

On RAM Priority Queues

Mikkel Thorup

SIAM Journal on Computing, Sept., 1999

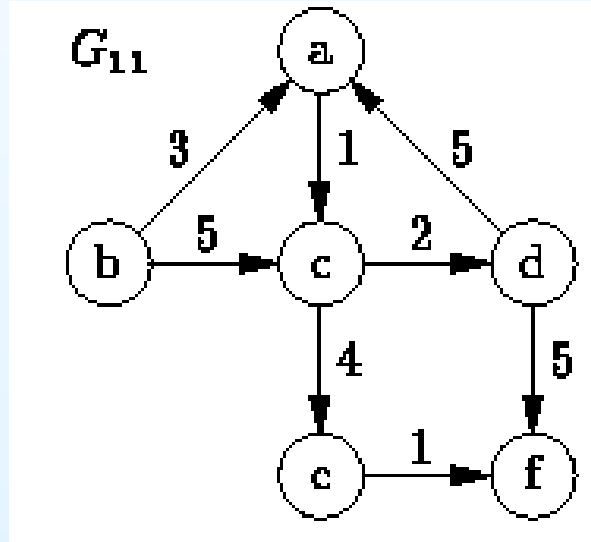
Overview

- Dijkstra's SSSP
- Short-Integer Queue
- Arbitrary Integer Queue
- Deletions
- Final Result

Single-Source Shortest-Path (SSSP)

Problem Description:

- Given directed graph $G = (V, E)$, source node $s \in V$:
 $\forall v \in V$ find minimum path cost $s \rightarrow v$
- *path cost* = sum of edge weights along path
- *non-negative* edge weights



Graphic taken from <http://www.brpreiss.com/books/opus5/html/page564.html>

Dijkstra's SSSP Algorithm

Data structures:

- Adjacency matrix E (edge weights)
- Path cost vector d : $d[i] = \text{min cost, node } i$
- Queue Q of “unfinished” nodes ordered by path cost
- Set S of completed nodes

Dijkstra's SSSP Algorithm

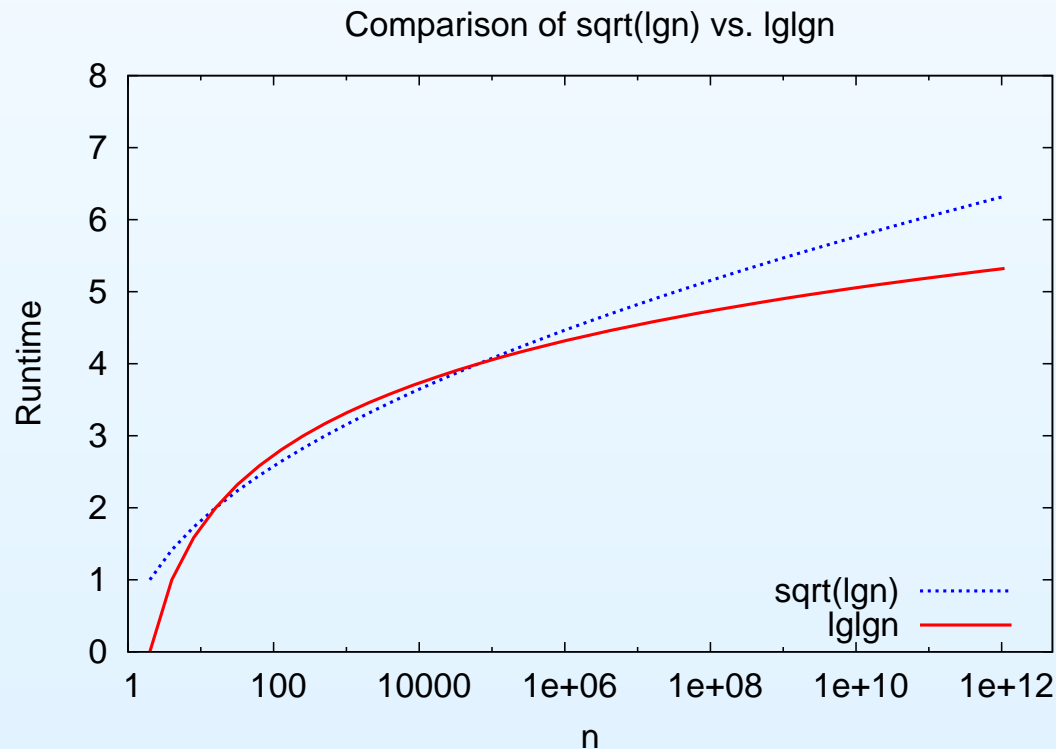
Algorithm:

1. $\forall v \in V, d[v] \leftarrow \infty; d[s] \leftarrow 0$
2. $S \leftarrow \emptyset$
3. $Q \leftarrow V[G]$
4. **while** $Q \neq \emptyset$
5. $w \leftarrow$ **Extract-Min**(Q)
6. $S \leftarrow S \cup \{w\}$
7. **for** $v \in Adj[w]$:
8. **if** $d[w] + E[w, v] < d[v]$
9. **Decrease-Key**($Q, v, d[w] + E[w, v]$)

Dijkstra's SSSP Algorithm

Queue Efficiency (graph with n nodes, m edges)

- Fredman/Willard (1993):
 $O(\sqrt{\lg m})$ per operation $\rightarrow O(m\sqrt{\lg m})$ overall
- Thorup: $O(\lg \lg m)$ $\rightarrow O(m \lg \lg m)$



Small-Integer Priority Queue

Albers and Hagerup (1997):

- Given two sorted lists containing $\leq k$ keys
 - key size $\leq w/k$ bits
 - list size ≤ 1 word
- Can merge in $O(\lg k)$ time
 - method similar to bitonic sorting

Insight: if $k = \lg n$, we get $O(\lg \lg n)$

Small-Integer Priority Queue

Data Structures:

- “reception” buffer B_0
- buffers $B_i, i = 1, \dots, \lg n - 1$
 - each $B_i, i \geq 1$ stores 0-2 sorted lists
 - each list holds $\lg n \cdot 2^{i-1}$ keys (2^{i-1} words)
- heap M of size $\lg n$
 - stores min value for each B_i
- multi-word merge broken into *merge steps*

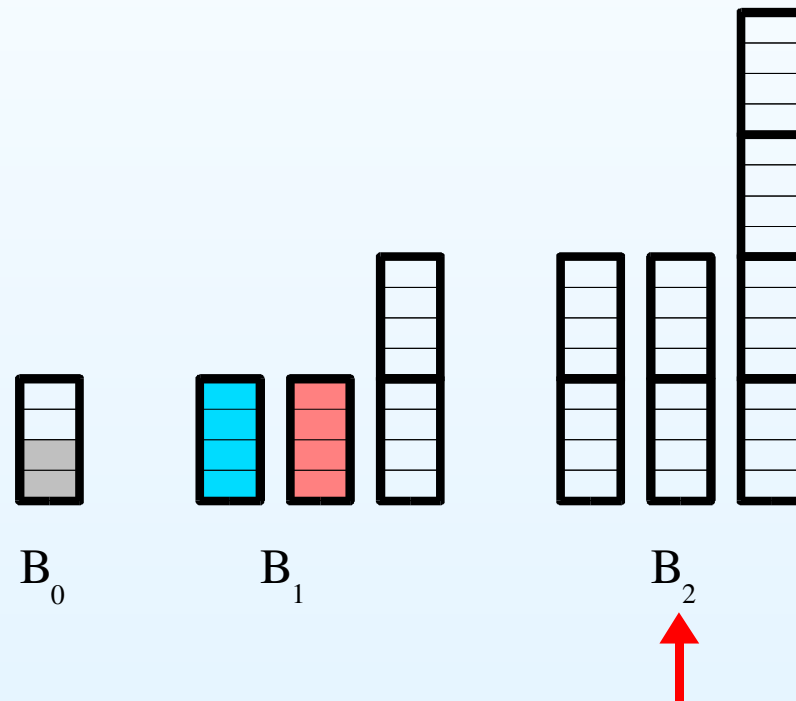
Small-Integer Priority Queue

Note: i initialized to $\lg n - 1$
 $\text{insert}(x)$

1. insert x into B_0
2. if $i = 0$ // “reception” buffer
3. move B_0 list to B_1 buffer
4. else if B_i has 2 lists
5. apply one merge step
6. if merge is complete
7. move merged list to B_{i+1}
8. $i \leftarrow i - 1 \bmod \lg n$

