CS180 HW7

Siddharth Joshi

TOTAL POINTS

100 / 100

QUESTION 1

1 Problem 1 **25 / 25**

✓ **- 0 pts** Correct

- 10 pts failed to union points

- 10 pts run time not correct

- 20 pts wrong answer but showed efforts

- 25 pts wrong answer or no answer

QUESTION 2

2 Problem 2 **25 / 25**

✓ **- 0 pts** Correct

- 10 pts failed to transfer the problem space from G to G'

- 10 pts failed to use max flow min cut

- 20 pts wrong answer but showed efforts

- 25 pts wrong answer or no answer

- 0 pts Click here to replace this description.

QUESTION 3

3 Problem 3 **25 / 25**

✓ **- 0 pts** Correct

- 25 pts No answer found

QUESTION 4

4 Problem 4 **25 / 25**

✓ **- 0 pts** Correct

- 6 pts Need more explanation

- 4 pts Fail to prove time complexity

- 25 pts No answer found

① Algorithm:

→ Sort the requests in descending order of rate of flow

→ Let breakpoints be an array of 2n breakpoints as discussed by the question and let each breakpoint be a tree rooted at itself

→ For each request:

- Run 'find' on the start of this request's interval and on its ending point
- If find(start) = start and find(end) = end
  - * commit the value of request.flow to the region [start, end]
  - * union all the breakpoints from 'start' of interval by following the next interesting breakpoint pointer until you reach the ~~edit~~ 'end' of interval
  - * also update the start breakpoint's next pointer of interest to end
- if find(start) != start
  - * if find(end) != end
    do nothing
  - * else
    - → commit the value of request.flow to region [find(start) → next ~~pointer~~ b.p. of interest, end]
    - → union all breakpoints between find(start) → next b.p till end in same way as above
    - → set find(start) → next break point of interest's pointer to next breakpoint to end
- else ~~if~~ (implies find(end) != end)
  - → commit the value of request.flow to region [start, find(end)] ~~→ next~~
  - → union all b.p.s ~~between~~ from start to find(end) in same manner as above
  - → set ~~for~~ start's next breakpoint of interest to find(end)

Correctness: Correctness is trivial as it is intuitively obvious that since requests are ordered by flow rate asked for, requests cannot ~~alter~~ ~~as~~ be the max in any region already covered by a request coming before it therefore all that remains is to check whether this is being ~~a~~ satisfied and the way that the algo commits makes sure that it doesn't try to commit to a region overlapping ~~w~~ a previously committed region.

Time Complexity:

→ Sorting takes $O(n \log n)$

→ The for loop also takes $O(n \log n)$, but this is a more interesting result. The ~~loop~~ runs ~~O(n)~~ n times · the steps of the loop must be $O(\log n)$ for this to be true. While it may seem that since we union ~~w~~ all ~~inde~~ breakpoints in a certain range [start, end] ~~ca~~ etc. that this could take $O(n \log n)$ but since once ~~as~~ a breakpoint has been consumed in such a union it ~~cannot~~ no longer is pointed to as a breakpoint of interest and hence will never be ~~use~~ a root again and hence in this fashion, it's easy to see how ~~there~~ ~~are~~ at most through all iterations of the loop the max # of unions is n, hence amortized time for union/~~oper~~ iteration $= O(\frac{n \log n}{n}) = O(\log n) \Rightarrow$ For loop runtime

$= O(n \log n)$ (as find takes $O(\log n)$ as well)

Hence, total time complexity $= O(n \log n)$

1 Problem 1 25 / 25

✓ **- 0 pts** Correct

- 10 pts failed to union points

- 10 pts run time not correct

- 20 pts wrong answer but showed efforts

- 25 pts wrong answer or no answer

HW 7

② Let G' be a new graph (directed) where every undirected ~~eg~~ edge in the original graph G ~~is~~ (u,v) is represented in the graph G' by 2 directed edges (u,v) and (v,u) each with capacity ~~c~~ set to 1

E.g. G G'

→ Computing the max flow on this graph ~~as~~ G' w source ~~s~~ s and sink t gives ~~the~~ us the # of edge disjoint paths from s to t.

