

Shortest Paths

Consider $G = (V, E)$ with edge costs $c : E \rightarrow \mathbb{R}$. Extend c to paths, i.e.,

$$c(\langle e_1, \dots, e_k \rangle) = \sum_{i=1}^k c(e_i). \text{ We look for a shortest path from } s \text{ to } t:$$

Queries: Find one shortest s - t path

SSSP: Single source shortest path from s to all $t \in V$.

APSP: All pairs shortest path

Overview

- SSSP for arbitrary edge weights
- DAGs
- nonnegative edge weights
- integer edge weights
- average case analysis
- all pairs shortest paths
- shortest path queries with preprocessing

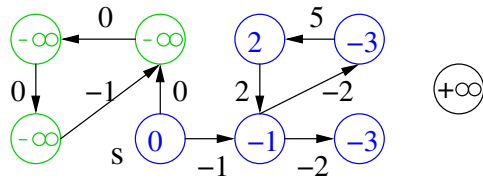
1 Algorithmic Framework

Shortest path distance:

$$\mu(s, v) := \begin{cases} +\infty & \text{if } v \text{ is not reachable from } s \\ -\infty & \\ \min \{c(p) : p \text{ is simple path from } s \text{ to } v\} & \text{else} \end{cases}$$



Simplicity is no restriction:

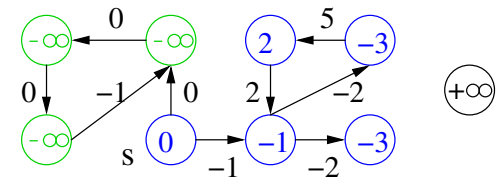


Some Simple SSSP Stuff

Lemma 1. $\langle s, \dots, v, v' \rangle$ is shortest path $\Rightarrow \langle s, \dots, v \rangle$ is shortest path

Lemma 2. No negative cycles \wedge all nodes reachable from s
 $\Rightarrow \exists$ tree T : all paths in T are shortest paths

Proof. Exercise □



A generic algorithm

$d = \langle +\infty, \dots, +\infty \rangle$: NodeArray of $\mathbb{R} \cup \{-\infty, +\infty\}$

$d(s) := 0$

invariant $\forall v \in V : d(v) \geq \mu(s, v)$

$in = \langle \perp, \dots, \perp \rangle$: NodeArray of Edge

invariant in -edges describe length $d(v)$ paths from s to v if $\mu(s, v) < \infty$

while $\exists v : d(v) \neq \mu(s, v)$ **do**

 pick some $e \in E$

 relax(e)

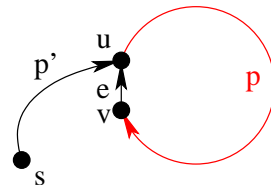
Procedure relax($e = (u, v) : Edge$)

if $d(u) + c(e) < d(v)$ **then** $(d(v), in(v)) := (d(u) + c(e), e)$

Cycles in the in -graph

Lemma 4. v on a cycle of in -edges $\Rightarrow \mu(s, v) = -\infty$

Proof.



$$d(v) + c(e) < d(u), d(v) = c(p') + c(p), d(u) = c(p')$$

$$\Rightarrow c(p') + c(p) + c(e) < c(p')$$

$$\Rightarrow c(p) + c(e) < 0$$

we can reach v using arbitrarily many traversals of the cycle. □

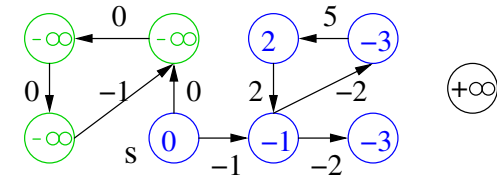
A (more concrete) termination condition

Lemma 3. $d(v) = \mu(s, v)$ if

\exists shortest path $\langle e_1, \dots, e_k \rangle, t_1 < \dots < t_k : e_i$ is relaxed at time t_i .

