香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Graph (V)

Matthew Chau {happychau}

2023-05-27

Introduction

Sometimes there are tree problems that can be solved efficiently by certain properties.

In today's lecture, we will take a look at Centroid Decomposition and Heavy-light Decomposition, algorithms that will allow us to break down the trees for our own convenience.

Our Powerful Weapons

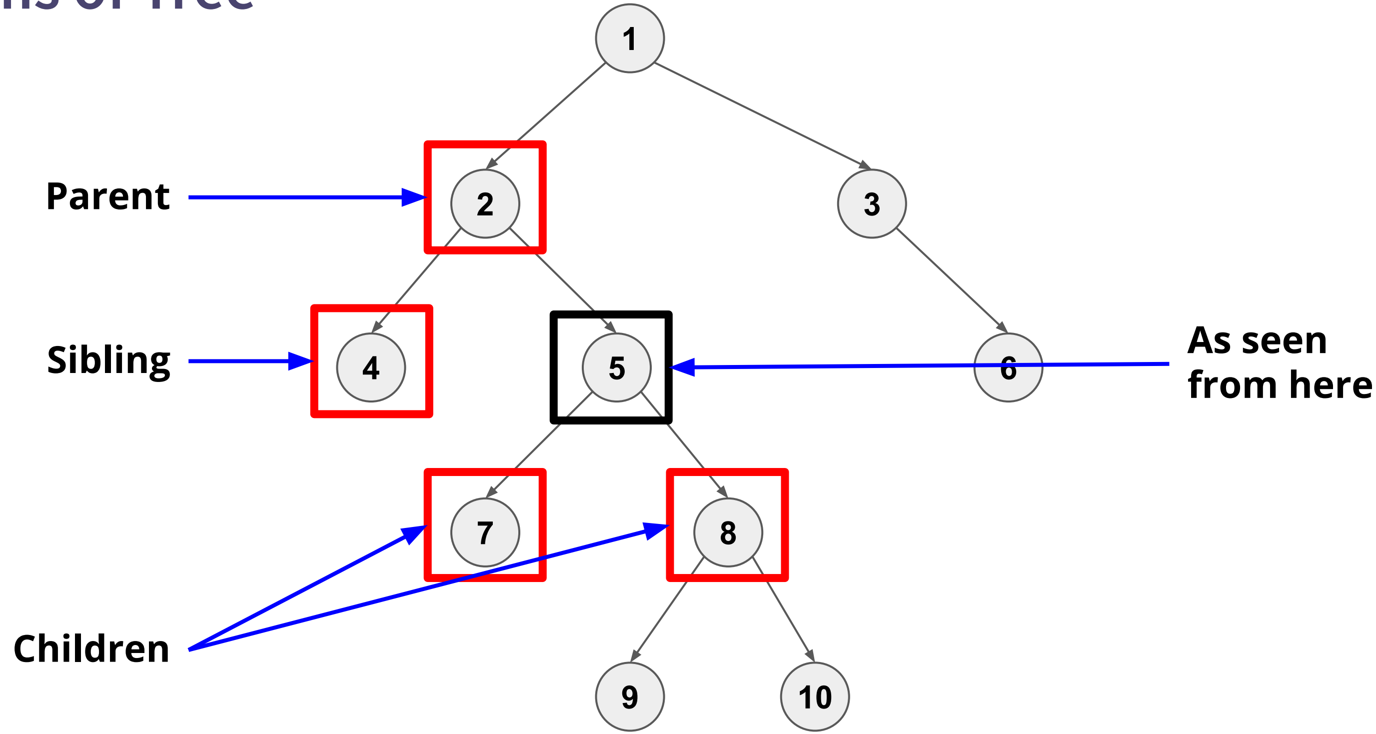
- Still DFS!

Revision: Terms of Tree

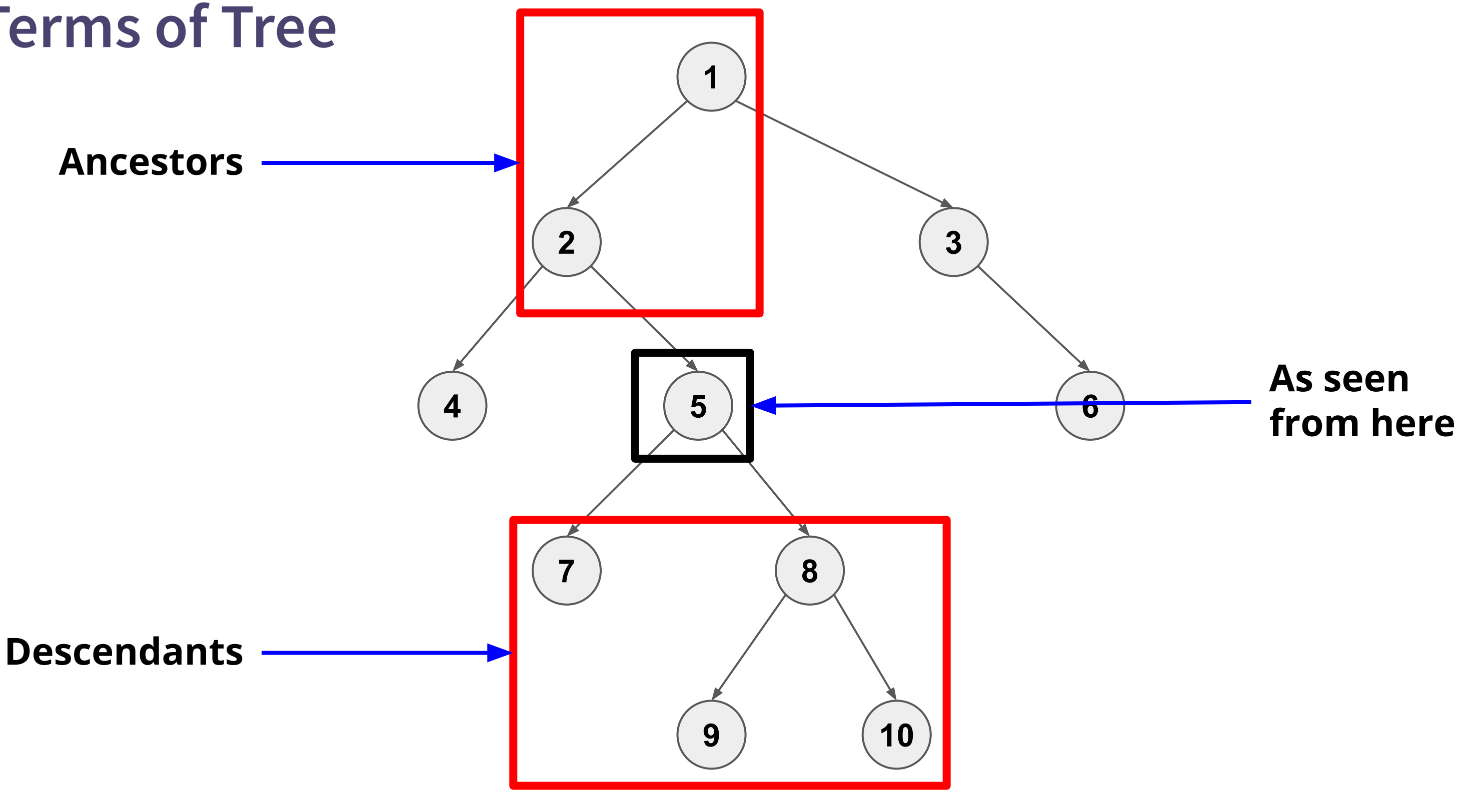
Refer to Graph (IV) (2023)

