# 201701436_AGT_Assignment_6

## Data Structures

1. Modified Adjacency Matrix $(G)$

- $G(u)(v)$ = set of edges connecting $u$ and $v$
- i.e. $G(u)(v) = \{e_1, e_2, e_3, ...\}$

2. Spanning Tree - Edge Set $(currentTree)$

- $currentTree$ will contain the set of $edges$ which are part of the current spanning tree

## Algorithm

- Nodes are numbered from $0, 1, 2, 3, .....n-1$ and Edges are labelled from $e_0, e_1, e_1, e_3, .....e_{m-1}$ for the convenience.
- The Algorithm is simply backtracking of the proof of Cayley's Formula: $\tau(G) = \tau(G - e) + \tau(G \cdot e)$

### Utility Functions

> add_edge $(G, u, v, e)$
>  **add** $e$ to $G(u)(v)$
>  **add** $e$ to $G(v)(u)$
>
> remove_edge $(G, u, v, e = null)$
>  if $e$ is equal to $null$
>  **delete** $G(u)(v)$
>  **delete** $G(v)(u)$
>  else
>  **remove** $e$ from $G(u)(v)$
>  **remove** $e$ from $G(u)(v)$

#### Collapse ($G \cdot e$)

- Let G be a graph and $e = uv$ be a link ( i.e. a non-loop edge). The graph $G \cdot e$ has
  vertex set $V(G) \setminus \{ u, v \} \cup \{ w \}$ where $w$ is a new vertex and the following edges:
  (1) If $f$ is an edge in $G$ with end vertices in $V(G) \setminus \{ u, v \}$ then $f$ is an edge in $G \cdot e$.
  (2) If $f \neq e$ is an edge with end vertices in $\{ u, v \}$ then $f$ is a loop in $G \cdot e$ at

$w$.
(3) If $f$ = $xy$ is an edge in $[\{ u, v \}, \ V(G) \setminus \{ u, v \}]$ then $f$ is an edge in $G \cdot e$ with one end
vertex as $w$ and the other end vertex as $y$.

- Here the condition is that $u$ and $v$ will be distinct, i.e. The edge that is getting removed will not be a loop. In other words a node will never collapse on itself.
- Without the loss of generality, instead of labelling the new vertex with label = $w$, we can use labels of one of the vertex $u$ or $v$, i.e. we will call the new vertex = $u$
- In other words, $v$ is collapsing into $u$, and all the loops and edges going from of $v$ are now given to $u$
- After the collapse , $|E(Ge)| = |E(G)| - 1$ and $|V(Ge)| = |V(G)| - 1$
- Pseudo Code

> Collapse $(G, u, v, e)$
> $Ge \leftarrow$ copy of $G$
> remove_edge$(Ge, u, v, e)$
> **FOR** each $node$ in $G(v)$
> **FOR** each $edge$ in $G(v)(node)$
> add_edge$(Ge, u, node, edge)$
> remove_edge $(Ge, v, node)$
> **delete** $v$ from $Ge$
> **return** $Ge$

## Backtracking Algorithm using Cayley's Formula

### Cayley's formula : $\tau(G) = \tau(G - e) + \tau(G \cdot e)$, where $e$ is an edge of $G$.

- Function $EnumerateTrees$ will take a graph G, number of nodes n, a set called $allTrees$, a set called $currentTree$ as the input
- Initially $currentTree$ set and $allTrees$ set will be empty
- When the function is terminated, $allTrees$ will contain set of Trees, where each $Tree$ will be represented by a set of $edges$ in that Tree.
- We always select the L**east Labelled Node**( i.e. $0$ here ) and one of its adjacent node to collapse into each other, and **relabel** the resulting node **with least node's label**( i.e. $0$ here )
- Node $0$ is SUPER NODE for us.