Proof. Induction:

at time t_i we have $d(\text{target}(e_i)) = c(\langle e_1, \dots, e_i \rangle)$. □



2 Arbitrary Edge Weights (Bellman-Ford Algorithm)

for $i := 1$ **to** $n - 1$ **do**

forall $e \in E$ **do** relax(e)

invariant all shortest paths with up to i hops are found

$d(w) := -\infty$ for all w reachable from edges $e = (u, v)$ with $d(u) + c(e) < d(v)$

Refinements

- Only relax edges out of nodes v where $d(v)$ changed last round
- Stop after a round without updates
- Detect cycles early, e.g., scan in -edges every $\mathcal{O}(n)$ scanned edges.

3 Acyclic Graphs

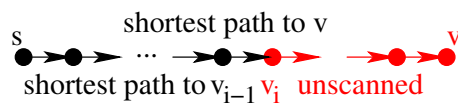
```
forall v ∈ V in topological order do
  forall e = (v, w) ∈ E do relax(e)
```

Correctness: the edges in any shortest path are relaxed in-order (Lemma 3)

time $\mathcal{O}(m + n)$

Correctness

Assuming incorrectness, consider the **first assertion violation** when deleting v .



$$d(v_i) = \mu(s, v_i) \leq \mu(s, v) < d(v)$$

hence v_i rather than v should be deleted !

4 Nonnegative Edge Weights

Dijkstra's Algorithm

q : *PriorityQueue* of Node with $d(\cdot)$ as priority

$d(s) := 0$

$q.insert(s)$

while $q \neq \emptyset$ do

$u := q.deleteMin()$

assert $d(u) = \mu(s, u)$

// label setting algorithm

foreach $(u, v) \in E$ do

if $d(u) + c((u, v)) < d(v)$ then

$d(v) := d(u) + c((u, v)); in[v] := (u, v)$

if $v \in q$ then $q.decreaseKey(v)$ else $q.insert(v)$

Analysis

$\mathcal{O}(n + m)$ outside priority queue.

$\leq n \times deleteMin, insert$

$\leq m \times decreaseKey$

Using different **integer** priority queues with max key C :

□ $\mathcal{O}(m + n^2)$

naive queue

□ $\mathcal{O}((m + n) \log n)$

binary heaps, **pairing heaps** →

□ $\mathcal{O}(m + n \log n)$

Fibonacci heaps

□ $\mathcal{O}(m + nC)$

bucket queues →

□ $\mathcal{O}(m + n \log C)$

radix heaps →

□ $\mathcal{O}((m + n) \log \log C)$

vebTrees →

□ $\mathcal{O}(m + n \log \log C)$

[Thorup 2003]

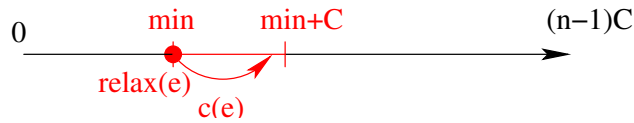
5 Monotone Discrete Priority Queues

A PQ q is **monotone** if “minima never decrease”

Assume edges costs are in $0..C$

⇒ Dijkstra keeps keys in $0..(n-1)C$ and

in $\min q.. \min q + C$



5.2 Radix Heaps

Idea: use narrow buckets close to min, use wide buckets further out

$K := 1 + \lceil \log C \rceil$

use buckets $B[-1], B[0], \dots, B[K]$

store v in $B[\min(K, msd(d(v), \min q))]$ where

$msd(a, b) :=$ most significant distinguishing bit index ($msd(a, a) := -1$)

Algorithm: *insert, decreaseKey*: analogous to bucket queue

Function *deleteMin()* : Node

$i := \min \{j \in -1..K : B[j] \neq \langle \rangle\}$

if $i \geq 0$ then

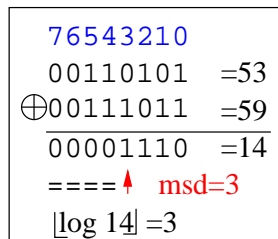
$minQ := \min B[i]$

foreach $v \in B[i]$ do

$j := \min(K, msd(d(v), minQ))$

$B[j].pushFront(B[i])$

return $popFront(B[-1])$



5.1 Bucket Queues

insert(v):

