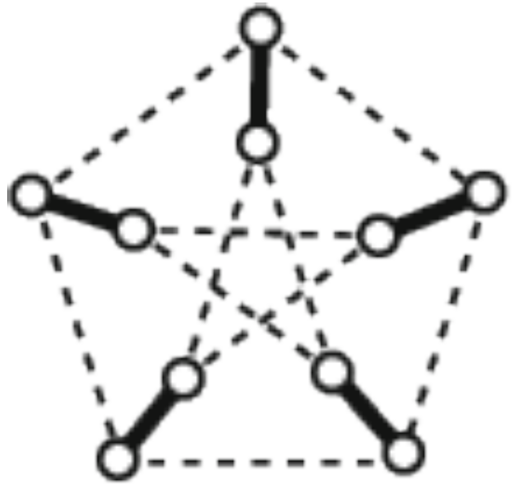
Cuckoo Hashing and Skip Lists

Admin

- Assignment 9 Thursday
- You can take the final at any time from Sat Dec 12- Sun Dec 20
- Distribution not yet clear.
- Any questions?

Problem 4 Hint



- Just to get a head start since we've only seen a couple approximation algorithms

- Maximum matching: largest possible set of edges such that each vertex is adjacent to at most one edge

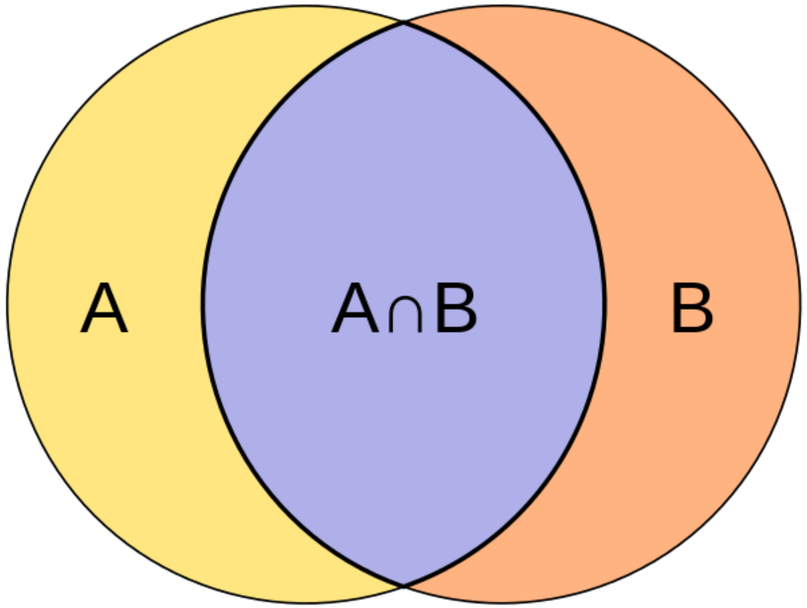
- How does the number of edges in a maximum matching (each vertex is adjacent to only one edge) compare to the size of the optimal vertex cover?

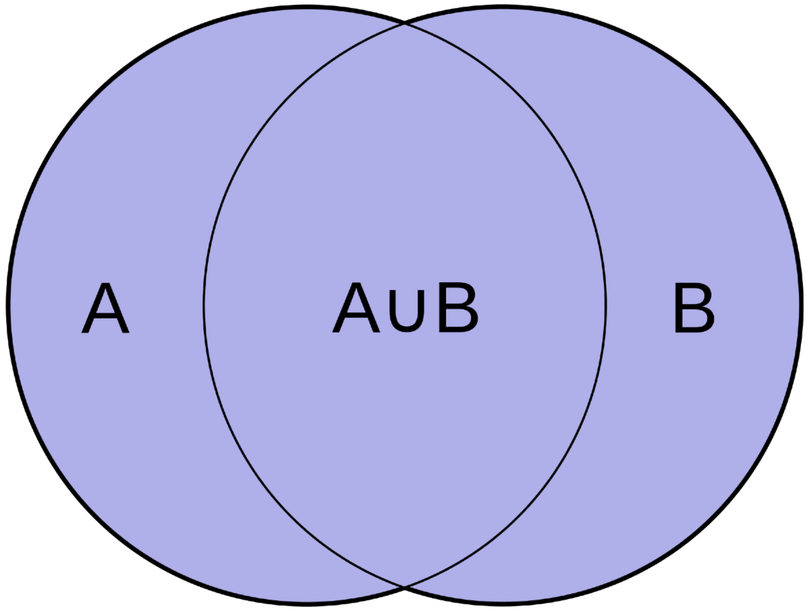
- How does the number of edges in a maximum matching compare to the size of the output of the algorithm?

Back to Linear Probing

Union Bound

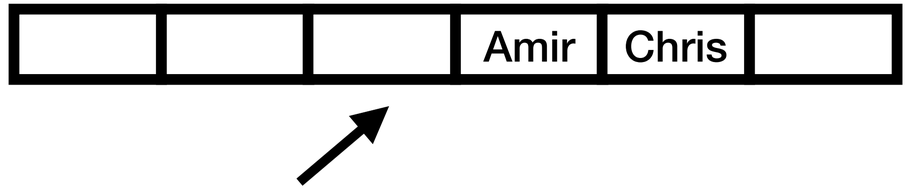
- Upper bound on the probability that two events happen
- Remember: events are a set of outcomes. So for *any* events A, B:
- $\Pr\left[A \text{ or } B\right] = \Pr[A] + \Pr[B] - \Pr[A \text{ and } B]$
- Probabilities are nonnegative. So:
- $\Pr[A \text{ or } B] \leq \Pr[A] + \Pr[B]$
- Union bound!





Linear Probing: w.h.p. Analysis

- All operations are $O(\log n)$ w.h.p.

