Learning Goals

- Understand the design idea of skip lists
- Carry out more involved probabilistic runtime analysis using Chernoff bound and union bound
- Understand the idea of SkipNet in Peer-to-Peer systems

Skip List

- Problem with storing ordered data with linked list: FIND takes $O(n)$ time.

Skip List

- Problem with storing ordered data with linked list: FIND takes $O(n)$ time.
- Imagine building faster links among the nodes:
  - At the bottom level L_0, we have the original linked list, sorted;

Skip List

- Problem with storing ordered data with linked list: FIND takes $O(n)$ time.
- Imagine building faster links among the nodes:
  - At the bottom level L_0, we have the original linked list, sorted;
  - One level above, at L_1, we have a linked list storing every other node, also sorted, with $\lfloor n/2 \rfloor$ nodes;

Skip List

- Problem with storing ordered data with linked list: FIND takes $O(n)$ time.
- Imagine building faster links among the nodes:
    - At the bottom level L_0, we have the original linked list, sorted;
    - One level above, at L_1, we have a linked list storing every other node, also sorted, with $\lfloor n/2 \rfloor$ nodes;
    - One level above, at L_2, we have a linked list storing every four node from L_0, or every other node from L_1, also sorted, with $\lfloor n/4 \rfloor$ nodes, etc..
- Each copy of the node in L_i stores pointers to its copies in L_{i-1} and L_{i+1} (if they exist), and also the nodes the precede and follow it in L_i.

Skip List: Illustration

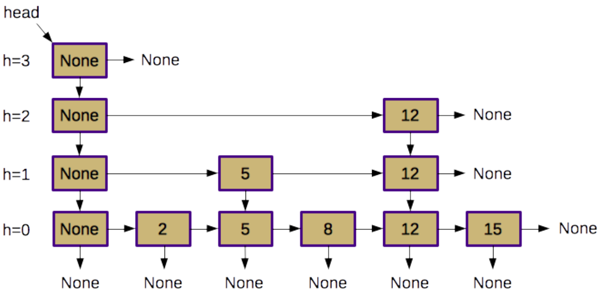


Image credit: Mike Lam at James Madison University

Find in Skip List

- Now Find takes time $O(\log n)$.
  - The highest level is L_H, where $H = \lceil \log n \rceil$.

FIND in Skip List

- Now FIND takes time $O(\log n)$.
    - The highest level is L_H, where $H = \lceil \log n \rceil$.
    - To find a key x, first walk in L_H as far as we can, finding the largest node whose key is still less than x;

FIND in Skip List

- Now FIND takes time $O(\log n)$.
  - The highest level is L_H, where $H = \lceil \log n \rceil$.
  - To find a key x, first walk in L_H as far as we can, finding the largest node whose key is still less than x;
  - Then walk down one level from that copy,and continue walking till we find again the node in level L_{H-1} with the largest key that is still smaller than x;

FIND in Skip List

- Now FIND takes time $O(\log n)$.
  - The highest level is L_H, where $H = \lceil \log n \rceil$.
  - To find a key x, first walk in L_H as far as we can, finding the largest node whose key is still less than x;
  - Then walk down one level from that copy,and continue walking till we find again the node in level L_{H-1} with the largest key that is still smaller than x;
  - Repeat, until we reach the node with x in level L_0.
- In actual implementation, we may store only the keys in levels other than L_0, and store the actual content only in nodes of L_0.

FIND in Skip List

- Now FIND takes time $O(\log n)$.
  - The highest level is L_H, where $H = \lceil \log n \rceil$.
  - To find a key x, first walk in L_H as far as we can, finding the largest node whose key is still less than x;
  - Then walk down one level from that copy,and continue walking till we find again the node in level L_{H-1} with the largest key that is still smaller than x;
  - Repeat, until we reach the node with x in level L_0.
- In actual implementation, we may store only the keys in levels other than L_0, and store the actual content only in nodes of L_0.
- Problem: INSERT and DELETE are combersome.

Skip List with Randomization

- Idea: Use randomization to construct the upper levels.