$B[d(v) \bmod (C + 1)].pushFront(v)$

decreaseKey(h):

$v := remove(h); insert(v)$

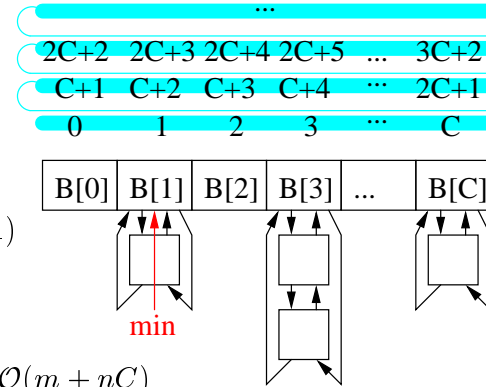
deleteMin:

```
while B[i] = ⟨ ⟩ do
  i := (i + 1) mod (C + 1)
return B[i].popFront()
```

space: $\mathcal{O}(n + C)$

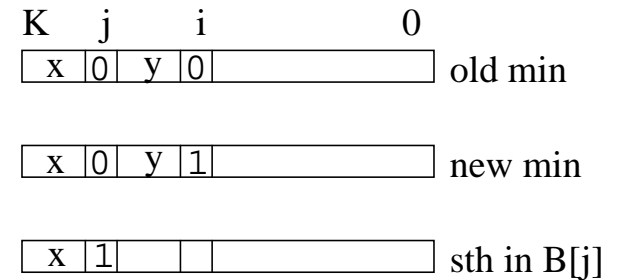
time: $\mathcal{O}(m + \max_{v \in V} \mu(s, v)) = \mathcal{O}(m + nC)$

bit parallel: $\mathcal{O}(m + \frac{\max_{v \in V} \mu(s, v)}{w})$, $w = \Omega(\log n)$ is word size



Correctness

in *deleteMin*, the $B[j], j > i$ remain correct



Analysis

insert, decreaseKey: $\mathcal{O}(1)$

deleteMin: $\mathcal{O}(1)$ if $i = -1$, $\mathcal{O}(K + |B[i]|)$ else

Amortized analysis: ($\mathcal{O}(K)$ insert, $\mathcal{O}(K)$ deleteMin, $\mathcal{O}(1)$ decreaseKey)

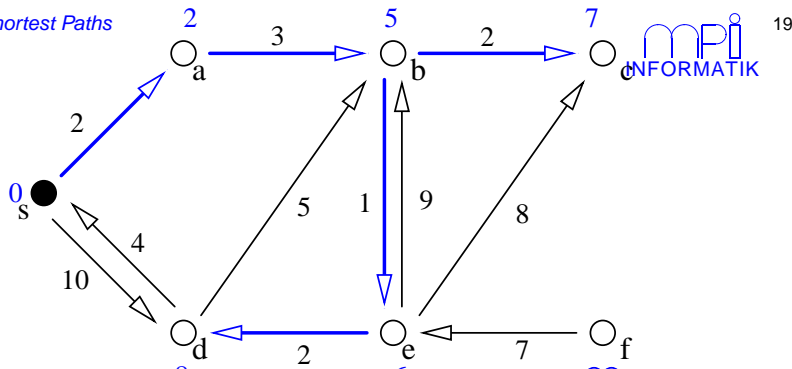
insert pays $K+1$ units insurance fee

insurance pays $|B[i]|$ to deleteMin (one unit per element moved)

Lemma 5. The insurance never goes bankrupt

Proof. We show that each element is moved at most $K + 1$ times.

Hence, it suffices to show that $j < i$ in the **foreach** loop □



operation	minQ	b_{-1}	b_0	b_1	b_2	b_3
insert(a, 2)	0			(a, 2)		
insert(d, 10)	0			(a, 2)		(d, 10)
deleteMin → a	2					(d, 10)
insert(b, 5)	2				(b, 5)	(d, 10)
deleteMin → b	5					(d, 10)

Lemma 6. v moves $B[i] \rightarrow B[h] \Rightarrow i > h$

Proof:

Case $i < K$:

all v 's agree by one add. bit

Case $i \geq K$.

