Codechef Learn, Episode Lecture-2 TULIPS & IOI-18 Werewolf

https://youtu.be/DTCvKPdWH2M

tanujkhattar@

https://www.codechef.com/problems/TULIPS

https://ioi2018.jp/wp-content/tasks/contest1/werewolf.pdf

TULIPS - Problem Statement

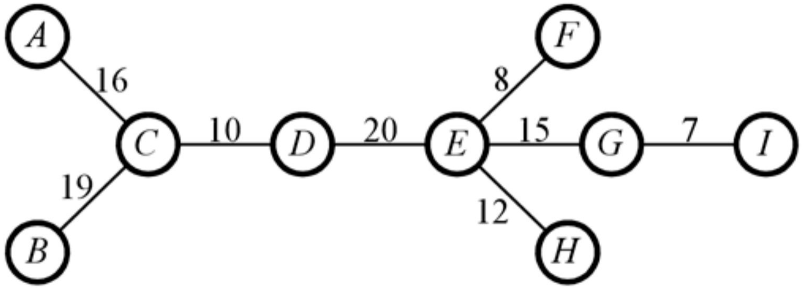
- https://www.codechef.com/problems/TULIPS
- Given a tree with N nodes, each node can have a TULIP growing on it. It takes X consecutive days for a TULIP to fully grow on any node. Answer Q queries of the form
  - D U K - How many Tulips can you pick if you start on day D from node U and visit all nodes reachable by only traversing edges of length <= K
  - Note that visiting a node also disturbs the TULIPS growing on that node, and hence resets the day on which next TULIP will become available to D + X.

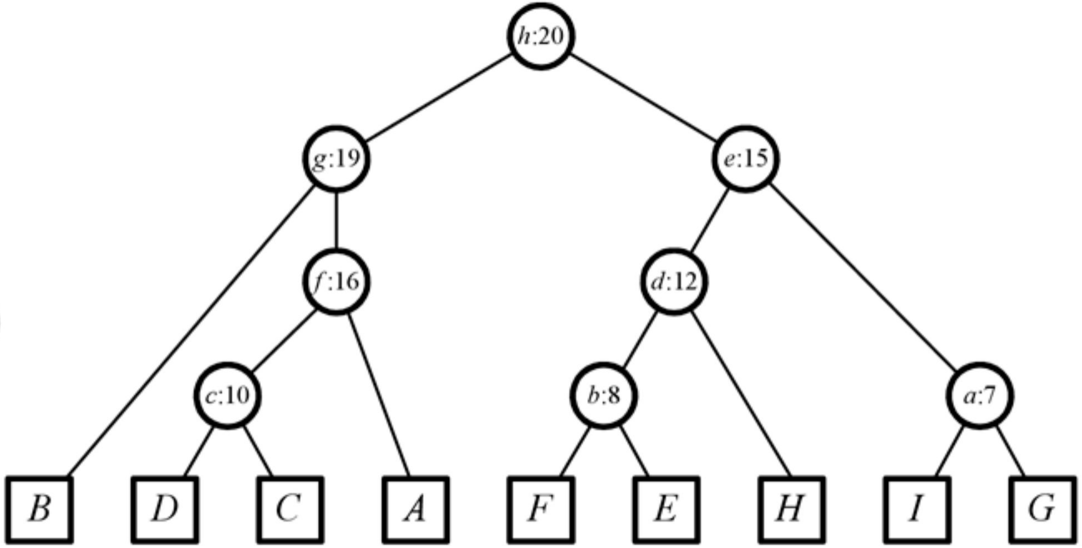
Edge Decomposition Trees - Quick Recap

- Decompose the original tree by picking an edge, removing it and recursing on the resulting subtrees T1 and T2.
  - The removed edge gets added as a new node in the Reachability Tree (RT) / Edge Decomposition Tree / DSU Tree
  - RT(T1) and RT(T2) get's attached as left and right children of this newly added root node.
- Nodes reachable by traversing edges with length <= K correspond to a subtree in the Reachability Tree.
  - Hence, such queries can be reduced to subtree queries on RT which can be answered via Euler Tour Technique (ETT).