https://assets.hkoi.org/training2023/g-iv.pdf

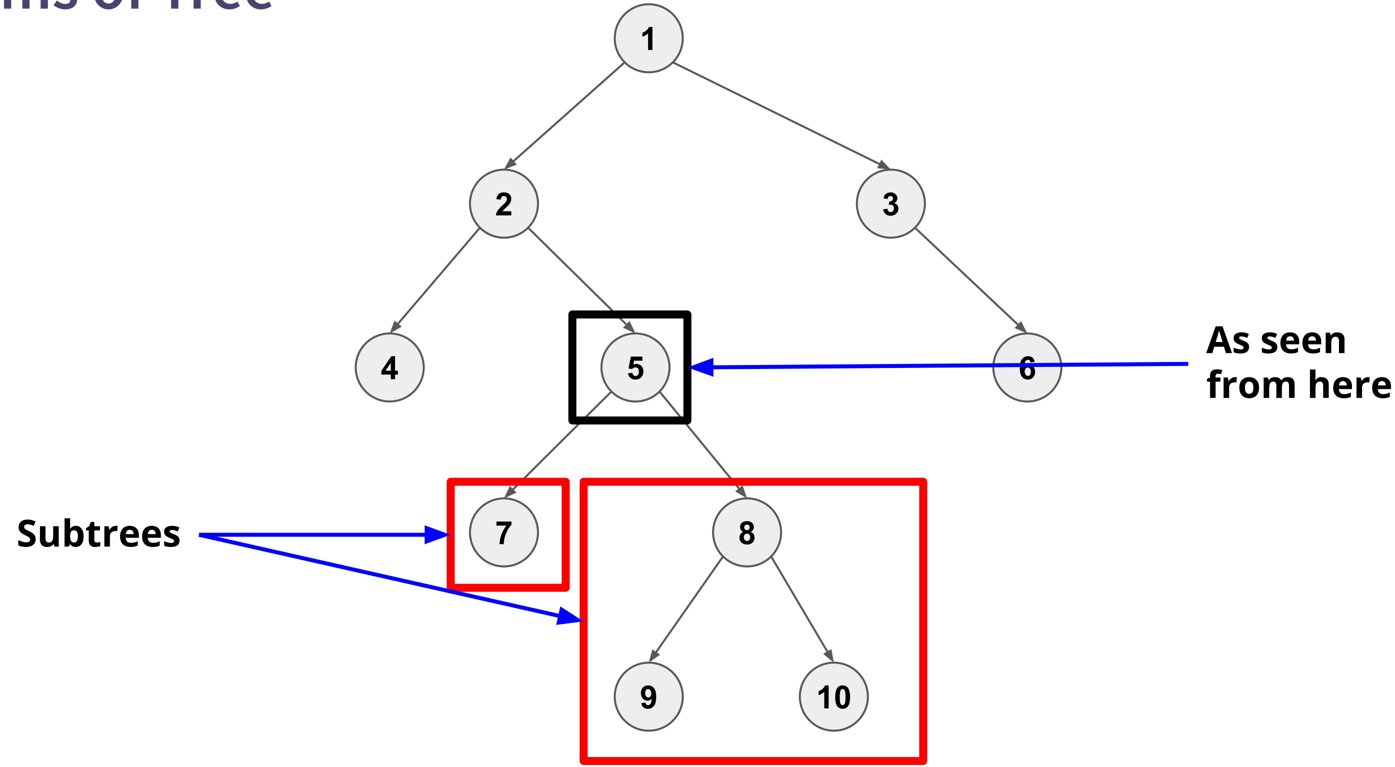
Revision: Terms of Tree



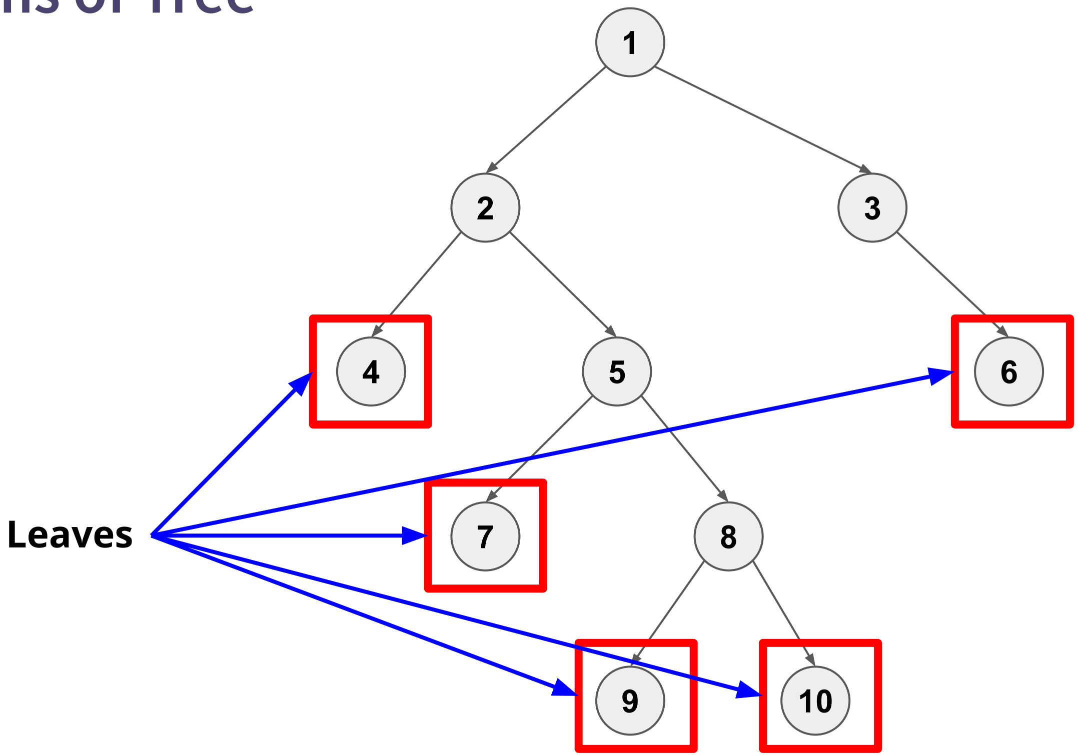
Revision: Terms of Tree



Revision: Terms of Tree



Revision: Terms of Tree



Revision: DFS

```
vector<vector<int>> G;  // Adjacency List
vector<bool> vis;

void dfs(int u) {
  vis[u] = true;
  for (int v : G[u])
    if (!vis[v])
      dfs(v);
}
```

Centroid Decomposition

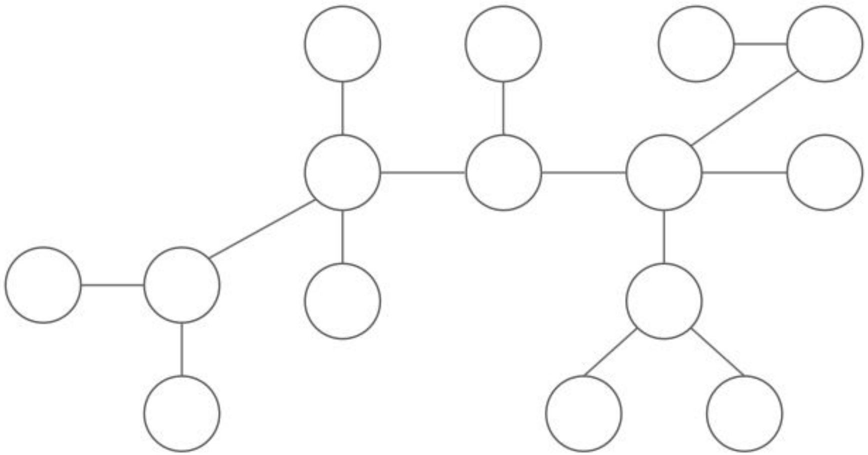
Centroid on tree

A **centroid** of a tree is defined as:

- a node such that when the tree is rooted at it, no other nodes have a subtree of size greater than N/2.

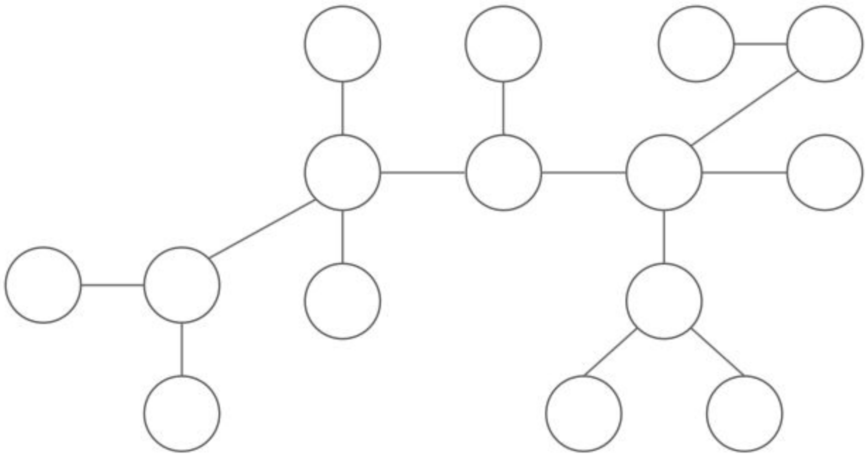
Centroid on tree

A visual example: which is centroid?



Centroid on tree

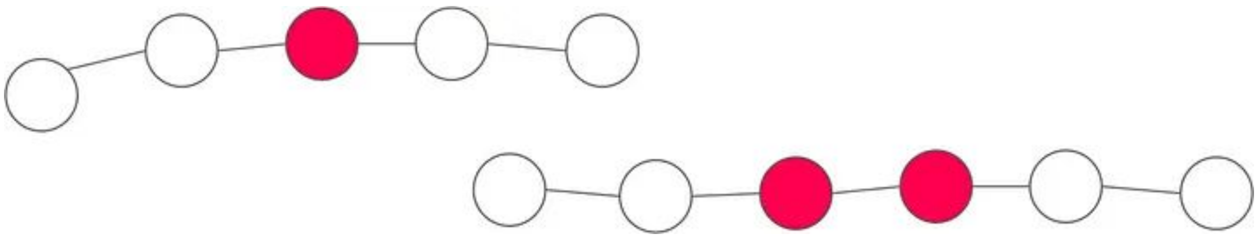
Answer:



Centroid vs Center

Centroid is different from the **center** of a tree:

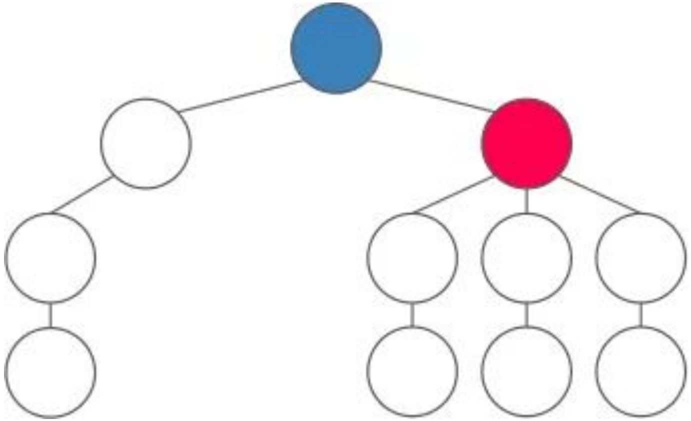
- the middle nodes (either 1 or 2) in every longest path along the tree.



Centroid vs Center

Example:

The red node is the centroid but the blue node is the center.



Why?

It may be unintuitive why we would need centroid decomposition.

Let's take a look at CF342E: https://codeforces.com/problemset/problem/342/E

E. Xenia and Tree

time limit per test: 5 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

Xenia and Tree

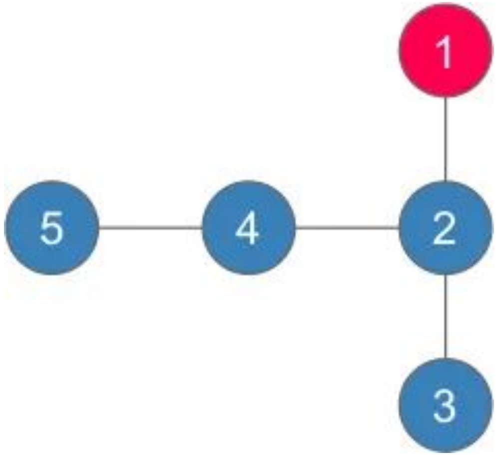
Problem statement:

- Consider a tree with nodes indexed from 1 to n.
- The first node is initially painted red, and the other nodes painted blue.
- Two types of operation:
  a. Update(u): Paint blue node u red
  b. Query(u): Find the distance to the closest red node for node u
- n, # of queries <= 10^5

Xenia and Tree

Example:

n = 5 as follow

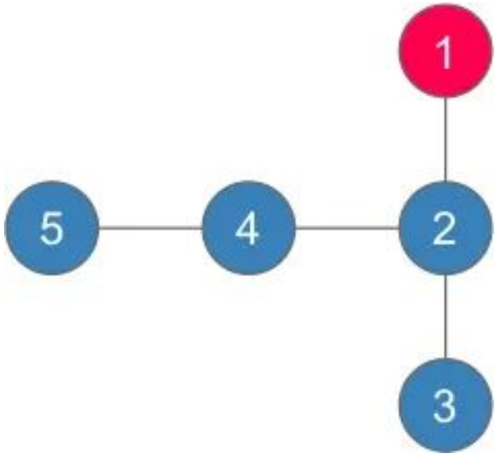


Xenia and Tree

Example:

n = 5 as follow

query(5) = ?

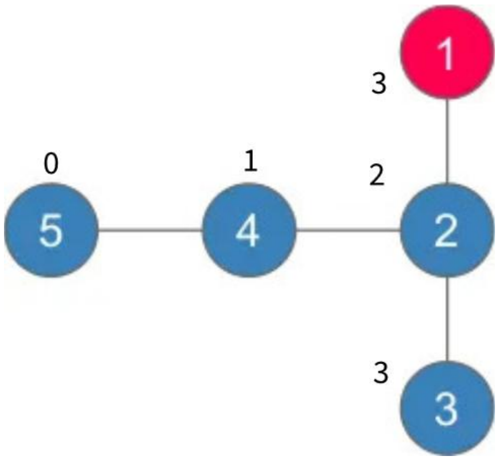


Xenia and Tree

Example:

n = 5 as follow

query(5) = 3



Xenia and Tree

How do we approach this problem?

Xenia and Tree

How do we approach this problem?

1. BFS/DFS on query

Xenia and Tree

How do we approach this problem?

