Persistent Centroid Decomposition

Special class

Tanuj Khattar • Nov 22, 2020

Persistent Centroid Decomposition

Course: https://unacademy.com/a/i-p-c-advanced-track

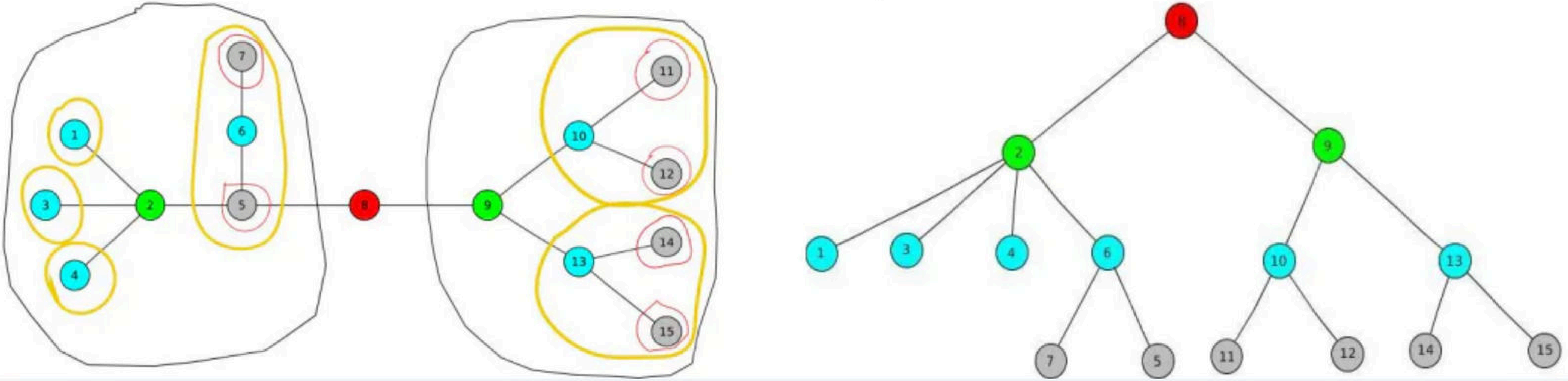
tanujkhattar@

Objective

- Centroid Decomposition - Quick Revision
- Problems on Centroid Decomposition
  - Discuss simpler problems.
  - Modify to motivate the need for Persistent Centroid Tree
- Persistent Centroid Tree
  - Key Ideas
  - Challenges due to high degree of a single node
- Binarizing the Input Tree
  - Adding dummy nodes & 0 weight edges to binarize input tree
  - Centroid tree of a Binary Tree would also be binary (each node has <= 3 children)
- Making the Centroid Tree Persistent
  - Use Path-Copying Persistence to make the Centroid Tree Persistent
  - Handle adjacent swap updates in Path-Copying persitence
- Conclusion

Centroid Decomposition - Quick Revision

- The Centroid Tree is constructed by decomposing the original tree by
  - Find Centroid and make it the root of the centroid tree.
  - Delete the centroid and connected edges.
  - Recurse on new smaller subtrees and attach them as children of root in centroid tree.
- The Centroid Tree has a height of O(logN) and represents O(NlogN) “special paths” which start at a centroid and go to other nodes in it’s component.
- Any path A--B in original tree can be written as concatenation of two special paths A -- C and C -- B where C is LCA of A, B in the centroid tree.