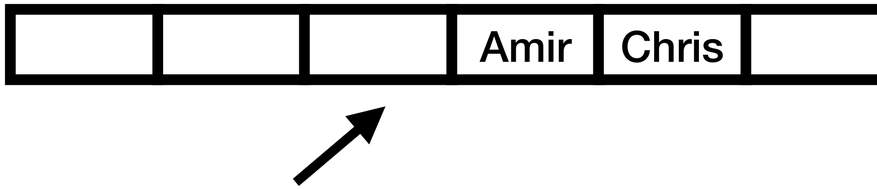


- Here's a sketch of why this is the case:
- What is the probability that, given that this slot is empty, the next $8 \log n$ slots are full?
- Must have exactly $8 \log n$ elements hashing to those $8 \log n$ slots
- Probability:

$$\binom{n}{8\log n}\left(\frac{8\log n}{m}\right)^{8\log n}\left(1-\frac{8\log n}{m}\right)^{n-8\log n} \leq \left(\frac{ne}{8\log n}\right)^{8\log n}\left(\frac{8\log n}{m}\right)^{8\log n}\left(e^{\frac{-8\log n}{m}}\right)^{n-8\log n}$$

- $\leq (9/10)^{8\log n} \leq 1/n^2$ so long as $\frac{n}{m}e^{1+(8\log n)/m-n/m} = \frac{e^{.5+(8\log n)/m}}{2} \leq 9/10$

Linear Probing: w.h.p. Analysis



- The probability that a given slot is before $O(\log n)$ consecutive full slots is $\leq 1/n^2$

- What is the probability that in the *entire table*, there is *any* slot is an empty slot before $O(\log n)$ consecutive slots?

Union bound

- $\Pr[P_1 \vee P_2 \vee \ldots \vee P_m]$
- Since $P_i = 1/n^2$ for all i, $\Pr[P_1 \vee P_2 \vee \ldots \vee P_m] \leq m/n^2 = O(1/n)$

- So the probability that *any* insert in the hash table takes more than $\log n$ probes is $O(1/n)$

Improving the Bounds

- We need randomness in order to hash, but can we get worst-case bounds?

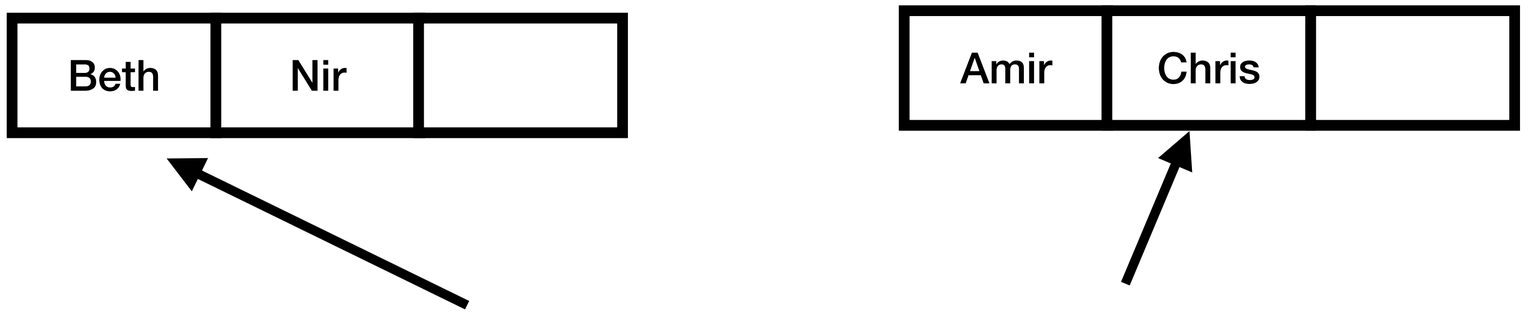
- We saw that we can get $O(1)$ worst-case insert, with $O(1)$ expected lookup

- But lookups are often what we care about more!

- Can we do the reverse? $O(1)$ worst-case lookup, with $O(1)$ expected insert (and $O(\log n)$ insert with high probability)?
- Yes—cuckoo hashing!

Cuckoo Hashing [Pagh, Rodler ’01]

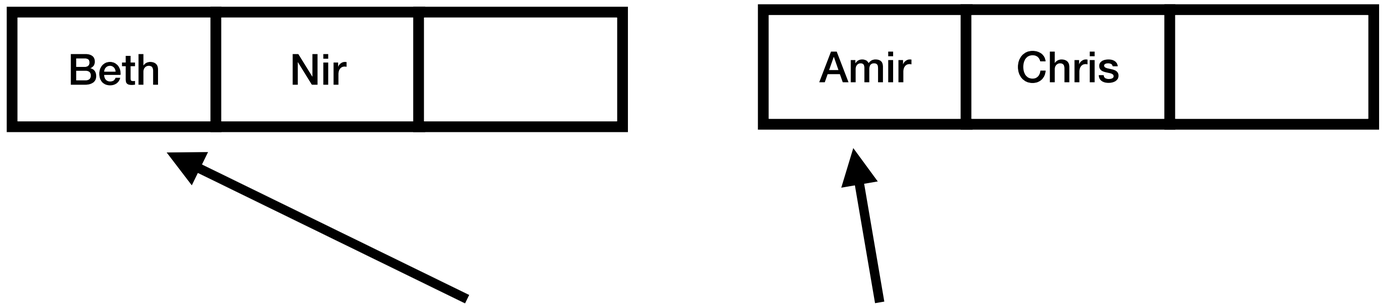
- Uses two hash functions, h_1 and h_2, two hash tables
- Each table size n
- Item i is guaranteed to be in $A[h_1(i)]$ or $A[h_2(i)]$
- So we can lookup in $O(1)$
- How can we insert?



