CSE565-Algo-HW1

Ömer Faruk Özdemir, ofo5108

September 2022

Collaborators

1. Göktan Güdükbay

Reference

1. https://stackoverflow.com/questions/50972404/minimum-number-of-edges-to-create-to-make-directed-graph-connected-from-root

1 Processing the Graph

a.) Prove or disprove the following statement: if u,v E is an edge in an undirected graph G = (V,E), and during depth-first search the post-visit numbers of u, v V satisfy ***post*$(u) <$ *post*(v), then v is an ancestor of u in the DFS tree.

Solution: Since there is an edge between u and v, we know that they are in the same SCC. That's why they will be in the same DFS Tree.
During DFS there are 2 cases:

a) u was visited using the v-u edge

In this case v is the direct parent and ancestor of u.

b) u was not visited using the v-u edge

Then we know $pre(v) < pre(u)$, proof:

Assume pre(u) < pre(v), then v would be visited either by u-v or a different path from u(because there are in the same SCC). That would result with $post(v) < post(u)$ which would contradict with *. Hence $pre(v) < pre(u) < post(u) < post(v)$, which means v is an ancestor of u.

b.) You are given a tree T = (V,E) (in adjacency list format), along with a designated root node r V. Recall that u is said to be an ancestor of v in the rooted tree if the path from r to v in T passes through u. You wish to preprocess the tree so that queries of the form "is u an ancestor of v?" can be answered in constant O(1) time. The preprocessing itself should take linear time. How can this be done?

Solution: This can easily done with post numbers. We run Explore on r and assign pre,post numbers to each vertex. $pre(u) < pre(v) < post(v) < post(u)$ means v is an ancestor of u. Pre and post numbers can be stored in an array. Explore takes $O(|V|+|E|)$, linear time. Comparison of post numbers takes $O(1)$ constant time.

2 One-way Streets

State College Police Department has decided to make all the streets one-way. The mayor of State College Borough claims that there is still a way to drive legally from any intersection in the city to any other inter- section, but the opposition is not convinced. You have been asked to write a computer program to determine whether the mayor is right. However, the mayor's term is about to end and he has said that he will not run for another term , and there is just enough time to run a linear-time algorithm.

a.) Formulate this problem graph-theoretically, and explain how it can indeed be solved in linear time (on the number of roads and intersection).

Solution: We can create a graph using Intersections as vertexes and roads as directed edges. We can find all the SCCs in the graph in $O(|V| + |E|)$ and if all the vertexes are in the same SCC then the mayors claim is correct.

b.) Suppose it now turns out that the mayor's original claim is false. He then claims something weaker: if you start driving from Westgate building, navigating one-way streets, then no matter where you reach, there is always a way to drive legally back to Westgate Building. Formulate this weaker property as a graph-theoretic problem, and carefully show how it too can be checked in linear time.

Solution: This claim means that if v is reachable from W(Westgate) then W is reachable from v. So W lives in a SCC called S , $\forall v \in S, e = (v, x) \in E \rightarrow x \in S$, ie. there is no edge that goes out of S.

1. We can find SCCs in $O(|V| + |E|)$
2. Run Explore on W and check if there is any edge going out of S(ie. $\exists (v, u) \in E \wedge v \in S \wedge u \notin S$, in $O(|V| + |E|)$

3 Make It Strongly Connected

Imagine you are given a directed graph G = (V,E) and you have been asked to find the minimum number of (directed) edges that are needed to be added to G such that the resulting graph is strongly connected. Give an algorithm that solves this problem in $O(|V| + |E|)$ time. What is the space complexity of your algorithm?

Solution: If we find all SCCs inside our graph and connect those SCCs to each other, then we would have a single SCC for the whole graph. Now the question is how to find minimum number of edges that is necessary to do this.

