that the very first edge on the path in DFS search tree from u to v is a bridge in G, which is in contradiction with our assumption that there is no bridge in G.
So in this case u, v are mutually reachable.
If u, v are not in the relationship of ancestor and descendant, then we there exists at least one common ancestor s of u and v. Then we can see s and u are mutually reachable, s and v are mutually reachable, thus u and v are mutually reachable. So in this case u, v are also mutually reachable to each other.
So G' must be strongly connected.

3.4

We describe the procedure by which the edges are provided.
Firstly we generate G' by a DFS process described in this problem.
Then we use Kosaraju's Algorithm taught in class which calculates the strongly connected components of our graph G'. Kosaraju's Algorithm only requires DFS twice.
Finally we scan the graph once and all the edges (u, v) where u and v belong to two different strongly connected components are the edges we want.
Since all the sub-procedures have time complexity $O(|V| + |E|)$, the total time complexity of our algorithm is also $O(|V| + |E|)$.

4

4.1

No. Consider a simple graph G where $V = \{1, 2, 3\}$ and $E = \{(1 \to 2) = -2, (2 \to 3) = 2, (1 \to 3) = 1\}$. Clearly the shortest path from 1 to 3 is passing by 2. But if we add a

weight 3 and generate G' where $E = \{(1 \to 2) = 1, (2 \to 3) = 5, (1 \to 3) = 4\}$ then Dijkstra' Algorithm will tell us to go to 3 directly from 1. So this algorithm does not work.
Here the reason is that our shortest path in G may contain a large number of edges. If we add a positive weight and generate G' this path will be added too many times of weights and the length is hence "exaggerated".

4.2

It works.
Suppose we have got a shortest path in G' denoted by P, we now prove that it is also the shortest path in G. If there is another path Q from u to v in G with total weight smaller than P, then path Q must have total weight smaller than P in G' as well. The reason is that by the property of directed grids we know that any two paths from u to v contain exactly same amount of edges. So the weight added to path P and Q must be the same.

5 Comments

5.1

This time about ten hours.

5.2

Actually the most difficult question from my perspective is problem 1. I do not think I will be able to work out this problem without the hint. The hint actually reduces the time spent on thinking largely. But I will still give this problem a big 5.
As for problem 2 and 3 they are applications of algorithms with some knowledge about discrete maths. So my recommendation is 3.
Problem 4 has difficulty in between the previous questions, so the rating is 4.

5.3

No collaborators this time.(Big thanks to the hint.)