Universal hashing: (recall)
- polynomial-size 2-universal hash family:
  - Pr[two elements collide] = \( \frac{1}{n} \)
  - \( \Rightarrow E[\#\text{collisions with an element}] = O(1) \)
  - e.g. \( x \mapsto (ax+b \mod p) \mod n \)
  - \( [a, m] \) \( \rightarrow \) random \( \rightarrow \) prime in \([m, 2m]\)
  - but max load can be \( \Theta(\sqrt{n}) \)

Perfect hashing: hashing with no collisions
- perfect hash function has no collisions on key set
- random hash function on \( n \) keys likely collides
  \( (n! \text{ perfect permutations, } n^n \text{ functions, ratio } = \Theta(\frac{\sqrt{n}}{e^n}) ) \)
- what about random hash functions on \( s < n \) keys?
  - \( E[\#\text{collisions}] = \left(\frac{s}{n}\right) \cdot \frac{1}{n} < \frac{1}{s} s^2 n \)
  - Markov's Inequality:
    - \( \Pr\{X \geq x\} \leq E[X]/x \)
    - \( \Pr\{\#\text{collisions} \geq 1\} < \frac{1}{s} s^2 n \)
    - \( \leq \frac{1}{s} \) if \( s = \sqrt{n} \).
  - \( \Rightarrow O(1) \) expected trials before success if \( s = \sqrt{n} \)

- Birthday Paradox shows optimality:
  - \( s \) samples of \( \{1, 2, \ldots, n\} \)
  - \( \Pr[\text{collision}] = (1-\frac{1}{n})(1-\frac{2}{n})\cdots(1-\frac{s}{n}) \)
  - \( \approx e^{-\frac{1}{2}s^2/n - \cdots - s/n} \approx e^{-\frac{1}{2}s^2/n} \)
  - \( \Rightarrow \) when \( s = \sqrt{n} \), \( \Theta(1) \) chance of collision
  - \( 23 \) for birthdays (even if independent)
2-level perfect hashing: reducing to linear space

- so far have quadratic-space perfect hashing: $s = \sqrt{n}$ keys in size-$n$ table
- hash $n$ keys into table of size $n$ using 2-universal hashing ($\Rightarrow$ collisions)
- build quadratic-size perfect hash table on keys in each bucket $b_i$ (randomly+repeat)

$\Rightarrow \text{total space } = \sum_{k=1}^{n} b_k^2 = \sum_{k=1}^{n} \left( \sum_{i=1}^{n} [i \in b_k] \right)^2 = \sum_{i=1}^{n} (b_i + 2 \sum_{j=1}^{n} C_{ij})$

$\Rightarrow \mathbb{E}[\text{total space}] = n + 2 \cdot \mathbb{E}[\text{total # collisions}] = O(s)$

- Markov Inequality $\Rightarrow \Pr[\text{space} \geq c \cdot n^2] \leq \frac{1}{2}$ for some $c$

$\Rightarrow O(1)$ expected trials until $\sum b_k^2 = O(n)$

- then build perfect hash table on each bucket using 2-universal hashing ($O(1)$ expected trials/bucket)

- expected linear time, guaranteed linear space
- easy: $6n$ cells of space
- hard: $n + o(n)$ cells of space [succinct data str.]

- Las Vegas algorithm: guaranteed correct, just time bound guarantee is probabilistic
- vs. Monte Carlo algorithm: answer correct with probability $\geq 2/3$, say
Derandomization:
- only $m^2$ top-level 2-universal hash functions
- try them all
- ditto for bottom-level hash functions
\[ \Rightarrow \text{time polynomial in } m, \text{ not } n \]

Dynamic:
- top-level 2-universal hash is dynamic
- allow for growth in bottom-level perfect hash by factor of 2 (up to $2b_k \Rightarrow 4b_k^2$ space)
- choose perfect hash for new element randomly
- execution as if element there in first place
- if collision, rebuild entire bucket
- if bucket overflows, rebuild 2x larger
- watch if $\sum b_k^2$ grows too big
  \[ \Rightarrow \text{rebuild entire structure} \]
Dijkstra's shortest-paths algorithm (recall)
- priority queue on vertices, keyed on distance estimate from source
- keys only decrease $\rightarrow m$ decrease-key's
- min only increases $\rightarrow n$ delete-min's
  **MONOTONICITY PROPERTY**
- standard run time:
  - $O(m \lg n)$ via binary heap
  - $O(m+n\lg n)$ via Fibonacci heap