• This can be seen by ~~using the~~ iteratively trying to find max flow by at each stage attempting to find an augmenting path P from s to t and adding ~~that~~ the capacity of said path to the current max flow. However, since every edge has capacity 1, capacity of every path $=1$ and hence the ~~f~~ additional flow at every iteration 1. As a result, the residual network G_f' at each iteration doesn't include any of the edges from path P (as flow from P 1 and each edge ~~has~~ in P has capacity 1 and hence each edge gets 'used up' completely ∴ doesn't exist in the residual network). This guarantees that any the paths found across iterations are

① edge-disjoint

② all ~~th~~ the edge-disjoint paths from s to t as the algo keeps running until none remain.

Moreover, since ~~each~~ each path contributes flow $=1$, max flow from s to t is equivalent to the # of ~~path~~ paths (edge disjoint) from s to t

Note know that for 2 vertices s and t to be disconnected in G' the max flow from s to t must equal 0
Hence

(*) ∴ Proving that max # of edge disjoint paths from s to t = s-t min # of edges to disconnect s and t is easy ~~to see~~ now as it is evident that if you ~~were to~~ ~~we~~ must <u>at least</u> delete one edge in every edge-disjoint path

But the # of edge disjoint paths in G = # of edge disjoint paths in G' (as every path P in G' can be seen as a path P' in G s.t. that the edges in P are represented by their undirected counterparts in P'). And hence by the aforementioned reasoning i.e max flow → 0 when one edge is deleted from all paths P, this is the s-t ~~connectivity~~ edge connectivity in G.

~~Proving~~ Additional inductive proof for (*):
→ ~~Assume~~ The # of edges needed to disconnect s and t in G' = 1 + # of edges —"— in G' − e where e is an edge belonging to one of the paths P found. But this decrements max flow by 1 as well. Hence in exactly max flow # of iterations, the max flow 0 in the remaining graph and hence s & t are now disconnected

(3) Proof by induction/recursion:
→ Let G the original graph
→ ~~The reasons~~ Let P be a path from s to t in G, and let V_P be the vertices in P that are not s & t
→ The # of vertex disjoint paths in G = 1 + # of vertex disjoint paths in G − V_P
→ Recursing on the graph in this manner, gives us the # of vertex disjoint paths as no 2 paths can have

2 Problem 2 **25 / 25**

✓ **- 0 pts** Correct

- 10 pts failed to transfer the problem space from G to G'

- 10 pts failed to use max flow min cut

- 20 pts wrong answer but showed efforts

- 25 pts wrong answer or no answer

- 0 pts Click here to replace this description.

Note know that for 2 vertices s and t to be disconnected in G' the max flow from s to t must equal 0
~~Hence~~

(A) .. Proving that max # of edge disjoint paths from s to t = set min # of edges to disconnect s and t is easy ~~to see~~ now as it is evident that if you ~~were to~~ ~~we~~ must <u>at least</u> delete one edge in every edge-disjoint path

But the # of edge disjoint paths in G = # of edge disjoint paths in G' (as every path P in G' can be seen as a path ~~B~~ P' in G s.t. that the edges in P are represented by their undirected counterparts in P'). And hence by the aforementioned reasoning i.e max flow → 0 when one edge is deleted ~~to~~ from all paths P, this is the s-t ~~connectivity~~ edge connectivity in G.

