Jason Less
404-640-158
CS180

CS180 Homework 7

Question 1

Algorithm:

- Sort intervals in flow decreasing order
- Create breakpoints array of size 2n for which elements are tuples consisting of either the start or end time, and a pointer to the next break point of "interest"
- Initialize disjoint sets (maintained as an array) where each start time of an interval is its own parent, and each end time of an interval points to the start time as its parent
    - With each parent, also store the interval
- For the first interval, iterate through the breakpoints array and union any overlapping intervals
- Commit the value v_i (corresponding to interval i) to the graph for interval $[s_i, e_i]$
- For each interval i
    - Using the find part of the Union-Find algorithm, check to see whether s_i or e_i overlap with another interval (i.e. parent of s_i is not s_i or parent of e_i is not s_i)
        - In doing this, also grab the interval associated with each parent:
            - $[s_{i,parent,start}, s_{i,parent,end}]$ and $[e_{i,parent,start}, e_{i,parent,end}]$
    - If there is an overlap:
        - If s_i is the one that overlaps:
            - Do nothing, we already committed this interval
        - Else s_i doesn't overlap:
            - Update interval for s_i's parent to be $[s_i, e_{i,parent,start}]$
            - Commit value v_i to interval $s_i, e_{i,parent,start}]$
        - If e_i is the one that overlaps:
            - Do nothing, we already committed this interval
        - Else e_i doesn't overlap:
            - Update interval for e_i's parent to be $[s_{i,parent,end}, e_i]$
        - Commit value v_i to interval $s_{i,parent,end}, e_i]$
    - Else there is no overlap:
        - Need to iterate through breakpoints array and union any overlapping intervals
        - Commit value v_i to interval $[s_i, e_i]$

Jason Less
404-640-158
CS180

Time Complexity:

The time complexity of the algorithm is O(N log N), and is dominated by the sort at the beginning. Iterating through each interval takes O(N) time, and the amortized time complexity for each iteration's union operations is O(log N). Each union operation itself takes O(log N). In the worst case, in any single iteration, you may have to union all the breakpoints, which would seem to be O(N^2 log N). However, the number of intervals is fixed at N; and hence, when n intervals do get unioned, there are none left next time to union. In addition, across all iterations, we can have at most N unions (regardless of whether all N unions happen in the same iteration or in a couple iterations). Therefore, the amortized cost of the unioning can be thought of as O(total time to union / number of iterations) are O(N log N / N) = O(log N). Thus, the O(N) loop does O(log N) work for each iteration, and the overall time complexity is O(N log N).

Jason Less
404-640-158
CS180

Question 2

Utilizing Menger's Theorem, which can be shown by using applications of max flow and the Pigeonhole Principle, can prove this. Let P be any set of edge-disjoint s-t paths. Let D be any set of edges whose removal from G disconnects s and t.

By the Pigeonhole Principle, |P| <= |D|. Every path in P (i.e. pigeon) must use an edge in D (i.e. pigeonhole), otherwise, the removal of D from G doesn't disconnect s and t. Also, no two paths in P (i.e. pigeon) share the same edge in D (i.e. pigeonhole), otherwise P is not edge-disjoint. Thus, there are no more pigeons than pigeonholes. Therefore |P| <= |D|. The claim is that (i) P is a max cardinality set of edge-disjoint s-t paths, and (ii) D is a min cardinality set of edges whose removal disconnects s and t.

Let G' be the flow graph obtained from G by creating source at s and a sink at t, and setting the capacity of all edges to be 1. Let f be the max flow of G', and (S, T) be a min s-t cut of G'. Let P' be the max cardinality set of edge-disjoint s-t paths and D' be the set of edges with one vertex in S and the other in T. By definition, the removal of D' from G' disconnects s and t, and similarly, the removal also disconnects s and t in G. This is the case because:

$$\begin{aligned} |P'| &= v(f) \\ &= c(S, T) \\ &= |D'| \end{aligned}$$

From (ii) of the claim, D' is a min cardinality set of edges whose removal from G disconnects s and t. Therefore, the max number of edge-disjoint s-t paths is equal to the min number of edges whose removal from G disconnects s and t. The set of such edges can be found using the Ford-Fulkerson algorithm to find max flow, which can be done in O(mn) time.

Jason Less
404-640-158
CS180

Question 3

Reducing the problem to a max-flow problem, and then applying the same proof from question 2 with slight modifications can show this. The idea is to split each node v into two nodes v_{in} and v_{out}. For each node v, add an edge of capacity 1 from v_{in} to v_{out}. Replace each edge (u, v) in G with edge from u_{out} to v_{in} of capacity 1. Then, by finding the max-flow from s to t, this is equal to the number of vertex-disjoint s-t paths, which is also equal to the minimum number of nodes in order to disconnect s and t (as already proved in question 2). The idea for this construction works because all edges have capacity 1, and since all capacities are integral, there exists an integral max-flow. No two flows can pass through the same vertex because in passing through a vertex v, the flow must pass through the edge from v_{in} to v_{out} with capacity restricted to 1. As the capacity from v_{in} to v_{out} is restricted to 1, then after one flow path passes through such a vertex v, the leftover capacity gets reduced to 0, thus restricting any other flow path from passing through v. This ensures that flow paths represent vertex-disjoint s-t paths.

Jason Less
404-640-158
CS180

Question 4

O(n + m) time is not enough to recomputed the max flow, thus must use the flow given. The idea is simple and boils down to the fact that the max flow is either v(f) or v(f) + 1. Using the flow given, try to find a single augmenting path from s to t in the residual graph. If such a path exists, then update the flow. Otherwise, no path exists, so the flow remains unchanged. This algorithm can be executed in the time it takes to construct the residual graph (i.e. to find one augmenting path), which is O(n + m).