Strongly Connected Components

Components with Direction

Research

Part of my research involves looking into Finite State Machines (FSMs).

Research

Part of my research involves looking into Finite State Machines (FSMs).

FSMs can be thought of as digraphs with conditions on the edges and are sometimes protected by some logic. The logic is usually in the form of extra nodes (states) that are traversed prior to entering the real states. An early problem involved determining which nodes were real and which were inserted, where we know,

Research

Part of my research involves looking into Finite State Machines (FSMs).

FSMs can be thought of as digraphs with conditions on the edges and are sometimes protected by some logic. The logic is usually in the form of extra nodes (states) that are traversed prior to entering the real states. An early problem involved determining which nodes were real and which were inserted, where we know,

- Real nodes could only ever reach a real node;
- AND all real nodes were able to reach all other real nodes.

Solution

Strongly Connected Components AKA Transitive closure

Solution

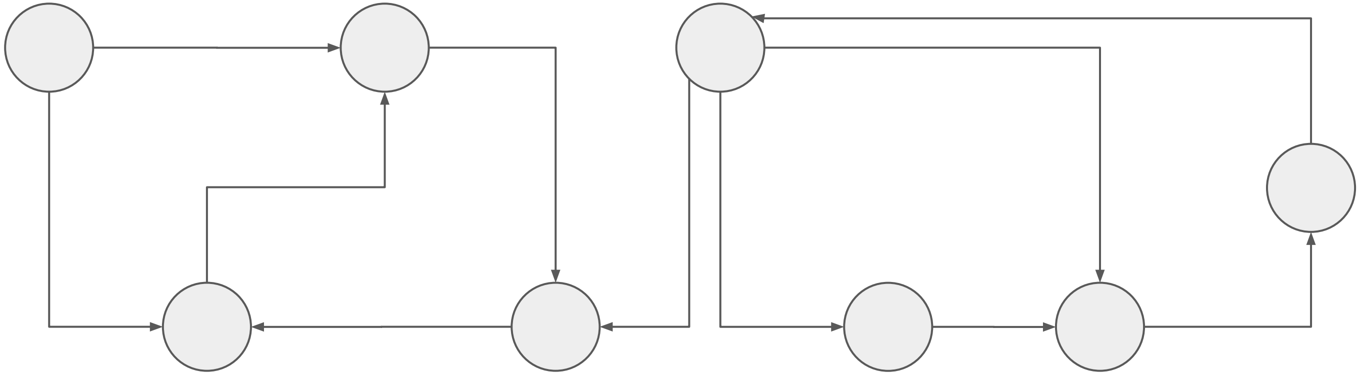
Strongly Connected Components AKA Transitive closure

Transitive closure means the groups of nodes that can reach each other in both directions through any combination of edges that exist.

Solution

Strongly Connected Components AKA Transitive closure

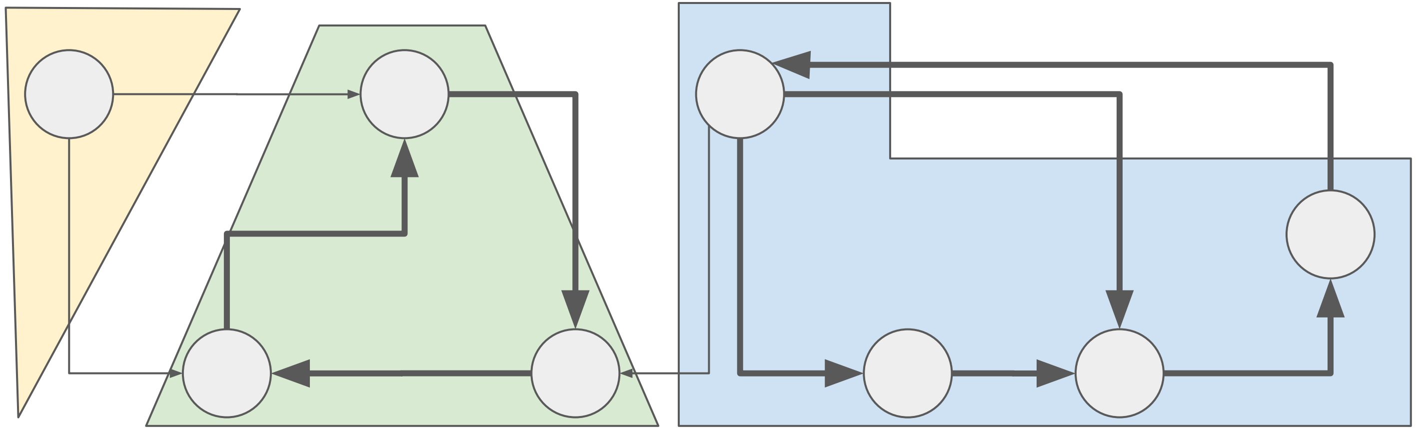
Transitive closure means the groups of nodes that can reach each other in both directions through any combination of edges that exist.



