OUTLINE

- Definitions
- Relation of MBST with MST
- Camerini’s Algorithm for finding an MBST in an undirected connected graph
- Camerini’s Algorithm for finding an MBSA in a directed connected graph
- Gabow’s and Tarjan Algorithms for finding an MBSA in a directed connected graph
Bottleneck in a spanning tree

A bottleneck edge(s) in a spanning tree are the edge(s) with the maximum weight in the tree.

Minimum bottleneck spanning tree is a spanning tree which has the minimum bottleneck edge value among all other spanning trees.
MBST and MST

Well MST (minimum spanning tree) is necessary an MBST while the opposite is not.

Therefore any algorithm that finds an MST is also an algorithm for finding an MBST.
Given an undirected connected Graph $G=(V,E)$ with cost function $w$ mapping edges to positive real numbers.

Find a Minimum Bottleneck Spanning Tree
Camerini’s Algorithm for finding MBST of an undirected connected Graph

Given a connected undirected graph $G = (V, E)$ with a cost function $w$ mapping edges to positive real numbers. We define the follow:

1. $A = \text{UH}(E)$ // Function UH takes E set of edges in G and returns $A \subseteq E$ such that:
   
   (a) $|A| = \left\lfloor \frac{|E|}{2} \right\rfloor$
   
   (b) $\forall e_k \in A$ and $\forall e_h \in (E - A)$ \hspace{0.5cm} $w(e_k) \geq w(e_h)$

2. Let $B = E - A$

3. $F = \text{FOREST}(G_B)$ // where $G_B = (V, B)$ and FOREST returns F such that:
   
   (a) $F$ the maximal forest of $G_B = (V, B)$

4. $\eta = \{N_1, \ldots, N_c\}$ where $N_i$ is the $i_{th}$ component of $F$
Theorem 1

(a) If $F$ is a spanning tree of $G$ then the MBST of $G$ is given by the MBST of $G_B$

(b) Else MBST of $G$ is given by $F \cup$ any MBST of Graph $G'$

$G'$ is Graph $G_A$ Collapsed into $\eta$
Camerini’s Algorithm for finding MBST of an undirected connected Graph

1: procedure MBST(G, w)
2: Let E be the set of edges of G
3: if |E| = 1 then
4: Return E;
5: else
6: \( A \leftarrow UH(E, w) \);
7: \( B \leftarrow E - A \)
8: \( F \leftarrow FOREST(G_B) \)
9: Let \( \eta = \{N_1, N_2, ..., N_c\} \) where \( N_i(i = 1, 2, ..., c) \) is the set of nodes of the \( i-th \) component of \( F \);
10: if \( c = 1 \) then
11: Return \( MBST(G_B, w) \);
12: else
13: Return \( F \cup MBST((G_A)_{\eta}, w) \);
14: end if
15: end if
16: end procedure
Camerini’s Algorithm for finding MBST of an undirected connected Graph

\[ G \]

\[ G_B \]

\[ F \]

\( e_k \in A \)

\( e_h \in B \)

\( F \) is not a spanning Tree of \( G \)
Camerini’s Algorithm for finding MBST of an undirected connected Graph

$(G_A)_\eta$

Recursive Call
MBST Of

$(G_A)_\eta$

$G_{B'}$

$F'$ is a spanning Tree of $(G_A)_\eta$ i.e. MBST of $G_{B'}$ is an MBST of $(G_A)_\eta$

$e_k \in A$

$e_h \in B$

$\in A'$

$\in B'$

$\in A'$

$\in B'$
Camerini’s Algorithm for finding MBST of an undirected connected Graph

$G_{B'}$

$G_{B''}$

$(G_{A''})_\alpha$

F'' is not spanning Tree of $G_{B'}$

F'' U MBST of $(G_{A''})_\alpha$ gives the MBST of therefore $(G_A)_\eta$
Camerini’s Algorithm for finding MBST of an undirected connected Graph

\[
\text{MBST OF } G = F \cup \text{MBST of } (G_A)_\eta
\]
Camerini’s Algorithm for finding MBST of an undirected connected Graph

**Time Complexity**

UH, FOREST, $G_B$, $(G_A)_\eta$ all require $O\left(\frac{m}{2^i}\right)$ at the $i^{th}$ iteration, where $m$ is the number of edges at the first call.