1. BFS/DFS on query

```
int query(int u, int p) {
    if (colour[u] == 1) return 0;
    int mn = INF;
    for (auto v : tree[u])
        if (v != p)
            mn = min(mn, query(v, u));
    return mn + 1;
}

void update(int a) {
    colour[a] = 1;
}
```

Xenia and Tree

How do we approach this problem?

1. BFS/DFS on query
   a. Update: O(1)
   b. Query: O(n)

This surely will TLE

```
int query(int u, int p) {
    if (colour[u] == 1) return 0;
    int mn = INF;
    for (auto v : tree[u])
        if (v != p)
            mn = min(mn, query(v, u));
    return mn + 1;
}

void update(int a) {
    colour[a] = 1;
}
```

Xenia and Tree

How do we approach this problem?

1. BFS/DFS on query
   a. Update: O(1)
   b. Query: O(n)
2. BFS/DFS on update

```
int query(int u) {
    return ans[u];
}

void update(int u, int p, int d) {
    ans[u] = min(ans[u], d);
    for (auto v : tree[u])
        if (v != p)
            update(v, u, d + 1);
}
```

Xenia and Tree

How do we approach this problem?

1. BFS/DFS on query
   a. Update: O(1)
   b. Query: O(n)
2. BFS/DFS on update
   a. Update: O(n)
   b. Query: O(1)

This will also TLE

```
int query(int u) {
    return ans[u];
}

void update(int u, int p, int d) {
    ans[u] = min(ans[u], d);
    for (auto v : tree[u])
        if (v != p)
            update(v, u, d + 1);
}
```

Xenia and Tree

Is it possible to balance the two operations?

Xenia and Tree

Is it possible to balance the two operations?

- Centroid Decomposition

Finding centroid

Recall the definition of **centroid**:

- a node such that when the tree is rooted at it, no other nodes have a subtree of size greater than N/2.

How to find the centroid with program?

Finding centroid

Consider a node u:

Finding centroid

Consider a node u:

- If all of its neighbor nodes have subtree size <= N/2:
  - Centroid

Finding centroid

Consider a node u:

- If all of its neighbor nodes have subtree size ≤ N/2:
  - Centroid
- Otherwise:
  - there will only be one neighbor node with subtree size > N/2

Finding centroid

Therefore we propose an algorithm:

1. Arbitrarily take a node as root

Finding centroid

Therefore we propose an algorithm:

1. Arbitrarily take a node as root
2. Calculate the subtree sizes

Finding centroid

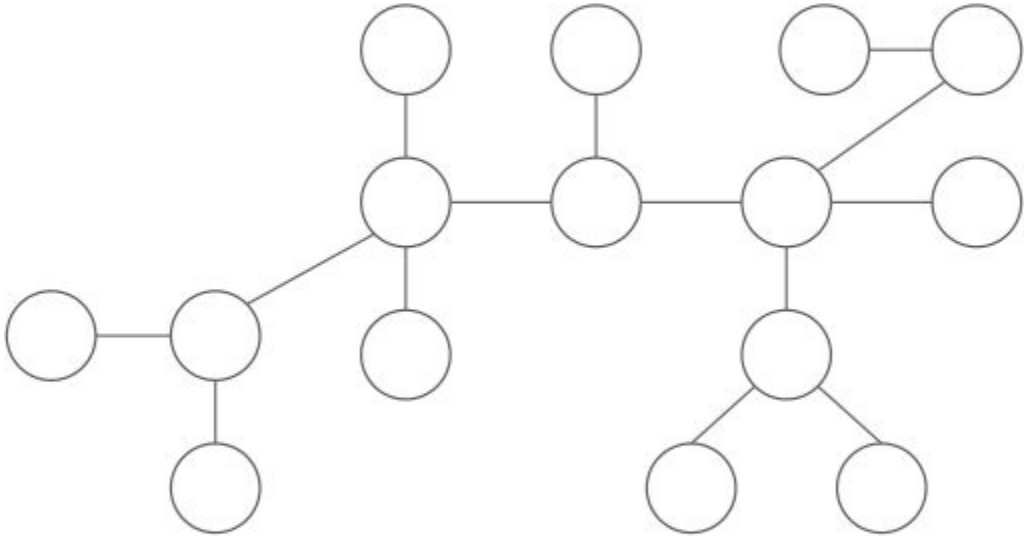
Therefore we propose an algorithm:

1. Arbitrarily take a node as root
2. Calculate the subtree sizes
3. Start considering from root node:
    a. If a neighbor node have subtree size > N/2: consider such node
    b. Otherwise the current node is centroid

Finding centroid

Visualization:

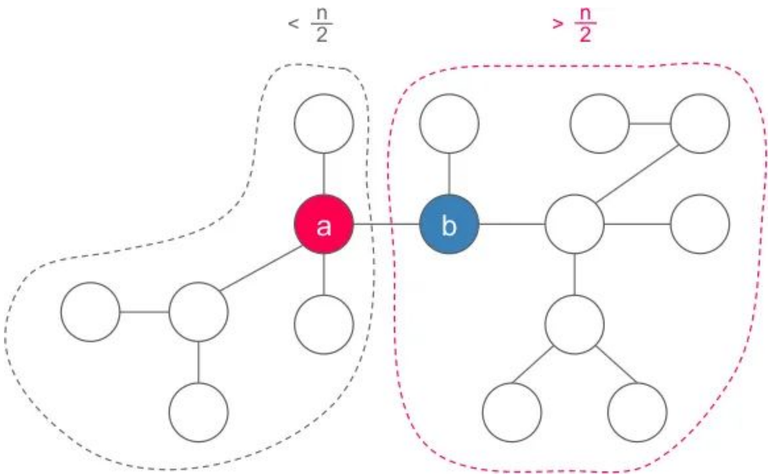
1. Arbitrarily root a node
2. Calculate the subtree sizes
3. Start considering from root node:
    a. If a neighbor node have subtree size > N/2: consider such node
    b. Otherwise the current node is centroid



Finding centroid

Some idea on why this work:

- it doesn’t visit a visited node (it doesn’t go back from b to a).
  - If it does, it will visit a node with subtree size < N/2.



Finding centroid

Let's code this out

1. Precompute
    a. DFS!
2. Search

```
int dfs(int u, int p) {
    for (auto v : tree[u])
        if (v != p) sub[u] += dfs(v, u);
    return sub[u] + 1;
}

int centroid(int u, int p) {
    for (auto v : tree[u])
        if (v != p and sub[v] > n/2) return centroid(v, u);
    return u;
}
```

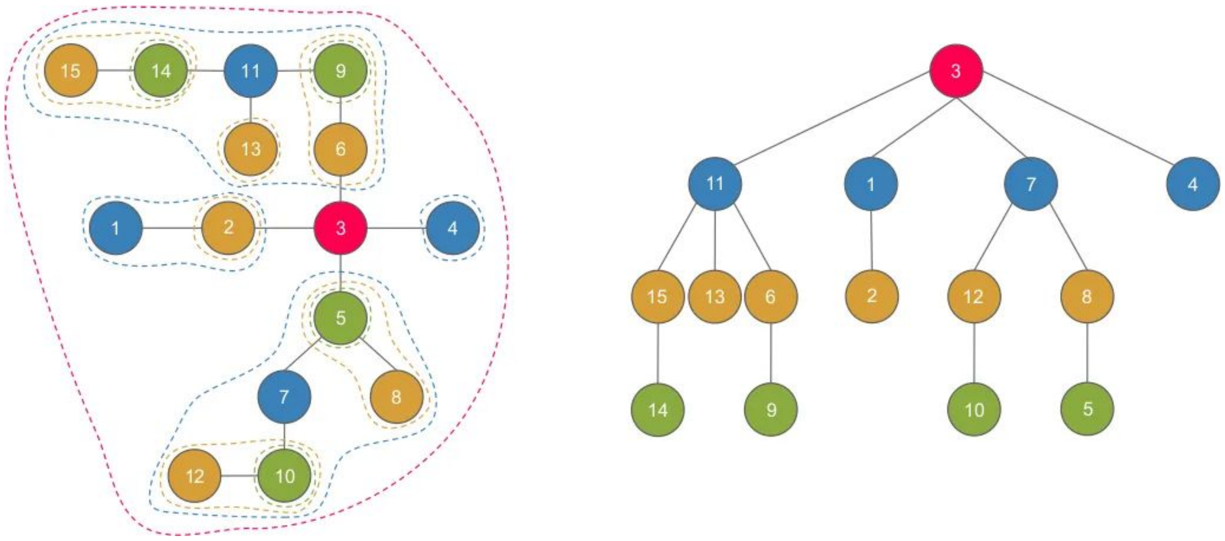
Centroid Decomposition

The centroid decomposition of a tree is another tree defined recursively as:

- Its root is the centroid of the original tree.
- Its children are the centroid of each tree resulting from the removal of the centroid from the original tree.

Centroid Decomposition

Visualization:



Centroid Decomposition

To code that: apply the definitions

```
void build(int u, int p) {
   int n = dfs(u, p); // find the size of each subtree
   int cent = centroid(u, p); // find the centroid
   if (p == -1) p = cent; // dad of root is the root itself
   dad[cent] = p;

   // for each tree resulting from the removal of the centroid
   for (auto v : tree[cent])
       tree[cent].erase(v), // remove the edge to disconnect
       tree[v].erase(cent), // the component from the tree
       build(v, cent);
}
```

Centroid Decomposition

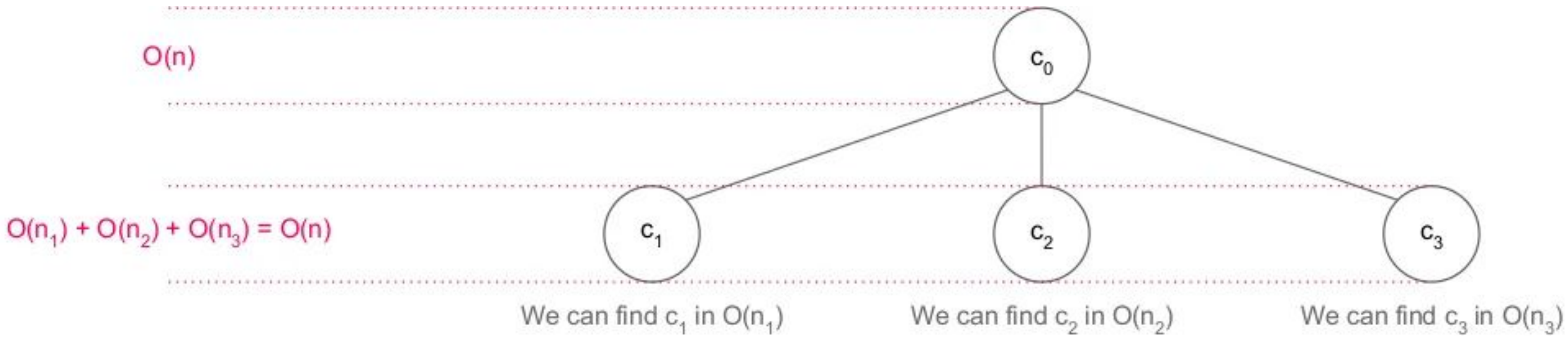
Time complexity: O(n log(n))

- Similar to merge sort, the maximum depth for decomposition is log n.
- Each level contains less than n nodes in total

Centroid Decomposition

Time complexity: O(n log(n))

- Similar to merge sort, the maximum depth for decomposition is log n.
- Each level contains less than n nodes in total



Properties of Centroid Decomposition

What can we do with the results?