$j := \text{msd}(\text{old min}Q, \text{min}Q)$

$h := \text{msd}(\text{min}Q, v)$:

Subcase $h < j$:

$\text{old min}Q \leq A + 2^j - 1$

$v \geq A + 2^j + 2^h$

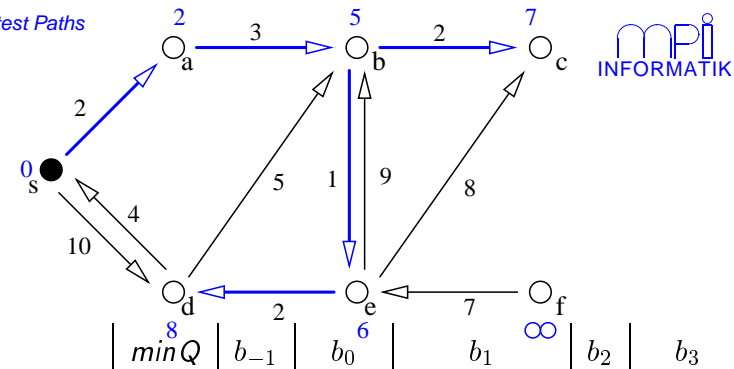
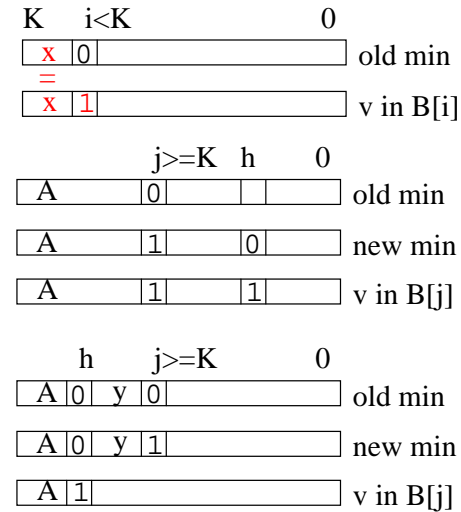
$\Rightarrow v - \text{old min}Q > 2^h$

$\Rightarrow h \leq \lfloor \log C \rfloor < K$

Subcase $h > j$:

$\text{old min}Q \leq A + 2^h - 1 - 2^j, v \geq A + 2^h$

$\Rightarrow v - \text{old min}Q \geq 2^j \geq 2^K > C$ contradiction



operation	minQ	b_{-1}	b_0	b_1	b_2	b_3
insert(c, 7)	5			(c, 7)		(d, 10)
insert(e, 6)	5			(c, 7), (e, 6)		(d, 10)
deleteMin → e	6		(c, 7)			(d, 10)
decreaseKey(d, 8)	6		(c, 7)			(d, 8)
deleteMin → c	7					(d, 10)
deleteMin → d	8					

5.3 Two-Level Radix Heaps

view keys as sequences of **radix- k digits**.

$B[i] := \{v : \text{the most significant differing digit of } v \text{ and } k \text{ is } i\}$

split $B[0]$ into k L2 buckets

Additional bit fiddling to find nonempty L2 buckets

gives time $\mathcal{O}\left(m + n \left\lceil \frac{\log C}{\log \log n} \right\rceil\right)$

Recursive description: A -bit integers, A a power of two

$|M| = 1$ or $A = 1$: store directly. Otherwise let $K = A/2$.

$|M| > 1$: store $\min M, \max M$,

t : store $\{x \text{ div } 2^K : x \in M\}$ (top recursion)

r_i : store $\{x \bmod 2^K : x \in M, x \text{ div } 2^K = i\}$ (bottom recursion) use hash table for space efficiency

5.4 Van Emde Boas Search Trees

maintain a set $M \subseteq 0..2^{2^K} - 1$

Operations: $\text{insert}(x)$, $\text{remove}(x)$, $\text{locate}(x) = \min \{y \in M : t \geq x\}$

time $\mathcal{O}(\log K)$ for all operations

space $\mathcal{O}(|M| \log K)$

Variants

- Priority queue for keys within a range of size $C = 2^{2^K} - 1$
- Sorted list of items
- Space $\mathcal{O}(|M|)$