- Since UH is similar to finding the median and then splitting the edges of $E$ with respect to that median, and finding the median can be done in $O(m)$ time.

- FOREST can be computed using DFS,

- $G_B$, $(G_A)_\eta$ since it only consist of building the adjacency list which contains $O\left(\frac{m}{2^i}\right)$ edges at iteration $i$.

Therefore runs in $O(m + \frac{m}{2} + \frac{m}{4} + \ldots + 1) = O(m)$
Arborescence and Spanning Arborescence

G=(V,E)

- An *arborescence* of G is a directed tree of G which contains a directed path from a specified node, say node L, to each node of a subset \( V' \) of \( V \setminus \{L\} \).
  
  Node L is called the root of arborescence.

- An arborescence is a spanning arborescence if \( V'=V \setminus \{L\} \)

- MBST in this case is a spanning arborescence with the minimum bottleneck edge.
Camerini’s Algorithm for finding MBSA of Directed Graph

In the directed graph $G=(V,E)$. Assumed to have a spanning arborescence rooted at node $L$. We define the follow

1. $A= UH(E-T)$ // Function $UH$ takes $(E-T)$ set of edges in $G$ and returns $A \subset (E-T)$ such that:

   (a) $|A| = \left\lfloor \frac{|E-T|}{2} \right\rfloor$

   (b) $\forall e_k \in A$ and $\forall e_h \in ((E - T) - A)$  $w(e_k) \geq w(e_h)$

2. $T$ a subset of $E$ for which it is know that $G_T$ does not contain any spanning arborescence rooted at node $L$. Initially $T=\emptyset$

3. Let $B = (E - T) - A$

4. $F = BUSH(G_{B\cup T})$ // where $G_{B\cup T} = (V, B \cup T)$ and BUSH returns $F$ such that:

   (a) $F$ the maximal arborescence of $G_{B\cup T}$ rooted at $L$. 


Camerini’s Algorithm for finding MBSA of Directed Graph

- “If F is a spanning arborescence of G then an MBSA of G can be given by any MBSA of $(G_B, T)$

- Else call MBSA($G, T \cup B$) in which now T is increased by B

```
Algorithm 3 Compute a MBSA of A Directed Graph G
1: procedure MBSA($G, w, T$)
2:     Let E be the set of edges of G
3:     if $|E - T| > 1$ then
4:         A ← UH(E-T);
5:         B ← (E-T)-A;
6:         F ← BUSH($G_{B\cup T}$);
7:         if F is a spanning arborescence of G then S ← F; MBSA($(G_{B\cup T}, w, T)$);
8:         else
9:             MBSA($G, w, T \cup B$);
10:         end if
11:     end if
12: end procedure
```
Camerini’s Algorithm for finding MBST of Directed Graph

1. \( e_1 \) to \( e_n \)

2. Case if \( F \) is a spanning arborescence

3. Case if \( F' \) is not a spanning arborescence

4. Case if \( F'' \) is a spanning arborescence

5. Case if \( F''' \) is not a spanning arborescence

This continues until \( M - |T| = 1 \) where \( M \) is the number of edges of last Graph.

Note that \( F \) is the arborescence of the Graph \( T \cup U \cup B \)
Time Complexity

- UH requires $O(m)$

- BUSH requires $O(m)$ at each execution and the number of these executions is $O(\log m)$ since $|E - T|$ is being halved at each call of MBSA,

Total total time complexity is $O(m\log m)$
Gabow and Tarjan noticed that if Dijkstra’s single-source shortest path algorithm was modified slightly it will produce an MBSA.