1. Find all SCCs inside the graph. $O(|V| + |E|)$
2. Now if we think SCCs and edges between SCCs(any edge that goes from a SCC to another SCC at the original graph) this will create a graph. The vertexes in this graph will have
   a. Either 0 edge
   b. Only in edges
   c. Only out edges
   d. The graph won't have any cycles, it would contradict with SCC classification.

3. For it to be reduced into a single SCC every SCC vertex should have at least one in and one out edges. So the output will be $max(|SCCs - without - in - edge|, |SCCs - without - out - edge|)$. We will connect SCCs with each other and result will be a single SCC.

Edge addition Algorithm

1. If there is a SCC with out-edges select it as the root(R). If there is not any SCC like that, that means there is no edges between SCCs. Select a random SCC as the root and draw an out-edge (R-K) to another random SCC(K). This edge does not break SCC classification.
2. Select a child(C) of R, (there is an edge (R-C)), C has no out-edges due to SCC classification.
3. If there is any vertex(T) without an in-edge; draw (C-T). Otherwise jump to last step.
4. Explore from T until finding vertexes without out-edges. Connect each SCC(S) to R with (S-R) except one(P).
5. Jump to step 3 via putting P into the place of C.
6. Draw (S-R) for all vertexes(S) without an out-edge.

This algorithm adds the minimum number of edges and results with a single SCC.

Minimumity: The trick is simple, keep a single path starting from R for connecting while traversing vertexes without a in-edge. And connect all other remaining vertexes without out-edges to the R.

Correctness: Every SCC is connected to the R both ways because

a. If it has 0 edge, then we add connections that connects to or from R with step 3 and 6.

b. If it has only-out edges we connect it to R by step 3.

c. If it has only in-edges, we connect it to R by out-edge paths by step 5 or 6. And connect its highest ancestor with a path from R(check b).

Time Complexity: It is $O(|V|+|E|)$. Finding SCCs is $O(|V|+|E|)$. Edge addition algorithm traverses edges and vertexes, so it is $O(|V|+|E|)$ as well.

Space Complexity: We keep vertexes and edges in memory. So that is either $O(|V|+|E|)$ or $O(|V|^2)$ depending on using adjacency matrix or list.

4 2SAT

In the 2SAT problem, you are given a set of clauses, where each clause is the disjunction (OR) of two literals (a literal is a Boolean variable or the negation of a Boolean variable). You are looking for a way to assign a value **true** or **false** to each of the variables so that all clauses are satisfied – that is, there is at least one true literal in each clause. For example, here's an instance of 2SAT:

$$(x_1 \vee \hat{x}_2) \wedge (\hat{x}_1 \vee \hat{x}_3) \wedge (x_1 \vee x_2) \wedge (\hat{x}_3 \vee x_4) \wedge (\hat{x}_1 \vee x_4).$$

This instance has a satisfying assignment: set x_1, x_2, x_3, and x_4 to **true**, **false**, **false**, and **true**, respectively.

Given an instance $\mathcal{I}$ of 2SAT with n variables and m clauses, construct a directed graph $G_{\mathcal{I}} = (V, E)$ as follows:

- $G_{\mathcal{I}}$ has $2n$ nodes, one for each variable and its negation.
- $G_{\mathcal{I}}$ has $2m$ edges: for each clause $(\alpha \vee \beta)$ of $\mathcal{I}$ (where α and β are literals), $G_{\mathcal{I}}$ has an edge from from the negation of α to β, and one from the negation of β to α.

(a.) Show that if GI has a strongly connected component containing both x and its negation xˆ for some variable x, then I has no satisfying assignment.

Solution: Realize that $(avb) \equiv (\neg a \to b) \wedge (\neg b \to a)$. This creates 2 edges, $(\neg a, b), (\neg b, a)$.

1. $\to$ operator has traversal property. $(a \to b) \wedge (b \to c) \ \to (a \to c)$
2. $(a \to \neg a) \ \wedge \ (\neg a \to a)$ is a contradiction.

