Complexity Theory of Polynomial-Time Problems

Lecture 9: Dynamic Algorithms I

Sebastian Krinninger

Today's Plan

1. Decremental SSSP via Even-Shiloach tree

2. Decremental APSP

Floyd-Warshall Algorithm

$O(n^3)$ algorithm for computing All-Pairs Shortest Paths (APSP)

n: number of nodes

Put some order $v_1, \dots, v_n$ on the nodes
Set $d(v_i, v_j) = w(v_i, v_j)$ for every pair of nodes $v_i \neq v_j$

For $k = 1$ to n:
    For every pair of nodes v_i, v_j:
        $d(v_i, v_j) \leftarrow \min\left(d(v_i, v_j), d(v_i, v_k) + d(v_k, v_j)\right)$

Running Time: n iterations, each takes time $O(n^2)$

Correctness: After iteration i, $d(\cdot,\cdot)$ gives correct distance in graph restricted to $\{v_1, \dots, v_k\}$
$\Rightarrow$ Correct in full graph after iteration n

Dynamic View

Why stop after n iterations?

Floyd-Warshall allows *insertions* of new nodes

<u>Insert(v, In_v, Out_v):</u> // (Insert node with incident edges and weights)
Set $d(v', v) = w(v', v)$ for every incoming neighbor v' of v
Set $d(v, v') = w(v, v')$ for every outgoing neighbor v' of v

For every incoming neighbor s of v and every node t

$$d(s,t) \leftarrow \min\big(d(s,t), d(s,v) + d(v,t)\big)$$

For every node s outgoing neighbor t of v

$$d(s,t) \leftarrow \min\big(d(s,t), d(s,v) + d(v,t)\big)$$

For every other pair of nodes s, t:

$$d(s,t) \leftarrow \min\big(d(s,t), d(s,v) + d(v,t)\big)$$

Update Time: $O(n^2)$ *per insertion*

Dynamic Algorithms

A dynamic graph algorithm is a **data structure** supporting:

- `Preprocess`(G): preprocess the graph G
- `Insert`(u,v): insert the edge (u,v) into G
- `Delete`(u,v): delete the edge (u,v) from G
- `Query`(G): return result of algorithm for **current** graph G

Terminology:

- Incremental: only insertions are supported
- Decremental: only deletions are supported
- Fully dynamic: both insertions and deletions are supported

Some algorithms also support insertions and deletions of nodes

Goal:

- Time spent per update or query less than recomputing from scratch
- (Polynomial preprocessing time)

Measuring Update Time

Two Measures

- Worst-case update time
  Fixed upper bound on running time per update
- Amortized update time
  "On average" upper bound on running time per update

Formally: Amortized update time $u(n,m)$ if total time spent for a sequence of t updates is at most $t \cdot u(n,m)$.

Very common in incremental/decremental algorithms:

- Amortize update time over m insertions/deletions
- "Total update time"