Procedure $\text{insert}(x)$

```

if  $|M| = \{e\}$  then                                     //  $|M| = 1 \rightsquigarrow |M| > 1$ 
     $\min M := \max M := e$ 
    create  $t$  with the single element  $e \text{ div } 2^k$ 
    create  $r[e \text{ div } 2^K]$  with the single element  $e \bmod 2^K$ 
if  $r[x \text{ div } 2^k] = \emptyset$  then
    create  $r[x \text{ div } 2^K]$  with the single element  $x \bmod 2^K$ 
     $t.\text{insert}(x \text{ div } 2^K)$ 
else
     $r[x \text{ div } 2^K].\text{insert}(x \bmod 2^K)$ 
    update  $\min M$  and  $\max M$ 

```

Function $locate(x) : 0..2^{2K} - 1$

if $x > maxM$ then return ∞

if $x \leq minM$ then return $minM$

$(h\ell) := (x \text{ div } 2^K, x \text{ mod } 2^K)$

if $r[h] \neq 0 \wedge r[h].maxM \geq \ell$ then return $2^K \cdot h + r[h].locate(\ell)$

else

$h' := t.locate(h + 1)$

return $2^K h' + r[h'].minM$

Analysis

- $\mathcal{O}(1)$ time outside recursion
- at most on recursive call
- $\mathcal{O}(\log \#bits)$ levels of recursion

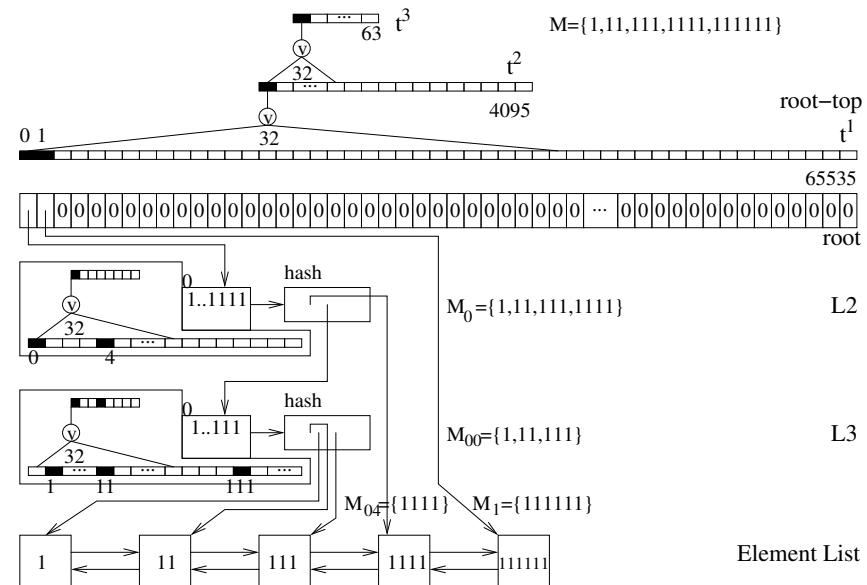
$\mathcal{O}(\log \log C)$ time for elements from $0..C$

what is the worst case space consumption?