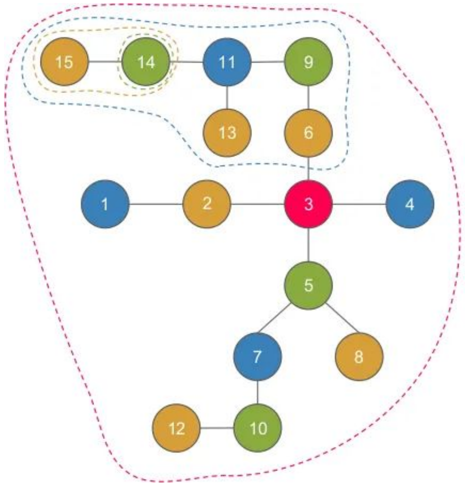
Properties of Centroid Decomposition

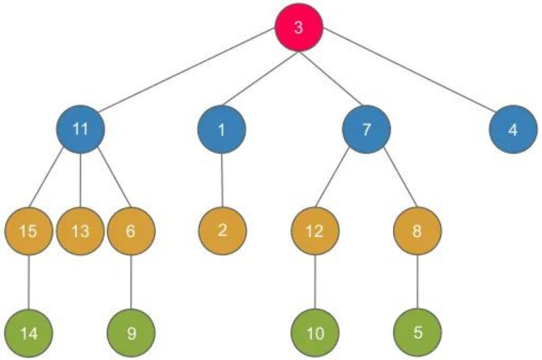
Property 1: A vertex belongs to the component of all its ancestors.

Properties of Centroid Decomposition

Property 1: A vertex belongs to the component of all its ancestors.

- The node 14 belongs to the component of 14, 15, 11 and 3.





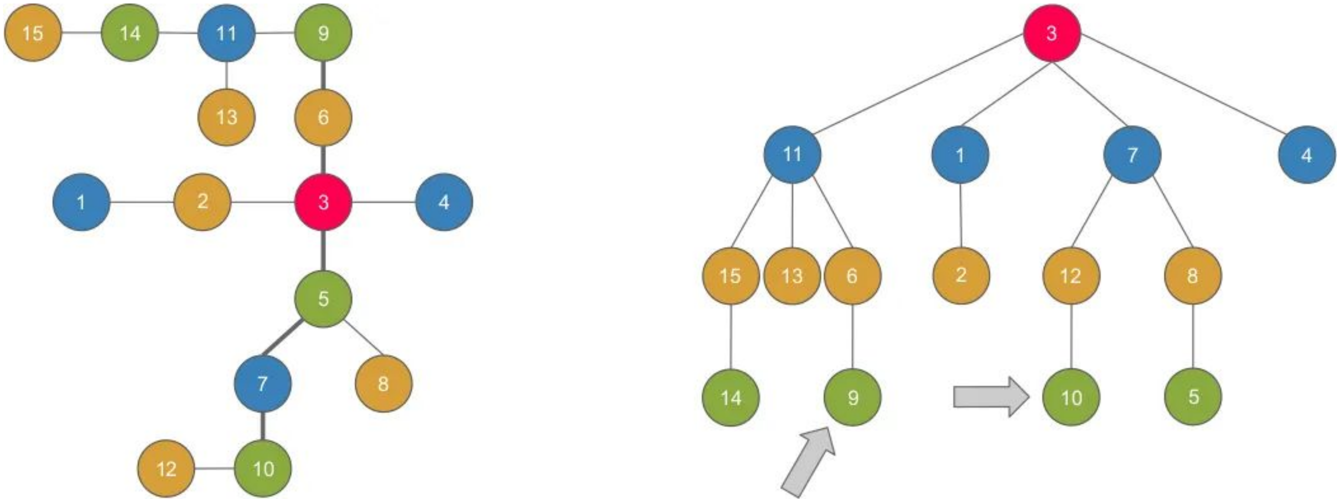
Properties of Centroid Decomposition

Property 2: the path from a to b can be decomposed into the path from a to lca(a,b) and the path from lca(a,b) to b.

Properties of Centroid Decomposition

Property 2: the path from a to b can be decomposed into the path from a to lca(a,b) and the path from lca(a,b) to b.

- The path from 9 to 10 in the original tree can be decomposed into the path from 9 to 3 and the path from 3 to 10.



Properties of Centroid Decomposition

Property 2: the path from a to b can be decomposed into the path from a to lca(a,b) and the path from lca(a,b) to b.

Proof:

- Both a and b belong to the component where the node lca(a,b) is the centroid.

Properties of Centroid Decomposition

Property 2: the path from a to b can be decomposed into the path from a to lca(a,b) and the path from lca(a,b) to b.

Proof:

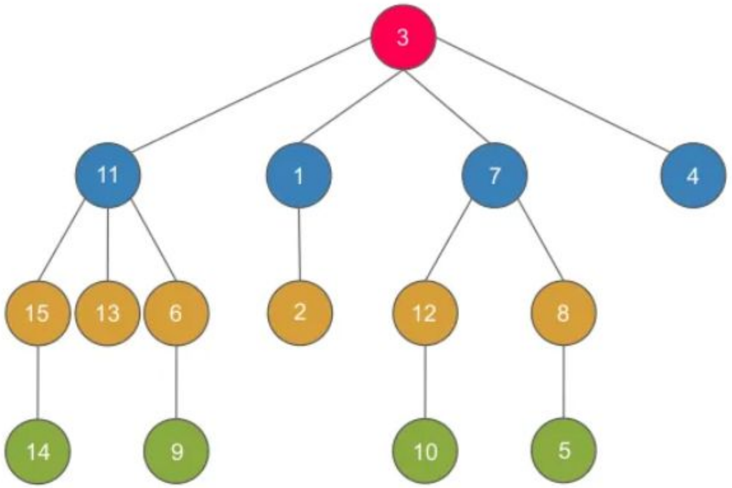
- Both a and b belong to the component where the node lca(a,b) is the centroid.
  - i.e. removal of lca(a,b) will split them into different components.

Properties of Centroid Decomposition

Property 3: Each one of the n^2 paths of the original tree is the concatenation of two paths in a set of $O(n \log(n))$ paths from a node to all its ancestors in the centroid decomposition.

Properties of Centroid Decomposition

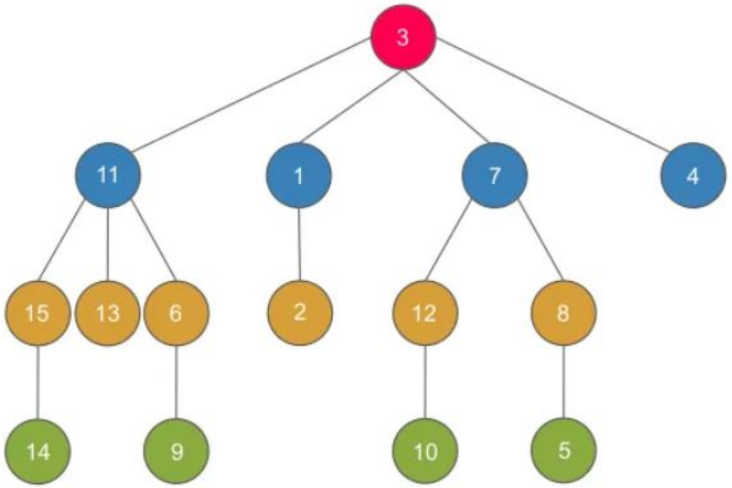
Take node 14 as example, to reach node a:



Properties of Centroid Decomposition

Take node 14 as example, to reach node a:

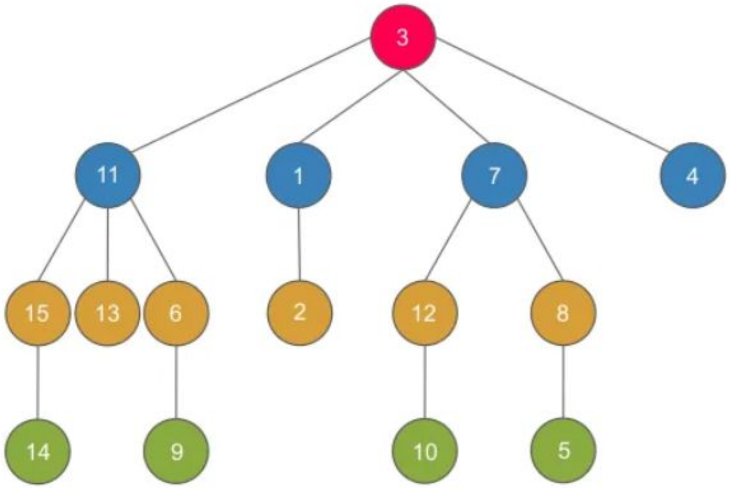
- a = {14}: (14, 14) + (14, a)



Properties of Centroid Decomposition

Take node 14 as example, to reach node a:

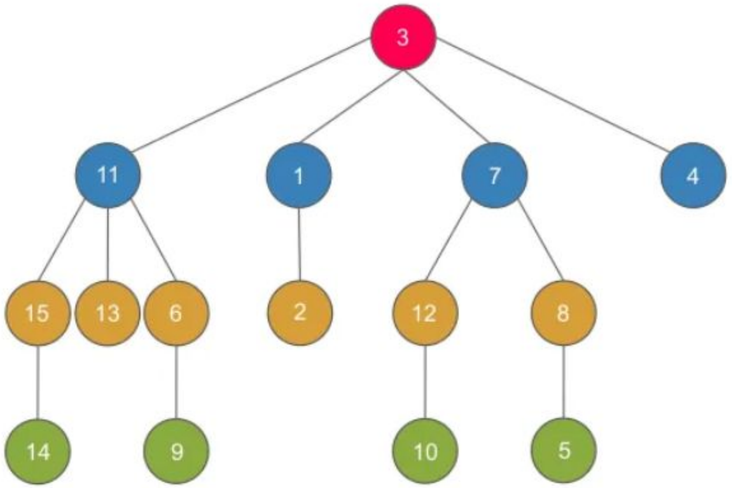
- a = {14}: (14, 14) + (14, a)
- a = {15}: (14, 15) + (15, a)



Properties of Centroid Decomposition

Take node 14 as example, to reach node a:

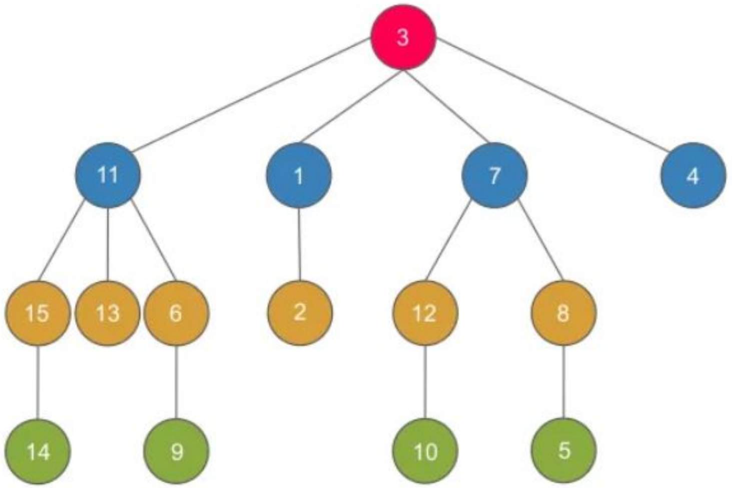
- a = {14}: (14, 14) + (14, a)
- a = {15}: (14, 15) + (15, a)
- a = {6, 9, 13}: (14, 11) + (11, a)



Properties of Centroid Decomposition

Take node 14 as example, to reach node a:

- a = {14}: (14, 14) + (14, a)
- a = {15}: (14, 15) + (15, a)
- a = {6, 9, 13}: (14, 11) + (11, a)
- a = {1, 2, 4, 5, 6, 7, 10, 12}: (14, 3) + (3, a)

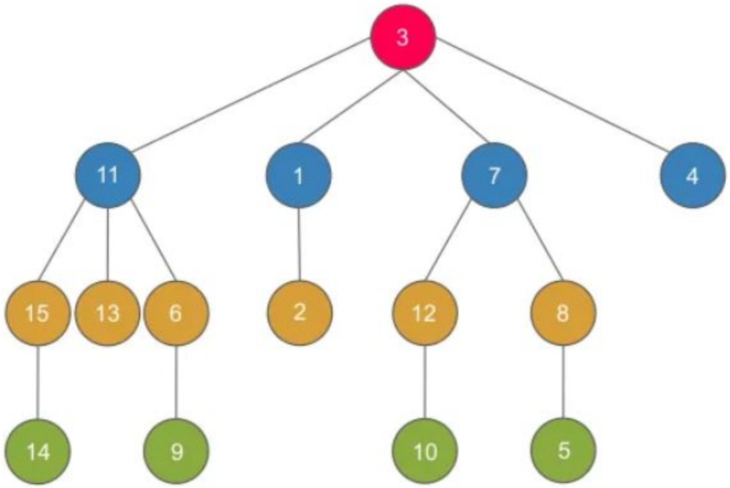


Properties of Centroid Decomposition

Take node 14 as example, to reach node a:

- a = {14}: (14, 14) + (14, a)
- a = {15}: (14, 15) + (15, a)
- a = {6, 9, 13}: (14, 11) + (11, a)
- a = {1, 2, 4, 5, 6, 7, 10, 12}: (14, 3) + (3, a)

As we can see, they can all be expressed as two paths, where one of them passes through 14's four ancestors.



Properties of Centroid Decomposition

Property 3: Each one of the n^2 paths of the original tree is the concatenation of two paths in a set of O(n log(n)) paths from a node to all its ancestors in the centroid decomposition.

- As each node contains at most log(n) ancestors, the total number of paths is n*log(n).

Xenia and Tree

Back to the problem.

How do we optimize the problem with centroid decomposition?

Xenia and Tree

Define ans[a] as the distance to the closest red node to a in the component where node a is centroid.

Xenia and Tree

Define ans[a] as the distance to the closest red node to a in the component where node a is centroid.

- Initially, ans[a] = ∞ because all nodes are blue (we'll update the first node before reading the operations).

Xenia and Tree

Define ans[a] as the distance to the closest red node to a in the component where node a is centroid.

- Initially, ans[a] = ∞ because all nodes are blue (we'll update the first node before reading the operations).

For update:

- Assume node a becomes red now
- It will only affect the ancestor of a
  => ans[b] = min(ans[b], dist(a, b)) for all ancestor b of a

Xenia and Tree

Define ans[a] as the distance to the closest red node to a in the component where node a is centroid.

- Initially, ans[a] = ∞ because all nodes are blue (we'll update the first node before reading the operations).

For update:

- Assume node a becomes red now
- It will only affect the ancestor of a

  => ans[b] = min(ans[b], dist(a, b)) for all ancestor b of a

log(n) ancestor * log(n) dist calculation = O(log(n)^2)

Xenia and Tree

Define ans[a] as the distance to the closest red node to a in the component where node a is centroid.

- Initially, ans[a] = ∞ because all nodes are blue (we'll update the first node before reading the operations).

For query:

- We can consider all nodes by ancestors of a
- ans = min(dist(a,b) + ans(b)) for all ancestor b of a
    - ans[b]: answer from b to other nodes c that lca(a,c) = b

Xenia and Tree

Define ans[a] as the distance to the closest red node to a in the component where node a is centroid.

- Initially, ans[a] = ∞ because all nodes are blue (we'll update the first node before reading the operations).

For query:

- We can consider all nodes by ancestors of a!
- ans = min(dist(a,b) + ans(b)) for all ancestor b of a
    - ans[b]: answer from b to other nodes c that lca(a,c) = b

log(n) ancestor * log(n) dist calculation = O(log(n)^2) as well

Xenia and Tree

Through centroid decomposition, we can split every possible paths into two paths that are easier to manage.

Related problems

- IOI'11 - Race
- 321C - Ciel the Commander
- 766E - Mahmoud and a xor trip
- 716E - Digit Tree
- 161D - Distance in Tree
- 776F - Sherlock's bet to Moriarty
- 379F - New Year Tree
- 342E - Xenia and Tree
- 293E - Close Vertices
- 150E - Freezing with Style
- 348E - Pilgrims
- Codechef - Prime Distance On Tree

Reference

https://medium.com/carpanese/an-illustrated-introduction-to-centroid-decomposition-8c1989d53308

Questions?

break;

Heavy-light Decomposition

Heavy Light Decomposition

- A way to split a tree into several paths
- Each node can reach the root node through at most log(n) paths

Why

It would allows us to effectively solve many problems that require queries on a tree .

NOI'21: Heavy-light edge

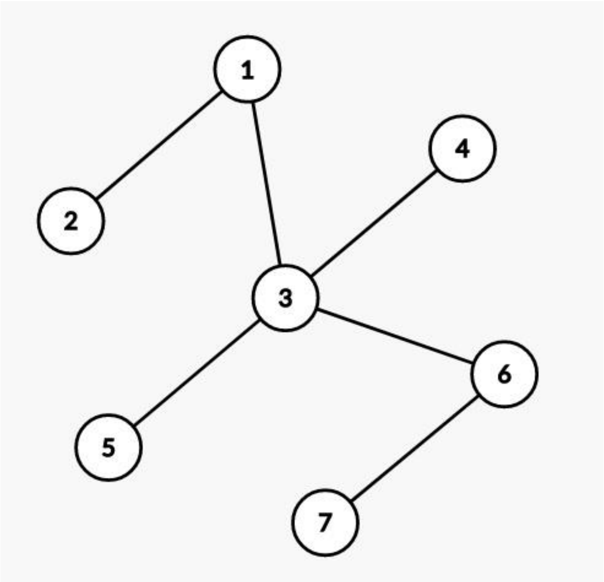
Given a tree with n node. Initially, all edges are "light edge".

There are two types of operations:

1. update(a,b): for all node x on the path from a to b, first assign all connected edges of x as "light edge", then assign the path from node a to node b as "heavy edge".
2. query(a,b): count how many edges from node a to node b are "heavy edge".

NOI’21: Heavy-light edge

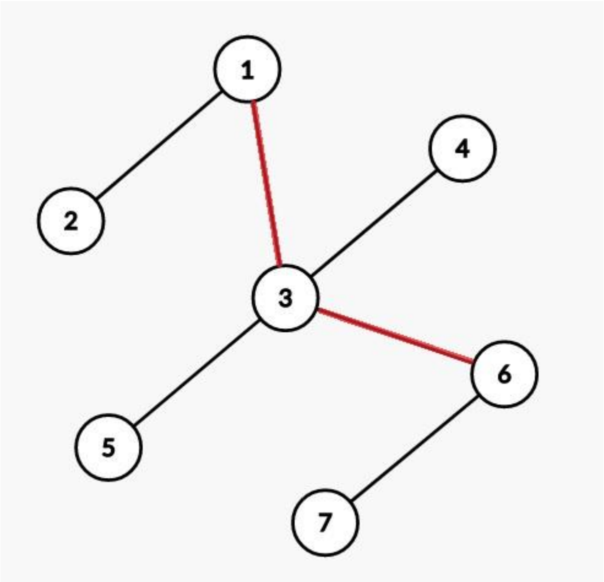
Example: consider the following tree:



NOI'21: Heavy-light edge

Example: consider the following tree:

update(1,6);

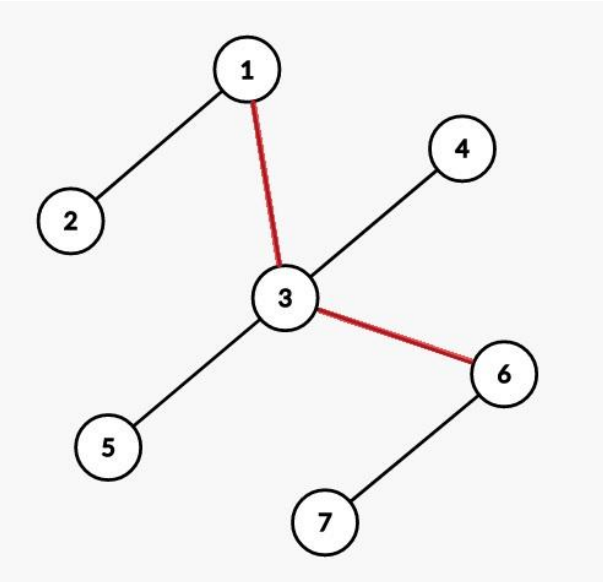


NOI'21: Heavy-light edge

Example: consider the following tree:

update(1,6);

query(2,4) = ?

