# Centroid Decomposition

Jason Chiu

February 11, 2015

# Overview

1. The Problem
   - Problem description
   - Solution
   - Proof of correctness
   - Time complexity
2. Centroid Decomposition
3. Example Problem

# Problem description

Let $T$ be an undirected tree. Find a node $v$ such that if we delete $v$ from the tree, spliting it into a forest, each of the trees in the forest would all have fewer than half the number of vertices from the original tree.

# Solution

Let $T$ be the given undirected tree with $n$ nodes.

1. Root the tree arbitrarily
2. Perform DFS to obtain, for every node $v$, the size of the subtree rooted at $v$: $S(v) = 1 + \sum_i S(adj[v][i])$
3. For each node $v$, check if
   $\max(n - S(v), S(adj[v][0]), S(adj[v][1]), \cdots) < n/2$
   halt and return $v$ if this is satisfied

Note: we can combine steps 2 and 3 into a single DFS.

# Proof of correctness

**Theorem.** There is always a solution

1. If the root works, great
2. If the root doesn't work, then we can recurse on the lop-sided subtree, because the other piece must be $< n/2$
3. Maximum subtree size gets smaller, so it must terminate eventually

**Theorem.** The algorithm produces a solution
This is obvious from the description of the algorithm.

# Time complexity

$O(n)$ to compute the size of subtrees, and $O(n)$ to find the correct node, because the cost of the node search is $\sum_{v \in V}(1 + \deg(v)) = 2n - 1$. Therefore, the time complexity is $O(n)$.

# Easy! But why?

# Centroid Decomposition

The solution of the previous problem finds a node $v$ which we shall call a **centroid** of the tree. Now what happens if we apply the algorithm recursively to each subtree split by the centroid?

# Centroid Decomposition

- We get a tree of centroids, which we shall call the **centroid decomposition** of the tree.
- Runtime is $O(n \log n)$ because we will recurse at most $\log_2 n$ times.
- Notice this decomposition has $\log n$ depth, so we can essentially do divide and conquer on the tree!

# Questions?

# Problem (IOI 2011)

Given a weighted tree with $N$ nodes, find the minimum number of edges in a path of length $K$, or return $-1$ if such a path does not exist.

- $1 \leq N \leq 200000$
- $1 \leq \text{length}(i, j) \leq 1000000$ (integer weights)
- $1 \leq K \leq 1000000$

# Solution

Brute force solution:

- For every node, perform DFS to find distance and number of edges to every other node
- Time complexity: $O(n^2)$

Obviously fails because $N = 200000$.

# Solution

Better solution:

- Perform centroid decomposition to get a "tree of subtrees"
- Start at the root of the decomposition, solve the problem for each subtree as follows
  - Solve the problem for each "child tree" of the current subtree
  - Perform DFS from the centroid on the **current subtree** to compute the minimum edge count for paths that include the centroid
    - Two cases: centroid at the end or in the middle of path
    - Use a timestamped array of size 1000000 to keep track of which distances from centroid are possible and the minimum edge count for that distance
  - Take the minimum of the above two

Time complexity: $O(n \log n)$

# The End