$$h_1(\text{Beth}) = 0, \quad h_2(\text{Beth}) = 1$$

Cuckoo Hashing: Insert

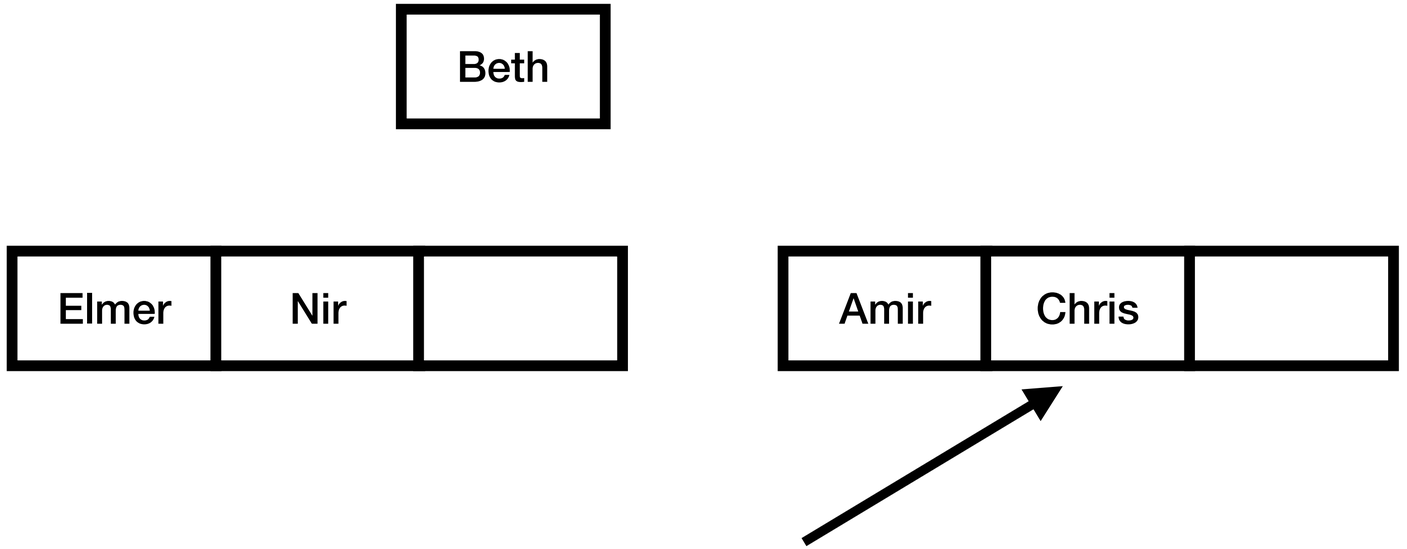
- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash



$$h_1(\text{Elmer}) = 0, \quad h_2(\text{Elmer}) = 0$$

Cuckoo Hashing: Insert

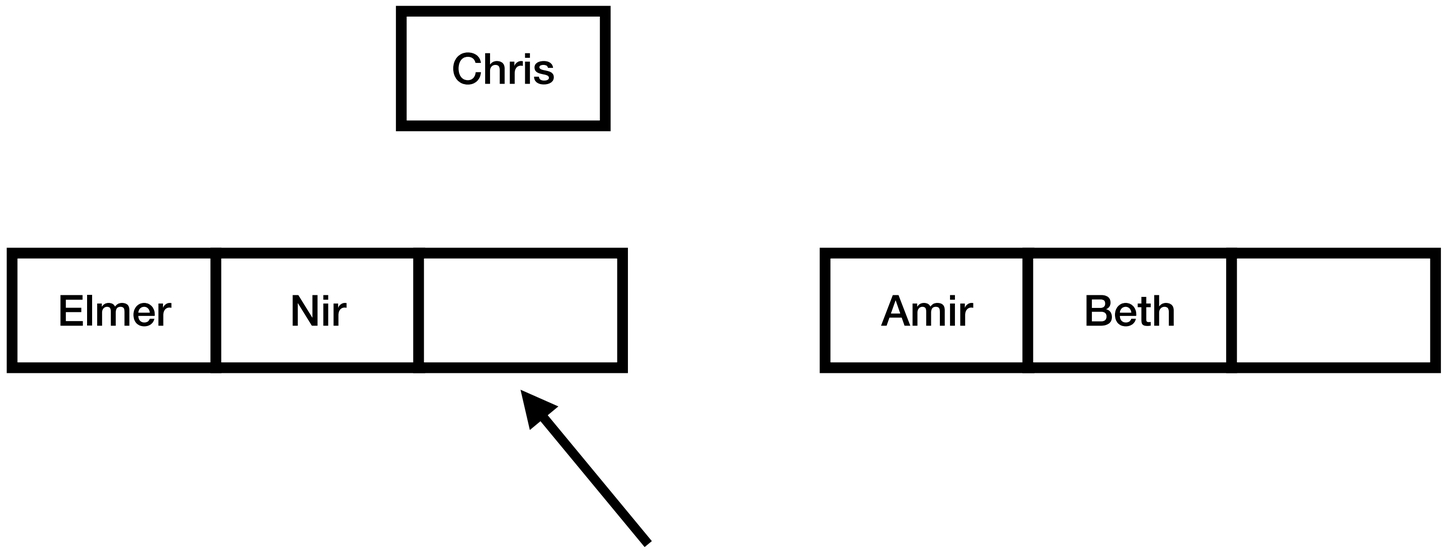
- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash



$h_1(\text{Beth}) = 0, \quad h_2(\text{Beth}) = 1$

Cuckoo Hashing: Insert

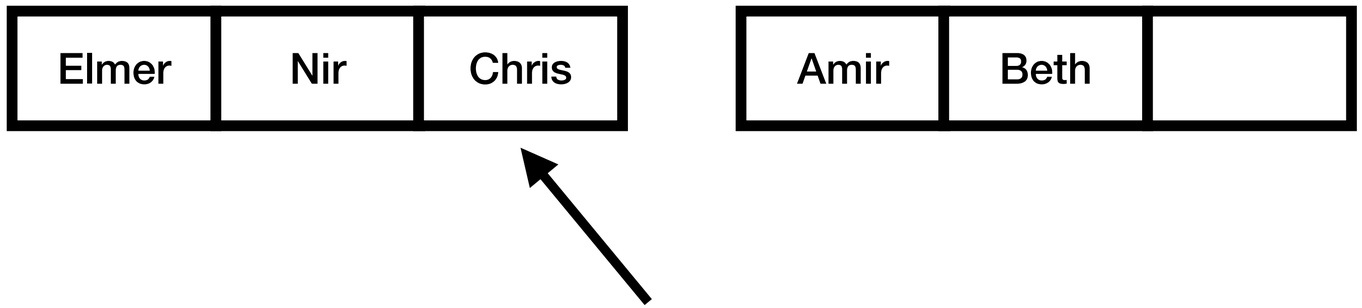
- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash



$h_1(\text{Chris}) = 2, \quad h_2(\text{Chris}) = 1$

Cuckoo Hashing: Insert

- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash



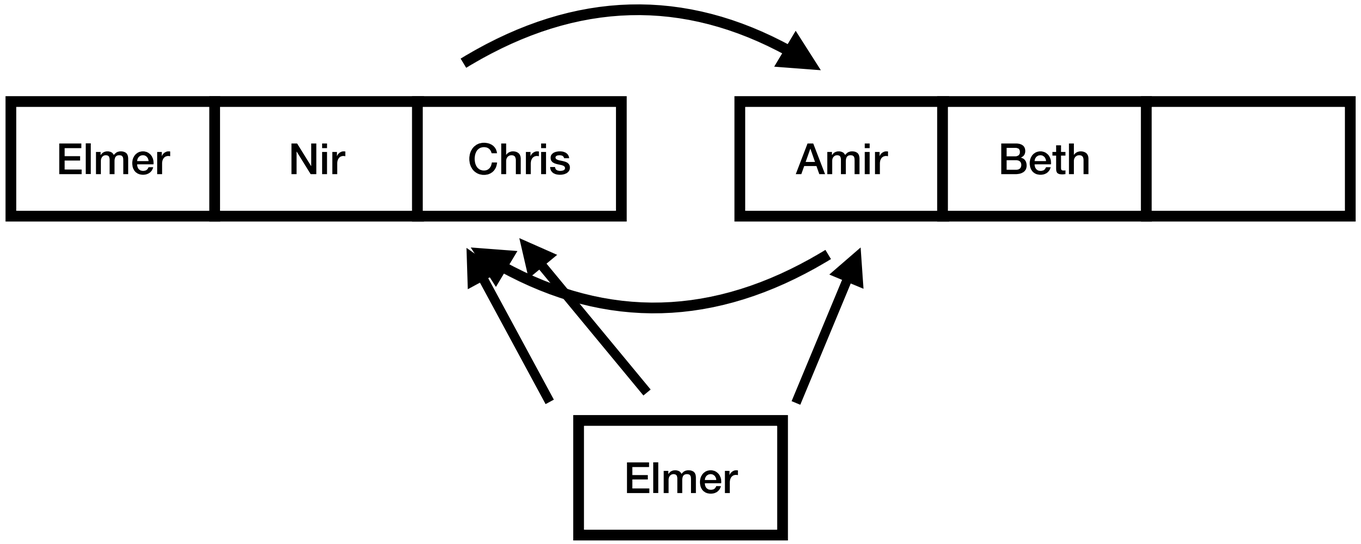
$$h_1(\text{Chris}) = 2, \quad h_2(\text{Chris}) = 1$$

Cuckoo Hashing: Insert

- What can go wrong?
- This process may not end
- Example: 3 items hash to the same two slots
- What is the probability that we have an insert to two slots, where each item in those slots only hashes to those two slots?
- $\binom{n}{2}\left(\frac{1}{n}\right)^4 = \Theta(1/n^2)$

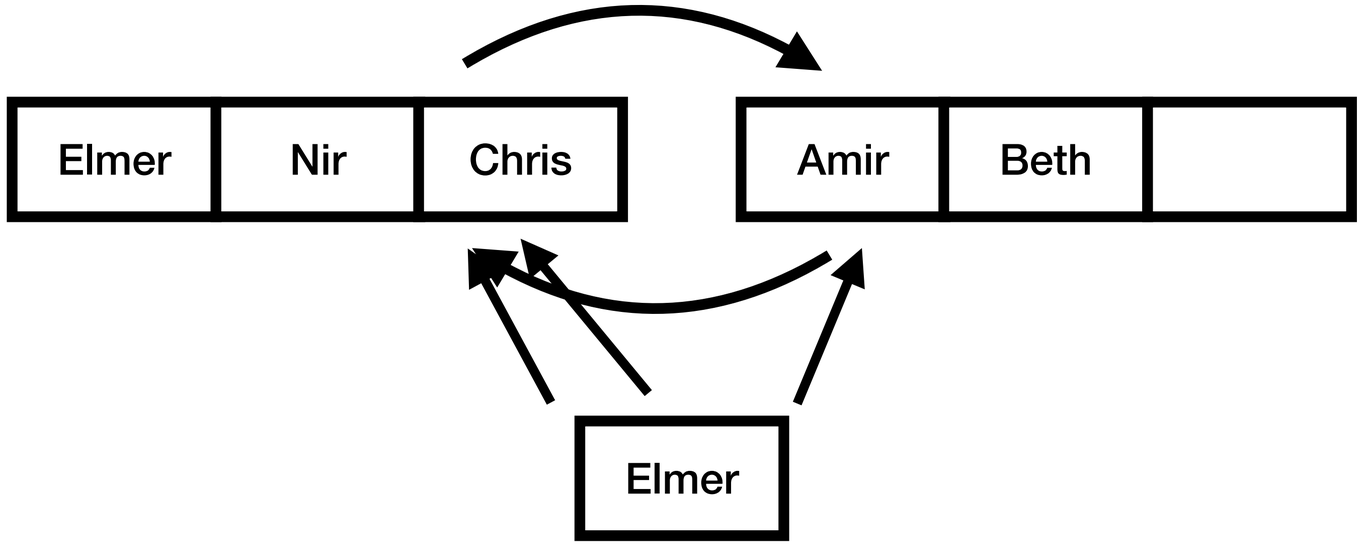
Ways to choose 2 items out of the n inserted

Probability that those two items hash to the given two slots



Cuckoo Hashing: Insert

- More complicated analysis:
- Cuckoo hashing fails with probability $O(1/n^2)$
- What happens when we fail?
- Rebuild the whole hash table
- (Expensive worst-case insert operation)



Cuckoo Hashing: Insert

- How long does an insert take on average?
- One idea: each time we go to the other table, what is the probability the slot is empty?
- $1/2$. (This analysis isn't 100% right due to some subtle dependencies, but it's the right idea)
- So need two moves to find an empty slot in expectation
- At most $O(\log n)$ with high probability

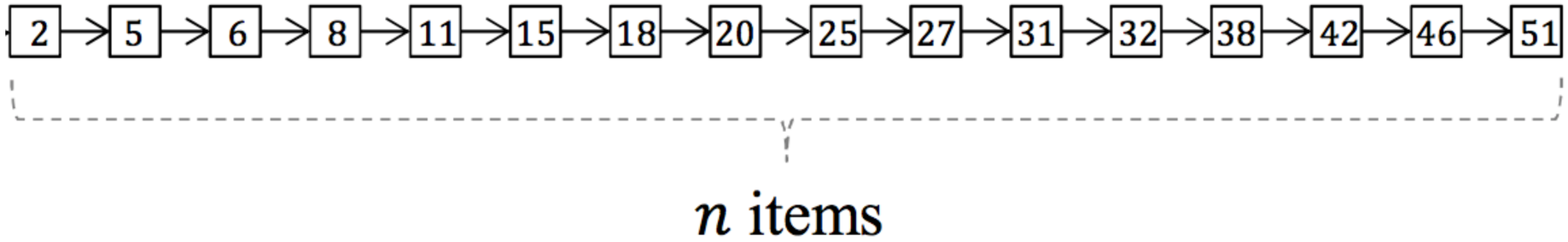
Skip Lists

Skip Lists: Randomized Search Trees

- Invented around 1990 by Bill Pugh
- Idea: binary search trees are a pain to implement
- Skip lists balance randomly; no rules to remember, no rebalancing
- Build out of simple structure: sorted linked lists
- Inserts, deletes, search, predecessor, successor are $O(\log n)$ with high probability
- No rebalancing makes them useful in concurrent programming (e.g. lock-free data structures)

