Define the problem:

Main a collection of rooted trees, each node of which has a cost. The following operations:

+ and cost (V)
- and cost (V)
+ and min (V)
- and min (V)
add cost (V, x)
link (V, W)
cut (V)

Application to network flow:

Dinitz algorithm

Sequence of layered networks
find/continue flow in each one

Sequence of augmenting paths
(cost elimination at each)
Reduce this to m log n
Close to a different type of tree,

delay the tree.

A binary search tree is a balanced list.
(Symmetric order: traversal of tree is the list.) Define symmetrical successors:

Balance, maintain symmetric.

Insert, rotation: rotate an item to the root.

Examples:

Assign each item a weight \( W(x) \).
The size of a node = total wt of its sub. \( s(x) \).

Access lemma: \( W(x) \) amortized to rotation:

\[
3 (\log s(w) + \log s(x)) + 1
\]

Sincerely, Stiq. A. etc.
Relation between \( V \) and \( T \) (two cases)

\[
x = p(z)
\]

\[
y = \pi(x)
\]

- In solid tree
- \( y \) is defined in solid tree
- In solid tree

much procedure in representation:
1. Selection of virtual tree (path in \( T \))
2. Unique tree structure

Parent & left & right pointer to store \( V \).

\[
\text{cost}(x) = \text{the cost of node } x.
\]

\[
\text{min cost}(x) = \text{the minimum cost of any descendant of } x \text{ in the same solid subtree}.
\]

Could do this directly, then find cost and find minimum will be efficient. Add cost now.

show how to traverse the virtual tree, to find this. also note that rotations are easy to do.
(5)

To make old cost `cost` into new tree costs in "difference form".

\[
\Delta \text{cost}(X) = \begin{cases} 
\text{cost}(X) & \text{if } X \text{ is a solid subtree root} \\
\text{cost}(X) - \text{cost}(P(X)) & \text{otherwise}
\end{cases}
\]

\[
\Delta \text{min}(X) = \text{cost}(X) - \min \text{cost}(X)
\]

The virtual tree can be modified with two constant time primitives:

- Rotation
- Splicing

The actual/virtual correspondence is maintained, we must show how to update \(\Delta \text{cost}\) and \(\Delta \text{min}\).
How do we update \( \Delta \text{cost} \) and \( \Delta \text{min} \):

\[
\Delta \text{cost}'(v) = \text{cost}(v) - \text{cost}(p(v))
\]

\[
= \text{cost}(v) - \text{cost}(w) + \text{cost}(w) - \text{cost}(p(w))
\]

\[
= \Delta \text{cost}(v) + \Delta \text{cost}(w)
\]

\[
\Delta \text{cost}(w) = \text{cost}(w) - \text{cost}(v)
\]

\[
= -\Delta \text{cost}(v)
\]

\[
\Delta \text{cost}'(b) = \text{cost}(b) - \text{cost}(w)
\]

\[
= \text{cost}(b) - \text{cost}(v) + \text{cost}(v) - \text{cost}(w)
\]

\[
= \Delta \text{cost}'(b) + \Delta \text{cost}(v)
\]

\[
\Delta \text{min}'(w) = \text{cost}(w) - \min \text{cost}(w)
\]

\[
= \text{cost}(w) - \min \{ \text{cost}(w), \text{cost}(b), \text{cost}(c) \}
\]

\[
= \max \{ 0, \text{cost}(w) - \min \text{cost}(b) \}
\]

\[
= \text{cost}(w) - \min \text{cost}(w)
\]
The scaling operation on a node in a virtual tree:

- a sequence of rotations and scaling.

3 passes:

**Pass 1:** Walk up from $x$, expanding on each valid subtree.

**Pass 2:** Walk up the (now) dotted path from $x$, galleries all the way.

**Pass 3:** Scaling $x$ to the root of its valid tree (the whole tree).

The amortized time of scaling is $O(\log n)$ (an $n$ node tree).

**Proof:** Assign a "mass" of 1 to each node.

The weight of a node in the $U$ area is $\#$ number of descendants, below the dashed edge.

The size of a node = $\#$ descendants of it in $U$. 
Proof: Recall the access lemma:

AC: Amortized cost is $\leq 3(r(\rho_{\text{opt}}) - r(x)) + 1$

The proof uses the $\Phi = \sum_{x} r(x)$

when $r(x) = \text{rank}(x) = \log_2 \text{size}(x)$

Let's modify this using a potential that is twice this, $\Phi = 2 \sum_{x} r(x)$.

For the zig-zag zig-zag steps this can most double the amortized cost bound. So it's $6(r'(x) - r(x))$ for the step.

