Introduction to Algorithms

Meng-Tsung Tsai

12/24/2019

Reminder

23:59, Dec 27	13:30-17:30, Dec 28
programming assignment #3	programing quiz #2

Reference

Chapter 8 in "Randomized Algorithms" by Motwani and Raghavan.

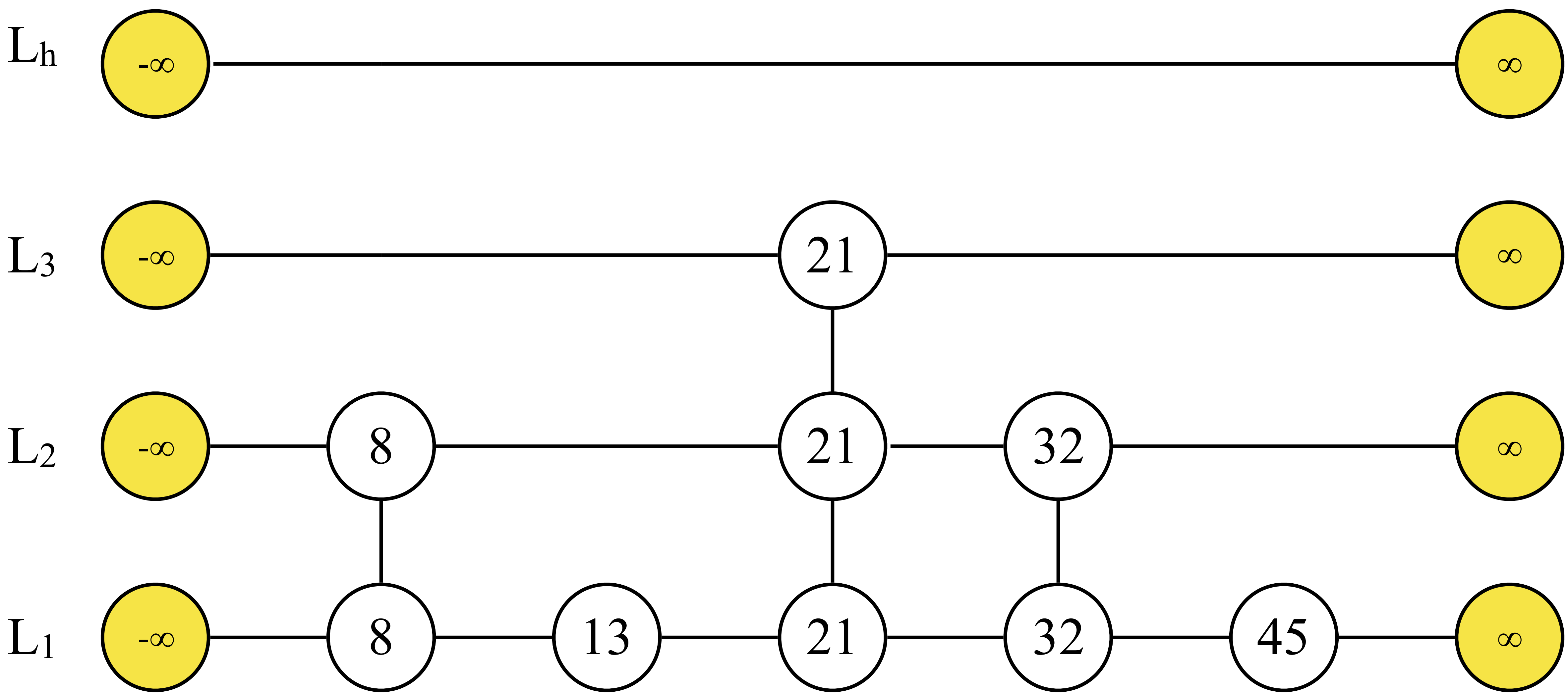
Skip Lists

What is a skip list?

Skip list is a membership container that can support the three operations, (1) insertion, (2) deletion, and (3) search. Each operation takes O(log n) time on average.

