Algorithm Design & Analysis

Biconnectivity

Let $G = (N, E)$ be a connected, undirected 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 $w$ goes through $a$.

Articulation point must lie in two components.

Biconnected Component

Maximal subgraph that is still connected after removal of any one node.

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
\( v \) is an ancestor
\( w \) is a descendant of \( a \)

vice versa

Lemma:
\( G = (N, E) \) is connected, undirected graph
\( S = (N, T) \) is DFS Tree of \( G \)
Node \( a \) is an articulation point if and only if
1. \( a \) is the root of \( S \) and
\( a \) has two or more children
or
2. \( a \) is not the root and \( a \) has some child \( c \) in \( S \) where there is no back edge from \( c \) or any descendant to a proper ancestor of \( a \).
Low Numbers

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

$T =$ Tree edges from a DFS
$B =$ Back edges

For simplicity, assume $\text{DFNum}[v] = v$

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

$\text{LOW}[v] =$

$\min\{\{v\} \cup \\
\{w | \text{there is } \{x,w\} \text{ in } B \text{ where } x \text{ is a descendant of } v \} \}$

Calculating Low

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

$w$ is an ancestor of $v$.

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

$\text{LOW}[c] \geq v$

Calculate $\text{LOW}$ during DFS

$\text{LOW}[v]$ is minimum of

1. $v$
2. $\text{LOW}[c]$ for all children of $v$
3. $w$ where $\{v, w\} \in B$
Biconnected Components Algorithm

Count = \( \text{Global} \), \( j = 0 \)
Parent[v] = parent of node v in T

BI(v)
mark v
DFNum[v] \( \leftarrow \) Count, increment Count
Low[v] \( \leftarrow \) DFNum[v]

Algorithm Cont.
for each edge \( \{v, w\} \) do
if w not marked then
  add \( \{v, w\} \) to T
  PARENT[w] \( \leftarrow \) v
  BI(w)
  if \( \text{LOW}[w] \geq \text{DFNum}[v] \)
    and v not root then
    Print (v is an articulation point)
    LOW[v] \( \leftarrow \) \( \min \) (LOW[v], LOW[w])
  else if \( \text{PARENT}[v] \neq w \) then
    LOW[v] \( \leftarrow \) \( \min \) (LOW[v], DFNum[w])

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

\[ G = (N, E) \]

Cost function \( c : E \rightarrow \mathbb{R} \) gives edge weights

A spanning tree is a graph \( S = (N, E') \) where \( E \subseteq E' \) and \( S \) is a tree

Complexity is \( O(|E|) \) in a connected graph \( |E| \geq |N|-1 \)
Cost of a Spanning Tree

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

$$\sum_{e \in E'} c(e)$$

Want minimum cost

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

a. for $v_1, v_2$ in $N$ there is a unique simple path $v_1 \rightarrow v_2$

b. adding an edge from $E - E'$ to $S$ causes a unique simple cycle

Spanning Forest

A set of trees

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

Where

$N = N_1 \cup N_2 \cup \ldots \cup N_k$

$E_i \subset E$

$E_i \cap E_j = \emptyset$
Expanding a Spanning Forest

Lemma: Suppose we have a spanning forest of \( k > 1 \) trees. Let \( E' = E_1 \cup E_2 \cup \ldots \cup E_k \).

Let \( e = \{v, w\} \) be a lowest cost edge in \( E - E' \) 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 \( S'' = (N, E''), E' \subseteq E'', e \notin E'' \) have lower cost than any spanning tree that contains \( E' \) and \( e \).

Adding \( e \) to \( S'' \) causes a cycle.

\[ c(e) \leq c(e') \]

\( e' \in E'' \)
Proof of Lemma 2

$S''$ must have an edge $e' = \{v', w'\}$
with $v' \in N_1, w' \notin N_1$

We know $c(e) \leq c(e')$
Let $S = S''$ with $e'$ removed and $e$ added
$S$ has no cycles
$e$ created a cycle, removing $e'$ broke it
$S$ is still a tree
$S$ costs no more than $S''$

Greedy Strategy

Take least cost edge $e_1$ in $E$. By last lemma,
there is some spanning tree that contains $e_1$

[Consider an initial spanning forest
$\{ (\{v_1\}, \emptyset), (\{v_2\}, \emptyset), \ldots, (\{v_n\}, \emptyset) \} \}$]
Iterative Step

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

Add the lowest cost edge that connects two trees in the forest.

By the last lemma, the result is a spanning forest that is a subset of a MST.