- Let S be the distinguished root vertex of G, and F be the collection of vertices and their inclusion cost \( c(v) \).
- Start with F having only node S
- Select minimum \( c(v) \) in F and delete it
- For every edge \((v,w)\) if w is not in F and not in Tree add it along with cost \( c(w) = c(v,w) \) and make v its parent
  - If w already in F check if \( c(v,w) < c(w) \) -> \( c(w) = c(v,w) \) and make v parent of w
procedure MBSA-GT(G, w, T)
for $|V|$ times do
Select $v$ with minimum $c(v)$ from F
Delete it from F
for $\forall edge(v, w)$ do
if $w \notin F$ or $w \notin Tree$ then
add $w$ to F
$c(w) = c(v, w)$
$p(w) = v$
else
if $w \in F$ AND $c(w) > c(v, w)$ then
$c(w) = c(v, w)$
$p(w) = v$
end if
end if
end for
end for
end procedure
### Tarjan and Gabow Algorithm for finding MBSA

<table>
<thead>
<tr>
<th>Vertex v</th>
<th>c(v)</th>
<th>p(v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-∞</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph Diagram]
<table>
<thead>
<tr>
<th>Vertex ( v )</th>
<th>( c(v) )</th>
<th>( p(v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-( \infty )</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>1</td>
</tr>
</tbody>
</table>

Tarjan and Gabow Algorithm for finding MBSA
### Tarjan and Gabow Algorithm for finding MBSA

<table>
<thead>
<tr>
<th>Vertex v</th>
<th>c(v)</th>
<th>p(v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-∞</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>2</td>
</tr>
</tbody>
</table>

C(3)=9<10=C(2,3) keep 1 as parent
Add 4 to F and make 2 parent of 4
Tarjan and Gabow Algorithm for finding MBSA

<table>
<thead>
<tr>
<th>Vertex v</th>
<th>c(v)</th>
<th>p(v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-∞</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>2</td>
</tr>
</tbody>
</table>

Diagram:

- Vertices: 1, 2, 3, 4, 5, 6, S
- Edges with weights:
  - (1, 2) 7
  - (1, 3) 9
  - (1, 4) 11
  - (1, 5) 15
  - (2, 3) 10
  - (3, 6) 14
  - (4, 6) 9
  - (5, 6) 2
Tarjan and Gabow Algorithm for finding MBSA

<table>
<thead>
<tr>
<th>Vertex v</th>
<th>c(v)</th>
<th>p(v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-∞</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>3</td>
</tr>
</tbody>
</table>

C(4)=15>11=C(3,4) make 3 parent of 4
C(6)=14>2=C(3,6) make 3 parent of 6
**Tarjan and Gabow Algorithm for finding MBSA**

<table>
<thead>
<tr>
<th>Vertex v</th>
<th>c(v)</th>
<th>p(v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-∞</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

F

![Graph diagram with vertex labels and edges]
Add 5 to F and make 6 parent of 5
Tarjan and Gabow Algorithm for finding MBSA

<table>
<thead>
<tr>
<th>Vertex v</th>
<th>c(v)</th>
<th>p(v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-∞</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>
5 has no neighbors and last element in F
delete

Tarjan and Gabow Algorithm for finding MBSA
since the only neighbour for 4 is 5 and 5 is in the tree but not in F do nothing
This algorithm runs in $O(|V| \log |V| + |E|)$ time if Fibonacci heap was used to implement $F$.

The table on the right shows the time complexity of the operations done on $F$.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Fibonacci$^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>find-min</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>delete-min</td>
<td>$O(\log n)^b$</td>
</tr>
<tr>
<td>insert</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>decrease-key</td>
<td>$\Theta(1)^b$</td>
</tr>
<tr>
<td>merge</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>

The Total edges visited are $|E|$ and we are iterating in the outer for loop for $|V|$ times the operation done inside the inner loop are in $O(\log |V|)$, therefore time complexity is $O(|V| \log |V| + |E|)$.
Another approach proposed by Tarjan and Gabow is to find the cost of the bottleneck edge $\lambda^*$ and keep the graph with all edges’ costs at most $\lambda^*$.

As in Camerini’s algorithm it keeps partitioning the edges until we find the subgraph $G(\lambda^*)$ such that all edges in $G$ are less than $\lambda^*$ then any spanning arborescence of $G$ is an MBSA.

