Lecture: Other Topics

Thursday, March 10, 2022 2:15 AM

Lecture: Flow Networks
Unit: 7.5
Instructor: Carlos C.L

CPCFI

Edge connectivity

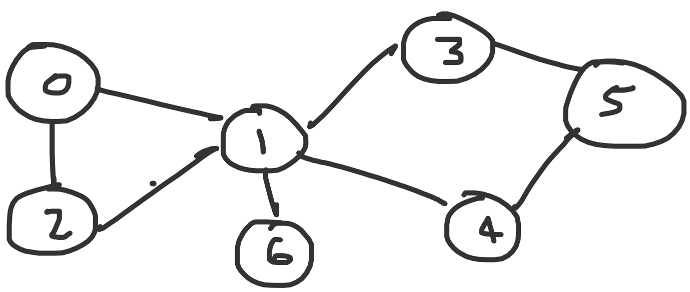
$$\lambda = min\ \#\ edges\ that\ need\ to\ be\ deleted,\ such\ that\ G\ get's\ disconnected$$

- **A graph that is already disconnected has $\lambda = 0$**
- **A connected graph with at least one bridge has $\lambda = 1$**

Vertex connectivity

$$\kappa = min\ \#\ nodes\ that\ need\ to\ be\ deleted,\ such\ that\ G\ get's\ disconnected$$

- **A graph that is already disconnected has $\kappa = 0$**
- **A connected graph with one articulation point has $\kappa = 0$**



Solve edge connectivity using max flow

a. **Iterate through all pair of vertices (u1, u2), for each pair find the largest number of disjoints paths between them, then the answer is the minimum.**

b. **To find the number of disjoint paths between u1, u2 run maximum flown on a network graph whose source is u1 and sink u2, with capacity of 1 for all edges.**

Complexity:

Using Dinic: O(V^2V^2E) = O(V^4E)
Using Stoer-Wagner = O(V^3)

Problems

CSES Police Chase

2-SAT problem

Motivation:

Say we want to order a pizza, we can choose any number of ingredients from a list, each person can give two wishes in the following way:

$$\pm x \quad \pm y$$

For example, if someone wishes:

$$+x \quad -y$$

This means she wants ingredient x and doesn't want ingredient y on the pizza.

Our problem is to find a way to pick ingredients such that at least one wish for every person is satisficed, notice there may not be solution.

The 2-SAT problem

SAT is the problem of assigning boolean values to variables that satisfy a boolean formula.

2-SAT is the problem where the formula is a conjunction of clauses where each clause is a disjunction of literals:

$$(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$$

Start by noticing these two are equivalent

$$a \vee b$$

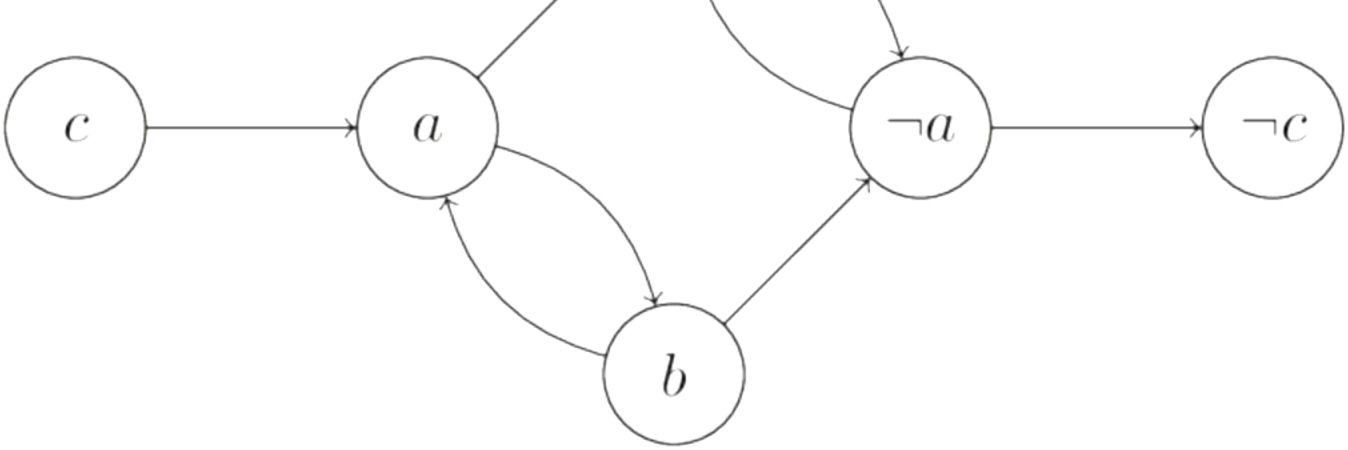
$$\neg a \Rightarrow b \wedge \neg b \Rightarrow a$$