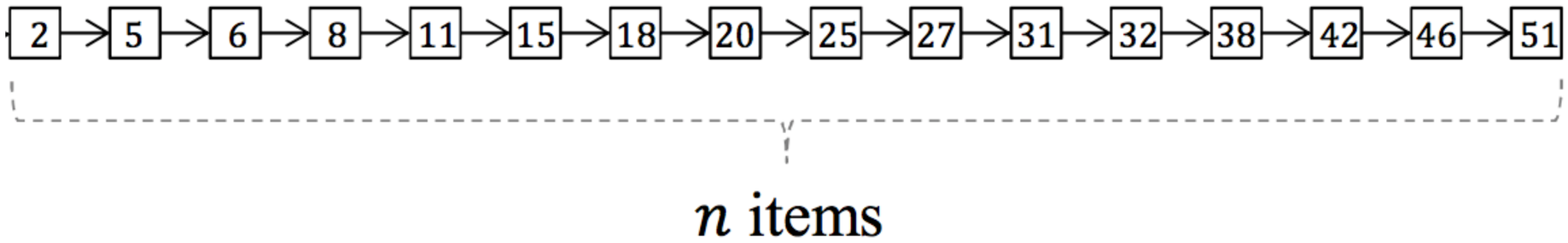
One Linked List

- Start from simplest data structure: (sorted) linked list
- Search cost?
  - $\Theta(n)$
- How can we improve it?



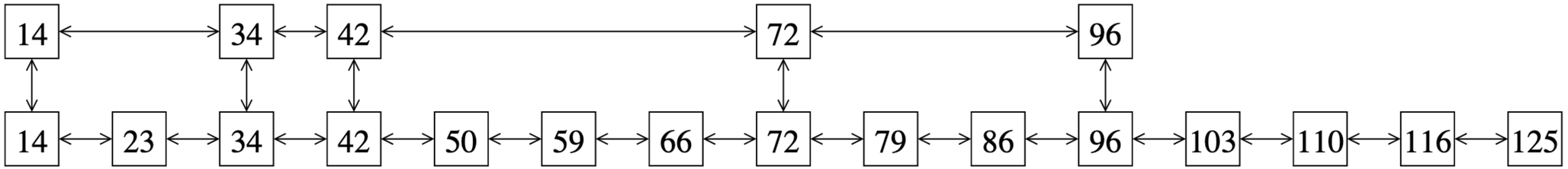
Two Linked List

- Suppose you had *two* sorted linked list
- Each element can appear in one or both lists
- How can you use two lists to improve search cost?



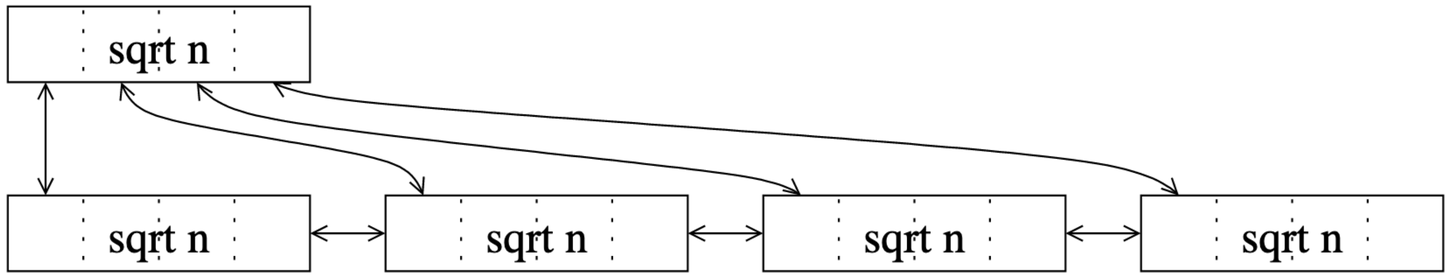
Two Linked List

- Suppose you had *two* sorted linked list
- Each element can appear in one or both lists
- How can you use two lists to improve search cost?
- Idea: have one “express” linked list, and one “local” linked list



Two Linked List

- How much gap between elements?
- If gap between elements in top list is g, then the number of elements traversed is at most $g + n/g$
- Optimized by setting $g = \sqrt{n}$. So the total cost is at most $2\sqrt{n}$



K Linked Lists

- Can you extend the previous idea to k linked lists?
- What is the cost of traversing them?
- $k(1 + n^{1/k})$
- Minimized at $k = \Theta(\log n)$
- Cost is $O(\log n(1 + n^{1/\log n})) = O(\log n)$

Skip List Details

- This is good, but how can we insert?
- Every new element disrupts our spacing
- Idea: use randomness!

Skip List Details

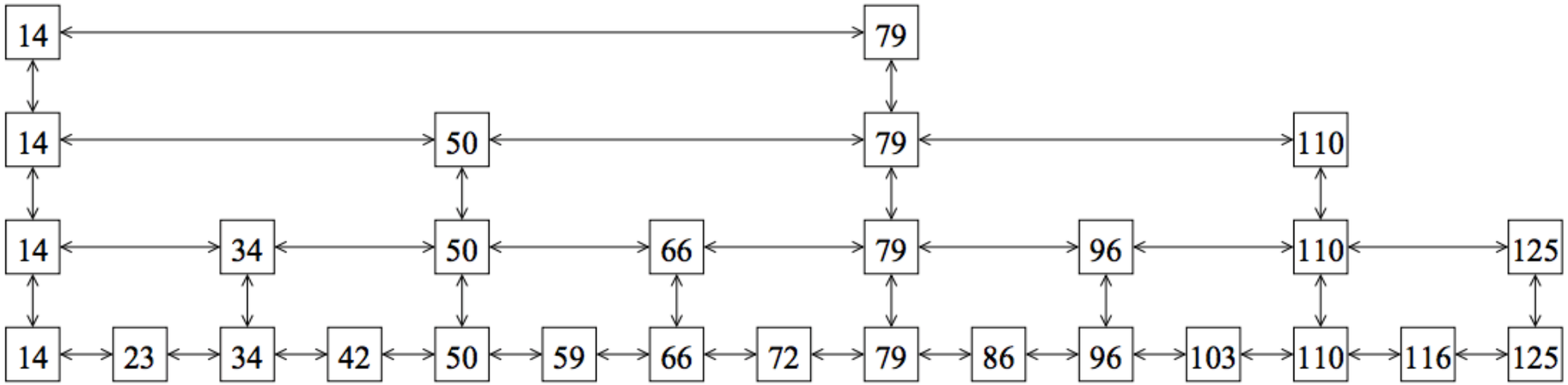
- Insert(x)
  - New element should certainly be added to the bottommost list
  - Invariant: Bottommost list contains all elements
  - Which other lists should a new item to added to?
  - Insert x at level 1 and flip a coin (idea we want half of the elements to go next level, similar to a balanced binary tree)
  - If heads: element gets promoted to next level, and we repeat
  - If tails element stays put at current level and we are done.

