Biconnectivity

Let $G = (N, E)$ be a connected graph.

A node $a \in N$ is an articulation point if there are $v$ and $w$ different from $a$ such that every path from $v$ to $w$ must pass through $a$.

Biconnected Component

Maximal subgraph that is still connected after

Express in terms of edges

Articulation point must lie in two components
DFS and Articulation Points

Tree, back edges in DFS give clue to articulation points

If $a$ is an articulation point that splits $v$ and $w$, then

Lemma:

$G = (N, E)$ is
$S = (N, T)$ is

Node $a$ is an articulation point if and only if
1. $a$ is the root of $S$ and

or
2. $a$ is not the root and $a$ has some child $c$ in $S$ where there is no
Low Numbers

For graph $G = (N, E)$, let

$T =$
$B =$

For simplicity, assume $DFNum[v] =$

How high (lowest $DFNum$) can we hop from a node on a back edge?

$LOW[v] =$
$\min \{ \{v\} \cup \{w| \text{there is } \{x, w\} \text{ in } B \} \text{ where}$

Calculating Low

If $\{x, w\} \in B$ and $w < v$ then

Also, $v$ is an articulation point if for some child $c$ of $v$,

$LOW[c]$

Calculate $LOW$ during DFS

$LOW[v]$ is minimum of

1.
2.
3.
Biconnected Components Algorithm

Count —
Parent[v] —
BI(v)
  mark v
  DFNum[v] ← Count, increment Count
  Low[v] ← DFNum[v]

Algorithm Cont.

for each edge {v, w} do
  if w not marked then
    if LOW[w] ≥ DFNum[v]
       and v not root then
      LOW[v] ←
    else if PARENT[v] ≠ w then
      LOW[v] ←

Need a check at end if root is an articulation point.
Minimum Cost Spanning Tree

\[ G = (N, E) \]

Cost function \( c \) gives edge weights.

A spanning tree is a graph \( S = ( \quad ) \)
where \( E \) and \( S \) is a subset of \( E' \).
Cost of a Spanning Tree

Cost of a spanning tree $S = (N, E')$ is

$$\sum_{e \in E'}$$

Want minimum cost

**Lemma**: Let $S = (N, E')$ be a spanning tree for $G = (N, E)$. Then

a. for $v_1, v_2$ in $N$

b. adding an edge from $E - E'$ to $S$

Spanning Forest

A set of trees

$\{(N_1, E_1), (N_2, E_2), \ldots, (N_k, E_k)\}$

Where

$N = \subseteq E$

$E \subseteq E_j$

$E_i \subseteq E_j$

Diagram of a graph with nodes labeled 1 to 6 and edges labeled 1 to 7.
Expanding a Spanning Forest

Lemma: Suppose we have a spanning forest of \( k > 1 \) trees. Let \( E' = \)

Let \( e = \{v, w\} \) be

such that \( v \in N_1, w \notin N_1. \)

Then there is a spanning tree for \( G \)

that includes \( E' \cup E'' \) and has minimal cost among trees that include \( E' \).

What kind of algorithm is this leading us towards?

Proof of Lemma

Suppose not. Let

\[ S'' = (N, E''), \quad E' \subseteq E'', \]

have lower cost than any spanning tree.

Adding \( e \) to \( S'' \) causes

\[ v \rightarrow w \]

\[ v' \rightarrow w' \]
Proof of Lemma 2

$S''$ must have an edge $e' = \{v', w'\}$ with

We know $c(e) \leq c(e')$

Let $S = S''$

$S$ has no cycles

$S$ is still a tree

Greedy Strategy

Take least cost edge $e_1$ in $E$. By last lemma,

[Consider an initial spanning forest

$\{ (\{ \}, \emptyset), (\{ \}, \emptyset), \ldots, (\{ \}, \emptyset) \}$]
Iterative Step

Let
\[ \{(N_1,E_1), (N_2,E_2), \ldots, (N_k,E_k)\} \]
be a spanning forest that is

Add the lowest cost edge that

By the last lemma, the result is

Kruskal’s Algorithm

\[
\text{KRUSKAL}(G(N,E))
\]
\[
S \leftarrow \emptyset
\]
\[
\text{NS} \leftarrow \{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\}
\]
\[
Q \leftarrow
\]
\[
\text{while } |\text{NS}| > 1 \text{ do}
\]
\[
\{v,w\}
\]
\[
\text{if } \text{Find}(v) \neq \text{Find}(w) \text{ then}
\]
\[
S \leftarrow S
\]
\[
\text{Union}(\quad)
\]
"
Example