Skip List with Randomization

- Idea: Use randomization to construct the upper levels.
  - When we insert a new node, after we find its position in L_0 and inserting it there, we toss a coin, and with probability $\frac{1}{2}$ insert a copy in L_1, otherwise stop;

Skip List with Randomization

- Idea: Use randomization to construct the upper levels.
  - When we insert a new node, after we find its position in L_0 and inserting it there, we toss a coin, and with probability $\frac{1}{2}$ insert a copy in L_1, otherwise stop;
  - If we made a copy in L_1, then toss another coin, insert with probability $\frac{1}{2}$ a copy to level L_2, etc.

Skip List with Randomization

- Idea: Use randomization to construct the upper levels.
  - When we insert a new node, after we find its position in L_0 and inserting it there, we toss a coin, and with probability $\frac{1}{2}$ insert a copy in L_1, otherwise stop;
  - If we made a copy in L_1, then toss another coin, insert with probability $\frac{1}{2}$ a copy to level L_2, etc.
- The expected number of copies we insert for each node is 2.

Skip List with Randomization

- Idea: Use randomization to construct the upper levels.
  - When we insert a new node, after we find its position in L_0 and inserting it there, we toss a coin, and with probability $\frac{1}{2}$ insert a copy in L_1, otherwise stop;
  - If we made a copy in L_1, then toss another coin, insert with probability $\frac{1}{2}$ a copy to level L_2, etc.
- The expected number of copies we insert for each node is 2.
- We just need to show that this randomized construction yields similar performance for Find as the previous deterministic structure.

Randomized Skip List: ILlustration

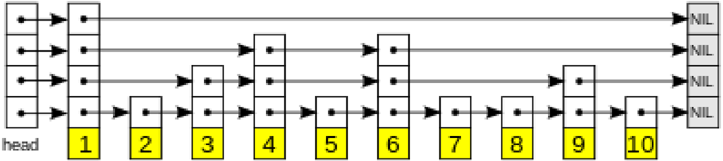


Image credit: Wikipedia

Analysis of FIND on Skip List

- There are two reasons that a FIND can take long: there can be too many layers, and the find takes too many horizontal steps.
- Let's first bound the number of levels H.

Analysis of Find on Skip List

- There are two reasons that a Find can take long: there can be too many layers, and the find takes too many horizontal steps.
- Let's first bound the number of levels H.
- The probability that a particular node has a copy at a level at least as high as H is 2^{-H}.

Analysis of FIND on Skip List

- There are two reasons that a FIND can take long: there can be too many layers, and the find takes too many horizontal steps.
- Let's first bound the number of levels H.
- The probability that a particular node has a copy at a level at least as high as H is 2^{-H}.
- By the union bound, when $n2^{-H} \leq \frac{1}{n^2}$, i.e., $H \geq 3 \log n$, with probability no more than $\frac{1}{n^2}$, there are no more than H levels.

Analysis of FIND on Skip List

- There are two reasons that a FIND can take long: there can be too many layers, and the find takes too many horizontal steps.
- Let's first bound the number of levels H.
- The probability that a particular node has a copy at a level at least as high as H is 2^{-H}.
- By the union bound, when $n2^{-H} \leq \frac{1}{n^2}$, i.e., $H \geq 3 \log n$, with probability no more than $\frac{1}{n^2}$, there are no more than H levels.

Bounding the number of horizontal steps

- For a fixed node x, we try to bound the number of steps it takes to reach x via a search path from the top level.

Bounding the number of horizontal steps

- For a fixed node x, we try to bound the number of steps it takes to reach x via a search path from the top level.
- It is easier to think of the path from node x up to the top level.