### Algorithm

1. Choose one of the adjacent nodes of $0$ will be collapsing into $0$, and node $0$ will have edges and loops of both the nodes, except one edge on which we are collapsing the nodes
2. The edge which we chose to collapse on, will be the part of $currentTree$.
3. Recurs for the new Graph $Ge$, and pass the set $currentTree$ along with it.
4. After the recursion remove the edge from the $currentTree$ set as well as from the graph $G$
5. If there are multiple edges, we will repeat step 2-4 for all of them.
6. Repeat above steps for all the adjacent nodes of $0$
7. We will never collapse the node $0$ on itself
8. Termination Condition: Whenever during the recursion, if size of Graph becomes equal to $1$(i.e. #nodes = 1), we add the $currentTree$ to $allTrees$ set, and Return

> $EnumerateTrees\ (G, allTrees, currentTree = set())$
> **IF** $length(G) == 1$
> **add** $currentTree$ to $allTrees$
> **return**
> **FOR** each $node$ in $G(0)$
> **IF** $node$ is not equal to $0$:
> **FOR** each $edge$ in $G(0)(node)$
> $Ge \leftarrow$ collapse $(G, 0, node, edge)$
> **add** $edge$ to $currentTree$
> $EnumerateTrees$
> $(Ge, allTrees, currentTree)$
> **remove** $edge$ from $currentTree$
> remove_edge$(G, 0, node, e)$

## Proof of Correctness

### Termination Condition :

- Since at every level of Recursion we are removing one node from the graph using Collapse function, and we return when there is only one node in the graph:
- Recursion tree can't have depth more than $n - 1$
- Also number of processes at each level will be equal to $d(0)$, where $d(0)$ = degree of node $0$ at the previous level
- We can not have degree of the $0$ more than $|E|$, since we are not adding any extra edges from the outside

- Hence, every processes in each level is a finite number i.e. $\leq |E|$
- Hence the algorithm always terminates in finite time

## Proof of Cayley's formula : $\tau(G) = \tau(G - e) + \tau(G \cdot e)$, where $e$ is an edge of $G$

- Let $T(G)$ = set of all spanning trees of $G$. Then $\tau(G)$ = $|T(G)|$.
- Further, $T(G) = \{T \in T(G) : e \notin T\} \cup \{T \in T(G) : e \in T\}$ and also $\{T \in T(G) : e \notin T\} \cap \{T \in T(G) : e \in T\} = \phi$, Therefore, $\tau(G) = |T(G)| = |\{T \in T(G) : e \notin T\}| + |\{T \in T(G) : e \in T\}|$

## Resulting Set forms a Spanning Tree :

1. $|E| = n - 1$

- **At every level** we remove one edge($e$) from the graph, and **add it to the current Tree**
- Now for the next level $E(Ge) = E(G - e)$, hence the edge which was added to the current Tree will never get added again since it will is removed by Collapse Function
- Also termination is when is only one node in the graph, hence in other word we will terminate when **Recursion Level =** $n - 1$, we are not adding any edge further
- Hence at the end, current Tree has n-1 edges

2. **Connectedness**

- We are Collapsing the current node with other node (i.e. not itself), hence only the edge which was removed gets deleted, but all the other edges remain intact
- Also the node which was deleted, passes all the edges and loops to the least Node (i.e. $0$ here)
- Also we are only traversing through adjacent nodes, (i.e. BFS tree), Hence the **next edge that is going to be added to the current Tree will always be adjacent to one of the edges that are already in the current Tree**
- Hence given a connected tree, after the traversal is over (at termination), the edges that were removed be also a part of connected graph

## Uniqueness and Completeness using Cayley's Proof:

- Our algorithm is based upon the proof of Cayley's formula : $\tau(G) = \tau(G - e) + \tau(G \cdot e)$
- Logic is simple : A tree will either include the edge $e$ or it will not.
- We are fixing the one of the collapsing node as $0$, and we will relabel the node after collapse equal to $0$.