Q  c  NS
{2,5} 1  {1}  {2}  {3}  {4}  {5}  {6}
{3,4} 2
{1,2} 3
{1,5} 3
{1,3} 4
{4,6} 4
{2,3} 6
{5,6} 7

Complexity

Priority Queue: sort, make a list
Union/Find
  # Union's =
  # Find's =

Can do a little better—
    keep a heap of edges by cost
Path Problems

Directed graphs with labels on edges

Shortest Path

Labels could be non-negative integers
Use for
Connectivity

Labels could be Boolean values
Use for

![Graph example]

Finite Automata

Labels could be symbols from an alphabet
(as in a finite automaton). Use for

![Graph example with labels]
Path Algorithms

Can produce generic algorithms, just need definitions for
Can take advantage of identities
Two kinds of algorithms
1.
2.

Can do 2. with

Dijkstra’s Algorithm

Single-source, shortest path in a digraph

S—array of distances
**General Step**

Add node not in $S$ with minimum $D$ value to $S$.

Update distances to nodes in $N-S$.

Is $S$ still correct?

**Full Algorithm**

$G = (N,E)$

$\text{lab}(v,w) - \text{label}$

represent with matrix

$S \leftarrow \{v_0\}$

$D[v_0] \leftarrow 0$

$D[v] \leftarrow \text{lab}(v_0,v)$ for rest

while $S \neq N$ do

choose $w$

$S \leftarrow$

for all $v$ in $N-S$ do

$D[v] \leftarrow$
Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>0</td>
<td>2</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
</tr>
</tbody>
</table>

Warshall’s Algorithm

Transitive Closure

Basic op

Consider each k only once
Algorithm

for k = 1 to n do
  for v = 1 to n do
    if a[v,k] then
      for w = 1 to n do
        if a[k,w] then

Example

```
   1  2  3  4  5
  1 ( 1  1  0  0  1 )
  2 | 0  1  1  0  0 |
  3 | 0  0  1  1  0 |
  4 | 0  0  0  1  0 |
  5 ( 0  0  1  1  1 )
```
Floyd’s Algorithm

All-pairs shortest path

\[
\begin{align*}
\text{for } & k = 1 \text{ to } n \text{ do} \\
& \text{for } v = 1 \text{ to } n \text{ do} \\
& \quad \text{if } d[v,k] \neq \infty \text{ then} \\
& \quad \text{for } w = 1 \text{ to } n \text{ do} \\
& \quad \quad d[v,w] \leftarrow \\
\end{align*}
\]

Example

\[
\begin{pmatrix}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 4 & \infty \\
2 & 1 & 0 & 2 & \infty \\
3 & \infty & 2 & 0 & 3 \\
4 & 2 & \infty & \infty & 0
\end{pmatrix}
\]

\[d[i,j]^0\]
### Example 2

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 4 & \infty \\
2 & 1 & 0 & 2 & \infty \\
3 & \infty & 2 & 0 & 3 \\
4 & 2 & \infty & \infty & 0 \\
\end{array}
\]

\[d[i,j]^0\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & \leftarrow & \leftarrow & \leftarrow \\
2 & \leftarrow & \leftarrow & \leftarrow \\
3 & \leftarrow & \leftarrow & \leftarrow \\
4 & \leftarrow & \leftarrow & \leftarrow \\
\end{array}
\]

### Example 3

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 4 & \infty \\
2 & 1 & 0 & 2 & \infty \\
3 & \infty & 2 & 0 & 3 \\
4 & 2 & 3 & 6 & 0 \\
\end{array}
\]

\[d[i,j]^1\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & \leftarrow & \leftarrow & \leftarrow \\
2 & \leftarrow & \leftarrow & \leftarrow \\
3 & \leftarrow & \leftarrow & \leftarrow \\
4 & \leftarrow & \leftarrow & \leftarrow \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 3 & \infty \\
2 & 1 & 0 & 2 & \infty \\
3 & 3 & 2 & 0 & 3 \\
4 & 2 & 3 & 5 & 0 \\
\end{array}
\]

\[d[i,j]^2\]
Example 3

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & ( & 0 & 1 & 3 & \infty \\
2 & & 1 & 0 & 2 & \infty \\
3 & 3 & 2 & 0 & 3 \\
4 & 2 & 3 & 5 & 0 \\
\end{array} \]

\[ d[i,j]^2 \]

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & ( & & & \\
2 & & & & \\
3 & & & & \\
4 & & & & \\
\end{array} \]

\[ d[i,j]^3 \]