TULIPS - Solution Idea

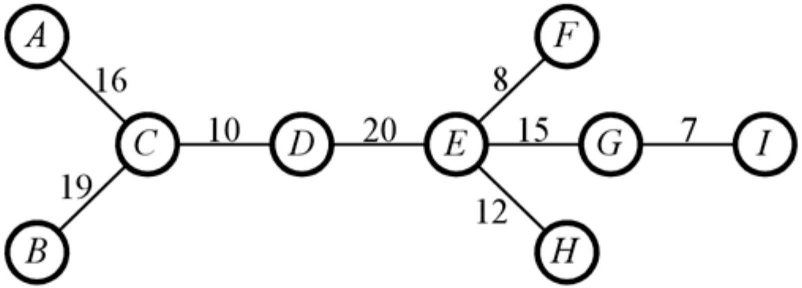
- Build a “Max-RT” of the given tree with the property that all nodes in the subtree of a node `e` are reachable by traversing edges of length <= W[e].

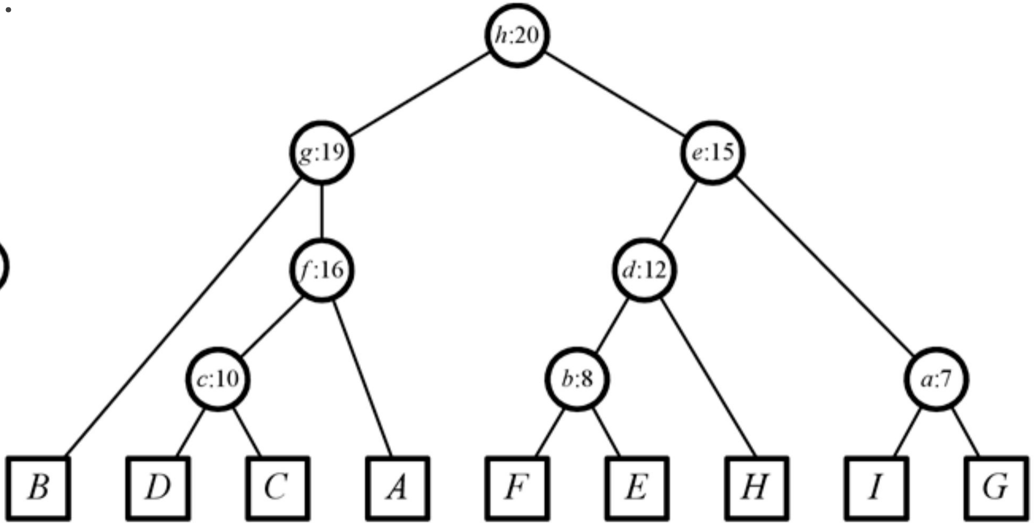




TULIPS - Solution Idea

- Build a “Max-RT” of the given tree with the property that all nodes in the subtree of a node `e` are reachable by traversing edges of length <= W[e].
- Linearize the RT using ETT such that a subtree in the RT reduces to a consecutive range in the ETT.





TULIPS - Solution Idea

- Build a “Max-RT” of the given tree with the property that all nodes in the subtree of a node `e` are reachable by traversing edges of length <= W[e].
- Linearize the RT using ETT such that a subtree in the RT reduces to a consecutive range in the ETT.
- A query D U K can now be reduced to
    - Tell the number of elements in [L, R] which have a grown TULIP on day D.
    - Update all elements in [L, R] such that next TULIP grows on day D + X.

TULIPS - How to answer the range queries?