- $\tau(G \cdot e)$ : For each edge($e$) in the graph, we first **collapse** it (meaning include in the spanning tree), and then **recurs over every possible combinations** of tree which include $e$
- Since we are always covering possibility for **every adjacent node** of $0$, we don't have option to collapse any more nodes since there must exist an edge between two nodes to collapse them.
- Note. that at every Level of recursion node $0$ will always be in the graph, since every node is getting collapsed into $0$, but the label always remain equal to $0$.
- $\tau(G - e)$ : Now, we have **remove that edge** from the original Graph itself, so that next sets of trees **does not repeat any of the trees** in the existing set of trees, this step is very crucial to prevent repeating of the trees in the final set.
- Also we remove the edge from the current Tree set, hence it does not get repeated.

## Time Complexity

- We already proved that there will be at $N - 1$ levels in the recursion tree.
- At level 0, we have only one processing node
- Lets say that Recursion Tree has $Tp$ processing Units
- At the next level there will be $d(0)$ processing nodes where , $d(0)$ = Degree of node 1.
- Now the worst case will be complete graph, hence $d(0) = N - 1$
- Now, for the second level, for each processing node $d(0) = 2 * N - 1$
  - Proof: The node collapsing into node $0$, has degree = N - 1
  - Now when two nodes $u$ and $v$ collide resulting degree of $d(w) = d(u) + d(v) - 2$
  - Here $w = 0$, $u = 0$ and for any other node $v$, $d(v) = N - 1$ (Complete Graph)
  - Initially $d_0(0) = 1$
  - At level 1, one processing unit:
    - $d_1(0)$ = $N - 1$
  - At level 2, for each processing unit: $N$ nodes in the previous graph
    - $d_2(v)$ = $N - 1$
    - $d_2(0) = d_2(0) + d_2(v) - 2$
    - $d_2(0) = N - 1 + N - 1 - 2 = 2N - 3$
  - Again At Level 3, for each processing unit : $N - 1$ nodes in the previous graph
    - $d_3(v) = N - 2$

    - $d_3(0) = d_3(0) + d_3(v) - 2$
    - $d_3(0) = 2N - 1 + N - 2 - 2 = 3N - 5$
- At level $k$ : Number of Child Processes for each process will be equal to $d_k(0)$
    - $d_k(0) = kN - 2k + 1$
    - $d_k(0) \leq kN$ , since $k > 0$
- Hence $d_0(0) = N - 1$, will be the processes generated by each node at level 1
- Total Number of Processes at $kth$ Level: $t_k = \prod_{i=0}^{i=k-1} d_i(0)$
    - $t_k \leq \prod_{i=0}^{i=k-1} iN = (k-1)!(N^k)$
- Total Number of Processes in whole Tree: $Tp = \sum_{k=1}^{k=N} t_k$
    - $Tp \leq \sum_{k=1}^{k=N} (k-1)!(N^k)$
      $\leq \sum_{k=1}^{k=N} (N-1)!(N^N)$
      $= N!(N^N)$
    - $Tp = O(N!(N^N))$, note here $Tp$ is the size of the recursion Tree
- Time Complexity for each processing Unit is Equal to Time Complexity of collapsing Two nodes: $= 2(d(u) + d(v)) = 2(d(0) + d(v))$ , remove edge from $v$, add edges to $u$
    - We already know that at level $k$
    - $d_k(0) = kN - 2k + 1$
    - $d_k(v) = N - k + 1$
    - $c_k = 2(d_k(0) + d_k(v))$
      $= kN - 2k + 1 + N - k + 1$
      $= (k+1)N - (3k - 2)$
      $\leq (k+1)N$
- Total Time Complexity : $T = \sum_{k=1}^{k=N} t_k \cdot c_k$
- $T \leq \sum_{k=1}^{k=N} (k-1)!(N^k) \cdot (k+1)N$
  $\leq \sum_{k=1}^{k=N} (N-1)!(N^N) \cdot (N+1)N$
  $= (N+1)!(N^{N+1})$
- $T = O((N+1)!(N^{N+1}))$, where $T$ is the time complexity