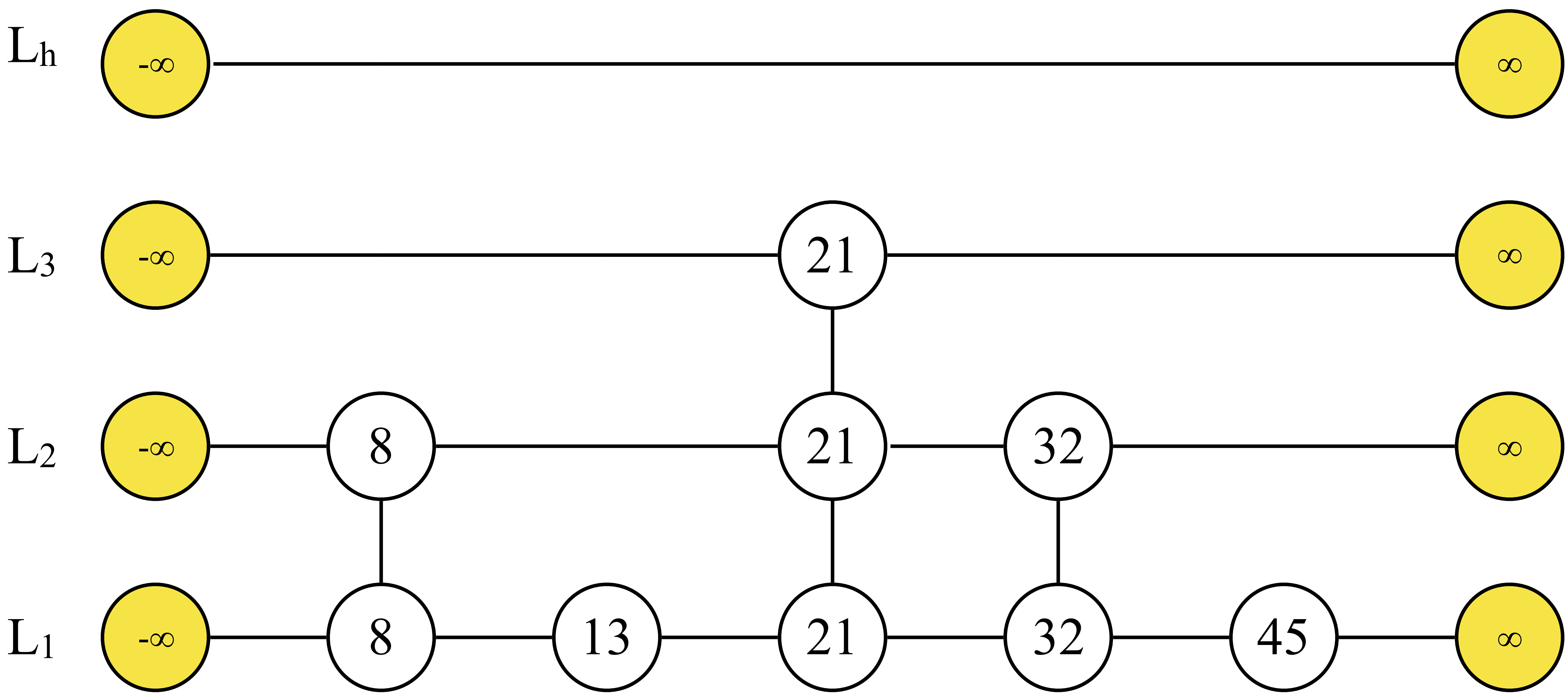
Basics

A skip list is a collection of sorted linked list L_1, L_2, ..., L_h, so that every element in L_i joins L_{i+1} with probability p for every i in [1, h-1] and L_h contains no element. You may assume p = 1/2.



Search Operation

Start from L_h and perform a linear scan to find the interval that contains the search key k. Once found, descend to L_{h-1} via vertical pointers and start another linear search in the subinterval. Repeat this while all lists are scanned.



Analysis

The height h is O(log n) with probability $1-1/n^{\Omega(1)}$.

Proof.

For each element, the probability that it joins at least k lists is p^k.

Hence, w.h.p. an element joins O(log n) lists.

By the Union bound, w.h.p. all elements joins O(log n) lists.

Analysis

The linear scan performed on a single list takes O(1) steps in expectation.

Proof.

For some list L_i, the probability that the algorithm incurs many probes is small. To see why, every element involves these probes cannot join L_{i+1}. This happens with a tiny probability. The expected length is O(1). Why?

The total search cost is thus O(log n) because w.h.p. there are O(log n) lists and each list has expected O(1) probes. It is possible to have more lists, but their total contribution to the expected number of probes is O(1). Why?

Conclusion. The search operation can be done in expected O(log n) time.

Insertion Operation

The insertion operation works as follows.

Use the search operation to locate the position where the current key shall be placed. Then, insert the key to L_1, L_2, ... until the random flipped coin tails up.

O(log n) time for search, and O(log n) time in expectation to added to the lists.

Conclusion. The insertion operation can be done in O(log n) time.

Deletion Operation

The deletion operation works as the insertion but in a reverse direction. The analysis works similarly.

Conclusion. The deletion operation can be done in O(log n) time.

FKS hashing

What is FKS hashing?

Given n static keys, FKS hashing can accomodate these n keys into O(n) space so that each search takes O(1) time in the worst case. The construction time is O(n) on average.