~~Proving~~ Additional inductive proof for (A):

→ ~~Assume~~ ~~a~~ The # of ~~st~~ edges needed to disconnect s and t in G' = 1 + # of edges -"- in G' - e where e is an edge belonging to one of the paths P found. But this decrements max flow by 1 as well. Hence in ~~the~~ exactly max flow ~~of~~ # of iterations, the max flow 0 ~~of~~ in the remaining graph and hence s & t are now disconnected

(3) Proof by induction/recursion:

→ Let G the original graph

→ ~~The reasons~~ Let P be a path from s to t in G, and let V_P be the vertices in P that are not s & t

→ The # of vertex disjoint paths in G = ~~#~~ ~~of~~ 1 + # of vertex disjoint paths in ~~G~~ G V_P

→ Recursing on the graph in this manner, gives us the # of vertex disjoint paths as no 2 paths can have

the same vertex. recursive
But also an identical relationship can be seen with the s-t vertex connectivity.
As the s-t vertex connectivity of G = 1 + s-t vertex ~~of~~ connectivity of $G - V_P$ (as the path P from s and t can deleting any vertex $v \in V_P$ from G)
→ Both recursions have the same base case though i.e. the trivial case where no paths exist from s to t.
Therefore since ~~the~~ the recursive relation and base case is identical, the ~~value~~ of vertex-disjoint paths ~~from~~ between s & t in G is # equal to the s-t vertex connectivity.

(4.) A flow network is a directed graph with edges having capacities and flows such that $\forall e \in E$, $flow(e) \leq capacity(e)$

(1) Create a new graph G' (residual of sorts) where capacity of every edge is its residual capacity i.e. ~~c~~ = original capacity − original flow.
If an edge's new 'residual capacity' = 0, delete the edge in G'
(2) Run BFS to find a path from s to t
(3) If one is found, new max flow is old max flow + 1, else ~~x~~ max flow remains unchanged

Time Complexity: Step (1) takes $O(m)$ time as each edge's capacity is updated. Step (2) takes $O(n+m)$ time (as BFS takes $O(|V|+|E|)$) and step (3) is $O(1)$ ∴ Total runtime = $O(n+m)$ as desired

3 Problem 3 **25 / 25**

✓ **- 0 pts** Correct

- 25 pts No answer found

the same vertex. recursive
But also an identical relationship can be seen with the s-t vertex connectivity.
As the s-t vertex connectivity of G = 1 + s-t vertex ~~of~~ connectivity of $G - V_p$ (as the path P from s and t can deleting any vertex $v \in V_p$ from G)
→ Both recursions have the same base case though i.e. the trivial case where no paths exist from s to t.
Therefore since ~~then~~ the recursive relation and base case is identical, the ~~value~~ # of vertex-disjoint paths ~~from~~ between s & t in G is equal to the s-t vertex connectivity.

(4.) A flow network is a directed graph with edges having capacities and flows such that $\forall e \in E$, $flow(e) \leq capacity(e)$

(1) Create a new graph G' (residual of sorts) where capacity of every edge is its residual capacity i.e. ~~e~~ = original capacity − original flow. If an edge's new 'residual capacity' = 0, delete the edge in G'
(2) Run BFS to find a path from s to t
(3) If one is found, new max flow is old max flow + 1, else ~~a~~ max flow remains unchanged

Time Complexity: Step (1) takes $O(m)$ time as each edge's capacity is updated. Step (2) takes $O(n+m)$ time (as BFS takes $O(|V|+|E|)$) and step (3) is $O(1)$ ∴ Total runtime = $O(n+m)$ as desired

6b ~~Corre~~

Correctness: The edge e having it's capacity increased by 1 is either the bottleneck edge or not i.e. to say for some path, the capacity of said path is either limited by e's capacity or not. If it isn't, then max flow remains the same and the algo actually won't be able to ~~see~~ find a path in the residual graph from s to t as some other edges are bottlenecks leading the path to be incomplete in G'. If it is ~~that~~ a bottleneck edge though, there will be a path in G' including e and that augmenting path can augment the max flow by exactly 1 (as logically the new capacity for e can at best let ~~1~~ 1 more packet across the network from s to t)

4 Problem 4 **25 / 25**

✓ **- 0 pts** Correct

- 6 pts Need more explanation

- 4 pts Fail to prove time complexity

- 25 pts No answer found