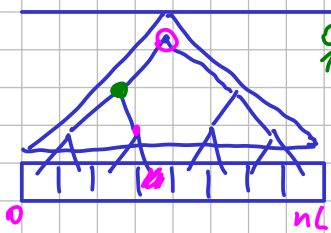
Insert
ExtractMin
Decrease

Dijkstra's alg. takes time $O(n \cdot T_I + n \cdot T_X + m \cdot T_D)$



0
1 = there's a non-empty bucket in the subtree

$O(\log nL)$ for Insert Decrease Ext.Min

$O(\log n + \log L)$

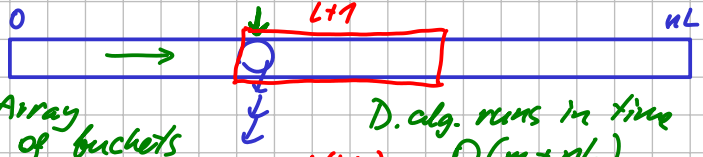
every op. on buckets translates to $O(1)$ ops on blocks

↳ $O(\log L)$ per op.

D.a. $O(m \log L)$

reduction from DS for $\{0 \dots nL\}$ to DS for $\{0 \dots L\}$

$\forall e \; \ell(e) \in \{0 \dots L\} \Rightarrow \forall v \; h(v) \le n \cdot L$



Array of buckets

D. alg. runs in time $O(m + nL)$
space $O(L)$ ... $O(n)$

van-Emde Boas Tree
(explained in Data Structures 2)

↳ $O(\log \log L)$ time per op.

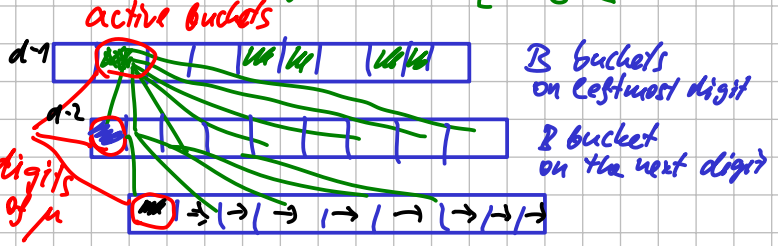
D.a. : $O(m \cdot \log \log L)$

Multi-Level Buckets

numbers in base B

B := base (a power of 2)

d := # of digits $d = \lfloor \log_B L \rfloor + 1 \in \Theta\left(\frac{\log L}{\log B}\right)$



State: parameters L, B, d

stack of levels, each is an array of B buckets, each b. contains a list of items

($\exists$ in array)

μ := last value extract-min'd

Invariant: At any given level, buckets preceding the active one are empty

Inv. #2: Items never move upwards.

ExtractMin does:

- scanning buckets $O(B)$
- removing levels $O(B)$
- find min $O(1)$ per item
- creating levels $O(B)$ → ExtractMin → Insert
- distributing items $O(1)$ per item
  ↳ $O(d)$ times during lifetime of the item

Insert(x): in time $O(1)$

$i := \left\lfloor \frac{MSB(x \oplus \mu)}{\log B} \right\rfloor$

Insert to level $\max(i,$ bottommost existing level$)$

$O(1)$ time

Decrease: Delete & Insert $O(1)$ time

ExtractMin:

- look for non-empty bucket at the last level, going from active bkt. to the right
  → if the level ends, discard the level & continue in the level above it
- have a non-empty bucket
  - if at level 0: remove any item from the bucket
  - otherwise:
    - scan all items in the bucket
    - find & remove min
    - create a new level & distribute items from current bucket there.

if bucket is not empty

$O(B+d)$ Insert
$O(1)$ Decrease
$O(1)$ ExtMin

Recall: $d \in \Theta\left(\frac{\log L}{\log B}\right)$ $\quad B = \frac{\log L}{\log\log L} \rightarrow B \log B = L$

Cost of Insert: $O\left(B + \frac{\log L}{\log B}\right)$ $\quad$ Set $B := \frac{\log L}{\log\log L}$ $\rightarrow$ amortized cost of Insert became

$\hookrightarrow O\left(\frac{\log L}{\log\log L} + \frac{\log L}{\log\log L - \log\log\log L}\right)$, $\Theta(\log\log L)$

$O\left(\frac{\log L}{\log\log L}\right)$

↓

D.a. runs in $O\left(m + n \cdot \frac{\log L}{\log\log L}\right)$

Even better: HOT queue (Goldberg et al.)

Heap On Top of $O(\sqrt{\log L})$ per Insert

Trick by Dinitz for non-integer lengths

If the $\ell(e) \geq \delta$, all open vertices with $h(v) < \left(\min_{v \text{ open}} h(v)\right) + \delta$ can be closed.

In the heap we can replace $h(v)$ by $\left\lfloor \frac{h(v)}{\delta} \right\rfloor$. & this is monotonous $\rightarrow$ D.S. with $L = \frac{\max_e \ell(e)}{\min_e \ell(e)}$

What to do with $\ell(e) < 0$?

① add some Δ to all $\ell(e)$ ... FAIL!

② Df: A potential $\varphi : V \to \mathbb{R}$.

Df: Reduced edge lengths wrt. φ:

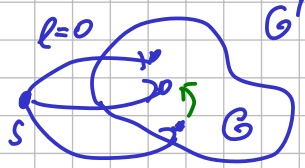
$\ell_\varphi(uv) := \ell(uv) + \varphi(u) - \varphi(v)$

Df: A potential φ is feasible $\equiv$ $\forall e: \ell_\varphi(e) \geq 0$.

↳ using a feasible φ, we can remove all neg. edges & run Dijkstra!

Thm: In every graph with no neg. cycles, we can find a feasible φ in time $O(nm)$.

Pf:



edges sv for all $v \in V$

use BFM to compute $d(s,v)$ for all v

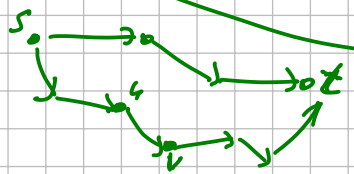
Set: $\varphi(v) := d(s,v)$

Why is φ feasible:

$\ell_\varphi(uv) = \ell(uv) + \varphi(u) - \varphi(v) \geq 0$ (want)

$\varphi(u) = d(s,u)$, $\varphi(v) = d(s,v)$

$\rightarrow d(s,v) \leq d(s,u) + \ell(uv)$ triangle ineq. for distance.



If P is a uv-path $u = v_1 v_2 \dots v_k = v$

$\ell_\varphi(P) = \sum_{e \in P} \ell_\varphi(e) = \sum_i \ell(v_i v_{i+1}) + \varphi(v_i) - \varphi(v_{i+1})$

$= \sum_{e \in P} \ell(e) + \varphi(u) - \varphi(v)$

All lengths of u,v-paths are increased by the same amount

↳ shortest paths are preserved

↳ we can find SP in G_φ instead!

→ APSP → A* (APSP)