**Bucketing:**
- often edge weights are small integers
  say in $\{1, 2, \ldots, C\}$
  $\implies$ keys in $\{0, 1, \ldots, (n-1)C\}$ $(\leq nC)$
- if $m$ is large, many equal keys (Pigeonhole)
- idea: keep equal keys together in a bucket
- store heap on $nC$ buckets
- but only $C+1$ buckets $\{x, x+1, \ldots, x+C\}$
  active at any time
  $\implies$ $O(m \lg C)$ via binary heaps ($\& O(C)$ space)
- $O(m+n\lg C)$ via Fibonacci heaps
- $O(m+nC)$ via simple array: [Dial 1969]
  - circular array of size $C+1$
  - pointer to current key $x$
  - advance pointer to next nonempty bucket
  - advance $\leq nC$ times by monotonicity
2-level bucketing:
- blocks of $b$ consecutive keys
  $\Rightarrow$ $nC/b$ blocks
- array of $nC/b$ block summaries (item counts)
- insert$(x)$ updates block $x \div b$ & its summary
- decrease-key$(x)$ ditto.
- delete-min scans summaries for nonempty block, then scans block
  - monotonicity $\Rightarrow$ $\leq nC/b$ total advancing through summaries
  - $\leq b$ to scan block (coalescing like keys)
  $\Rightarrow$ $n$ delete-min's cost $nC/b + nb$
  $\Rightarrow$ minimized when $C/b = b$, i.e. $b = \sqrt{C'}$
  $\Rightarrow$ $O(m + n\sqrt{C'})$ time for shortest paths
- as before, only $C+1$ active keys
  $\Rightarrow$ only need $O(C)$ space
- avoid cost of array initialization by re-using $C/b = \sqrt{C'}$ blocks

3-level bucketing:
- blocks, superblocks, summary
- block size $\sqrt[3]{C}$
  $\Rightarrow$ $O(m + nC^{1/3})$ time
- can we go to $O(m+n)$?
  NO: insert cost rises
k-level bucketing (tries)
- depth-k tree over key space of \( nc \)
  \( \Rightarrow \) branching factor of \( \Delta = (C+1)^{1/k} \)
  except at top where it's \( n\Delta \)
- insert, decrease-key: \( O(k) \) time
- delete-min: \( O(k\Delta) \) to navigate lower levels
  + \( O(n\Delta) \) overall to advance top level
\( \Rightarrow \) total cost: \( O(mk + nk\Delta) \)
\( \Rightarrow \) minimized when \( m = n\Delta \) i.e. \( C = (m/n)^k \)
  i.e. \( k = \frac{\lg C}{\lg \frac{m}{n}} \)
\( \Rightarrow \) \( O(m \log_{m/n} C) \) time
- space = \( kn = O(n \log_{m/n} C) \)

Q: \( nk\Delta \) vs. \( n\Delta \) seems sloppy...
  is a different division better? (by say \( \lg \lg \) factor)
Lazy version: [Denardo & Fox 1979]
- just one active root-to-leaf path to current minimum
- for each branch off of active path, accumulate linked list of stuff
- push stuff down "as necessary"
- insert(x):
  - walk down tree
  - if fall off active path: add x to list
  - else: add to leaf block
- decrease-key(x):
  - remove from current list
  - descend active path until fall off (can't go left of it by monotonicity)
  - add to list
- delete-min:
  - remove item from bottom of active path
  - walk up to first nonempty list hanging off right
  - expand & push elements down to form new active path (& find new min)
Analysis:
- each element descends k times
  \[ \Rightarrow \text{total descent cost (pushing down)} \leq n \cdot k. \]
- insert costs \( O(k) \) (and pays for future descent)
- decrease-key pays \( O(1) \) to go up 1,
  remaining \( O(k) \) charged to descent \( \Rightarrow O(1) \) am.
- delete-min: \( O(k) \) to rise, descent free,
  \( O(C^{1/k}) \) to scan leaf block
  & up-down transition block
  \[ \Rightarrow O(m + n (k+C^{1/k})) \text{ time} \]
  \[ \Rightarrow \text{minimized when} \quad k = C^{1/k}, \quad i.e. \quad C = k^k \]
  \[ i.e. \quad k = \lg C / \lg \lg C \]
  \[ \Rightarrow O(m + n \ \lg C / \lg \lg C) \text{ time} \]
  \[ \quad \text{better than Fibonacci!} \]
- \( O(n + k C^{1/k}) \) space \( \sim \ \text{typically} \ 0(n) \)

Further improvements:
- Heap On Top (HOT) queues:
  \[ O(m + n \ \lg^{1/3} C) \text{ time} \]
  \[ \text{[Cherkassky, Goldberg, Silverstein - SICOMP 1999]} \]
- van Emde Boas \[ \text{[Lecture 6]} \]
  \[ O(m \lg \lg (nC)) \text{ time} \]