Bounding the number of horizontal steps

- For a fixed node x, we try to bound the number of steps it takes to reach x via a search path from the top level.
- It is easier to think of the path from node x up to the top level.
- At every step, we go either left or up

Bounding the number of horizontal steps

- For a fixed node x, we try to bound the number of steps it takes to reach x via a search path from the top level.
- It is easier to think of the path from node x up to the top level.
- At every step, we go either left or up
    - If the current node has a copy in the level above, we step up: this happens with probability $\frac{1}{2}$;

Bounding the number of horizontal steps

- For a fixed node x, we try to bound the number of steps it takes to reach x via a search path from the top level.
- It is easier to think of the path from node x up to the top level.
- At every step, we go either left or up
  - If the current node has a copy in the level above, we step up: this happens with probability $\frac{1}{2}$;
  - Otherwise, we step left.

Bounding the number of horizontal steps

- For a fixed node x, we try to bound the number of steps it takes to reach x via a search path from the top level.
- It is easier to think of the path from node x up to the top level.
- At every step, we go either left or up
  - If the current node has a copy in the level above, we step up: this happens with probability $\frac{1}{2}$;
  - Otherwise, we step left.
- Once we reach level H, we declare success.

Bounding the number of horizontal steps

- For a fixed node x, we try to bound the number of steps it takes to reach x via a search path from the top level.
- It is easier to think of the path from node x up to the top level.
- At every step, we go either left or up
    - If the current node has a copy in the level above, we step up: this happens with probability $\frac{1}{2}$;
    - Otherwise, we step left.
- Once we reach level H, we declare success.
- The problem becomes: what's the probability that, after taking at least X steps, we haven't made H upward steps?

Bounding the number of horizontal steps

- For a fixed node x, we try to bound the number of steps it takes to reach x via a search path from the top level.
- It is easier to think of the path from node x up to the top level.
- At every step, we go either left or up
  - If the current node has a copy in the level above, we step up: this happens with probability $\frac{1}{2}$;
  - Otherwise, we step left.
- Once we reach level H, we declare success.
- The problem becomes: what's the probability that, after taking at least X steps, we haven't made H upward steps?

Apply Chernoff Bound

Take X to be, say, $36 \log n$, and let Y_i, $i = 1, \cdots, X$, be the indicator variable that the i-th step is upward. Then $\mathbf{E}[Y_i] = \frac{1}{2}$. Let Y be $\sum_i Y_i$.

Apply Chernoff Bound

Take X to be, say, $36\log n$, and let Y_i, $i = 1, \cdots, X$, be the indicator variable that the i-th step is upward. Then $\mathbf{E}[Y_i] = \frac{1}{2}$. Let Y be $\sum_i Y_i$.
By Chernoff bound,

$$\begin{aligned}\mathbf{Pr}\left[Y \leq 3\log n\right] &= \mathbf{Pr}\left[Y \leq \mathbf{E}\left[Y\right] - 15\log n\right] \\ &\leq \exp\left(-\frac{2\cdot(15\log n)^2}{36\log n}\right) < \frac{1}{n^2}.\end{aligned}$$

Apply Chernoff Bound

Take X to be, say, $36 \log n$, and let Y_i, $i = 1, \cdots, X$, be the indicator variable that the i-th step is upward. Then $\mathbf{E}[Y_i] = \frac{1}{2}$. Let Y be $\sum_i Y_i$.
By Chernoff bound,

$$\begin{aligned} \mathbf{Pr}\left[Y \le 3 \log n\right] &= \mathbf{Pr}\left[Y \le \mathbf{E}\left[Y\right] - 15 \log n\right] \\ &\le \exp\left(-\frac{2 \cdot (15 \log n)^2}{36 \log n}\right) < \frac{1}{n^2}. \end{aligned}$$

This analysis was performed for a specific node x. By the union bound, with probability at least $1 - \frac{1}{n}$, no node takes more than $36 \log n$ steps to reach level H.

Putting Everything Together

Let A be the bad event that the highest level is more than $3 \log n$, and B be the bad event that, starting from some node, out of X steps there are fewer than $3 \log n$ steps.

Putting Everything Together

Let A be the bad event that the highest level is more than $3\log n$, and B be the bad event that, starting from some node, out of X steps there are fewer than $3\log n$ steps.
We have bounded $\mathbf{Pr}[A] \leq \frac{1}{n^2}$, and $\mathbf{Pr}[B] \leq \frac{1}{n}$.