Skip List Details

- Thus, on average
  - $1/2$ of the elements go up 1 level
  - $1/4$ of the elements go up 2 levels
  - $1/8$ go up to 3 levels
  - Etc.
- Search(x):
    - Start at top list, go right just before value gets $>$ target
    - Go down and repeat until element is found or hit bottom right

Skip List Details

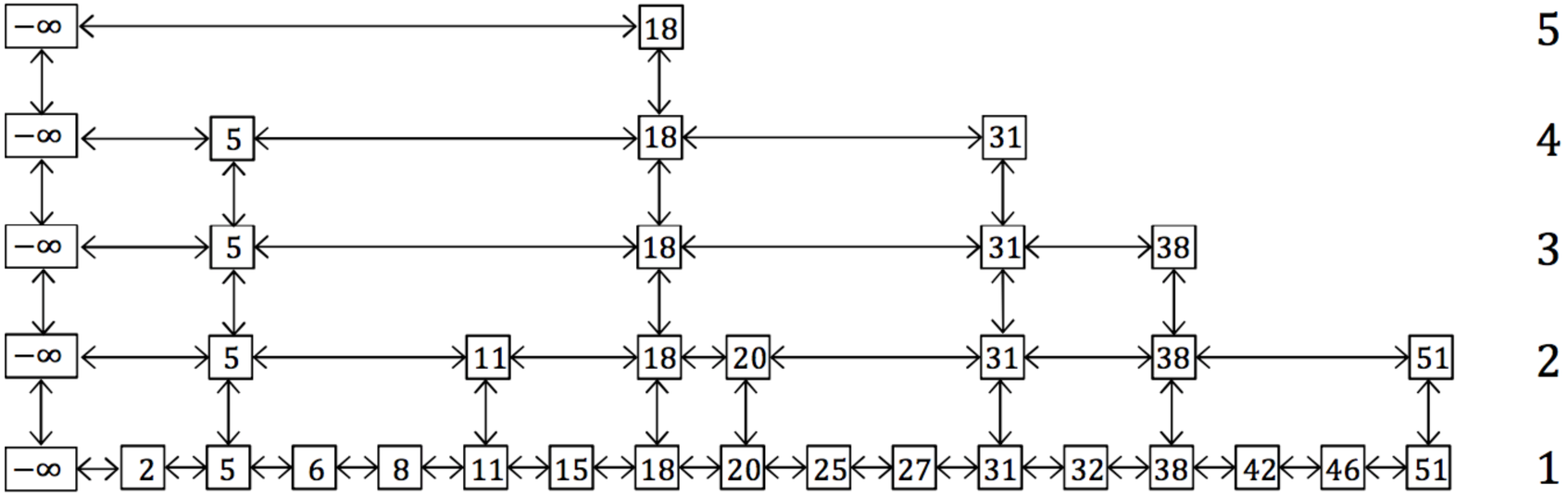
- Search(x):
  - Start at top list, go right just before value gets > target
  - Go down and repeat until element is found or hit bottom



- **Example:** Search for 72
  - Level 1: 14 too small, 79 too big; go down 14
  - Level 2: 14 too small, 50 too small, 79 too big; go down 50
  - Level 3: 50 too small, 66 too small, 79 too big; go down 66
  - Level 4: 66 too small, 72 spot on

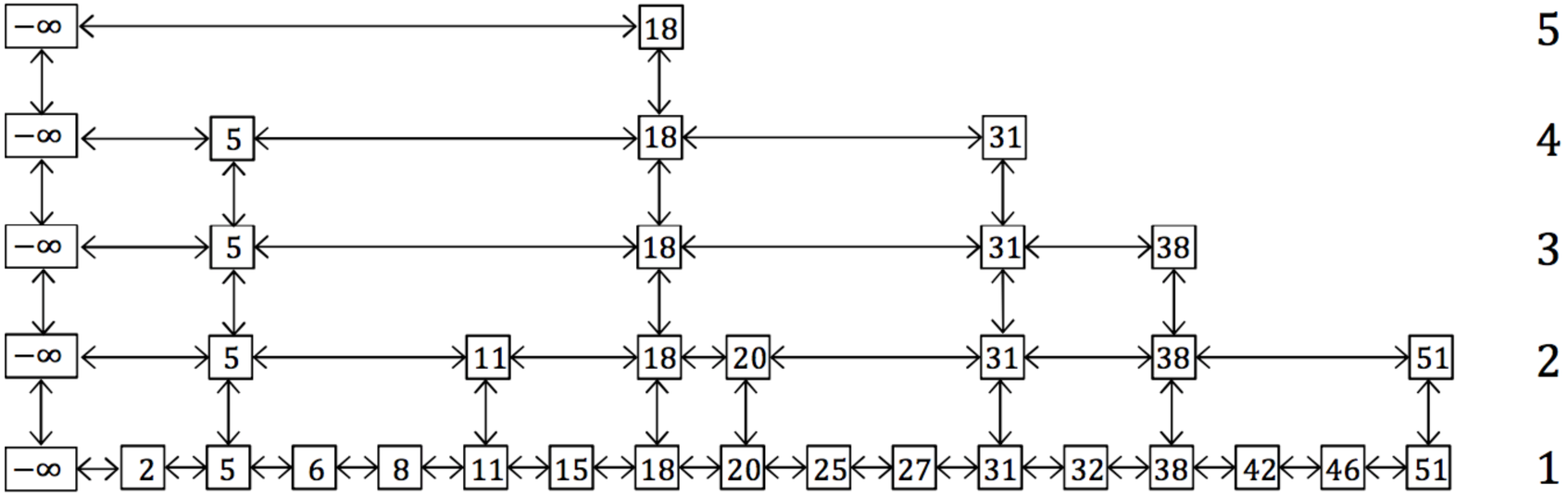
Skip List Analysis

- Let us first define the height of a skip list formally.
- Let L_k be the set of all items in level $k \geq 1$.
- Height of an element. $\ell(x) = \max\{k \mid x \in L_k\}$
- Height of a skip list. $h(L) = \max\{\ell(x) \mid x \in L_0\}$
-