1. Decremental SSSP

Even-Shiloach Algorithm

Goal: Decremental SSSP in unweighted graphs from source s

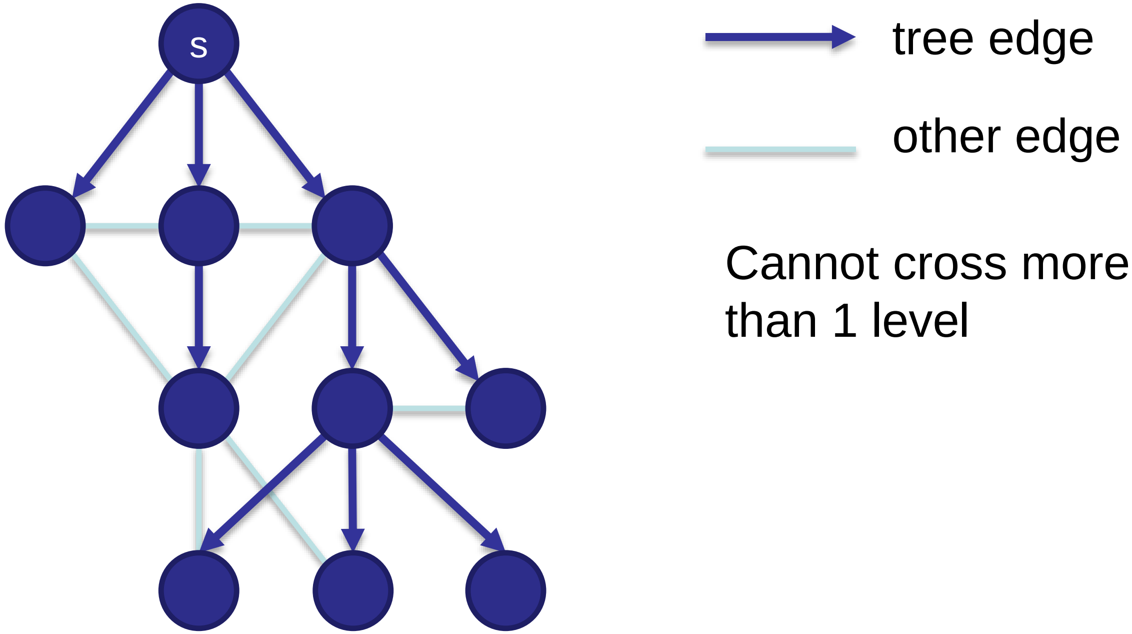
Example of shortest path tree from s:

Level 0

Level 1

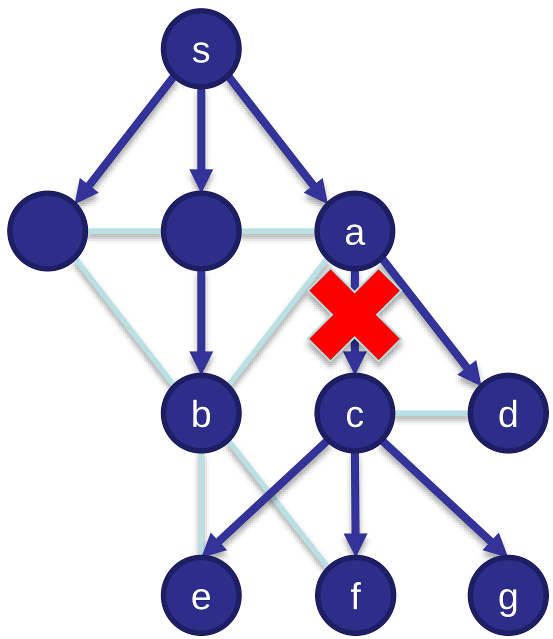
Level 2

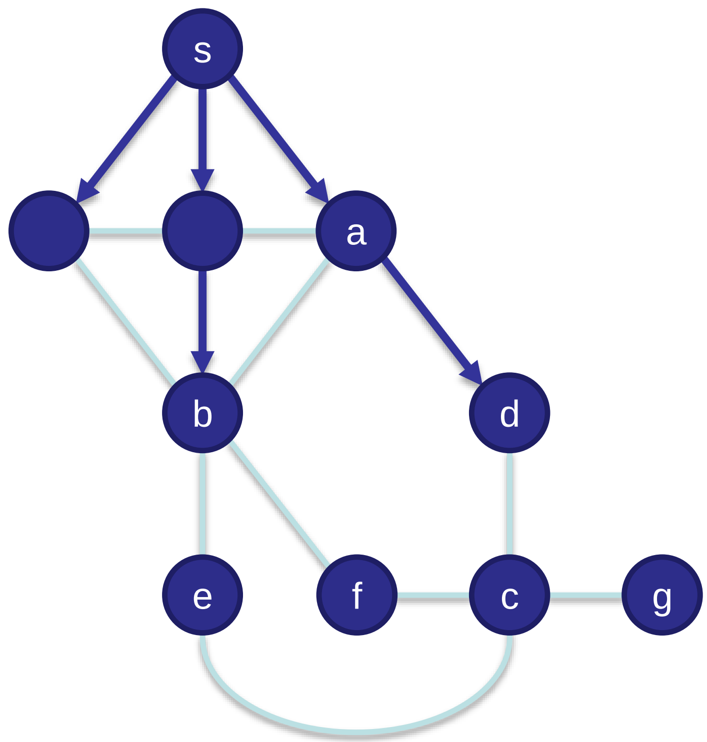
Level 3



Cannot cross more than 1 level

Deletion Procedure I



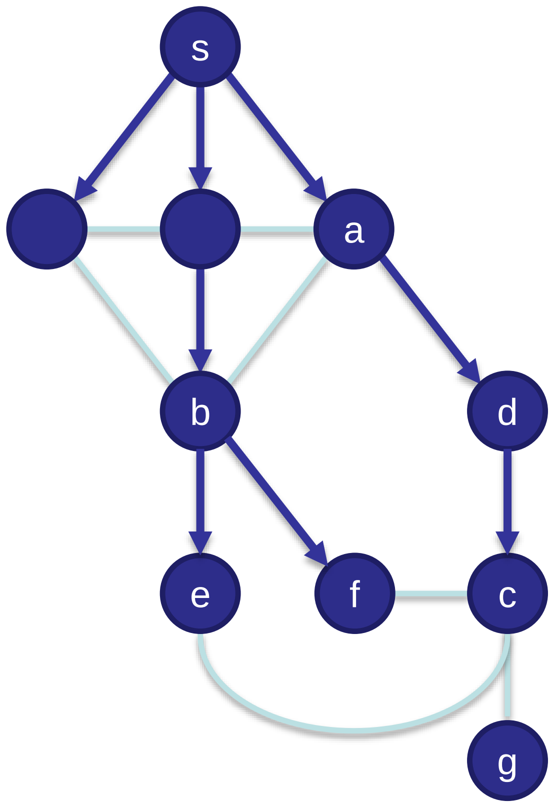


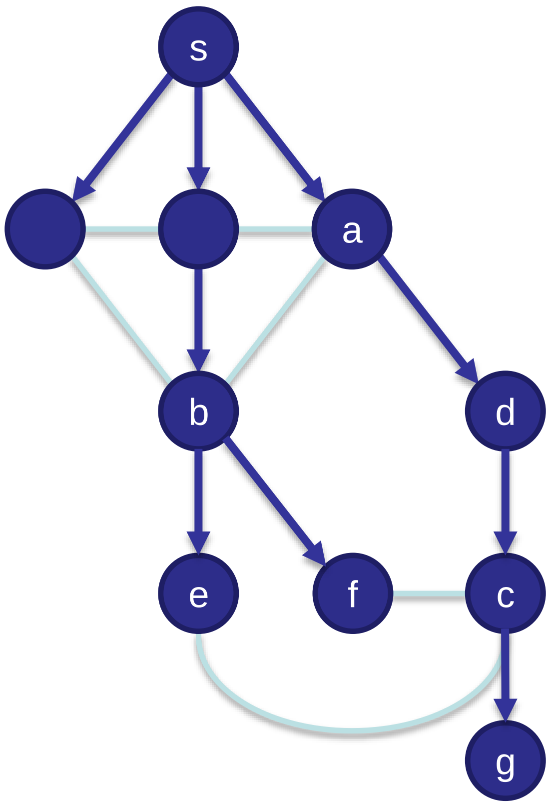
- c loses its parent
- c finds no new parent at level 2
- c increases level to 3
- c informs neighbors about level increase
- Children of c lose their parent

- c finds new parent d
- e finds new parent b
- f finds new parent bg finds no new parent at level 3
- g increases level to 4

max planck institut informatik

Deletion Procedure II





- g finds new parent c
- Now we are done because all nodes have a parent again

Internal Data Structures and Initialization

Data Structures:

For every node v:

- Number neighbors of v from 1 to $\deg(v)$ (initial degree of v)
- $n_i(v)$ Pointer to i-th neighbor of v
- $p(v)$ Index of parent of v (among neighbors) in tree
- $\ell(v)$ Level of v in tree (will correspond to distance from root)

Global:

- Q Priority queue with levels as keys (used in update procedure)

Initialization:

Compute BFS tree from source s such that each node takes parent with minimum index among neighbors.