Putting Everything Together

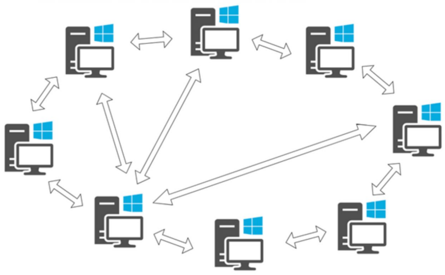
Let A be the bad event that the highest level is more than $3 \log n$, and B be the bad event that, starting from some node, out of X steps there are fewer than $3 \log n$ steps.
We have bounded $\mathbf{Pr}[A] \leq \frac{1}{n^2}$, and $\mathbf{Pr}[B] \leq \frac{1}{n}$.
Now by a final union bound, with probability at least $1 - \frac{2}{n}$, there are no nodes beyond level $L_{3 \log n}$ and every node reaches that level within $36 \log n$ steps. So Find takes time $O(\log n)$ for every node with high probability.

Application in Distributed Systems: Peer-to-Peer Systems

- A peer-to-peer (P2P) system has n nodes, each maintaining a host of connections to its neighbors, and none having global knowledge.

Application in Distributed Systems: Peer-to-Peer Systems

- A peer-to-peer (P2P) system has *n* nodes, each maintaining a host of connections to its neighbors, and none having global knowledge.
  - Keeping everything fully connected is way too expensive.

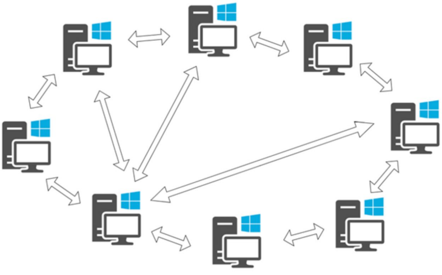


A simulation of a peer-to-peer network

Image credit: mysterium.network

Application in Distributed Systems: Peer-to-Peer Systems

- A peer-to-peer (P2P) system has n nodes, each maintaining a host of connections to its neighbors, and none having global knowledge.
  - Keeping everything fully connected is way too expensive.



A simulation of a peer-to-peer network

Image credit: mysterium.network

- A request of a node to communicate with another can take $O(n)$ time to traverse the network if we are not careful.

Idea of SkipNet

- We can use the idea of skip list to organize nodes in a P2P network.

Idea of SkipNet

- We can use the idea of skip list to organize nodes in a P2P network.
- Give each node an *identifier*, similar to the key value of a node in the database.

Idea of SkipNet

- We can use the idea of skip list to organize nodes in a P2P network.
- Give each node an *identifier*, similar to the key value of a node in the database.
- Given each node a bitstring of length $O(\log n)$.

Idea of SkipNet

- We can use the idea of skip list to organize nodes in a P2P network.
- Give each node an *identifier*, similar to the key value of a node in the database.
- Given each node a bitstring of length $O(\log n)$.
- There are multiple levels. Nodes sharing the same prefixes of length k are connected by an (ordered) linked list on level k.

Idea of SkipNet

- We can use the idea of skip list to organize nodes in a P2P network.
- Give each node an *identifier*, similar to the key value of a node in the database.
- Given each node a bitstring of length $O(\log n)$.
- There are multiple levels. Nodes sharing the same prefixes of length k are connected by an (ordered) linked list on level k.
- The resulting structure is similar to a skip list, except that on each level there are multiple lists.

Idea of SkipNet

- We can use the idea of skip list to organize nodes in a P2P network.
- Give each node an *identifier*, similar to the key value of a node in the database.
- Given each node a bitstring of length $O(\log n)$.
- There are multiple levels. Nodes sharing the same prefixes of length k are connected by an (ordered) linked list on level k.
- The resulting structure is similar to a skip list, except that on each level there are multiple lists.
- To access a node, we go as far as possible on a high level, then descend and continue.