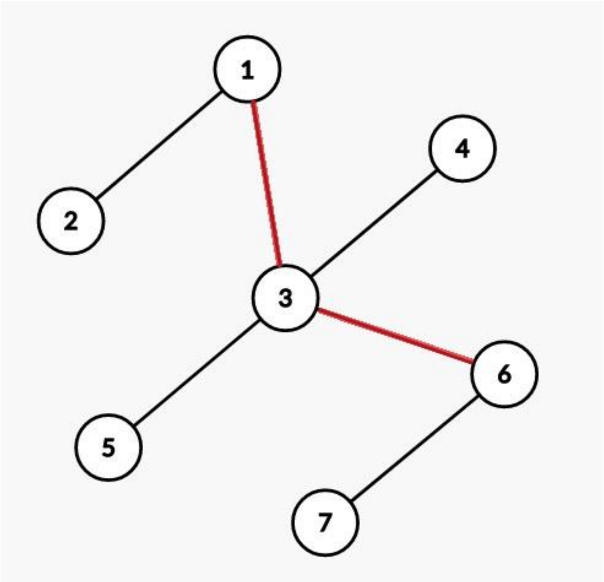


NOI’21: Heavy-light edge

Example: consider the following tree:

update(1,6);

query(2,4) = 1



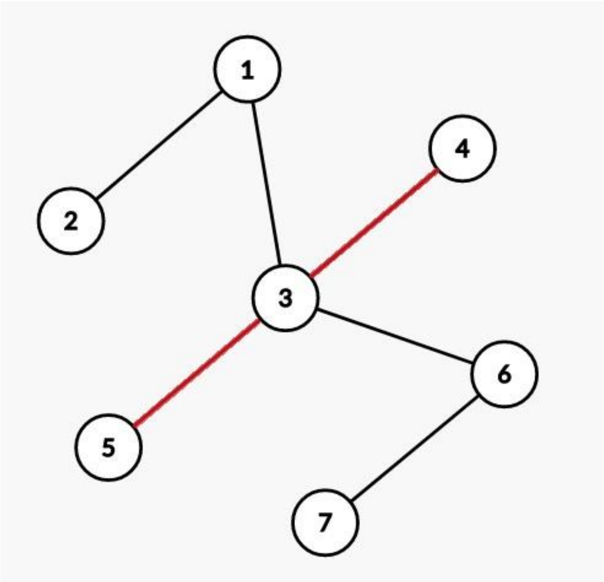
NOI’21: Heavy-light edge

Example: consider the following tree:

update(1,6);

query(2,4) = 1

update(4,5);



NOI’21: Heavy-light edge

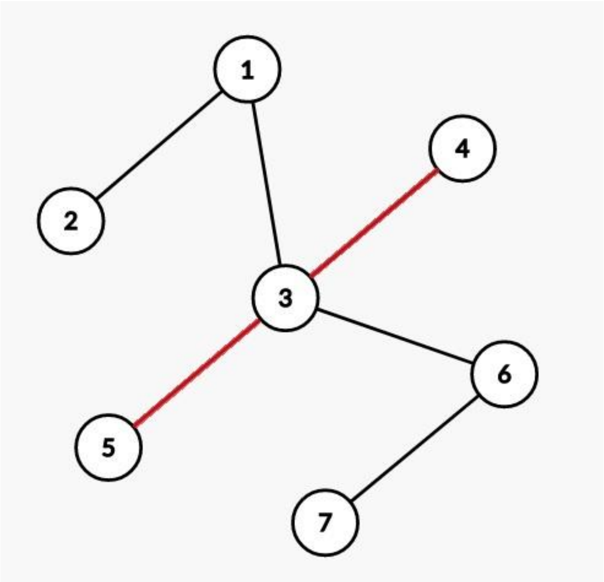
Example: consider the following tree:

update(1,6);

query(2,4) = 1

update(4,5);

query(2,4) = ?



NOI’21: Heavy-light edge

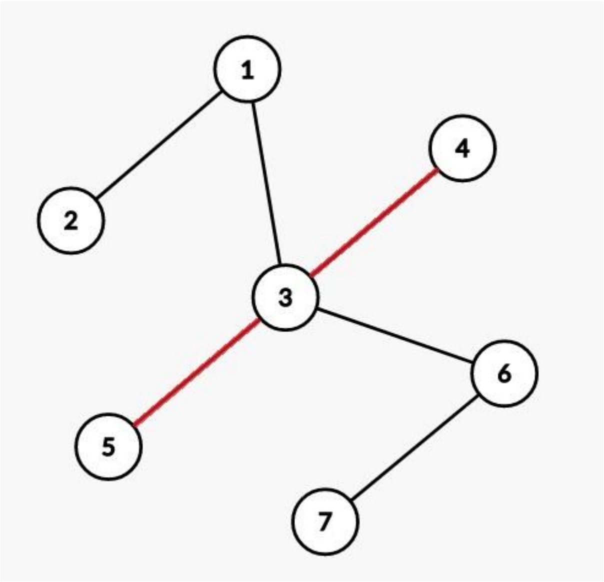
Example: consider the following tree:

update(1,6);

query(2,4) = 1

update(4,5);

query(2,4) = 1

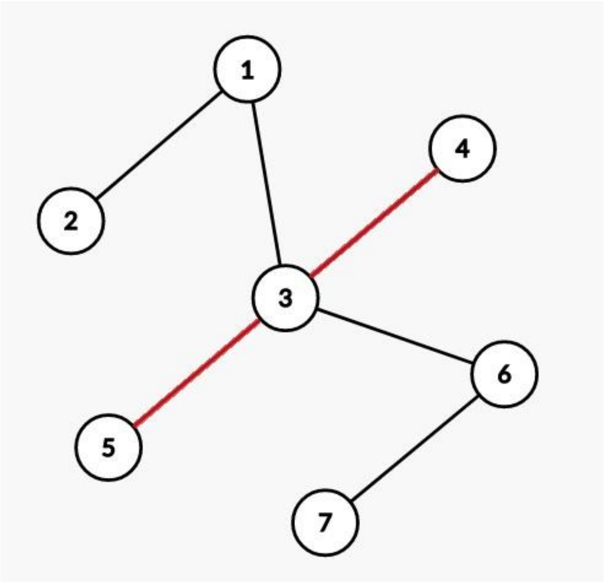


NOI’21: Heavy-light edge

Naive solution: update the paths accordingly.

Time complexity: O(n)

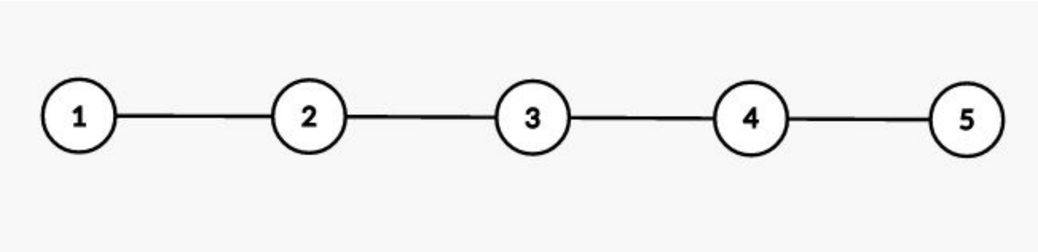
TLE :(



NOI'21: Heavy-light edge

This looks hard, let's figure out how to solve it on a simpler problem.

Consider a chain:

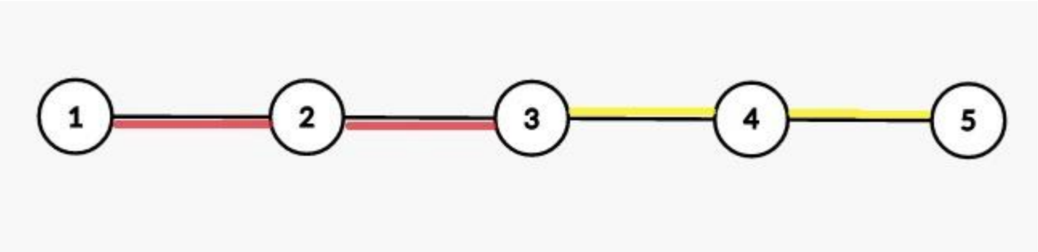


NOI'21: Heavy-light edge

update(1,3)

update(3,5)

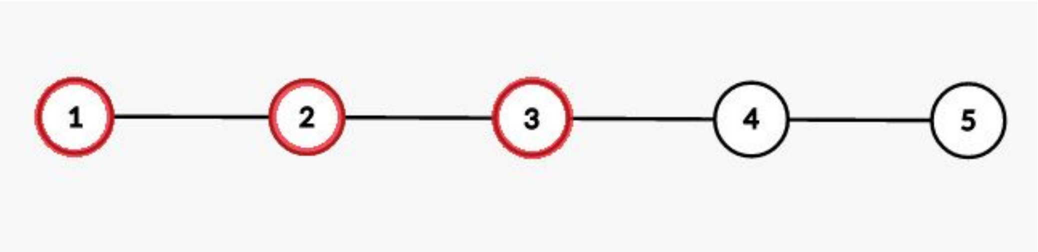
How can we handle them?



NOI’21: Heavy-light edge

Instead of updating edges, consider colouring the nodes:

update(1,3);

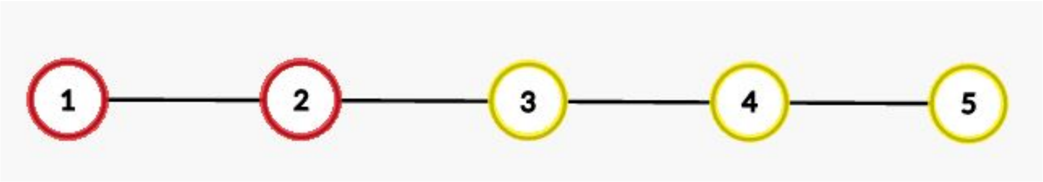


NOI’21: Heavy-light edge

Instead of updating edges, consider colouring the nodes:

update(1,3);

update(3,5);



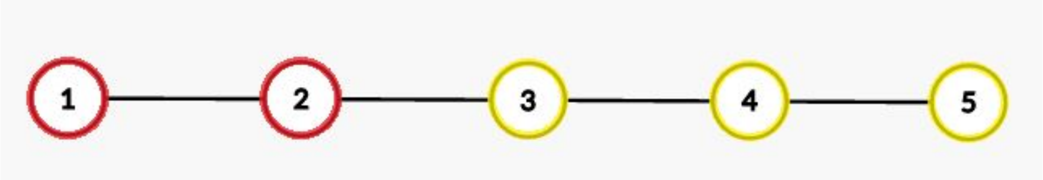
NOI'21: Heavy-light edge

Instead of updating edges, consider colouring the nodes:

update(1,3);

update(3,5);

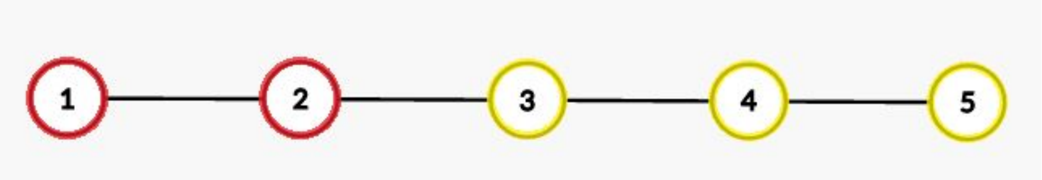
Number of heavy edges becomes: number of adjacent nodes with same colour



NOI’21: Heavy-light edge

This can be solved with segment tree.