Time: $O(m)$

Pseudocode

```
Delete(u,v):
Add u and v to Q
While Q ≠ ∅
        Take node v with minimum level from Q
        Process(v)

FindNewParent(v):
// Check if neighbor with index p(v) is a valid parent
While G does not contain edge (v, n_{p(v)}(v)) or l(v) < (ℓ(n_{p(v)}(v)) + 1):
        p(v) ← p(v) + 1                    // If not, try next neighbor as parent
        Add v to Q
        If p(v) = deg(v) + 1               // Check if all neighbors exhausted
                l(v) ← l(v) + 1            // Increase level
                If ℓ(v) ≥ n − 1:           // Check if level too big
                        Set ℓ(v) ← ∞
                        Remove v from Q
                p(v) ← 1                   // Reset parent index
                Add neighbors of v to Q    // Process neighbors
```

Correctness I

Claim 1: Initially, and after each update is finished: $\ell(v) \geq \mathrm{dist}(s, v)\ \forall v$

Proof:
If $\ell(v) = \infty$, then certainly true

Otherwise:
Consider path π from v induced by following parents
Levels of nodes on π are strictly decreasing:

- When parent of a node is set, parent has strictly smaller level
- When level of a node changes it informs all potential children

Thus, π ends at s because s is the only node at level 0

$\ell(v) =$ length of π
π cannot be shorter than shortest path from s to v
Thus, $\ell(v) \geq \mathrm{dist}(s, v)$

Correctness II

Claim 2: At any time: For every node v with neighbor u, $\ell(v) \leq \ell(u) + 1$ if $\ell(u) + 1 \leq n - 1$.

Proof:
By induction on #level increases of v (in total over all deletions)
Induction Base: True after initialization

Induction Step:
$\ell(v)$: level of v directly before level increase
$\ell'(v)$: level of v directly after level increase

By IH: $\ell(v) \leq \ell(u) + 1$
Algorithm guarantees: $\ell(v) < \ell(u) + 1$ (otherwise no level increase of v)
(Detail: no candidate parent for v at level $\ell(v)$ anymore by processing order according to levels)
Thus: $\ell(v) + 1 \leq \ell(u) + 1$
Since $\ell'(v) = \ell(v) + 1$ we have $\ell'(v) \leq \ell(u) + 1$

Inequality remains true until next level increase of v because level of u never decreases

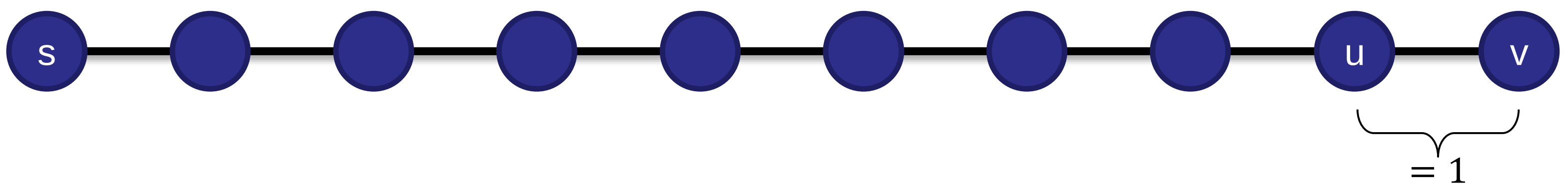
Correctness III

Lemma: Initially and after each update is finished, $\ell(v) = \mathrm{dist}(s, v)\ \forall v$

