Evan Zhang | z5383657

Question 4

4.1

Begin by constructing G and run Edmonds-Karp to find the maximum flow and the minimal cut, dividing all vertices in G either into S or T.

- Store whether a vertex belongs in S with a 0 and T with a 1 in an array with its index corresponding to its vertex number, making it so checking whether a vertex is in S or T takes O(1) time.

Additionally, use a BFS to check whether any/all vertices in S is reachable from s. Follow only edges that are below max flow and mark visited vertices in an array with vertices as indexes. This will take O(n + m) time as one pass across (worst case) all vertices and edges is needed and will take O(1) time to check whether a vertex in S is reachable from s (i.e., there is a 1 capacity flow path to a from s).

Therefore, the checks we perform when queried is as follows:

- If a is in T (taking O(1) time) -> max flow will remain the same.
- If a is not in T (taking O(1) time):
    - Check whether a is reachable from s (O(1) time). If it is, max flow will increase by 1. If it isn't, max flow will stay the same

The time complexity of Edmonds-Karp + BFS check for all vertices is $O(VE^2 + n + m)$, where V = n and E = m, so precomputation runs in $O(nm^2)$ as required.

4.2

A similar approach is used, with an additional step in precomputation:

- BFS is similarly used to check whether t is reachable from any vertex in T. Start from t and follow flow edges backwards (pretend the flow network is reversed and that t is s) – this is to ensure we do not start on any disconnected component (if this is even possible). Store the results similarly.

As such, the checks we perform when queried is as follows:

- If a and b are in S (taking O(1) time), max flow will remain the same.
- If a and b are in T (taking O(1) time), max flow will remain the same.
- If a is in T and b is in S (taking O(1) time), max flow will remain the same. (Additional flow from anywhere in T to S will not increase max flow. In a minimal cut, all edges going from T to S are empty, therefore the additional edge will not invalidate the current minimal cut).
- If a is in S and b is in T (taking O(1) time):
    - Check if a and b are reachable by s and t respectively (taking O(1) time). If they both are, max flow will increase by 1. If any one of them aren't, max flow will remain the same.

The time complexity of Edmonds-Karp + BFS check for all vertices in S and T is $O(VE^2 + n + m)$, where V = n and E = m, so precomputation runs in $O(nm^2)$ as required.

4.3 and 4.4

Begin by constructing G and run Edmonds-Karp to find the maximum flow and the minimal cut, dividing all vertices in G either into S or T.

- Run the Edmonds-Karp algorithm from s to every other vertex in S (i.e., for each vertex, pretend it is a new t). Record the max flow for each vertex in an array with indexes corresponding to vertices, making retrieving max flow for each vertex take O(1) time.
    - Mark every vertex unreachable as having 0 flow.
- Run the Edmonds-Karp algorithm from every vertex in T to t (i.e., for each vertex, pretend it is a new s). Record the max flow in the array described above.
    - Mark every vertex unreachable as having 0 flow as well.

As such, the checks we perform when queried is as follows:

- If a and b are in S (taking O(1) time), max flow will remain the same.
- If a and b are in T (taking O(1) time), max flow will remain the same.
- If a is in T and b is in S (taking O(1) time), max flow will remain the same. (Additional flow from anywhere in T to S will not increase max flow. In a minimal cut, all edges going from T to S are empty, therefore the additional edge will not invalidate the current minimal cut).
- If a is in S and b is in T (taking O(1) time):
    - Check the max flow rate of a and b (taking O(1) time). The bottleneck value (i.e., min(flowrate(a), flowrate(b))) is how much the max flow rate will increase by.

Precomputation time:

- Constructing and running the Edmonds-Karp algorithm on the graph G -> $O(VE^2)$ (V = n and E = m) -> $O(nm^2)$.
- Running the Edmonds-Karp algorithm on every vertex in S and every vertex in T (i.e. every vertex in G except s and t) -> $O(n(VE^2))$ -> $O(n(nm^2))$ -> $O(n^2m^2)$.

Therefore, precomputation time will be $O(n^2m^2)$ as required.