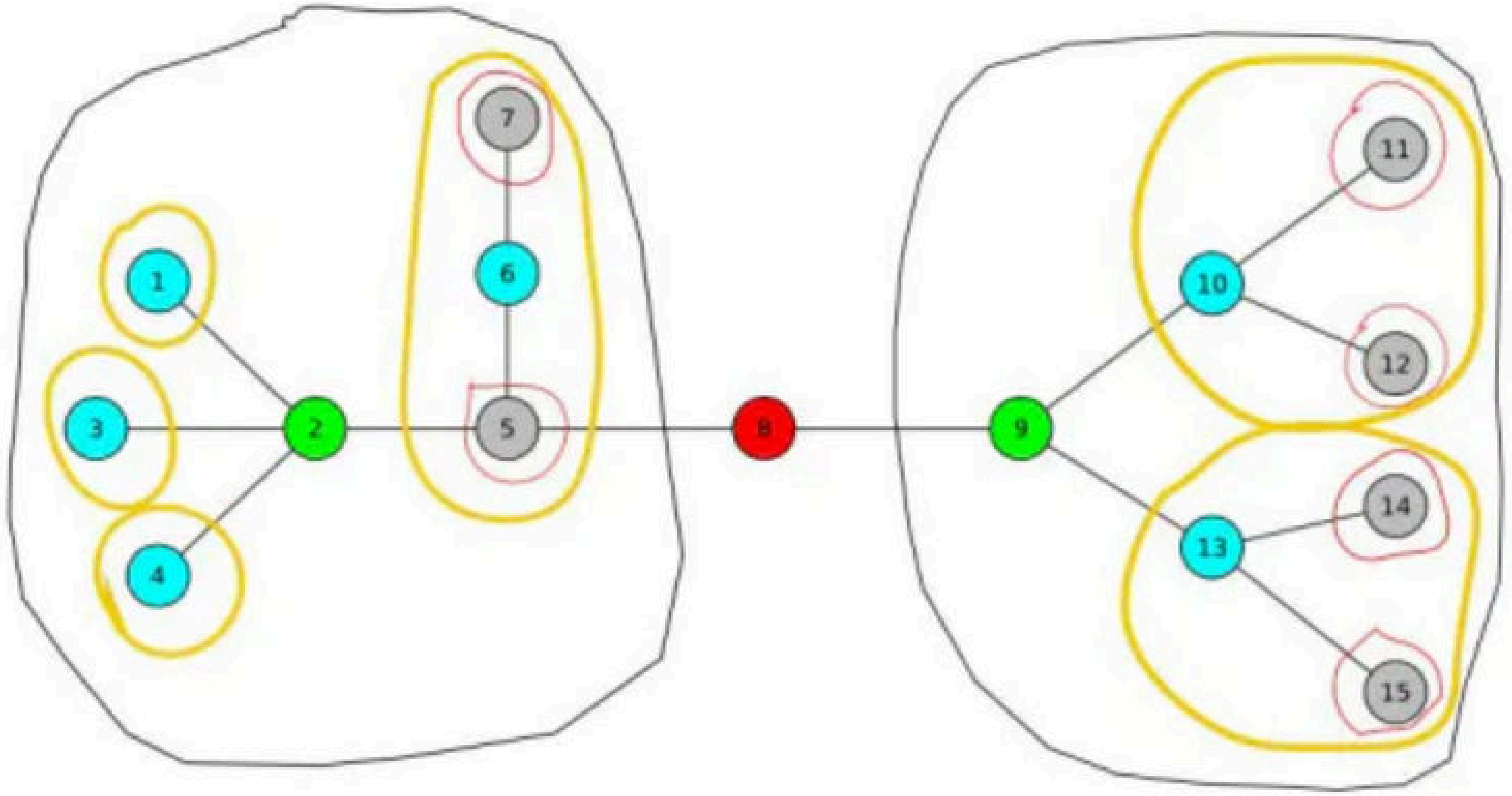
Problem Discussion

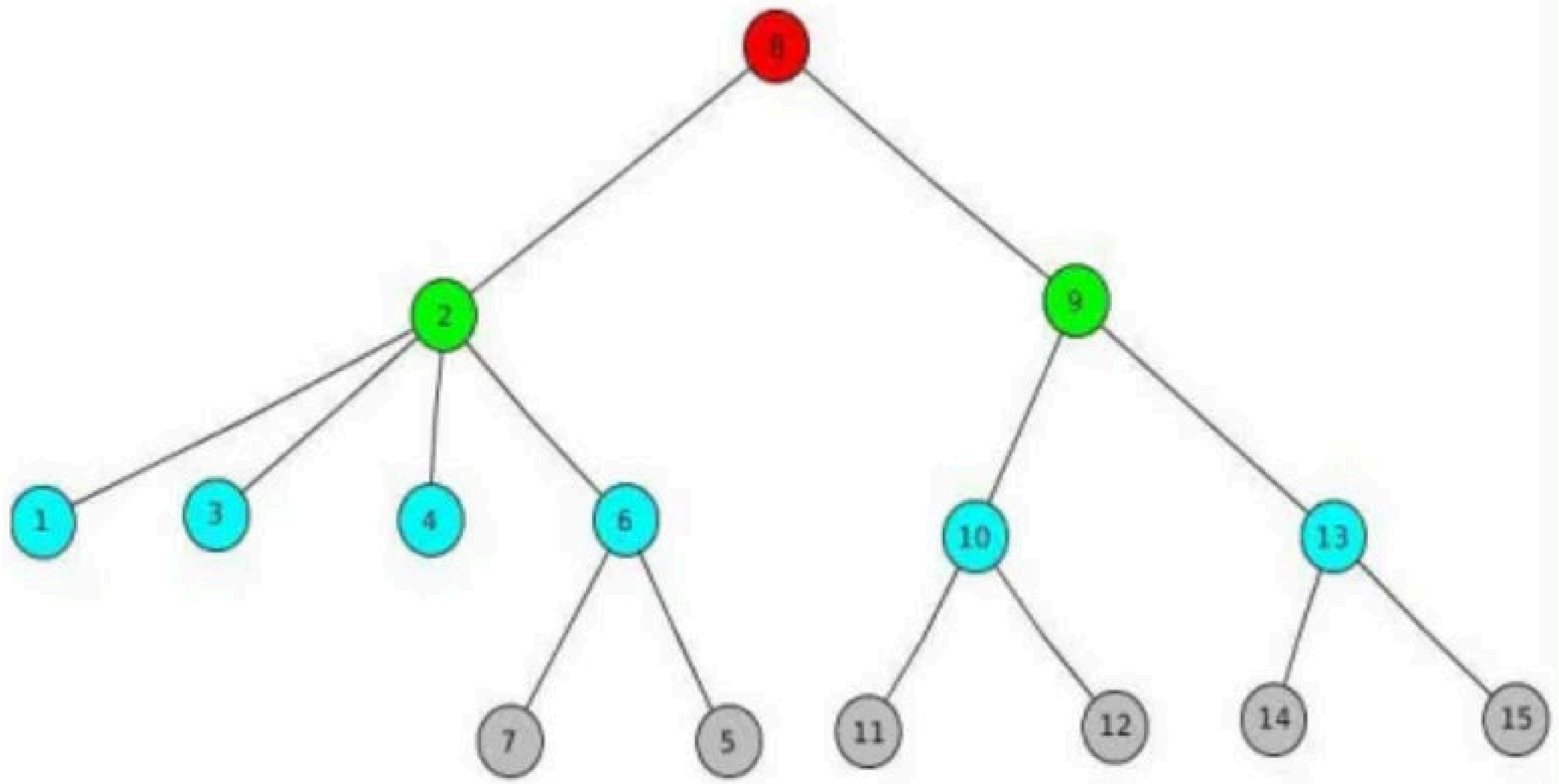
Q1: Given a weighted tree, initially all the nodes of the given tree are inactive. We need to support the following operations fast :

- **Query v :** Report the sum of distances of all active nodes from node v in the given tree.
- **Activate v :** Mark node v to be an active node.