Since a SCC contains both x and $\neg x$ there is a path from x to $\neg x$ and $\neg x$ to x. By 1. having path from a to b implies $a \to b$. Hence we have both $(a \to \neg a)$ *and* $(\neg a \to a)$, creates a contradiction by 2.

(b.) Now show the converse of (a): namely, that if none of GI's strongly connected components contain both a literal and its negation, then the instance I must be satisfiable.

Solution: 1. By a $(a \to \neg a) \ \wedge \ (\neg a \to a)$ results with a contradiction. And if a and $\neg a$ is not in the same SCC, this won't happen because either of them won't be able to reach the other one.

Claim: There is no other contradiction.

Proof: Assume there is, since it is a contradiction, when it is reduced into lesser terms there should be a term like this inside the and chain, $(a \to \neg a) \wedge (\neg a \to a)$ which means a and $\neg a$ are both reachable by each other by a1. and they are in the same SCC, hence a contradiction.

(c.) Conclude that there is a linear-time algorithm for solving 2SAT.

Solution: Making use of a and b, find all SCCs inside the graph, and search if a literal and its negation lives inside the same SCC. SCC creation take $O(|V| + |E|)$, search takes $O(|V|)$, so the complexity is $O(|V| + |E|)$.

5 Verify max flow

Suppose someone presents you with a solution to the maximum flow problem a flow network G. Give a linear time algorithm to determine whether the solution does indeed give a maximum flow.

Solution: Create residual edges from the current flows in the reverse direction, with the capacity as current flow. Check if there is any path from source to sink with non-zero capacity with a simple exploration algorithm using the original edges and residual edges. Complexity: $O(|E| + |V|)$

6 Ford Fulkerson

Assume you are given the following flow network (Figure 1) with source s, sink t and capacities on the edges.

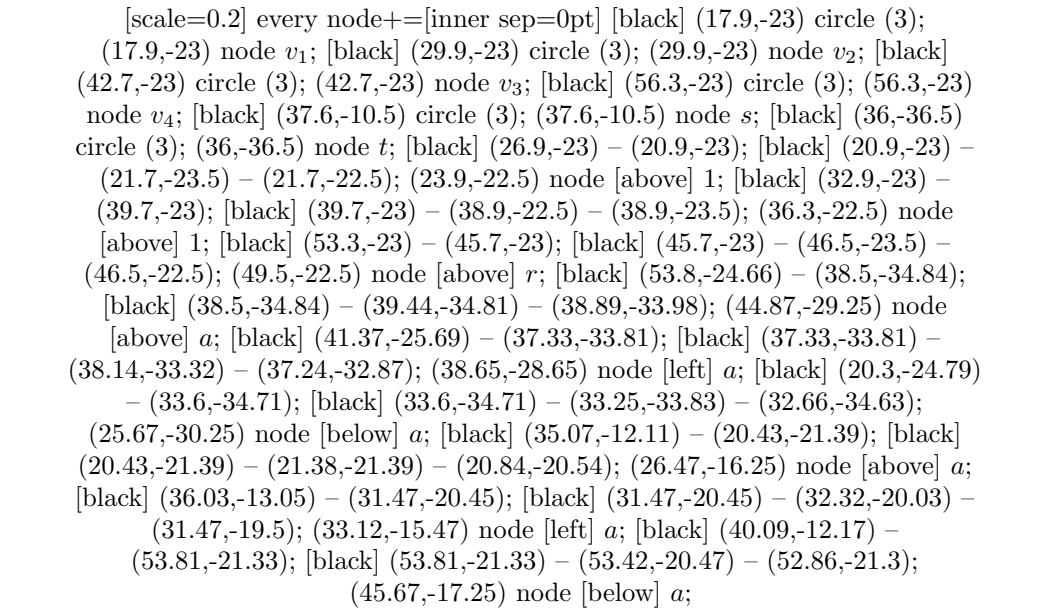
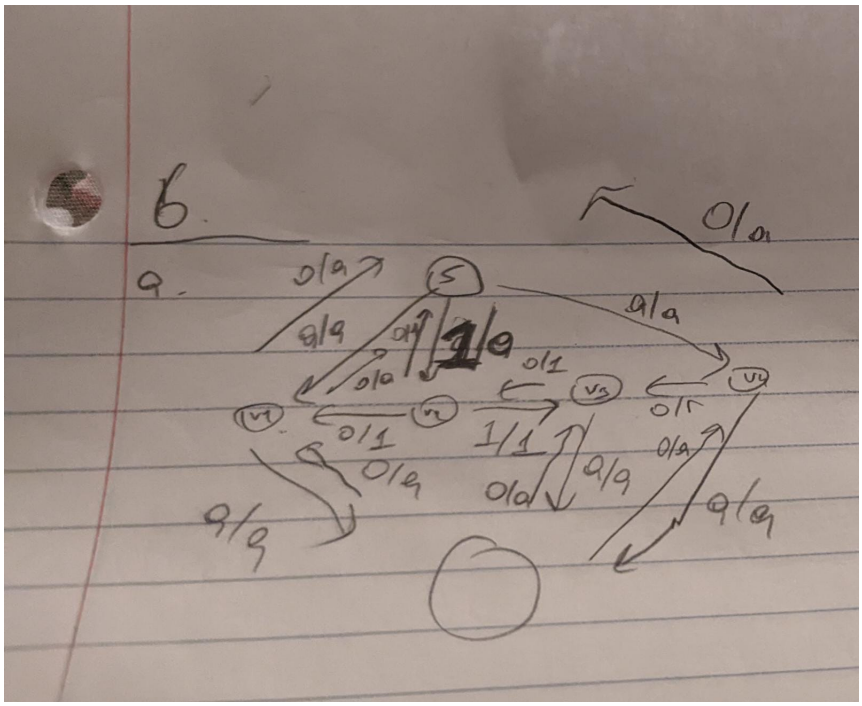


