Dynamic Graphs problems involve maintaining solutions under additions and removal of edges, while maintaining solutions to problems such as:

- **Connectivity** on undirected unweighted graph. Tree/forest can be used to maintain the connectivity.
- **2-connectivity**
- **matching**
- **max flow** this problem is kind of difficult.

When add, or remove edge \((u, v)\) within a forest, considering the connectivity problem, an important aspect about the edge is:

- whether \(u\) and \(v\) are in the same tree
- \(\Leftrightarrow\) whether root of \(u\) is the same with root of \(v\)
- \(\Leftrightarrow\) find the root of a vertex

To do this, we first need to root the trees.

However, the depth of a tree can be really high, \(O(n)\). For example, when there is a path of \(n\) vertices, and the root is at one of the end. Thus, next, we will solve this problem on paths by using BSTs.

## 1 Dynamic Paths: represent using BSTs

First, we consider the simpler case of the trees being single paths.

![Figure 1: A path to a BST](image)
To reduce the depth of a path, we can first choose the middle node $x$ as the root, and then recursively doing the same on the left and right of $x$, forming the left and right subtree. Figure 1 shows this process.

In this way, we can change a path to a BST tree, eg. an AVL tree, with depth $O(\log n)$ within $O(\log n)$ time.

When we are inserting a edge with end nodes having different root, we need to combine the two trees. For insertion (or combination) of two trees: split $a$ or $b$ (randomly).

Note: For all path related issues, use treaps.

When insert($r, x$), and $x < r$.

We can have the combined tree as in figure 2 which is balanced.

![Figure 2: Combine two trees](image)

2 Dynamic Trees, heavy-light decomposition, amortized analysis.

Path Decompositions of trees:

1. Each non-leaf node assigns/picks a preferred child, forms paths.
2. Each path maintained using a binary search tree.
3. Key quantity: number of different paths encountered along some root-to-leaf path.

Walk up:

1. Find root of $x$.
2. Repeatly: go from $x$ to the start of its path, take parent pointer.

Heavy-light decompositions:

**Lemma 1.** In any tree, can pick preferred children of nodes so that any node to root path has $O(\log n)$ non-preferred children.
1. $size(p)$, represents size of a node $p$, which means the number of nodes in its subtree.

2. For the current tree, rooted with node $p$, assign preferred child to the node of maximum size.

3. Each non-preferred child $q$: $size(q) \leq \frac{1}{2} size(p)$, so at most $O(\log n)$ of these along any path.

3 Expander Decompositions

The generalization of this to graphs is expander decompositions.

An $(\epsilon, \alpha)$-expander decomposition \cite{SW19} of a graph $G = (V, E)$ with $m$ edges is a partition of vertices into clusters such that

- each cluster induces subgraph with conductance at least $\alpha$, where $\alpha$ is the minimum conductance of the clusters.
  
  \[
  \frac{\text{vol} (E(S, \overline{S}))}{\min\{|S|, |\overline{S}|\}} 
  \]

- the number of inter-cluster edges is at most $\epsilon$, where $\epsilon$, the ratio of the weight of inter-cluster edges to the total weight of all edges

1. Simplification: ignore weights and degrees: $\phi(S) \approx \frac{|E(S,\overline{S})|}{\min\{|S|, |\overline{S}|\}}$.

2. Sparsest cut problem: find, or approximate, $\phi(G) = \min_{S \subseteq V} \phi(S)$.

3. Applications: expander partitioning, segmentation.

Expander decomposition: decompose so that each piece has conductance at least $\alpha$.

1. $E(S, V \setminus S)$: set of edges leaving $S$, $\text{vol}(S)$: total degree in $S$.

2. If $E(S, V \setminus S) \leq \beta \text{vol}(S)$, put $S$ into its own cluster.

3. Repeat until termination.

4. Can ensure: size of $S$ halves at each step.

5. Charge each cut edge to the volume of piece: $\alpha \log nm$ edges cut.

References