- Given an array of N elements, initially every index has a grown TULIP on Day-1. Process the following queries
  - Q D L R: Tell the number of elements in [L, R] which have a grown TULIP on day D.
  - U D L R: Update all elements in [L, R] such that next TULIP grows on day D + X.
- The days D are given in increasing order and an update is performed for every query.

TULIPS - Way-1 Using Segment Trees + Queues.

- Let A[i] = 0 represent that there's a fully grown TULIP on index i.

TULIPS - Way-1 Using Segment Trees + Queues.

- Let A[i] = 0 represent that there's a fully grown TULIP on index i.
- To process an update D L R,
    - Add 1 to all elements in the range [L, R] on day D
    - Subtract 1 from all elements in the range [L, R] on day D + X -- This can be done by pushing (D + X, L, R) in a queue and processing the deletions before answering any subsequent query on day Dj.

TULIPS - Way-1 Using Segment Trees + Queues.

- Let A[i] = 0 represent that there's a fully grown TULIP on index i.
- To process an update D L R,
  - Add 1 to all elements in the range [L, R] on day D
  - Subtract 1 from all elements in the range [L, R] on day D + X -- This can be done by pushing (D + X, L, R) in a queue and processing the deletions before answering any subsequent query on day Dj.
- To answer a query D L R
  - First process all deletions from the queue which are supposed to happen before day D.
  - Find the number of 0's in the range [L, R].

TULIPS - Way-2 Sqrt Decomposition

- Divide the array into blocks of size sqrt(N) and for each block, store
    - mp[b] : Map of <day, count> pairs storing number of fully grown tulips on day D.
    - block_val[b] : Lazy value storing the day at which all elements of the block will have a grown TULIP.
- To process a query & update D, [L, R], X:
    - Query Indices: For blocks in which L & R lie, iterate on individual indices [L, en[b[L]] and [st[b[R]], R] and compare max(val[i], block_val[b[i]]) with D to check whether the index has a fully grown TULIP on day D or not.
    - Query Blocks: For blocks in between, iterate on the map storing <day, count> pairs and sum all counts for days <= D. Reset the map and set mp[b][D + X] = sz[b] to perform an update.

TULIPS - Way-2 Sqrt Decomposition

- Note that storing and updating the whole map for a block still works well in amortized complexity O(Q * sqrt(N)).
  - The days are always increasing and we perform an update whenever we perform a query.
  - Therefore, whenever we iterate on an entry of day D in the map of a block, we end up resetting the map and hence popping that entry.
  - Therefore, every entry which is pushed into the map is iterated on at-most once, and hence the amortized complexity across all queries is O(Qsqrt(N)).

TULIPS Implementation

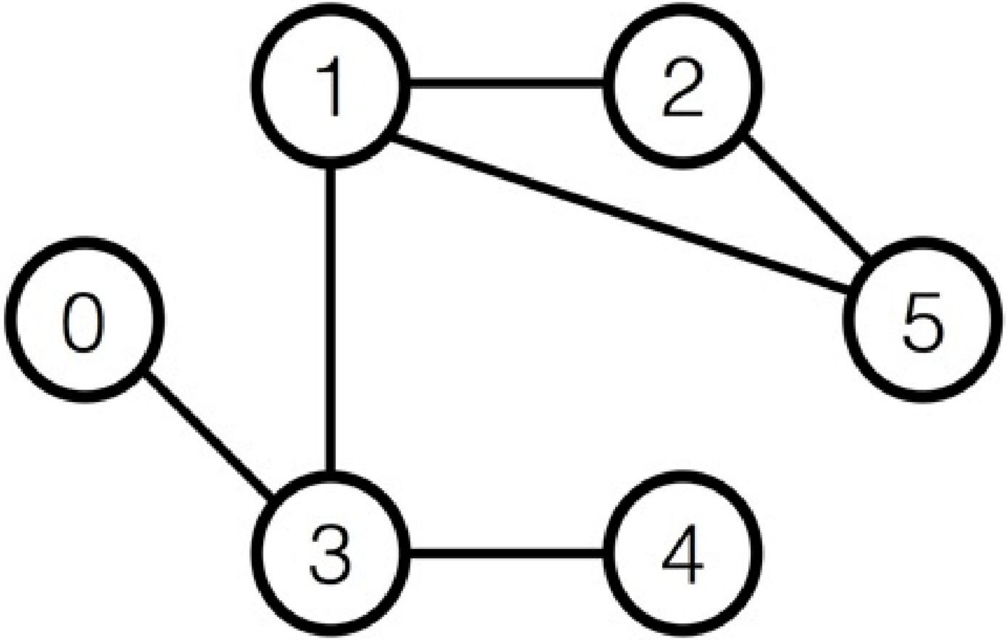
https://github.com/tanujkhattar/cp-teaching/blob/master/CWC/Ep01:%20Edge%20Decomposition%20Tree/tulips.cpp