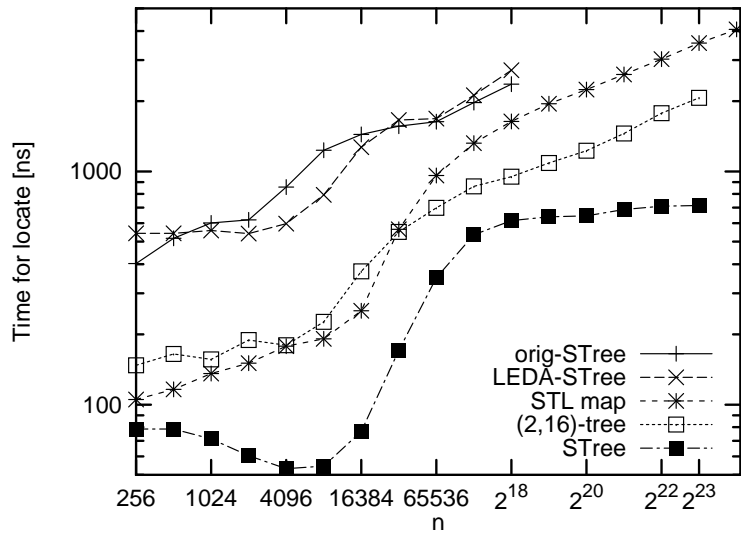
Refinements

- Add hash table for $\mathcal{O}(1)$ find, associated information
- Add sorted doubly linked list for succ, pred, range queries
- Break recursion early (use bit arrays)
- Do not recurse for small sets $|M| \leq 4$?
- Store only every $\mathcal{O}(\log \log |C|)$ elements (rest in doubly linked list) \rightsquigarrow linear space
- Use full array at root or somewhere in t
- Fast memory management
- Fast hash functions

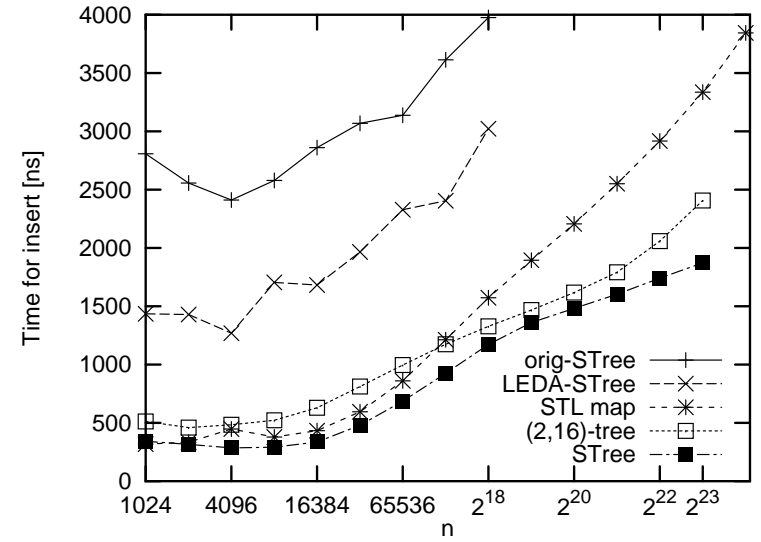
A tuned 32 bit Version



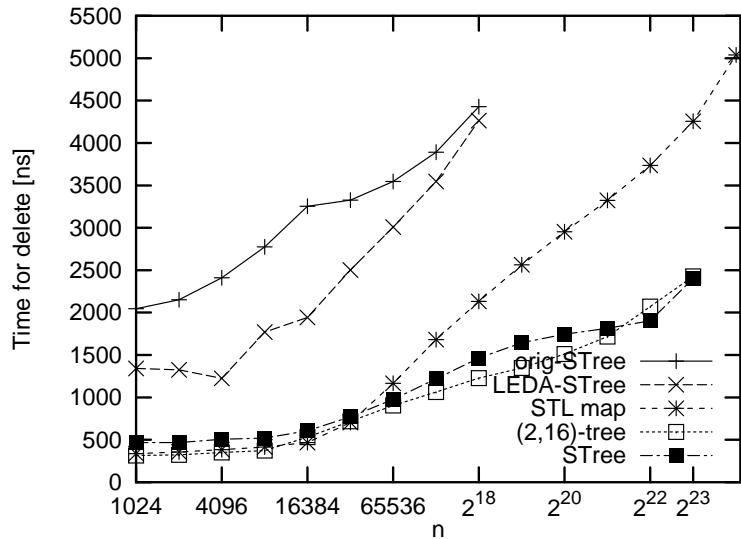
Running Time: *locate* (random 32 bit keys)



Running Time: *insert* (random 32 bit keys)



Running Time: *remove* (random 32 bit keys)



6 Average Case Analysis of SSSP

arbitrary (worst case) graphs

“random” edge weights

does this change the pecking order of algorithms?