Proof by induction on distance to s

If $\mathrm{dist}(s, v) = \infty$: Then $\ell(v) \geq \mathrm{dist}(s, v) = \infty$ by Claim 1

If $\mathrm{dist}(s, v) < \infty$:
Consider successor u of v on shortest path from s to v



When algorithm finished update:
$\ell(u) = \mathrm{dist}(s, u)$ by IH
In particular: $\mathrm{dist}(s, u) \leq n - 2$ and thus $\ell(u) + 1 \leq n - 1$

By Claim 2: $\ell(v) \leq \ell(u) + 1 = \mathrm{dist}(s, v)$
By Claim 1: $\ell(v) \geq \mathrm{dist}(s, v)$

$$\Rightarrow \ell(v) = \mathrm{dist}(s, v)$$

Running Time

Lemma: The total update time over **all** deletions is $O(mn)$

(where m is the number of edges at initialization)

Amortized analysis!

Idea: Every time the level of some node v increases, we charge running time of $O(\deg(v))$ to that level increase (see next slide).

(where $\deg(v)$ is the degree of v at initialization)

The level of every node can increase at most $n - 1$ times (max. distance).

Additionally, charge time $O(1)$ to every deletion

Total time: $O(\#\mathrm{del} + \sum_{v \in V} n \deg(v)) = O(m + n \cdot \sum_{v \in V} \deg(v)) = O(n \cdot m)$

Remember from kindergarten: sum of degrees ≤ twice #edges

Running Time Analysis

<u>`Delete(`u`,`v`):`</u>

Add u and v to Q $O(1)$ per deletion

While $Q \neq \emptyset$

Take node v with minimum level from Q

`Process(`v`)`

$O(1)$ charge to
- level increase of node that put v into queue or
- deletion that put v into queue

<u>`Process(`v`):`</u>

$O(1)$, charge to
- level increase of node that put v into queue or
- deletion that put v into queue or
- increase of parent index

While G does not contain edge $(v, n_{p(v)}(v))$ or $l(v) < \left(\ell\left(n_{p(v)}(v)\right) + 1\right)$:

$p(v) \leftarrow p(v) + 1$

Add v to Q

$O(1)$ per increase of parent index
(increases at most $\deg(v)$ times at each level)

If $p(v) = \deg(v) + 1$

$l(v) \leftarrow l(v) + 1$

If $\ell(v) \geq n - 1$:

Set $\ell(v) \leftarrow \infty$

Remove v from Q

$p(v) \leftarrow 1$

Add neighbors of v to Q

$O(\deg(v))$:
charge to level increase of v

Total: $O(\#\mathrm{del} + \sum_{v \in V} n \deg(v))$+

Banker's View

Every node v receives:
- $10\,\deg(v)$ coins at initialization
- 3 coins when deleting incident edges

Observation: Sufficient number of coins to pay 1 coin per operation.

(Note: give constant number of coins to each neighbor at level increase)

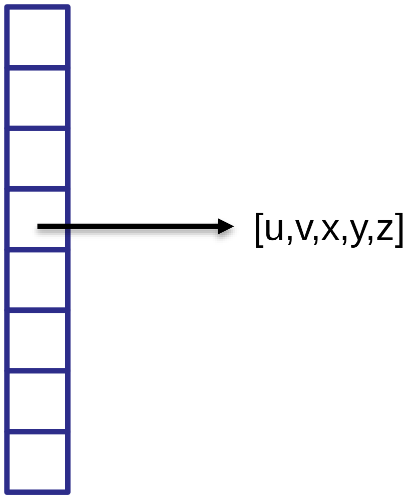
Total number of coins spent: $O(\#\mathrm{del} + \sum_{v \in V} n\,\deg(v))$

Implementing Priority Queue

Standard heap: Time $O(\log n)$ per operation

In our application we can get $O(1)$ per operation

Array A of size n, where $A[i]$ contains pointer to list of nodes at level i



In unweighted undirected graphs:

- At most two lists non-empty
- at consecutive levels

Extensions

Theorem: Maintaining SSSP under deletions takes total time
- $O(mn)$ in unweighted undirected graphs
- $O(mn)$ in unweighted directed graphs
- $O(mnW)$ in directed graphs with weights $\{1,2,\ldots,W\}$.