Skip List Expected Analysis

- Expected height of a node:
  - $E[\ell(x)] = 1 + \frac{1}{2} \cdot 0 + \frac{1}{2}(1 + E[\ell(x)])$
  - $E[\ell(x)] = 2$
- Worst-case height? $h(L) = \max\{\ell(x) \mid x \in L\}$



Skip List Analysis

- **Claim.** A skip list with n elements has height $O(\log n)$ levels with high probability.
- Recall. (Informally) An event happens with high probability if the probability that it does not happen is polynomially small in n, that is, $\leq 1/n^c$ where the constant $c \geq 1$
- (More formally) Skip list of size n has $O(\log n)$ levels with high probability if the probability that it has more than $d \log n$ levels is at most $1/n^c$ where the constants c, d usually depend on each other
- **Proof idea.** What is the probability that an element gets promoted to level 1?
  - $1/2$

Skip List Analysis

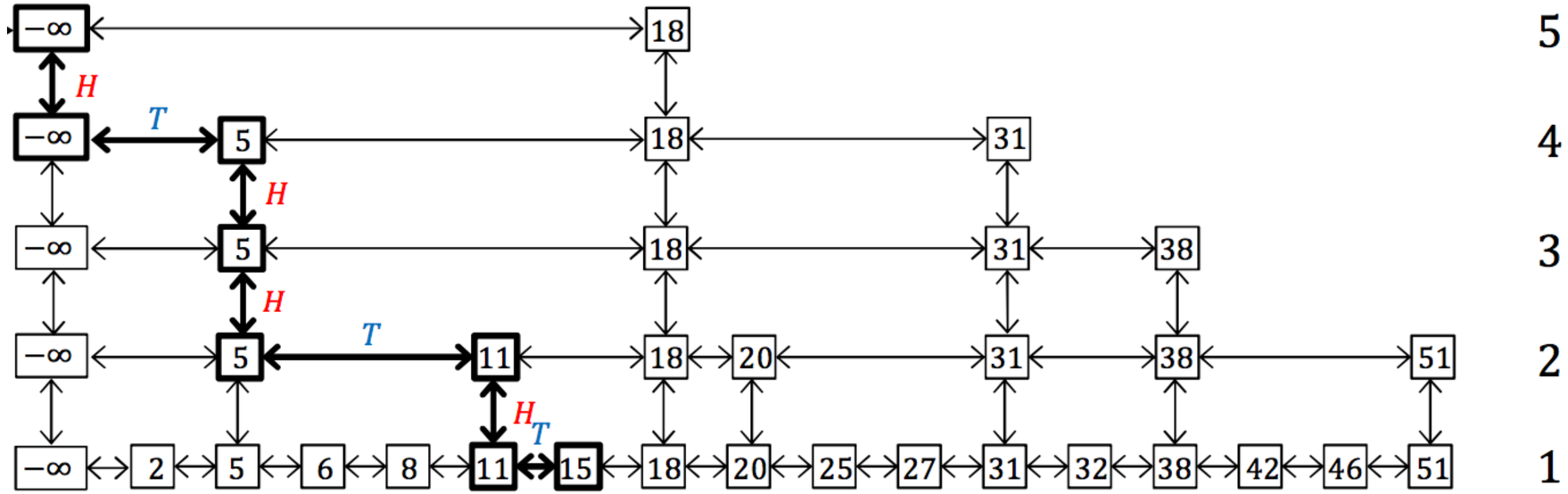
- **Claim.** A skip list with n elements has height $O(\log n)$ levels with high probability.
- **Proof.** For any $x \in L,\ k \geq 1$, the probability that height of x is k
- $\mathbf{Pr}[\ell(x) = k] = \frac{1}{2^k}$
- $\Pr[\ell(x) > k] = \sum_{k+1}^{\infty} \Pr[\ell(x) = i] = \sum_{i=k+1}^{\infty} \frac{1}{2^i} = \frac{1}{2^k}$
- $\Pr[h(L) > k] = \Pr[\cup_{x \in L} \ell(x) > k] \leq \sum_{x \in L} \Pr[\ell(x) > k] = \frac{n}{2^k}$

Union bound

- $\Pr[h(L) > c \log n] \leq \frac{1}{n^{c-1}}$ [pick any $c > 2$ for w.h.p.]
- Thus, height of skip is $O(\log n)$ with high probability