6.1 decreaseKeys in Dijkstra's Algorithm

Experimental Observation:

Binary heaps beat Fibonacci heaps

$\mathcal{O}((n + m) \log n)$ better than $\mathcal{O}(m + n \log n)$???

Closer inspection: actual number of decreaseKey ops is much smaller than worst case for most inputs.

arbitrary (worst case) graphs and edge weight

but random assignment to the edges ($m!$ possibilities)

Theorem 7. $E[\# \text{decreaseKey ops}] \leq n \ln \frac{m}{n}$

Proof

e_i causes a decreaseKey

$$\Leftrightarrow i \geq 2, d(u_i) + c(e_i) < \min \{d(u_j) + c(e_j) : j < i\}$$

$$\Rightarrow i \geq 2, c(e_i) < \min \{c(e_j) : j < i\}$$

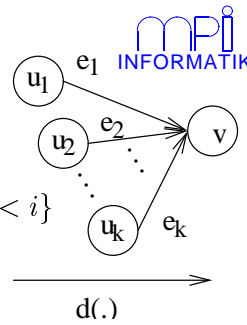
$c(e_i)$ s independent of $d(v_i)$

\Rightarrow all orders of $c(e_i)$ s equally likely

Lemma 8. $E[\# \text{of left-right maxima of } k \text{ els. in random order}] = H_k$

Hence,

$$E[\# \text{ops}] \leq \sum_{v \in V} (H_{\text{in-degree}(v)} - 1) \leq \sum_{v \in V} \ln \text{in-degree}(v) \leq n \ln \frac{m}{n}$$



Left-Right Minima

Consider randomly permuted values x_1, \dots, x_k

$$\text{prob}(x_i = \min \{x_1, \dots, x_i\}) = \frac{1}{i}$$

$$\sum_{i=1}^k \frac{1}{i} =: H_k \leq \ln k + 1$$

(k - th harmonic number)

6.2 Average Case Linear Time SSSP

q : RadixHeap of Node with $d(\cdot)$ as priority

$F = \emptyset$: Set of Node

// nodes with $d(v) = \mu(s, v)$

$d(s) := 0$

$q.insert(s)$

while $q \neq \emptyset$ **do**

while $F \neq \emptyset$ **do**

$u := F.deleteAny()$

foreach $(u, v) \in E$ **do**

if $d(u) + c((u, v)) < d(v)$ **then**

$d(v) := d(u) + c((u, v)); in[v] := (u, v)$

if $v \in q$ **then** $q.decreaseKey(v)$ **else** $q.insert(v)$

$F := F \cup q.deleteSettled()$

Let $\text{minIn}(v) = \min \{c((u, v)) : (u, v) \in E\}$

Function $\text{deleteSettled}() : \text{Node}$

$i := \min \{j \in -1..K : B[j] \neq \langle \rangle\}$

if $i = -1$ then return $\{\text{popFront}(B[-1])\}$

else

$F' := \emptyset$: Set of Node

$\text{minQ} := \min B[i]$

foreach $v \in B[i]$ do

if $d(v) \leq \text{minQ} + \text{minIn}$ then

$F' := F' \cup \{v\}$

else

$j := \min(K, \text{msd}(d(v), \text{minQ}))$

$B[j].\text{pushFront}(B[i])$

Lemma 10. $\sum_{i>1} i2^{-i} = 2$

$$\begin{array}{ccccccc}
 \frac{1}{2} & & & & & & \\
 \frac{1}{4} & \frac{1}{4} & & & & & \\
 \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & & & & \\
 \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & & & \\
 \vdots & \vdots & \vdots & \vdots & \vdots & & \\
 \hline
 1 + & \frac{1}{2} + & \frac{1}{4} + & \frac{1}{8} + & \dots = & 2
 \end{array}$$

Analysis

Model: arbitrary graph, random edge weights from $0..C$ ($K = 1 + \lceil \log C \rceil$)

Theorem 9. $E[\text{execution time}] = \mathcal{O}(m + n)$

Proof. node v only moves down to $B[\log \text{minIn}(v)]$ now

$$\begin{aligned}
 E[\sum \text{moves}] &\leq \sum_{v \in V} E[K + 1 - \log \text{minIn}(v)] \\
 &\leq n + \sum_{e \in E} E[K - \log c(e)]
 \end{aligned}$$

$$\begin{aligned}
 E[K - \log c(e)] &= \sum_{i=1}^K i \text{prob}(c(e) < 2^i) = \sum_{i=1}^K i2^{-i} \leq 2 \\
 &\leq n + 2m
 \end{aligned}$$

□

7 All Pairs Shortest Paths

find $\mu(s, t)$ for all $s, t \in V$.

output size $\Theta(n^2)$.