[Even/Shiloach '81, King '99, King/Thorup '01]

Theorem: Maintaining SSSP under deletions up to depth D takes total time $O(mD)$ in directed graphs with integer weights.

2. Decremental APSP

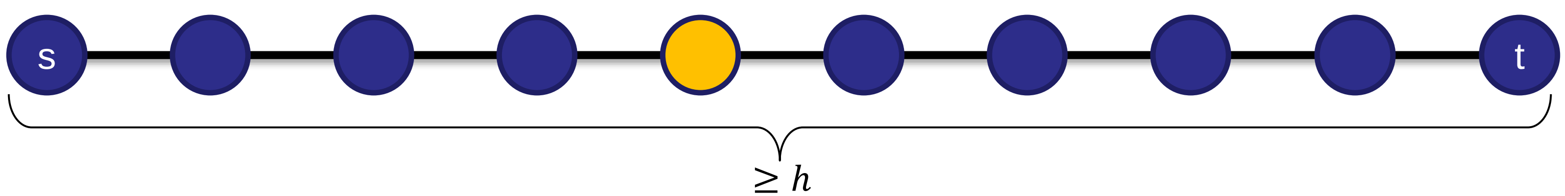
Hitting Set for Long Paths

Random process for picking a set of nodes S:

- Set $p = \min\left(\frac{10 \log n}{h}, 1\right)$
- Iterate over all nodes
- Pick each node with probability p independently (flip biased coin)
- Expected size of S: $O\left(\frac{n \log n}{h}\right)$

Lemma: For every pair of nodes s and t, if the shortest path from s to t contains at least h nodes, then one of them is from S with probability at least $1 - \frac{1}{n}$ (i.e., ‘with high probability’).

Caveat: There could be many shortest paths from s to t. We only guarantee to hit one of them (e.g. lexicographic shortest path).



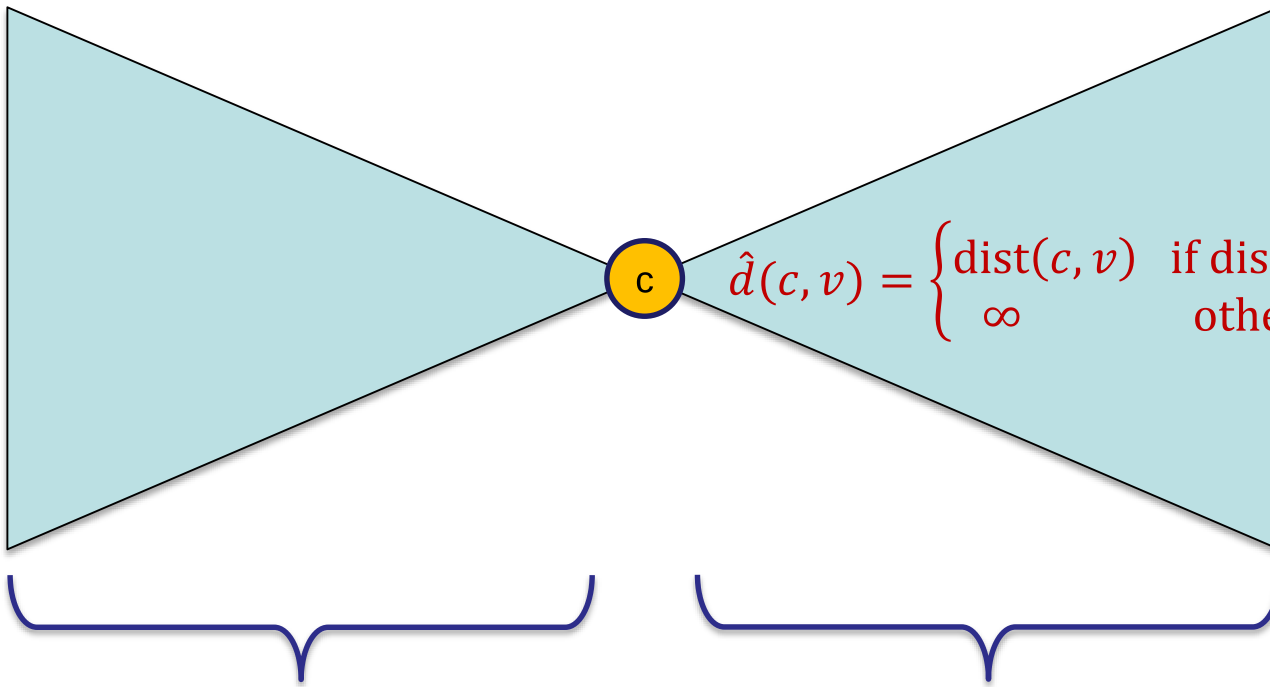
Lemma also holds for all graphs during a sequence of deletions (if sequence of deletions is independent from random choices of algorithm)