However the partitioning here is different in which a function $k(i)$ is introduced to determine the number of partitions we should have at iteration $i$.

$$k(1) = 2$$
$$k(i) = 2^{(k(i-1))}$$
Tarjan and Gabow Algorithm for finding MBSA in Sparse Graph

1: procedure MBSA-KPARTITIONING(G, w)
2:       \( \lambda_1 = c(u, v) \) where \( c(u, v) \leq c(x, y) \) \( \forall (x, y) \in E \) \ // \( \lambda_1 \) be the minimum edge cost in E
3:       \( \lambda_2 = c(u, v) \) where \( c(u, v) \geq c(x, y) \) \( \forall (x, y) \in E \) \ // \( \lambda_2 \) be the maximum edge cost in E
4:       i = 0;
5:     Step 1:
6:       i \leftarrow i + 1;
7:       Let \( S_0 \subseteq E \) such that \( \forall e_k \in S_0 \) \( c(e_k) \leq \lambda_1 \)
8:       Let \( E_1 \subseteq E \) such that \( \forall e_h \in E_1 \) \( \lambda_1 < c(e_h) \leq \lambda_2 \)
9:     Step 2:
10:      Partition \( E_1 \) into \( k(i) \) subsets such that \( S_1, S_2, S_3, \ldots, S(k(i)) \)
11:      and \( \forall e_r \in S_i \) and \( \forall e_p \in S_{i+1} \) \( e_r \leq e_p \)
12:      and the size of each subset is \( \lfloor \frac{|E_1|}{K(i)} \rfloor \) or \( \lceil \frac{|E_1|}{K(i)} \rceil \)
13:     Step 3:
14:      Find minimum \( j \) such that \( G(j) = (V, S_0 \cup S_1 \cup S_2 \cdots \cup S_j) \)
15:      and in \( G(j) \) \( \forall v \in V \) \( \exists \) path from \( s \) to \( v \) where \( s \) is the distinguished root
16:     Step 4:
17:      if \( j=0 \)
18:      \( \lambda^* = \lambda_1 \) Terminate;
19:   Else
20:      \( \lambda_1 = c(u, v) \) where \( c(u, v) \leq c(x, y) \) \( \forall (x, y) \in S_j \)
21:          \ // \( \lambda_1 \) be the minimum edge cost in \( S_j \)
22:      \( \lambda_2 = c(u, v) \) where \( c(u, v) \geq c(x, y) \) \( \forall (x, y) \in S_j \)
23:          \ // \( \lambda_2 \) be the maximum edge cost in \( S_j \)
24:   Go to Step 1
25: end procedure
$i = 1$

$K(i) = K(1) = 2$ partitions $S_1, S_2$

Each partition should be equal to $\left\lfloor \frac{|E_1|}{K(i)} \right\rfloor$ or $\left\lceil \frac{|E_1|}{K(i)} \right\rceil$

and $\forall e_r \in S_i$ and $\forall e_p \in S_{i+1}$ $e_r \leq e_p$

$S_0$ is the set of edges in $E$ with cost $\leq \lambda_1$

Initially

$\lambda_1 =$ minimum edge cost in $E$

$\lambda_2 =$ maximum edge cost in $E$
Tarjan and Gabow Algorithm for finding MBSA in Sparse Graph

Start from edges in $S_0$ and check if all nodes are reachable from vertex $s$ the distinguished root if not move to $S_1$

For example assume that all nodes are reachable from $s$ only when we reached set $S_1$
Tarjan and Gabow Algorithm for finding MBSA in Sparse Graph

\[ \lambda_1 \quad \lambda_2 \quad \rightarrow \quad E \]

\[ e_1, e_2, \ldots, e_{(n-1)}, e_n \]

\[ S_0 \quad \uparrow \quad S_1 \quad \uparrow \quad S_2 \quad \uparrow \quad S_3 \quad \uparrow \quad S_4 \]

\[ i = 2 \]

\[ k(i) = K(2) = 2^{k(2-1)} = 4 \quad \text{partitions} \]
Tarjan and Gabow Algorithm for finding MBSA in Sparse Graph

For example assume that the all nodes are reachable at \( S_3 \)

E1 will be partitioned to 16 partition

This will continue until all nodes are reachable from set \( S_0 \) then the arborescence of the graph \( G=(V,S_0) \) rooted at \( s \) is a minimum spanning arborescence

This algorithm runs in \( O(|E|\log^*|V|) \)
References