Figure 1: Flow network

where $r = (\sqrt{5} - 1)2$ and $a \geq 2$ is an integer.

(a.) Show that maximum flow of the network is 2a + 1.

Solution: Since we can't find any paths with nonzero capacity in the residual graph from source to sink, this is the max flow.



(b.) Let p0 = s, v2, v3, t, p1 = s, v4, v3, v2, v1, t , p2 = s, v2, v3, v4, t and p3 = s, v1, v2, v3, t be s t paths in the residual graph. Suppose that the Ford-Fulkerson algorithm chooses to augment along the paths p0, p1, p2, p1, p3, p1, p2, p1, p3, p1, p2, p1, p3, ... in this order. Show that after augmenting along p3 each time the residual capacities of edges (v2, v1), (v4, v3) and (v2, v3) are of the form rk, rk+1 and 0 respectively for some k N.

Solution:

I'll take 20% PASS

I was unable to find a corresponding answer even after hours of debugging, so I will just drop it here.

		s-v1	s-v2		s-v4	v1-t	v2-v1	v2-v3	v3-t	v4-v3	v4-t
step0	0	0		0	0	0	0	0	0	0	
p0		1						1	1		
p1		1		r	r	r	1-r	1	r		
p2		1+r		r	r	r	1	1	0	r	
p1		1+r		1	1	1	r	1	1-r	r	
p3	1-r	1+r		1	1	r	1	2-r	1-r	r	
p1	1-r	1+r		2r	2r	3r-1	2-2r	2-r	r	r	
p2	1-r	3r		2r	2r	3r-1	1	2-r	1-r	3r-1	
p1	1-r	3r		2-r	2-r	1	3r-1	2-r	3-4r	3r-1	
p3	3-4r	3r		2-r	2-r	3r-1	1	4-4r	3-4r	3r-1	
p1											

(c.) Prove that if we use the augmenting paths in the sequence above an infinite number of times, the total low converges to 3 + 2r. Note that since the max

flow is 2a + 1, this show that the Ford-Fulkerson algorithm never terminates and the flow doesn't even converge to the maximum flow.