Solution

Strongly Connected Components AKA Transitive closure

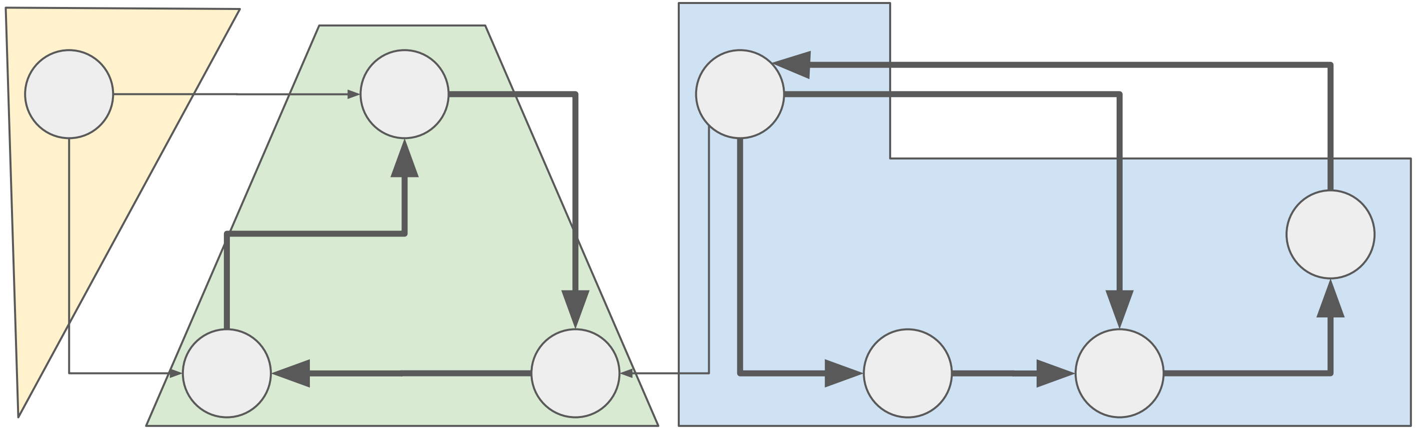
Transitive closure means the groups of nodes that can reach each other in both directions through any combination of edges that exist.



Solution

Strongly Connected Components AKA Transitive closure

Transitive closure means the groups of nodes that can reach each other in both directions through any combination of edges that exist.



<u>Note: SCCs should only be used on directed graphs</u>

Things to Note

Forming the SCCs of a graph can be thought of as compressing cycles into single nodes.

Things to Note

Forming the SCCs of a graph can be thought of as compressing cycles into single nodes.

If all nodes within all SCCs are turned into single nodes, then the resulting graph becomes a directed acyclic graph.

Things to Note

Forming the SCCs of a graph can be thought of as compressing cycles into single nodes.

If all nodes within all SCCs are turned into single nodes, then the resulting graph becomes a directed acyclic graph.