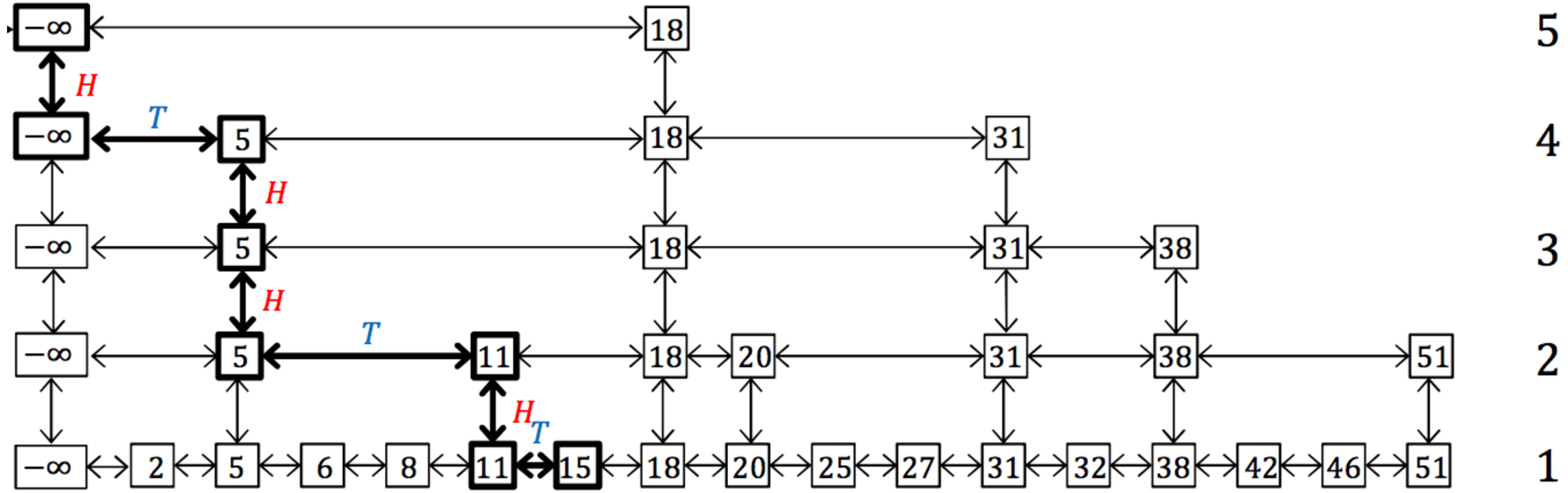
Skip List Search Cost

- **Claim.** Search cost in a skip list is $O(\log n)$ with high probability
- **Proof.**
- Idea think of the search path "backwards"
- Starting at the target element
- Going left or up until you reach root or sentinel node $(-\infty)$



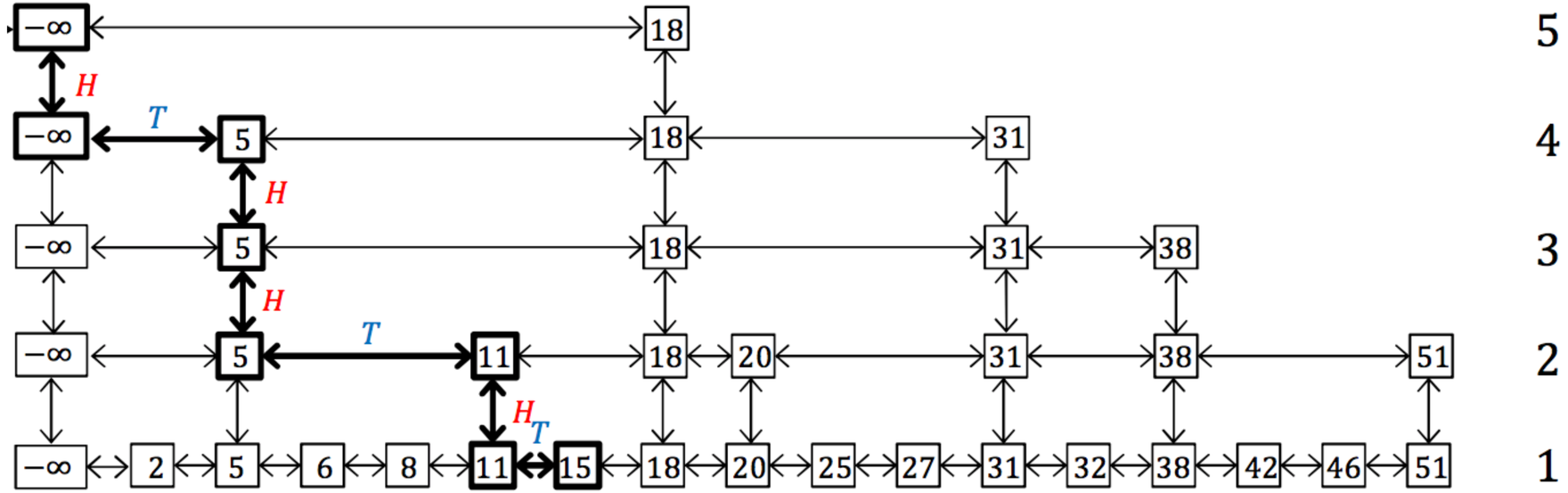
Skip List Search Cost

- Backwards search path, when do go up versus left?
- If node wasn't promoted (got tails here), then we go [came from] left
- If node was promoted (got heads here), then we go [came from] top
- How many consecutive tails in a row? (left moves on a level)
  - Same analysis as the height! $O(\log n)$
  - $O(\log^2 n)$ length overall—but I claimed $O(\log n)$ earlier



Skip List Search Cost

- We know height is $O(\log n)$ with high probability; say it is $c \log n$
- Thus, number of "up" moves is at most $c \log n$ with high probability
- Search path is a sequence of $HHHTTTHHTT\ldots$
- Search cost:
  - How many times do we need to flip a coin to get $c \log n$ heads with high probability?



Coin Flipping

- **Claim.** Number of flips until $c \log n$ heads is $\Theta(\log n)$ with high probability, that is, with probability $1 - 1/n^c$
- Note. Constant in $\Theta(\log n)$ will depend on c
- **Proof.**
  - Say we flip $10c \log n$ coins
  - When are there at least $c \log n$ heads?
  - Pr[exactly $c \log n$ heads]

$$= \binom{10c \log n}{c \log n} \cdot \left(\frac{1}{2}\right)^{c \log n} \cdot \left(\frac{1}{2}\right)^{9c \log n}$$

  - Pr[at most $c \log n$ heads] $\leq \binom{10c \log n}{c \log n} \cdot \left(\frac{1}{2}\right)^{9c \log n}$

Coin Flipping

- **Claim.** Number of flips until $c \log n$ heads is $\Theta(\log n)$ with high probability, that is, with probability $1 - 1/n^c$
- **Proof.**
  - Pr[at most $c \log n$ heads] $\leq \left(\frac{e \cdot 10c \log n}{c \log n}\right)^{c \log n} \cdot \left(\frac{1}{2}\right)^{9c \log n}$

$$= (10e)^{c \log n} \cdot \left(\frac{1}{2}\right)^{9c \log n}$$

$$= 2^{\log(10e) \cdot c \log n} \cdot \left(\frac{1}{2}\right)^{9c \log n}$$

$$= 2^{(\log(10e)-9) \cdot c \log n} = 2^{-d \log n}$$

$$= 1/n^d$$

- If we instead look at probability of at most $d'c \log n$ heads, as $d' \to \infty$, $d = 9 - \log(10e) \to \infty$, independent of c

Skip Lists

- Using $O(\log n)$ linked lists, achieve same performance as binary search tree
- No stored information about balance, no tricky balancing rules
- Just flip coins while inserting each new element to decide what lists it goes in

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
  - MIT course notes, 6.042/18.062J Mathematics for Computer Science April 26, 2005, Devadas and Lehman