Nonnegative edge weights

$n \times \text{SSSP} \rightsquigarrow \text{time } \mathcal{O}(nm + n^2 \log n), \dots$

Arbitrary edge costs

Naive: $\mathcal{O}(n^2m)$

Idea: One general SSSP + $n - 1$ nonnegative SSSPs

Node potentials

find $pot(v) : V \rightarrow \mathbb{R}$ such that the

reduced cost $\bar{c}((v, w)) = pot(v) + c(e) - pot(w)$

is nonnegative.

Lemma 11. Shortest paths wrt \bar{c} are the same as with respect to c .

Proof. all but the first and last potential “telescope out”

$$\begin{aligned} \bar{c}(p) &= \sum_{0 \leq i < k} \bar{c}(e_i) = \sum_{0 \leq i < k} (pot(v_i) + c(e_i) - pot(v_{i+1})) \\ &= pot(v_0) + \sum_{0 \leq i < k} c(e_i) - pot(v_k) \\ &= pot(v_0) + c(p) - pot(v_k) \end{aligned}$$

□

The Algorithm

- new node s , zero weight edges (s, v) $\mathcal{O}(m)$
- compute $pot(v) = \mu(s, v)$ $\mathcal{O}(mn)$
- $\forall v \in V$ solve SSSP from v wrt \bar{c} $\mathcal{O}(n(m + n \log n))$
- set $\mu(v, w) = \bar{\mu}(v, w) + pot(w) - pot(v)$

Finding node potentials

Lemma 12.

Assume G has no negative cycles and all nodes can be reached from s .

With $pot(v) = \mu(s, v), \forall e \in E : \bar{c}(e) \geq 0$

Proof. $\forall (v, w) \in E : \mu(s, v) + c((v, w)) \geq \mu(s, w) \Leftrightarrow$
 $\mu(s, v) + c((v, w)) - \mu(s, w) \geq 0$ □

8 Speeding Up s - t Path Queries

What we (often) really want:

- just one s - t path
- s and t may be close together (e.g. SB Uni Mensa \rightarrow Johanniskirche)
 . . . no need to explore all of Europe
- additional information (e.g., geometry)
- preprocessing OK but limited space

8.1 A* Search

bound function $f(v, t) \leq \mu(v, t)$, e.g., Euclidean distance

q : PriorityQueue of Node with $d(v) + f(v, t)$ as priority

$d(s) := 0$

$q.insert(s)$

while $q \neq \emptyset$ do

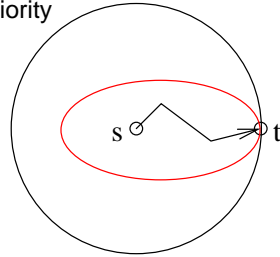
$u := q.deleteMin()$

 foreach $(u, v) \in E$ do

 if $d(u) + c((u, v)) < d(v)$ then

$d(v) := d(u) + c((u, v)); in[v] := (u, v)$

 if $v \in q$ then $q.decreaseKey(v)$ else $q.insert(v)$



When does this work?

The algorithm is equivalent to ordinary Dijkstra using the **node potential**

$pot(v) = -f(v, t)$.

Hence, it works if $\forall e \in E : \bar{c}(e) \geq 0$, i.e.,

$\forall e = (v, w) \in E : c(e) + f(w, t) < f(v, t)$

a kind of **triangle inequality**

8.2 Preprocessing Heuristics

- Compute interesting areas for each edge
- Hierarchies** e.g., no slow trains for most of a long trip (but careful Lux-Metz-Trier-SB?!)
- Bidirectional Search