We assume the keys are from the universe $U = \{1, 2, ..., m\}$ and assume that m+1 is a prime p.

FKS hashing is a 2-level hashing. That is, for each search key, we use two hash functions to locate the potential address where the key is placed.

The first level

The first level has s slots, and the hash function at the first level is in the following form:

$f_k(x) = ((kx) \bmod p) \bmod s$ where k is sampled uniformly at random from U.

The first level hashing is to (roughly) evenly distribute the keys into s slots.

Analysis

Collision is defined to be the number of pairs of keys in the input that falls within the same slot.

<u>Example</u>. Suppose that $f_k(x)$ hash 5 keys into slot 1 and 1 key into slot 2. Then, the number of collisions is C(5, 2) + C(1, 2) = 10.

Let B(s, r, k, j) be the collisions incurred in slot j by using $f_k(x)$ to distribute r keys.

$$\sum_{k=1}^{p-1}\sum_{j=1}^{s}\binom{B(s,r,k,j)}{2} < (p-1)r^2/s$$

Fix an (x, y) pair, there are at most 2(p-1)/s k's that can hashed them into the same slot. Summing over C(r, 2) pairs, one obtains the above bound.

Analysis

$$\sum_{k=1}^{p-1}\sum_{j=1}^{s}\binom{B(s,r,k,j)}{2} < (p-1)r^2/s$$

Lemma. By setting s = r, there exists some k so that $f_k(x)$ incurs only O(r) collisions.

Corollary. By setting s = r, there are a constant fraction of k from U so that $f_k(x)$ incurs only O(r) collisions. --- Morkov Inequaltiy

Conclusion. After $f_k(x)$ is applied, there are at most $O(r^{1/2})$ keys in a slot.

The second level

The second level has s hash function, each associated to the keys that are hashed into a certain slot. The hash function at the second level is in the following form, again:

$f_k(x) = ((kx) \bmod p) \bmod s$ where k is sampled uniformly at random from U.

The second level hashing is to distribute the keys into s slots without any collision.

Analysis

$$\sum_{k=1}^{p-1}\sum_{j=1}^{s}\binom{B(s,r,k,j)}{2} < (p-1)r^2/s$$

Lemma. By setting s = $2r^2$, there exists some k so that $f_k(x)$ incurs 0 collision.

Corollary. By setting s = $2r^2$, there are a constant fraction of k from U so that $f_k(x)$ incurs 0 collision. --- Morkov Inequaltiy

Conclusion. After $f_k(x)$ is applied, there are at most 1 key in a slot.

Space Usage

To set s = n, one needs O(n) space for the first level hashing.

To set $s = 2r_j^2$, one needs $O\left(\sum_{j=1}^{s} r_j^2\right)$ space, which is bounded by the number of collision incurred by the first level hashing up to a constant factor. Hence, the space usage of the s hash tables in the second level is O(n) as well.

Probabilistic Verifiers

Matrix Multiplication

Given a pair of n by n matrices A and B, and another matrix C.

Verify whether AB = C using $O(n^2)$ operations with a good prob.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{bmatrix}$$

Matrix Multiplication

Given a pair of n by n matrices A and B, and another matrix C.

Verify whether AB = C using $O(n^2)$ operations with a good prob.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{bmatrix}$$

If you multiply A with B directly, then it needs $O(n^3)$ operations.

Key Observation

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

1. Ax only uses $O(n^2)$ operations.

2. (AB)x = A(Bx)

3. If AB = C, then for every x, (AB)x = Cx.

4. If AB $\neq$ C, then for some x, (AB)x $\neq$ Cx.

Random Vector x

$$\begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

If AB $\neq$ C, then P = AB-C $\neq$ 0, say some entry $p_{ij} \neq 0$.

Sample x uniformly at random from $\{0, 1\}^n$.

$$\sum_{k=1}^{n} p_{ik} x_k = p_{ij} x_j + \sum_{k=1, k \neq j} p_{ik} x_k$$

Random Vector x

$$\begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

If AB $\neq$ C, then P = AB-C $\neq$ 0, say some entry $p_{ij} \neq 0$.

Sample x uniformly at random from $\{0, 1\}^n$.

$$\sum_{k=1}^{n} p_{ik} x_k = p_{ij} x_j + \sum_{k=1, k \neq j} p_{ik} x_k$$

With probability $\leq 1/2$, RHS vanishes.

Repetition

Repeat the above procedure multiple (say 3) rounds. Then the verifier succeeds with probability $\geq 1 - (1/2)^3 = 7/8$.

Exercise

Can you use a verifier to implement matrix multiplications?