1 Buckets

Cherkassky, Goldberg, and Silverstein. SODA 97. review shortest path algorithm.
In shortest paths, often have edge lengths small integers (say max $C$).
Observe heap behavior:

- heap min increasing (monotone property)
- max $C$ distinct values
- (because don’t insert $k + C$ until delete $k$).

Idea: lots of things have same value. Keep in buckets.
How to exploit?

- standard heaps of buckets. $O(m \log C)$ (slow) or $O(m + n \log C)$ with Fib (messy).
- Dial’s algorithm: $O(m + nC)$.

space?

- use array of size $C + 1$
- wrap around

2-level buckets.
Tries.

- depth $k$ tree over array of size $\Delta$
- depth $k$
- expansion factor $\Delta = (C + 1)^{1/k}$ (power of 2 simplifies)
- insert: $O(k)$ (also find, delete-non-min, decrease-key)
- delete-min: $O(k\Delta) = O(kC^{1/k})$ to find next element
- Shortest paths: $O(km + knC^{1/k})$
- Balance: $nC^{1/k} = m$ so $C = (m/n)^k$ so $k = \log(C)/\log(m/n)$
- Runtime: $m \log_{m/n}(C)$
- Space: $kn = n \log_{m/n} C$

Problems: space and time
Idea: be lazy!

- unique array on each level active
- keep other stuff piled up in list
• expand to buckets when reach
• each item descends one level per touch, never ascends
• charge to insert, pay for other ops by pushing items down
• In delete, need to traverse exactly one level to find next nonempty item
• (may also do pushdowns, but those are paid for)
• space to linear
• New time analysis:
  – $O(k)$ insert
  – $O(C^{1/k})$ delete
  – $O(1)$ decrease key
• paths runtime: $O(m + n(k + C^{1/k})) = O(m + n(\log C)/\log \log C)$
• Further improvement: heap on top (HOT) queues get $O(m + n(\log C)^{1/3})$ time
• Implementation experiments—good model for project

2 VEB


Idea

• idea: in bucket heaps, problem of finding next empty bucket was heap problem. Recurse!
• $b$-bit words
• $\log b$ running times
• thorup paper improves to $\log \log n$
• consequence for sorting.

Algorithm.

• need constant time hash table. non-trivial complexity theory, but can manage with randomization or slight time loss.
• queue $Q$ on $b$ bits is struct
  – $Q$.min is current min, not stored recursively
- Array $Q.low[]$ of $\sqrt{n}$ queues on low order bits in bucket
- $Q.high$, vEB queue on high order bits of elements other than current min in queue

• Insert $x$:
  - if $x < Q.min$, swap
  - now insert $x$ in recursive structs
  - expand $x = (x_h, x_l)$ high and low half words
  - If $Q.low[x_h]$ nonempty, then insert $x_l$ in it
  - else, make new queue holding $x_l$ at $Q.low[x_h]$, and insert $x_h$ in $Q.high$
  - note two inserts, but one to an empty queue, so constant time

• Delete-min:
  - need to replace $Q.min$
  - Look in $Q.high$.min. if null, queue is empty.
  - else, gives first nonempty bucket $x_h$
  - Delete min from $Q.low[x_h]$ to finish finding $Q.min$
  - If results in empty queue, Delete-min from $Q.high$ to remove that bucket from consideration
  - Note two delete mins, but second only happens when first was constant time.