IOI 2018 Werewolf

- https://ioi2018.jp/wp-content/tasks/contest1/werewolf.pdf
- You are a werewolf and start at a given start node in the human form. You can change your form exactly once from, human to wolf, while traversing a path. Given a graph with N nodes and M edges, you want want to answer Q queries of the form:
  - S E L R : Can you go from start node S to end node E such that you cannot visit the nodes numbered [1 … L - 1] while in human form and nodes numbered [R + 1 … N] while in wolf form.
  - Note that you start at node S in human form and on node E you must be in the wolf form. You can change your form exactly once while traversing the path from S to E.

IOI 2018 Werewolf

- Eg: 4 2 1 2
- Start node = 4
- End Node = 2
- Avoid in Human Form: [0..1]
- Avoid in Wolf Form: [3..5]

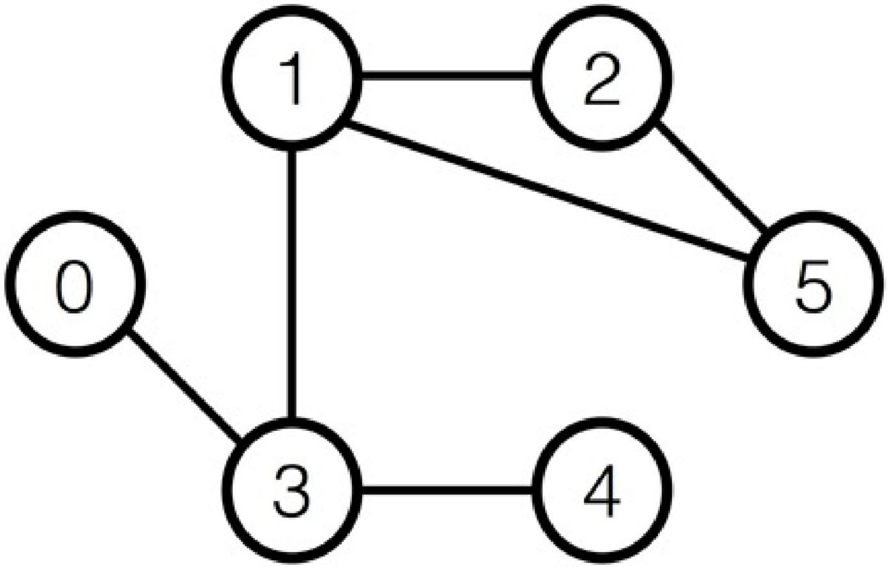


For the trip 0, you can travel from the city 4 to the city 2 as follows:

- Start at the city 4 (You are in human form)
- Move to the city 3 (You are in human form)
- Move to the city 1 (You are in human form)
- Transform yourself into wolf form (You are in wolf form)
- Move to the city 2 (You are in wolf form)