In the example before yellow could reach green, blue could reach green, but green could not reach other nodes.

Method

Two main ways

Method

Two main ways

- Low Link

Method

Two main ways

- Low Link
- Reverse DFS of the reverse of the post order of the graph

Method

Two main ways

- Low Link
- Reverse DFS of the reverse of the post order of the graph
  - Easier to code
  - Empirically worse

Method 1: Low Link

- We can use a low link method to help build SCCs.

Method 1: Low Link

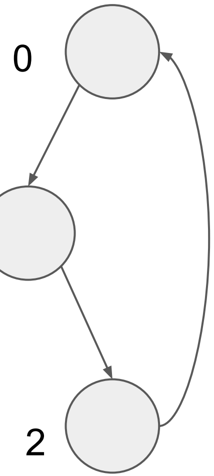
- We can use a low link method to help build SCCs.

Low-Link methods find the least pre-order node in the stack reachable by each node

Method 1: Low Link

- We can use a low link method to help build SCCs.

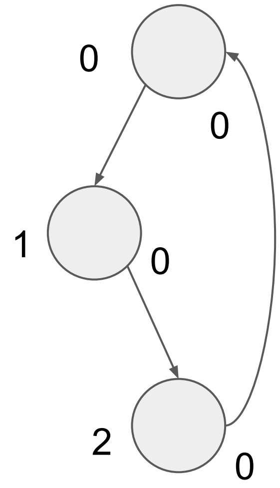
Low-Link methods find the least pre-order node in the stack reachable by each node



Method 1: Low Link

- We can use a low link method to help build SCCs.

Low-Link methods find the least pre-order node in the stack reachable by each node



Method 1: Low Link

- We can use a low link method to help build SCCs.
- Run a DFS

Method 1: Low Link

- We can use a low link method to help build SCCs.
- Run a DFS
- If a nodes low link is equal to its own pre order value, then we know that all nodes it can reach cannot reach a node with a smaller pre-order

Method 1: Low Link

- We can use a low link method to help build SCCs.
- Run a DFS
- If a nodes low link is equal to its own pre order value, then we know that **all nodes in its SCC have a pre order value greater than it**.

Method 1: Low Link

- We can use a low link method to help build SCCs.
- Run a DFS
- If a nodes low link is equal to its own pre order value, then we know that **all nodes in its SCC have a pre order value greater than it.**
- We create a stack to store all the nodes that have not found their SCC yet, but have been visited.

Method 1: Low Link

- We can use a low link method to help build SCCs.
- Run a DFS
- If a nodes low link is equal to its own pre order value, then we know that **all nodes in its SCC have a pre order value greater than it.**
- We create a stack to store all the nodes that have not found their SCC yet, but have been visited.

When a node has a pre order equal to its low link, we pop off the nodes from the stack until we reach the original node triggering this event. All those nodes belong to this SCC.

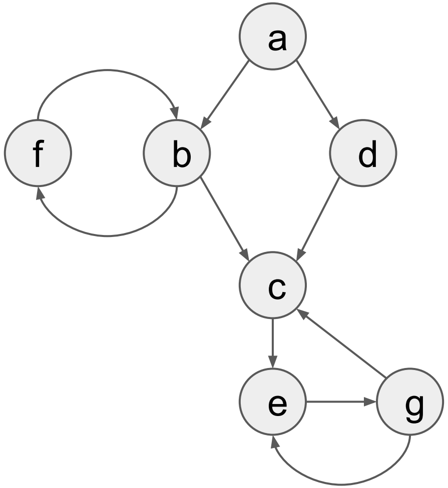
Method 1: Low Link

- We can use a low link method to help build SCCs.
- Run a DFS
- If a nodes low link is equal to its own pre order value, then we know that **<u>all nodes in its SCC have a pre order value greater than it.</u>**
- We create a stack to store all the nodes that have not found their SCC yet, but have been visited.