Problem Discussion

- Let sum[i] denote the sum of distances to all activated nodes for the centroid "i" in its corresponding part.
- Let contribution[i] denote contribution of all activated nodes in subtree of i to sum[par[i]] in centroid tree.
- Let cnt[i] denote the number of activated nodes in the subtree of i in the centroid tree.
- For each update, to activate a node **u**, we move up to all the ancestors **x** of **u** in the centroid tree and update their sum[x] += dist(x, u); contribution[x] += dist(par[x], u); cnt[x] += 1;
- For each query, we compute sum[u] + (sum[par[x]]-contribution[x] + (cnt[par[x]] – cnt[x]) * dist(par[x],u)) for all ancestors **x** of **u** .





Problem Discussion

Q2: Given a weighted tree and a sequence a_1, a_2, .. a_n (permutation of 1 .. n). There Q queries of the form:

- **Query l, r, v :** Report Sum(dist(a_i, v)) for l <= i <= r.
- **Update x:** Swap(a_x, a_{x+1}) in the given input sequence.

Link: https://codeforces.com/contest/757/problem/G

Solution Idea

- Each query of the form (L R v) can be divided into two queries of form (1 R v) - (1 L - 1 v). Hence it is sufficient if we can support the following query: (i v) : Report the answer to query (1 i v)

Solution Idea

- To answer a single query of the form (i v) we can think of it as what is the sum of distance of all active nodes from node v, if we consider the first i nodes to be active.
- Hence initially if we can preprocess the tree such that we activate nodes from 1 to n and after each update, store a copy of the centroid tree, then for each query (i v) we can lookup the centroid tree corresponding to i, which would have the first i nodes activated, and query for node v in time by looking at it's ancestors.
- To store a copy of the centroid tree for each i, we need to make it persistent.

Persistent Centroid Tree

- Important thing to note is that single update in the centroid tree affects only the ancestors of the node in the tree.
- Since height of the centroid tree is $O(\log N)$, each update affects only $O(\log N)$ other nodes in the centroid tree.
- The idea is very similar to that of a persistent segment tree BUT unlike segtree, here each node of the centroid tree can have arbitrarily many children and hence simply creating a new copy of the affected nodes would not work because linking them to the children of old copy would take O(number of children) for each affected node and this number could be as large as N, hence it could take $O(N)$ time in total !

Binarizing the Input Tree

- To overcome the issue, we convert the given tree T into an equivalent binary tree T' by adding extra dummy nodes such that degree of each node in the transformed tree T' is $<=3$, and the number of dummy nodes added is bounded by O(N).
- The dummy nodes are added such that the structure of the tree is preserved and weights of the edges added are set to 0.

Binarizing the Input Tree

- To do this, consider a node x with degree d > 3 and let c1, c2...cd be it's adjacent nodes. Add a new node y and change the edges as follows :
  - Delete the edges (x - c3), (x - c4) ... (x - cd) and add the edge (x - y) such that degree of node x reduces to 3 from d.
  - Add edges (y - c3), (y - c4) ... (y - cd) such that degree of node y is d - 1.
- Recursively call the procedure on node y.
- Since degree of node y is d - 1 instead of original degree d of node x, it can be proved that we need to add at most O(N) new nodes before degree of each node in the tree is < = 3.

Making the Centroid Tree Persistent

- Hence we perform centroid decomposition of this transformed tree T'. The centroid tree formed would have the following properties.
  - The height of the centroid tree is O(logN)
  - Each node in the centroid tree has ≤ 3 children.
- Now we can easily make this tree persistent by path-copying approach.

Handling Adjacent Swap Updates

- Observe that swapping A[i] and A[i + 1] would affect only the i'th persistent centroid tree, which can be rebuilt from the tree of i - 1 by a single update query. In this approach, for each update we add $O(\log N)$ new nodes.

Conclusion & Further Reading

- Fairly advanced trick with lots of nice ideas in formulation and correct efficient implementation.
- https://tanujkhattar.wordpress.com/2016/01/10/centroid-decomposition-of-a-tree/
- https://tanujkhattar.wordpress.com/2019/04/03/persistent-centroid-tree/