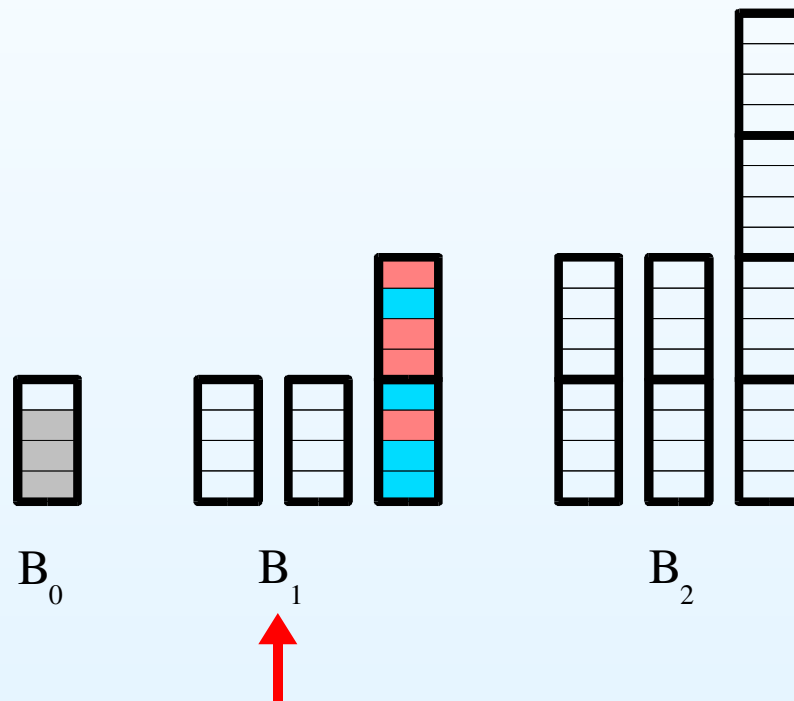
Small-Integer Priority Queue

Example: $i = 2$; (no merge for B_2)