When a node has a pre order equal to its low link, we pop off the nodes from the stack until we reach the original node triggering this event. All those nodes belong to this SCC.

All these nodes have links to nodes that eventually reach the top, and the top can reach all nodes that have greater pre order. Thus the SCC is formed.

Example



Method 1: Low Link

This is Tarjan's

Method 1: Low Link

This is Tarjan’s

Runtime O(V + E)

Method 2: Reverse DFS Reverse DFS

- Run a DFS on a graph an store the nodes in **post order**.

Method 2: Reverse DFS Reverse DFS

- Run a DFS on a graph an store the nodes in **post order**.
- Reverse the array that stores the **post order** of these nodes.

Method 2: Reverse DFS Reverse DFS

- Run a DFS on a graph an store the nodes in **post order**.
- Reverse the array that stores the **post order** of these nodes.
- From each node in the **reverse post order** run a Reverse DFS.

Method 2: Reverse DFS Reverse DFS

- Run a DFS on a graph an store the nodes in **post order**.
- Reverse the array that stores the **post order** of these nodes.
- From each node in the **reverse post order** run a Reverse DFS.
- Each node visited from this reverse DFS is in the same SCC.

Method 2: Reverse DFS Reverse DFS

- Run a DFS on a graph an store the nodes in **post order**.
- Reverse the array that stores the **post order** of these nodes.
- From each node in the **reverse post order** run a Reverse DFS.
- Each node visited from this reverse DFS is in the same SCC.

O(E + V)

SCC Application

Checkposts Problems

https://codeforces.com/problemset/problem/427/C

Problem 1: Checkposts

Codeforces Div. 2 C

Given a city composed of junctions and one way roads that connects junctions, place posts to protect the city. At each junction a post can be built. Posts protect the junction they are placed in. Additionally, posts can send police cars to other junctions to protect them as long as the police car can travel back to the post it came from. Cities have a unique cost for building a post.

Problem 1: Checkposts

Codeforces Div. 2 C

Given a city composed of junctions and one way roads that connects junctions, place posts to protect the city. At each junction a post can be built. Posts protect the junction they are placed in. Additionally, posts can send police cars to other junctions to protect them as long as the police car can travel back to the post it came from. Cities have a unique cost for building a post (non-negative up to 10^9).

Determine the minimum cost to protect all junctions **in the fewest number of posts**.

Idea

Find all SCCs.

Idea

Find all SCCs.

To protect a city it needs to be reached by a post and be able to reach a post.

Idea

Find all SCCs.

To protect a city it needs to be reached by a post and be able to reach a post.

How are the SCCs used to find the minimum cost?

Idea

Find all SCCs.

To protect a city it needs to be reached by a post and be able to reach a post.

How are the SCCs used to find the minimum cost?

The number of SCCs is the number of posts required. A post in an SCC can protect all nodes in the SCC.

Idea

Find all SCCs.

To protect a city it needs to be reached by a post and be able to reach a post.

How are the SCCs used to find the minimum cost?

The number of SCCs is the number of posts required. A post in an SCC can protect all nodes in the SCC.

We want to minimize the cost by finding the minimum cost post in each SCC.

Not Done

The real problem wants to know how many ways can we protect all junctions with the minimum cost.

Not Done

The real problem wants to know how many ways can we protect all junctions with the minimum cost.

We must use the minimum cost from an SCC.

Not Done

The real problem wants to know how many ways can we protect all junctions with the minimum cost.

We must use the minimum cost from an SCC.

BUT there might be many posts with an SCC that have the same cost.

Two SAT

A different class of problem.

Two SAT

A different class of problem.

We have a series of statements that ALL need to be true.

Two SAT

A different class of problem.

We have a series of statements that ALL need to be true.

Each statement has a simple form (X OR Y)

Two SAT

A different class of problem.

We have a series of statements that ALL need to be true.

Each statement has a simple form (X OR Y)

For example (a OR b) AND ($\neg$b OR c) AND (c OR $\neg$a)

Two SAT

A different class of problem.