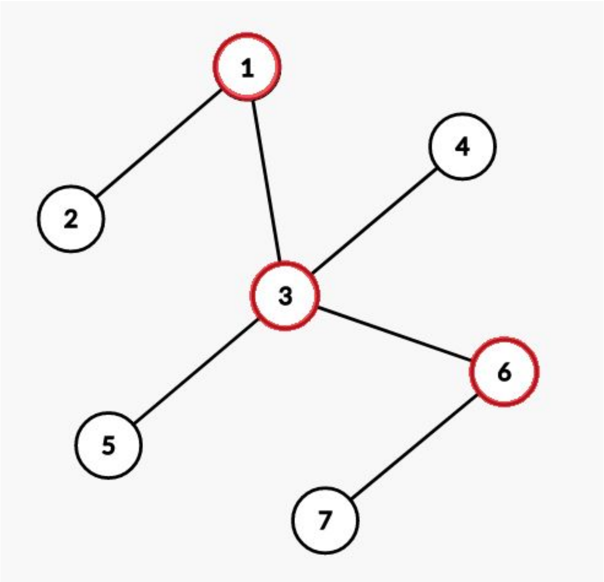
Time complexity = O(log(n))



NOI’21: Heavy-light edge

But how do we solve this problem on tree?

Can we divide the paths into segments of chains?

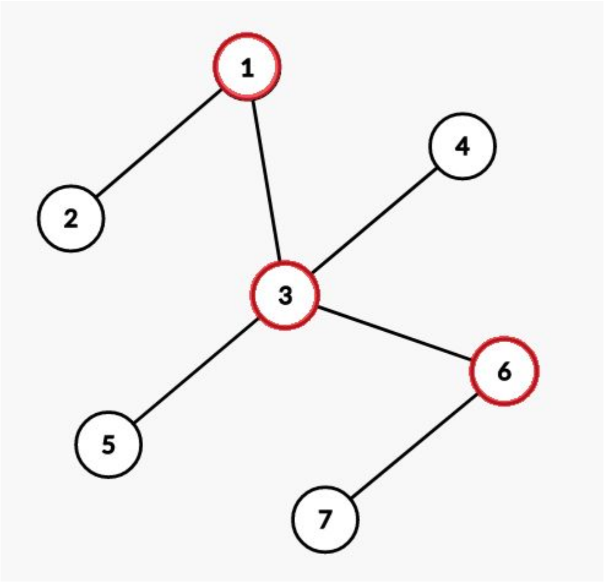


NOI'21: Heavy-light edge

But how do we solve this problem on tree?

Can we divide the paths into segments of chains?

Yes!



Heavy-light decomposition

This is why we need heavy-light decomposition.

Recall what it does:

- Split a tree into several paths
- Each node can reach the root node through at most log(n) paths

Heavy-light decomposition

This is why we need heavy-light decomposition.

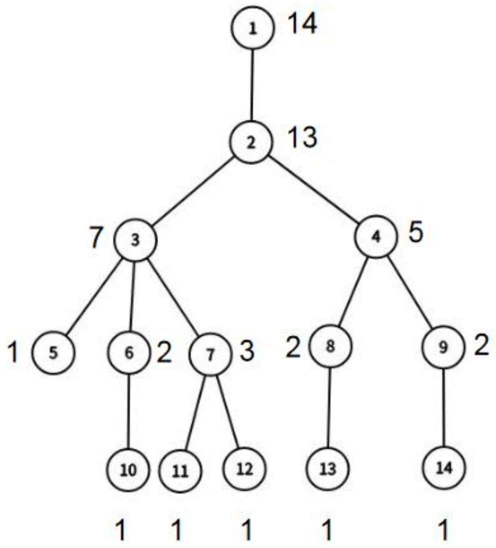
Recall what it does:

- Split a tree into several paths
- Each node can reach the root node through at most log(n) paths

Heavy-light decomposition

To understand how it works, let’s start with some definitions:

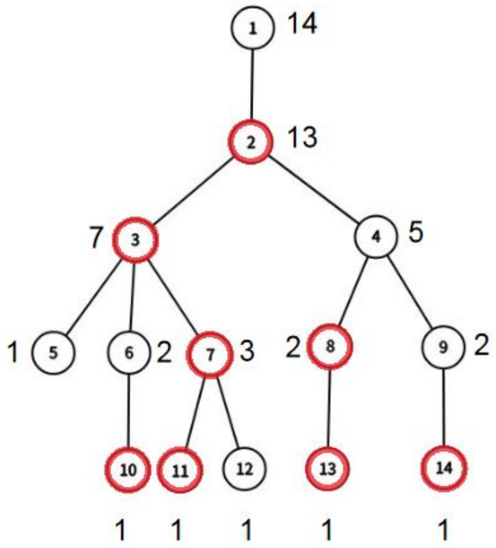
Assume the tree is rooted and we know every subtree sizes, then for a node u:



Heavy-light decomposition

To understand how it works, let's start with some definitions:
Assume the tree is rooted and we know every subtree sizes, then for a node u:

- Heavy edge: the edge to the child with the largest subtree size.

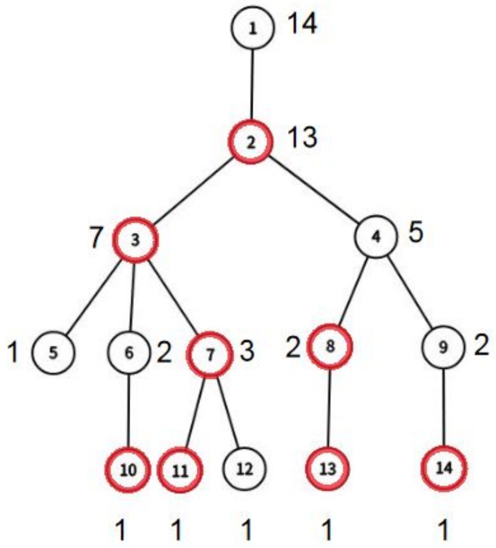


Heavy-light decomposition

To understand how it works, let's start with some definitions:

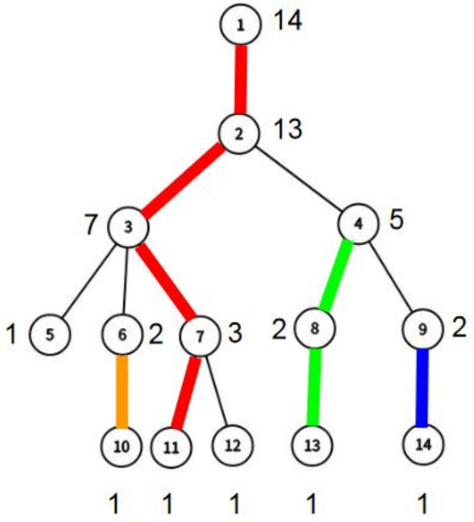
Assume the tree is rooted and we know every subtree sizes, then for a node u:

- Heavy edge: the edge to the child with the largest subtree size.
- Light edge: edges that are not heavy.



Heavy-light decomposition

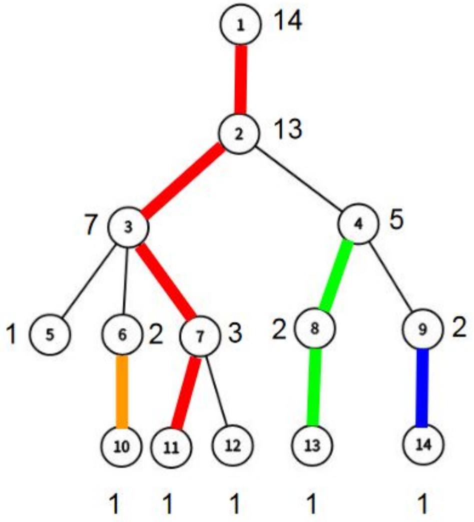
These are all the heavy edges labelled.



Heavy-light decomposition

These are all the heavy edges labelled.

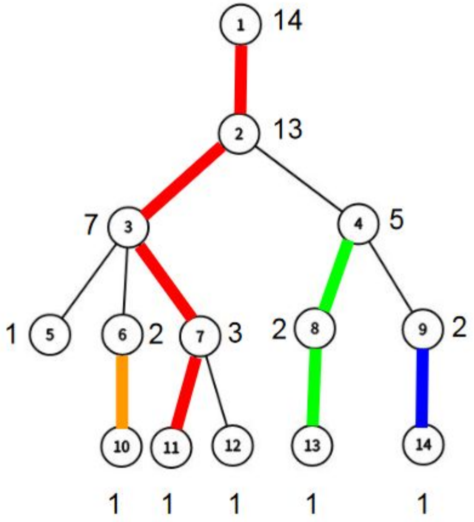
DFS on the tree with heavy edges first: 1 2 3 7 11 12 6 10 5 4 8 13 9 14



Heavy-light decomposition

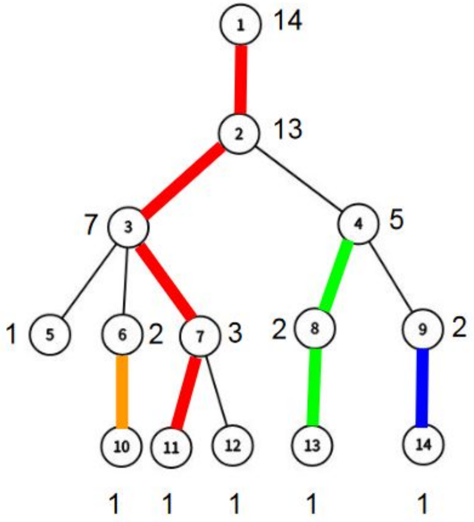
Consider visiting nodes from root:

How many light edge will we go through?



Heavy-light decomposition

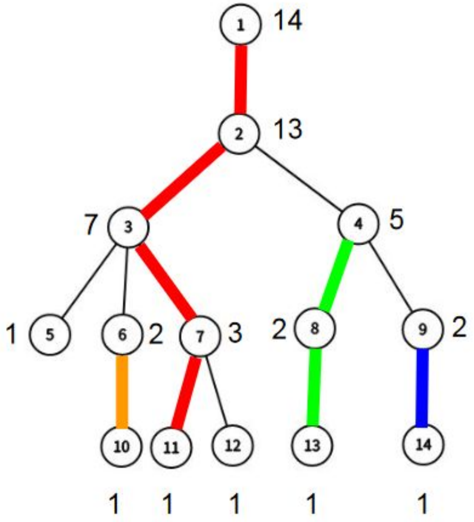
Observe that a light edge must connect to a node with subtree size halve that of the parent node.



Heavy-light decomposition

Observe that a light edge must connect to a node with subtree size halve that of the parent node.

- Otherwise it will be the heavy edge.



Heavy-light decomposition

Observe that a light edge must connect to a node with subtree size halve that of the parent node.

- Otherwise it will be the heavy edge.

Therefore every simple path from root at most will go through log(n) light edges.