IOI 2018 Werewolf Solution Ideas

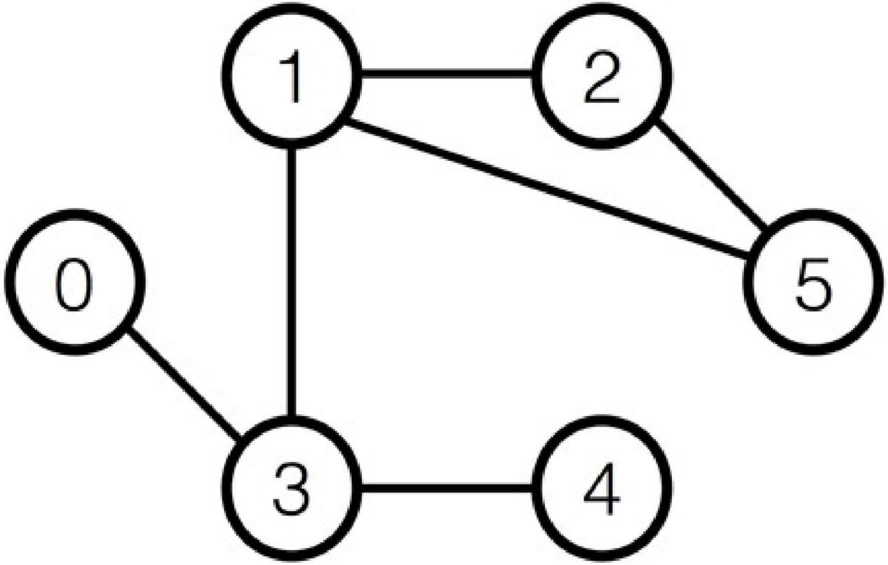
- Both S and E nodes have two different reachability conditions
  - Nodes [L .. N] can be reached by node S
  - Nodes [1 … R] can be reached by node E



IOI 2018 Werewolf Solution Ideas

- Both S and E nodes have two different reachability conditions
  - Nodes [L .. N] can be reached by node S
  - Nodes [1 … R] can be reached by node E
- Build two different reachability trees for human form and wolf form.
  - For Human Form RT, start adding nodes from the end such that subtree of the RT represents set of nodes reachable by visiting only nodes >= K.
  - For Wolf Form RT, start adding nodes from the start such that subtree of the RT represents set of nodes reachable by visiting only nodes <= K.

IOI 2018 Werewolf Solution Ideas



IOI 2018 Werewolf Solution Ideas

- Both S and E nodes have two different reachability conditions
  - Nodes [L .. N] can be reached by node S
  - Nodes [1 … R] can be reached by node E
- Build two different reachability trees for human form and wolf form.
  - For Human Form RT, start adding nodes from the end such that subtree of the RT represents set of nodes reachable by visiting only nodes >= K.
  - For Wolf Form RT, start adding nodes from the start such that subtree of the RT represents set of nodes reachable by visiting only nodes <= K.
- Find the subtrees in Human RT and Wolf RT based on S, L and E, R.
- A path exists if there’s a common node that lies in both of these subtrees -- this is node where we can change forms.

IOI 2018 Werewolf Solution Ideas

- It reduces the problem to a range query [L1, R1], [L2, R2] - Find whether there exists a common element between A[L1...R1] and B[L2...R2] where A and B are permutations of [1...N]
- This can easily solved via Segment Trees

IOI 2018 Werewolf Implementation

https://github.com/tanujkhattar/cp-teaching/blob/master/CWC/Ep01:%20Edge%20Decomposition%20Tree/werewolf.cpp

More Practice Problems

- https://www.codechef.com/problems/TULIPS
- IOI 2018 Werewolf: https://ioi2018.jp/wp-content/tasks/contest1/werewolf.pdf
- https://www.codechef.com/JULY19A/problems/MXMN (related)

Conclusion

- RT has a nice property that every subtree represents a connected component in the original tree and the edges in the connected component satisfy a certain property (eg: all edge weights <= K).
- The reachability property can be different based on the problems, for eg: Min / Max edge traversal, Min / Max node visitation etc.