We have a series of statements that ALL need to be true.

Each statement has a simple form (X OR Y)

For example (a OR b) AND (¬b OR c) AND (c OR ¬a)

We want to find out if a solution exists.

Two SAT Reduction

Suppose we need to satisfy ($\neg$b OR c)

Two SAT Reduction

Suppose we need to satisfy (¬b OR c)

We know that if b is TRUE, then to satisfy the constraint we need c to be TRUE as well.

Two SAT Reduction

Suppose we need to satisfy ($\neg$b OR c)

We know that if b is TRUE, then to satisfy the constraint we need c to be TRUE as well.

Suppose instead we know that c is FALSE, then we must have b be FALSE as well.

Two SAT Reduction

Suppose we need to satisfy ($\neg$b OR c)

We know that if b is TRUE, then to satisfy the constraint we need c to be TRUE as well.

Suppose instead we know that c is FALSE, then we must have b be FALSE as well.

We can say that b implies c and $\neg$c implies $\neg$b

Two SAT Reduction

Suppose we need to satisfy ($\neg$b OR c)

We know that if b is TRUE, then to satisfy the constraint we need c to be TRUE as well.

Suppose instead we know that c is FALSE, then we must have b be FALSE as well.

We can say that b implies c and $\neg$c implies $\neg$b

We use arrows to denote implication (b $\rightarrow$ c) ($\neg$c $\rightarrow$ $\neg$b)

Contradictions

Implications are transitive $(a \to b)$ and $(b \to c)$ means $(a \to c)$.

Contradictions

Implications are transitive $(a \rightarrow b)$ and $(b \rightarrow c)$ means $(a \rightarrow c)$.

Special Implication: If we find that $(a \rightarrow \neg a)$ then a must be false, because if a is true, then a is also false.

Contradictions

Implications are transitive $(a \to b)$ and $(b \to c)$ means $(a \to c)$.

Special Implication: If we find that $(a \to \neg a)$ then a must be false, because if a is true, then a is also false.

Even better: What if we find that both $(a \to \neg a)$ and $(\neg a \to a)$

Contradictions

Implications are transitive $(a \rightarrow b)$ and $(b \rightarrow c)$ means $(a \rightarrow c)$.

Special Implication: If we find that $(a \rightarrow \neg a)$ then a must be false, because if a is true, then a is also false.

Even better: What if we find that both $(a \rightarrow \neg a)$ and $(\neg a \rightarrow a)$

This would mean that a is TRUE AND FALSE. CONTRADICTION

Contradictions

Implications are transitive $(a \to b)$ and $(b \to c)$ means $(a \to c)$.

Special Implication: If we find that $(a \to \neg a)$ then a must be false, because if a is true, then a is also false.

Even better: What if we find that both $(a \to \neg a)$ and $(\neg a \to a)$

This would mean that a is TRUE AND FALSE. CONTRADICTION

Such a constraint would be impossible to satisfy impossible

Contradictions (cont.)

How do we find out if for some variable $(a \rightarrow \neg a)$ and $(\neg a \rightarrow a)$?

Contradictions (cont.)

How do we find out if for some variable ($a \rightarrow \neg a$) and ($\neg a \rightarrow a$)?

SCCs (obviously XD)

Contradictions (cont.)

How do we find out if for some variable (a→¬a) and (¬a→a)?

SCCs (obviously XD)

- Turn each variable assignment into a node (both a and ¬a become nodes).

Contradictions (cont.)

How do we find out if for some variable ($a \to \neg a$) and ($\neg a \to a$)?

SCCs (obviously XD)

- Turn each variable assignment into a node (both a and $\neg$a become nodes).
- Draw directed edges for each implication.

Contradictions (cont.)

How do we find out if for some variable ($a \to \neg a$) and ($\neg a \to a$)?