Kruskal’s Algorithm

KRUSKAL(G(N, E))

\[
\begin{align*}
S & \leftarrow \emptyset \quad \text{spanning forest edges} \\
NS & \leftarrow \{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\} \\
Q & \leftarrow \text{a priority queue of edges \ \text{sorted by increasing weight}} \\
\text{while } |NS| > 1 \text{ do} \\
& \{v, w\} \leftarrow \text{EXMIN}(Q) \\
& \text{if } \text{Find}(v) \neq \text{Find}(w) \text{ then} \\
& \quad S \leftarrow S \cup \{v, w\} \\
& \quad \text{Union( } \text{Find}(v), \text{Find}(w) \text{ )}
\end{align*}
\]
**Example**

<table>
<thead>
<tr>
<th>Q</th>
<th>c</th>
<th>NS</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2,5}</td>
<td>1</td>
<td>{1}</td>
</tr>
<tr>
<td>{3,4}</td>
<td>2</td>
<td>{1}</td>
</tr>
<tr>
<td>{1,2}</td>
<td>3</td>
<td>{1}</td>
</tr>
<tr>
<td>×{1,5}</td>
<td>3</td>
<td>{1,2,5}</td>
</tr>
<tr>
<td>×{1,3}</td>
<td>4</td>
<td>{1,2,3,4,5}</td>
</tr>
<tr>
<td>×{4,6}</td>
<td>4</td>
<td>{1,2,3,4,5,6}</td>
</tr>
<tr>
<td>{2,3}</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>{5,6}</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

**Complexity**

Priority Queue: sort, make a list  \( O(|E| \log |E|) \)

Union/Find

\[
\begin{align*}
\text{# Union's} &= n-1 \\
\text{# Find's} &= 2|E| + \text{at most} \quad O(|E| \log |E|)
\end{align*}
\]

Can do a little better—
keep a heap of edges by cost  \( O(d \log |E|) \)

\[\text{the number of edges we consider before we find a free}\]
Path Problems

Directed graphs with labels on edges

1 2 3 4 5 6
a b c
d e f

Shortest Path

Labels could be non-negative integers
Use for finding paths of minimum distance
• as addition
• as min

1 2 3
3 2 2
5 6
Connectivity

Labels could be Boolean values
Use for reachability

- is and
- or

\[(1 \lor 1) \lor (1 \lor 0) \lor 1\]

![Graph](image)

Finite Automata

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

- find strings that take you from one node to another

\[(a \cdot b) \cup (b \cdot a) \cup (a \cdot a)\]

![Graph](image)
Path Algorithms

Can produce generic algorithms, just need definitions for $+$, $*$

Can take advantage of identities $1 * \text{ anything } = 1$

Two kinds of algorithms

1. Single source:
   - From one $v$ to all the rest

2. All pairs:
   - From $v$ to $w$ for every pair $v, w$

Can do 2. with multiple calls to 1,
   - but there might be cheaper ways

Dijkstra’s Algorithm

Single-source, shortest path in a digraph

$S$ — Nodes for which shortest path is known
$D$ — Array of distances to nodes from $v_0$ only passing through nodes in $S$
General Step

Add node not in \( S \) with minimum \( D \) value to \( S \).

Update distances to nodes in \( N-S \) using \( w \).

Is \( S \) still correct?

Full Algorithm

\( G = (N,E) \)

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

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 \( w \) in \( N-S \) with minimum \( D[w] \) value

\( S \leftarrow S \cup \{w\} \)

for all \( v \) in \( N-S \) do

\( D[v] \leftarrow \min (D[v], D[w] + \text{lab}(w,v)) \)
Warshall's Algorithm

Transitive Closure

Basic op

Consider each \( k \) only once

Paths that go through nodes numbered \(< k\)

Is there a path with node numbered \( \leq k \)
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 \( a[v,w] \leftarrow 1 \)

\( O(n^3) \)

Example

\[
\begin{pmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 1 & 1 & 0 & 1 \\
2 & 0 & 1 & 1 & 0 \\
3 & 0 & 0 & 1 & 1 \\
4 & 0 & 0 & 0 & 1 \\
5 & 0 & 0 & 1 & 1 \\
\end{pmatrix}
\]

pivot on 1
pivot on 2
pivot on 3
pivot on 4
pivot on 5

no new paths
no new paths
1 \rightarrow 2, 1 \rightarrow 3
2 \rightarrow 3, 2 \rightarrow 4

1 \rightarrow 5

1 \rightarrow 2, 1 \rightarrow 3
1 \rightarrow 4, 1 \rightarrow 5

1 \rightarrow 2, 1 \rightarrow 3
2 \rightarrow 3, 2 \rightarrow 4

1 \rightarrow 5

no new paths

\( a[i,j] \)
Floyd’s Algorithm

All-pairs shortest path

for \( k = 1 \) to \( n \) do
  for \( v = 1 \) to \( n \) do
    if \( d[v,k] \neq \infty \) then
      for \( w = 1 \) to \( n \) do
        \( d[v,w] \leftarrow \min(d[v,w], d[v,k] + d[k,w]) \)

Algorithm Design & Analysis

Example

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

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

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

Example 3

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

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>∞</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

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

\[d[i,j]^4\]