Heavy-light decomposition

Let's try and code that out:

Compute the subtree sizes: DFS

```
void calc_size(int id) {
    size[id] = 1;
    for (int i = 0; i < edge[id].size(); ++i) if (edge[id][i] != parent[id]) {
        parent[edge[id][i]] = id;
        calc_size(edge[id][i]);
        size[id] += size[edge[id][i]];
        if (size[edge[id][i]] > size[edge[id][0]]) swap(edge[id][i],edge[id][0]);
    }
}
```

Heavy-light decomposition

Let's try and code that out:

Compute the subtree sizes: DFS

- We also mark the heavy edges for convenience.

```
void calc_size(int id) {
    size[id] = 1;
    for (int i = 0; i < edge[id].size(); ++i) if (edge[id][i] != parent[id]) {
        parent[edge[id][i]] = id;
        calc_size(edge[id][i]);
        size[id] += size[edge[id][i]];
        if (size[edge[id][i]] > size[edge[id][0]]) swap(edge[id][i],edge[id][0]);
    }
}
```

Heavy-light decomposition

Let's try and code that out:

Decompose the tree into an array: DFS by heavy edges first

- Save the order we visit the nodes into an array

```
void hld(int id) {
    a[++cnt] = id;
    st[id] = cnt;
    for (int i = 0; i < edge[id].size(); ++i) if (edge[id][i] != parent[id]) {
        head[edge[id][i]] = i == 0 ? head[id] : edge[id][i];
        hld(edge[id][i]);
    }
    ed[id] = cnt;
}
```

Heavy-light decomposition

Let's try and code that out:

Decompose the tree into an array: DFS by heavy edges first

- Save the order we visit the nodes into an array
- Save the starting and ending position of each heavy paths

```
void hld(int id) {
    a[++cnt] = id;
    st[id] = cnt;
    for (int i = 0; i < edge[id].size(); ++i) if (edge[id][i] != parent[id]) {
        head[edge[id][i]] = i == 0 ? head[id] : edge[id][i];
        hld(edge[id][i]);
    }
    ed[id] = cnt;
}
```

Heavy-light decomposition

Let's try and code that out:

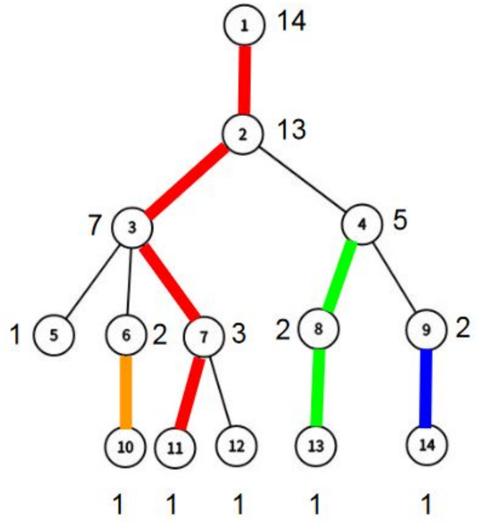
Decompose the tree into an array: DFS by heavy edges first

- Save the order we visit the nodes into an array
- Save the starting and ending position of each heavy paths
- Mark the parent node of each starting node into head[]

```
void hld(int id) {
    a[++cnt] = id;
    st[id] = cnt;
    for (int i = 0; i < edge[id].size(); ++i) if (edge[id][i] != parent[id]) {
        head[edge[id][i]] = i == 0 ? head[id] : edge[id][i];
        hld(edge[id][i]);
    }
    ed[id] = cnt;
}
```

Heavy-light decomposition

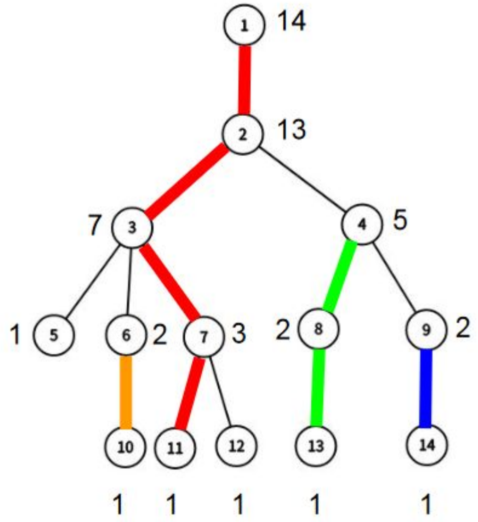
Now we have decomposed the tree, and every path to ancestors can be expressed in several heavy paths:



Heavy-light decomposition

Now we have decomposed the tree, and every path to ancestors can be expressed in several heavy paths:

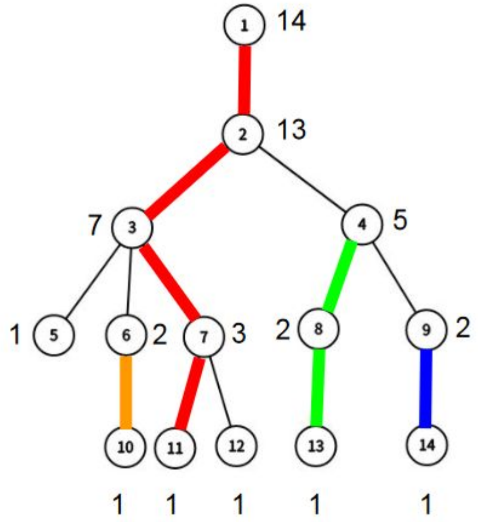
- 7 - 2: 7 - 2
- 10 - 2: 10 - 6, 3 - 2
- 14 - 1: 14 - 9, 4 - 4, 2 - 1



Heavy-light decomposition

Paths to other nodes can also be expressed into 2 paths to their lca.

- 7 - 10: 7 - 3 + 3 - 10
  - 7 - 3: 7 - 3
  - 10 - 3: 10 - 6, 3 - 3



Heavy-light decomposition

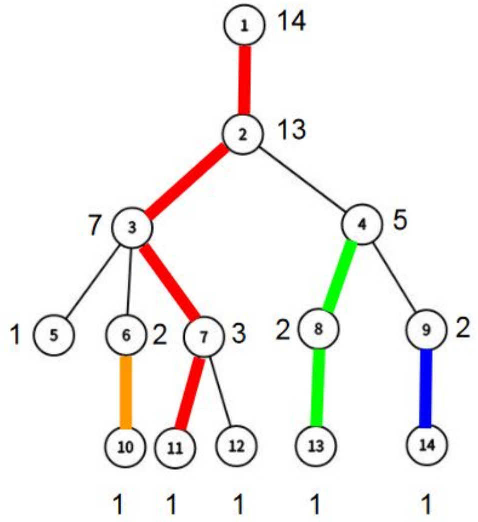
Paths to other nodes can also be expressed into 2 paths to their lca.

Time complexity:

- Lca: O(log(n))
- Retrieving path to ancestor: O(log(n))

Overall: O(log(n))

Implementation is somewhat long and will be left as exercise for readers.



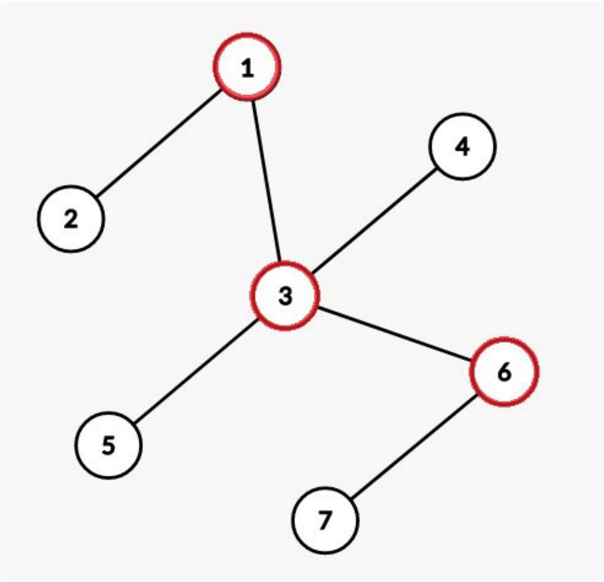
NOI’21: Heavy-light edge

Back to the problem.

We can now split each update and query into log(n) continuous segments.

Which can then be handled by segment tree.

Time complexity: O(log(n)^2)



Tips

Sometimes although a task can be solved by heavy light decomposition, it actually can be solved using some simpler ways.

Tips

Sometimes although a task can be solved by heavy light decomposition, it actually can be solved using some simpler ways.

Example: Given a tree

- Update: Change the value of a node
- Query: Find the sum of the values on the path between two nodes

Tips

Sometimes although a task can be solved by heavy light decomposition, it actually can be solved using some simpler ways.

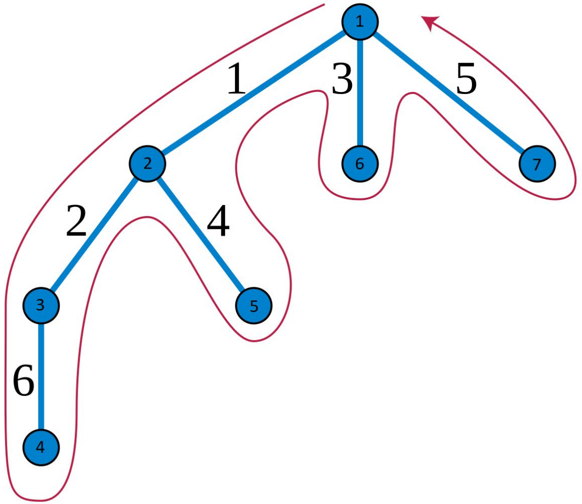
Example: Given a tree

- Update: Change the value of a node
- Query: Find the sum of the values on the path between two nodes
  - Store the value of the nodes in Euler tour
  - Additionally store the the value in negative when leaving the node
  - Use segment tree to query or update

Tips

- Store the value of the nodes in Euler tour
- Additionally store the the value in negative when leaving the node
- Use segment tree to query or update

Modified Euler tour = 1, 2, 6, -6, -2, 4, -4, -1, 3, -3, 5, -5

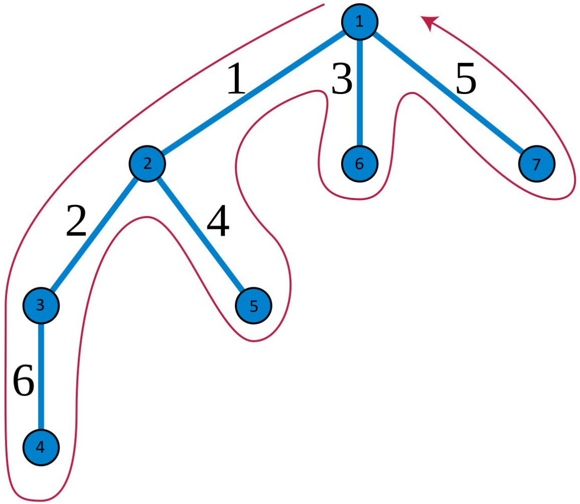


Tips

- Store the value of the nodes in Euler tour
- Additionally store the the value in negative when leaving the node
- Use segment tree to query or update

Modified Euler tour = 1, 2, 6, -6, -2, 4, -4, -1, 3, -3, 5, -5

(2, 6) = (1, 2) + (1, 6) = (1) + (1, 2, 6, -6, -2, 4, -4, -1, 3)
= 4



Related Problems

- Path Queries (CSES)
- QTREE (SPOJ): allows you to test modifications for edges
- GRASSPLA (SPOJ; original source is USACO but the judge doesn't work for that problem)
- GSS7 (SPOJ)
- QRYLAND (CodeChef)
- MONOPLOY (CodeChef)
- QUERY (CodeChef)
- BLWHTREE (CodeChef)
- Milk Visits (USACO)
- Max Flow (USACO)
- Exercise Route (USACO)

Questions?

end.