In the zig case we get

$$(x, y) \quad \Rightarrow \quad (x', y')$$

$$2(r'(x) - r(x)) \quad \text{pages with } \Delta \Phi$$

$$\downarrow$$

$$2(r'(x) - r(x)) + 1 \leq 6(r'(x) - r(x)) + 1$$

So we have the new access lemma:

AC': Amortized cost is $\leq 6(r(\rho_{\text{opt}}) - r(x)) + 1$

when $\Phi = 2 \sum_{x} r(x)$. 
Proof contd:

the potential we use in the VC case is

\[ \Phi = 2 \sum_{x \in V} r(x) \]

Let's analyze the amortized cost of each pass of the oracle on \( V \).

cost measure: one per rotation
one per spline

(this covers everything)
let \( k \) be the number of splines,

pass 1: let the node selected be \( x_0 \) \( x_1 \) \( \cdots \) \( x_k \)
let their corresponding \( r \) values be \( r_0 \) \( r_1 \) \( \cdots \) \( r_k \)

the cost of relay \( i \) is

\[ 6 (r(r_i) - r(x_i)) + 1 \]

Total is \( \sum_{0 \leq i \leq k} [6 (r(r_i) - r(x_i)) + 1] \)

\[ k + 6 \sum_{0 \leq i \leq k} (r(r_i) - r(x_i)) \]

note \( r(x_1) \geq r(r_0 - 1) \) (defining \( r(r_0) = r(x_0) \))
\[ \text{Pass 1} \leq k+1 + 6 \sum_{0 \leq i < k} r(r_i) - r(r_{i-1}) \]

\[ = k+1 + 6 (r(r_k) - r(r_{i})) \]

\[ = k+1 + 6 (r(v_k) - r(v_0)) \]

\[ = k+1 + \frac{6}{\log_2 n} \]

\[ \leq k+1 + \frac{6}{\log_2 n} \]

---

**Pass 2:** Scheduling does not change the \( \phi \). (The sizes of the tasks do not change, the weights do)

\[ \text{pass 2} = k \]

---

**Pass 3:** Scheduling a path of length \( k \).

\[ \Rightarrow \text{the cost is } k. \]

At most \( 6 \log_2 n + 1 \)
Putting it all together:

In the "average-case" case, we can recast \( K \) by \( 6 \log n + 1 \)

\[
\begin{align*}
\text{Pass 1} & \leq 2 + 12 \log n \\
\text{Pass 2} & \leq 1 + 6 \log n \\
\text{Pass 3} & \leq 1 + 6 \log n \\
\end{align*}
\]

\[
24 \log n + 4
\]

Q.E.D.

The recursion now is slightly different:

1) The cost with path length to \( X \) = \# rotations above this, if exists, and otherwise 0.

2) \( \Rightarrow \) Pass 2 has 0 cost.

3) Use the "doubled" access lemma:

\[ 2 \cdot \# \text{ rotations} \leq 6 \log n + 2 \]
half

Use one of these to cover the "K" of Page 1. The other half to review Page 3.

Thus our total becomes:

1 + 6/ln n + 6/ln n + 2 = 12ln n + 3

Note: the paper runs here. I think it's a bug.
If there are no dangling edges, then we get

Pass 1: \( O(\log n + h) \)

Pass 2: \( O(h) \)

Pass 3: \( O(\log n) \)

But \( 2h \leq \text{total Pass 3 cost} \)

\[ \Rightarrow \text{the amortized time is } O(\log n) \]

(Q.E.D.)

Now done was V tree & Delay to implement the evaluation?

Find cost (V): Delay at V, and

Return the cost (V)

Find cost (V): Delay at V

Go to minimum node of

The solid tree re. X

Delay X, return X.
\( \text{find } \min(v) \) : Delay at \( V \)

* Walk down the path from \( V \) using \( \Delta \text{cost} \text{ and } \Delta \text{min} \) to the last missing cost node after \( V \), call it \( X \)
* Delay at \( X \), and return.

\[
\Delta \text{min} = 0
\]

\[
\Delta \text{cost}(X) - \Delta \text{min}(X) = \text{cost}(X) - \text{cost}(P(X)) - \text{cost}(X) + \text{mincost}(X)
\]

\[
= \text{mincost}(P(X)) - \text{cost}(P(X))
\]

\[
\Rightarrow \text{mincost}(X) \leq \text{mincost}(Y)
\]

\[
\Delta \text{cost}(X) - \Delta \text{min}(X) \leq \Delta \text{cost}(Y) - \text{mincost}(Y)
\]

\( \Delta \text{cost}(X) - \Delta \text{min}(X) \) comes to explain the fact that we use known and reachable \( Y \), we can compute its \( \text{cost}() \) and \( \text{mincost}() \).
add left(v) : self(v)
add right(v) : self(v)

\[ \text{if } \text{left}(v) \neq \text{null} \]
\[ \text{then } \text{cut}(\text{left}(v)) \]

Link (v, w) : cut at v and cut at w

Make v a middle child of w

cut (v) : self at v

add self (v) to self (right(v))

Brush the link between v and right(v).

Thus: A sequence of m operations on n nodes is \( O(m \log n) \)

Proof: All the operations have \( \log n \) announce time. The middle potential is zero. The potential is always 2.0.

Q.E.D.