Implication graph

Create a graph, in the following way:

- Each variable and it's negative is a node.
- For each literal [Equation] , we create two edges:

$$\neg a \Rightarrow b \quad , \quad \neg b \Rightarrow a$$

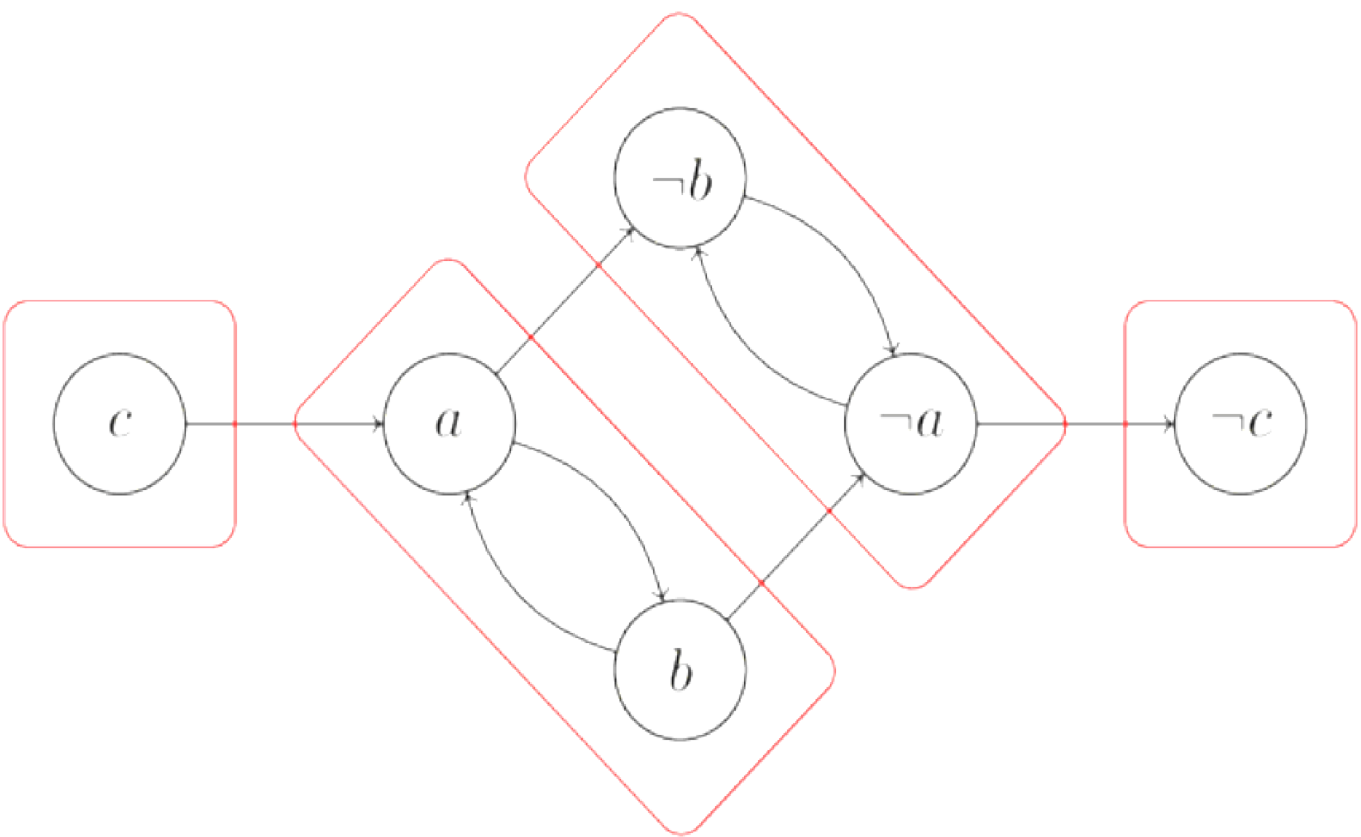


Note the following:

- **If there is a path from x to -x and a path form -x to x, then there is no solution**
- **Else, if there is a path from x to -x, then x has to be false**
- **Else, If there is a path from -x to x, then x has to be true**

Idea:

- **Find SCC on implication graph**
- **Remember, if we consider each SCC as a node, then the resulting graph will be a DAG**
- **Do topological sort on the SCCs**



Algorithm

a. **Find SCCs**

b. Create comp array, where:

comp[a] = # SCC of a

comp[a] < comp[b] if there is a path from a -> b and b is on a different SCC

c. If comp[a] $<$ comp[-a] choose a = false else true

d. If comp[a] = comp[-a] then there is no solution

Complexity:

O(V+E)

Problems

CSES Giant Pizza

Module summary

Topic	Notes	Complexity
Floyd Warshall	• Finds distance matrix, d where: $d\left[u\right]\left[v\right] = min\,distance\,(weighted)\,between\,u\,and\,v$ • Works with negative weights • Can detect negative cycles	$O\left(V^3\right)$
LCA	• Respond queries for the lowest common ancestor of any two nodes, where: $LCA\left(a,b\right) = deepesth\,node\,who\,is\,ancestor\,of\,bot\,a\,and\,b$ • Solve using Euler tour + Segment Tree	Preprocessing: $O\left(N\right)$ Answer query: $O\left(LogN\right)$
Binary Lifting	• Respond queries for the k-th ancestor of any node • Works with dynamic programming: $up\left[u\right]\left[j\right] = 2^j\,ancestor\,of\,u$ $up\left[u\right]\left[j\right] = up[up\left[u\right]\left[j-1\right], j-1]$	Preprocessing: $O\left(N\right)$ Answer query $O\left(LogN\right)$
Ford Fulkerson	• Finds maximum flow in a graph • Defines residual network (G + residual capacity and reverse edges) $residual\,capcity\left(e\right) = capacity\left(e\right) - flow\left(e\right)$ • Finds augmenting paths in residual network and updates flows using bottleneck capacity	$O\left(EF\right)$ $F = Max\,flow$
Edmonds Karp	• Ford Fulkerson using BFS to find augmenting paths	$O\left(VE^2\right)$
Dinic	• Finds maximum flow in a graph	Any graph:

	• Defines layered network = residual network + layer array + only keeps edges (u,v) where layer[u] = layer[v]+1 • Similar to Ford Fulkerson but finds augmenting paths in layered network	$O\left(V^2 E\right)$ Unit graphs: $O(\sqrt{V}E)$
Min Cost Max Flow	• Introduces cost function to a graph network: $cost\ (e) = cost\ per\ unit\ flow\ along\ edge\ e$ • Finds maximum flow with min cost total cost: $total\ cost\ (flow) = \sum_{e \in E} flow\ (e) * cost\ (e)$	Using Edmonds: $O\left(V^3 E\right) = O(V^5)$
Maximum Bipartite Matching	• Finds maximum matching matched on a bipartite graph, where there is and edge (u,v) y u can be matched with v • Use Dinic to solve this problem, construct network flow by adding source and sink and choose capacity equal to 1 for all edges	Using Dinic: $O(\sqrt{V}E)$
Assigment Problem	• Finds 1:1 matching on a bipartite graph with the lowest possible cost $cost = \sum_{(u,v) \in assigment} cost(u, v)$ • Solve using Min Cost Max Flow, construct network flow by adding source and sink, choose capacity equal to 1 for all edges and use the cost function already given	Using Edmonds: $O\left(V^3 E\right) = O\left(V^5\right)$ Hungarian: $O\left(V^3\right)$
Graph Connectivity	• Minimum number of edges/vertices that must be deleted to disconnect graph • Find disjoint number of paths between every pair of vertices, answer will be the minimum of these values	Using Dinic: $O\left(V^4 E\right)$ Stoer-Wagner: $O\left(V^3\right)$
2-SAT	• Find values for boolean variables satisfying a boolean formula • Boolean formula is a conjunction of clauses where each clause is a disjunction of literals	$O\left(V + E\right)$