SCCs (obviously XD)

- Turn each variable assignment into a node (both a and $\neg a$ become nodes).
- Draw directed edges for each implication.
- If a and $\neg a$ are in the same scc then they can imply each other using transitivity.

Contradictions (cont.)

How do we find out if for some variable (a→¬a) and (¬a→a)?

SCCs (obviously XD)

- Turn each variable assignment into a node (both a and ¬a become nodes).
- Draw directed edges for each implication.
- If a and ¬a are in the same scc then they can imply each other using transitivity.
- Check each variable and see if they belong to the same SCC.

Problem 2:

https://codeforces.com/problemset/problem/27/D

Problem 2: Ring Road 2

Codeforce Div. 2 D

There is a land with N cities in a circle. Road will be build inside or outside the circle. We don't want roads to cross (outside of a city). Determine if possible and if so print out a way to build the roads (state whether each road is inside or outside the circle.

Approach 1

Since the lecture is about SCCs we will probably use SCCs.

Approach 1

Since the lecture is about SCCs we will probably use SCCs.

It pretty obvious that the roads can form edges and cities can form nodes…

Approach 1

Since the lecture is about SCCs we will probably use SCCs.

It pretty obvious that the roads can form edges and cities can form nodes…

BUT the edges are not really directed. SCCs won’t make sense.

Approach 1

Since the lecture is about SCCs we will probably use SCCs.

It pretty obvious that the roads can form edges and cities can form nodes…

BUT the edges are not really directed. SCCs won’t make sense.

We need to find a different graph.

Approach 1

Since the lecture is about SCCs we will probably use SCCs.

It pretty obvious that the roads can form edges and cities can form nodes…

BUT the edges are not really directed. SCCs won't make sense.

We need to find a different graph.

If we use 2 SAT we typically want variables that can be set to TRUE or FALSE (two choices). We have this ability. We have roads that can be set to INSIDE or OUTSIDE.

Approach 1

Since the lecture is about SCCs we will probably use SCCs.

It pretty obvious that the roads can form edges and cities can form nodes…

BUT the edges are not really directed. SCCs won't make sense.

We need to find a different graph.

If we use 2 SAT we typically want variables that can be set to TRUE or FALSE (two choices). We have this ability. We have roads that can be set to INSIDE or OUTSIDE.

If certain roads are inside we could potentially have other roads that must become outside.

Approach 2

We can also greedily assign roads and BFS. :\

Problem 3: 2-SAT 2

TROY Query
https://codeforces.com/gym/100570/problem/D

Problem 3: TROY Query

Codeforces Div. 1 D

We have 2 grids of 10^{18}x10^{18}. Each spot is either -1 or +1. We can flip rows and columns by multiplying every spot in the row by -1 or every spot in the column by -1. We are told 1 spot at a time what the values in each grid are. We want to know for the information given so far if the two grids can be transformed into each other (assuming we choose the unknown bit).

Approach

If 2 SAT is the solution, what would be our decisions?

Approach

If 2 SAT is the solution, what would be our decisions?

How do the decisions affect each other?

Approach

If 2 SAT is the solution, what would be our decisions?

How do the decisions affect each other?

We could add an edge for each piece of information rerun SCC and check for contradictions, BUT…

Approach

If 2 SAT is the solution, what would be our decisions?

How do the decisions affect each other?

We could add an edge for each piece of information rerun SCC and check for contradictions, BUT…

We are told up to 10^5 different pieces of the grid. Why is that a problem?

Approach

If 2 SAT is the solution, what would be our decisions?

How do the decisions affect each other?

We could add an edge for each piece of information rerun SCC and check for contradictions, BUT…

We are told up to 10^5 different pieces of the grid. Why is that a problem?

What can we do to fix?

Note: at some point the Grids will be incapable of reaching each other, and regardless of the extra information it will always be impossible.

Binary Search

Look for the contradiction point build the graph and check

Binary Search

Look for the contradiction point build the graph and check

$O(\log(Q)*(V + E))$