Solution:

I'll take 20% PASS

As it can be seen in the sequence states(assuming the answer of b), the max flow enters into a loop and converges to 3+2r.

7 Removing an Edge

Suppose you are given an s-t flow network $G = (V, E)$. You are also given an integer maximum s-t flow in G, defined by a flow value $f(e)$ on each edge e. Now suppose we pick a specific edge $e \in E$ and increase its capacity by one unit. Show how to find a maximum flow in the resulting capacitated graph in time $O(|V| + |E|)$.

Solution: Create residual edges from the current flows in the reverse direction, with the capacity as current flow. Check if there is any path from source to sink with non-zero capacity with a simple exploration algorithm using the original edges and residual edges and update the flows whenever you found a path.

8 Reducing maximum flow

Imagine you are given a flow network with all edge capacities equal to 1. In other words, you have directed graph $G = (V, E)$, a source $s \in V$, and a sink $t \in V$; and $c_e = 1$ for every $e \in E$. You are also given a parameter k. Your goal is to delete k edges so as to reduce the maximum $s - t$ flow in G by as much as possible. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum $s - t$ flow in $G' = (V, E \setminus F)$ is as small as possible subject to this. Give an efficient algorithm to solve this problem.

Solution: This can be easily solved with min-cut max-flow duality.

1. Find the max-flow

Repeat up-to k many times:

2. Remove the edge with biggest capacity in the min cut(one of the edges that goes from source side to sink side). (To save the max-flow solution state as much as possible we can remove flows passing from this edge with Exploration from source-to-sink as well)

3. Find the max-flow again.

Complexity: If we use Orlion algorithm for finding the max-flow, $O(k.|V|.|E|)$

Rationale: Min-cut is the bottleneck of our flow. If we decrease the min-cut as much as possible in each step, the flow will decrease at most.

9 Moving Branches

A company has n branches in one city, and it plans to move some of them to another city. The expenses for operating the i-th branch is a_i per year if it stays in the original city, and is b_i per year if it is moved to the new city. The company also needs to pay c_{ij} per year for traveling if the i-th and j-th branches are not in the same city. Design a polynomial time algorithm to decide which branches should be moved to the new city such that the total expenses (including operating and traveling expenses) per year is minimized.

Solution: This can be converted into maxFlow-minCut duality problem. When a cut is drawn the vertexes will divide into 2, 1st part will stay in the B city, 2nd part will stay in the A city. *And we will draw the graph s.t the cut will pass from all of the costs, like this:

1. Source vertex is the B city(the other city)
2. Sink vertex is the A city
3. Draw edges from B to ith branch (B-i) = ai (cost of staying in A city)
4. Draw edges from A to ith branch (B-i) = bi (cost of staying in B city)
5. Draw edges between branches both-ways, (i,j) = (j,i) = cij (cost of having i and j in different cities)

When the max-flow is calculated and min-cut is drawn the branches are divided into 2 by finding the minimum cost incurred see *. There are $\binom{n}{2}$*2 edges from 5., 2*n edges from 3. and 4. In total there are n+2 Vertexes and $2 * n + n^2$ Edges. We can find polynomial complexity max-flow algorithms in the literature with respect to n. ie. Orlin, $O(n^3)$

10 Max Flow With Queries

You are given a directed graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. Each edge has a capacity. Exactly k edges $(e_1, e_2, \cdots, e_k)$ have capacities equal to zero, initially. Someone asks you q queries and suppose that q is much larger than k. In each query, she gives you k integers $w_1, w_2, \cdots, w_k$, where w_i is the capacity that she wants to be assigned to e_i, for $1 \leq i \leq k$. Give an algorithm to find the maximum flow that goes from a given vertex v to another given vertex u. The running time of your algorithm should be $\mathcal{O}(2^k T_{max\_flow}(n, m) + q2^k)$, where $T_{max\_flow}(n, m)$ is the time complexity of computing the max flow.

Solution:

I'll take 20% PASS