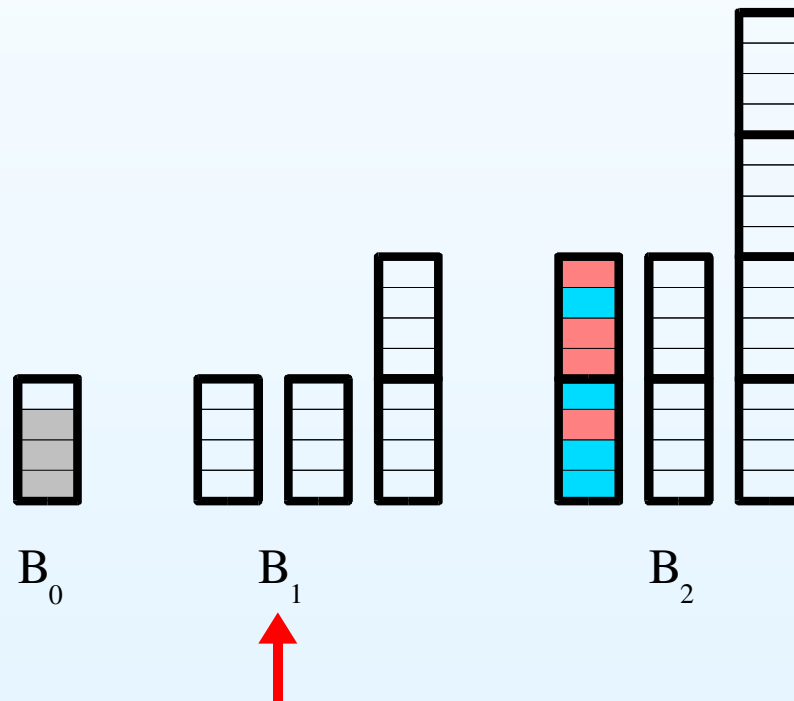
Small-Integer Priority Queue

Example: $i = 1$; merge lists in B_1



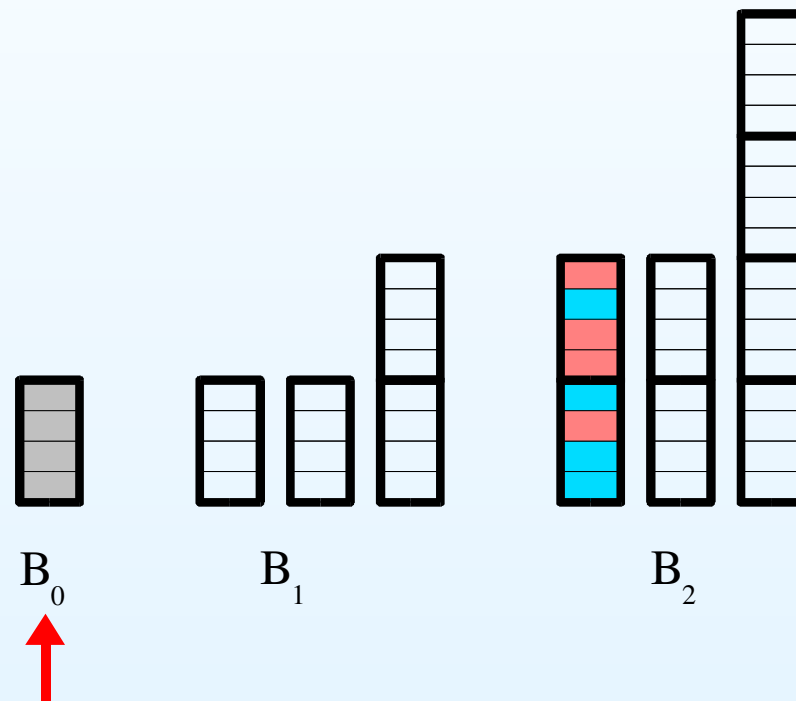
Small-Integer Priority Queue

Example: $i = 1$; move list $B_1 \rightarrow B_2$



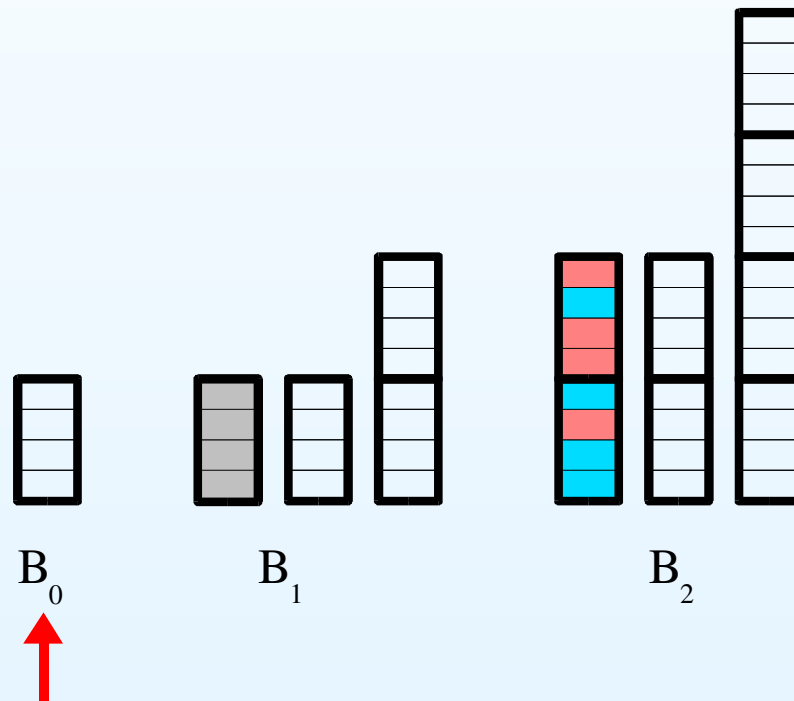
Small-Integer Priority Queue

Example: $i = 0$; B_0 full



Small-Integer Priority Queue

Example: $i = 0$; move B_0 into B_1



Arbitrary Integer Queues

Basic Strategy:

- Recursive range reduction
- b -bit integers where b is fixed
- Split key x into low $b/2$ bits, high $b/2$ bits:
 - $x = \text{high}(x) \cdot 2^{b/2} + \text{low}(x)$
 - high part $b/2$ -bit key queue
 - lookup tables for low part queues
- Leads to $T(n, b) = O(1) + T(n, b/2)$ for delete, insert-min
 - Recurrence halts after $O(\lg \lg n)$ times

Deletions

extract-min

1. remove min $\mu \in B_i$ from M
2. if B_i partially merged, mark μ
 - actual removal corrupts merging
 - defer until lists merged
3. new min μ' from $B_i \rightarrow M$
4. apply merge step to B_i
 - d delete steps \rightarrow merge done early
 - spend d to remove deferred items

Total = $O(\lg \lg n)$ amortized

Deletions

General deletions

- $\text{insert}(x)$ returns unique ID: i_x
 - NB: values $0-2^b \rightarrow i_x \sim \text{hashing}$
- $\text{delete}(i_x, Q)$ sets flag $D[i_x]$
- capacity $n \rightarrow 2n - 1$ indices
- rebuild when indices exhausted:
 - $O(n)$ extracts/inserts, total $O(n \lg \lg n)$
 - amortized $O(\lg \lg n)$

$\text{decrease-key}(x, i_x, Q) // O(\lg \lg n)$

1. $\text{delete}(i_x, Q)$
2. $\text{insert}(x)$

Punch Line

Queue operations:

- insert = $O(\lg \lg n)$
- delete $\sim O(\lg \lg n)$ (*amortized*)
- \therefore decrease-key $\sim O(\lg \lg n)$ (*amortized*)
- extract-min = $O(\lg \lg n)$

Dijkstra's SSSP - graph with n nodes, m edges:

- $O(m \cdot \lg \lg m)$ for $m \gg n$
- (previous best $O(m \cdot \sqrt{\lg m})$ for $m \gg n$)