Maintaining shortest paths in range $2^i \dots 2^{i+1}$

Pick set of nodes S_i ("i-centers"):

- Sampling probability $p = \min\left(\frac{10 \log n}{2^i}, 1\right)$
- Expected size of S_i: $O\left(\frac{n \log n}{2^i}\right)$

For every i-center $c \in S_i$:



$$\hat{d}(c, v) = \begin{cases} \mathrm{dist}(c, v) & \text{if } \mathrm{dist}(c, v) < 2^{i+1} \\ \infty & \text{otherwise} \end{cases}$$

Even-Shiloach tree **to** c
up to depth 2^{i+1}
(Reverse graph:
reverse direction of each edge)

Even-Shiloach tree **from** c
up to depth 2^{i+1}

Total time: $O\left(|S_i| m 2^{i+1}\right) = O(mn \log n)$

Decremental APSP algorithm

For $i = 1$ to $\lfloor \log n \rfloor$:

- Pick i-centers S_i with sampling probability $p = \min\left(\frac{10 \log n}{2^i}, 1\right)$
- For every i-center $c \in S_i$: Maintain ES-tree to and from c of depth 2^{i+1}

Total update time: $O\left(\sum_{i=1}^{\lfloor \log n \rfloor} |S_i| m 2^i\right) = O\left(\sum_{i=1}^{\lfloor \log n \rfloor} mn \log n\right) = O(mn \log^2 n)$

Query Algorithm:

- Question: What is the distance from s to t
- Return minimum value of $\hat{d}(s,c) + \hat{d}(c,t)$ among all centers $c \in \cup S_i$
- Query time: $O(n)$ (= number of centers)

Correctness:

- Let π be shortest path from s to t
- π has between 2^i and 2^{i+1} nodes for some $i = 1$ to $\lfloor \log n \rfloor$
- π contains a center $c \in S_i$ with high probability
- Subpaths from s to c and from c to s are also shortest paths and both have length $\leq 2^{i+1}$
- Thus, $\hat{d}(s,c) + \hat{d}(c,t) = \mathrm{dist}(s,t)$
- Other centers can never report a smaller value for $\hat{d}(s,c) + \hat{d}(c,t)$

Extensions

Result we just showed:

Theorem: There is a decremental algorithm for maintaining APSP in unweighted, directed graphs with total update time $O(mn \log^2 n)$ and query time $O(n)$.

By explicitly maintaining distances after each update, one can reduce query time.

Theorem: There is a decremental algorithm for maintaining APSP in unweighted, directed graphs with total update time $O(n^3 \log^2 n)$ and **constant** query time.

[Baswana et al. '02]