CS5112: Algorithms and Data Structures for Applications

Locality-sensitive Hashing

Ramin Zabih

Some content from: Wikipedia; Charles Kingsford

Cornell University

CORNELL TECH

Administrivia

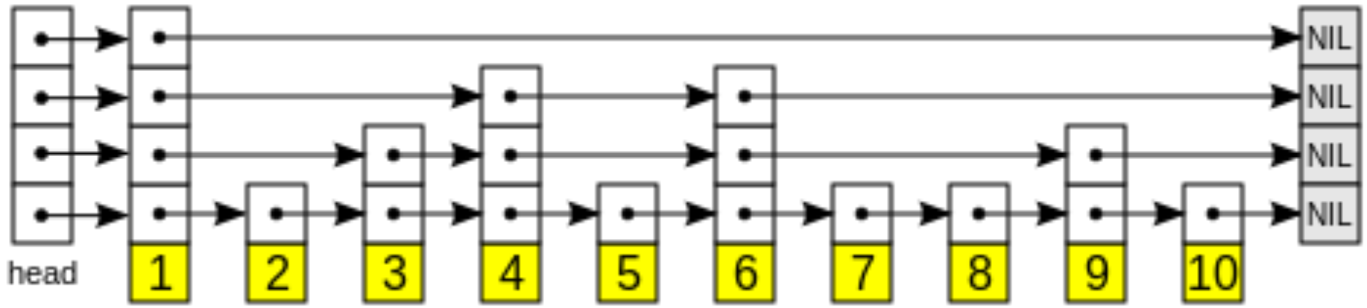
- A duck problem will be on the final exam
  - But lower bounds will not be, since it’s not core
- Extra office hours the week of December 1
- Final exam will be split among 2-3 nearby rooms
  - Students will be assigned to a room to avoid crowding

Lecture outline

- Chord and skiplists
- MinHash
- Locality sensitive hashing

Skip lists

- Can we find an element in a sorted list quickly?
  - Hierarchy of ‘express lanes’, randomly generated



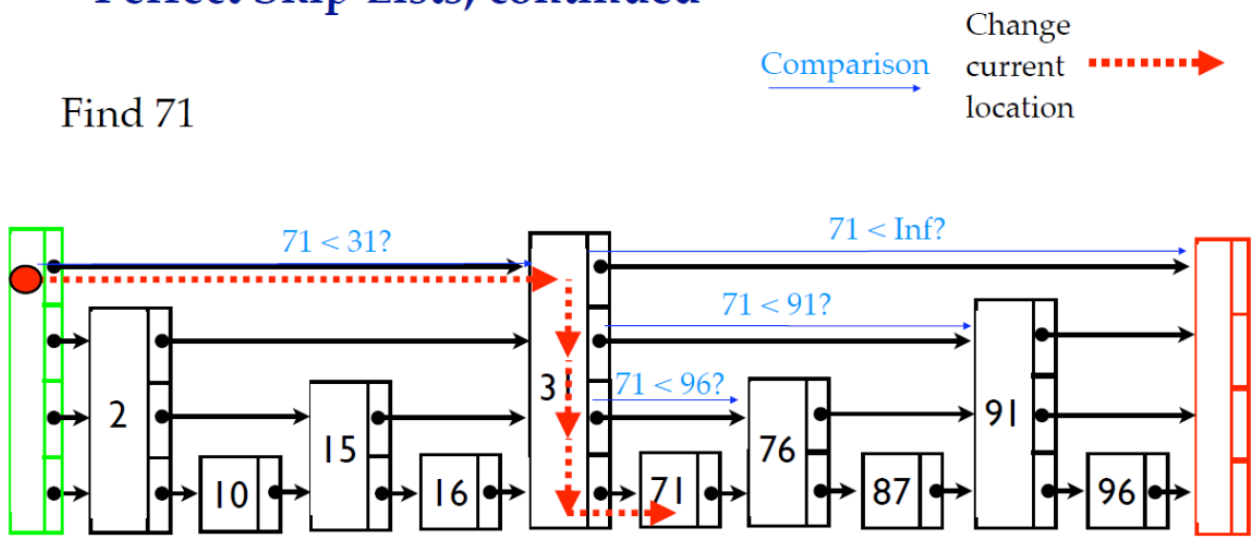
Source

  - Insertion at bottom level, promote a level ½ the time
  - No need for complex rebalancing schemes

Best skiplist slide (from Kingsford)

Perfect Skip Lists, continued

Find 71



When search for k:

If k = key, done!

If k < next key, go down a level

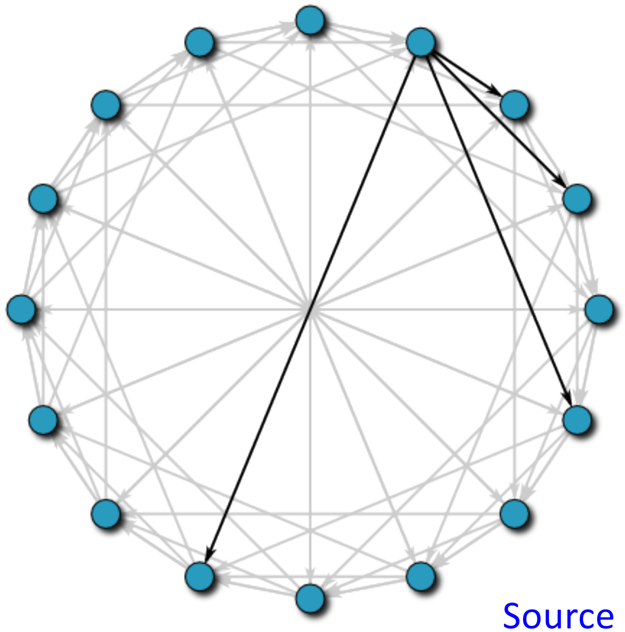
If k ≥ next key, go right

[Source](#)

CORNELL TECH

Chord: using local knowledge

- Can we do this without global information?
- Each node keeps a "finger table"
  - Like a skip list but circular
- Can efficiently find the right node
- Good ways of handling new nodes
  - Update finger tables in the background
- No great ideas on handling crashes



Source

CORNELL TECH

MinHash

- Suppose you want to figure out how similar two sets are
  - Jacard similarity measure is $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$
  - This is 0 when disjoint and 1 when identical
- Define $h_{min}(S)$ to be the element of S with the smallest value of the hash function h, i.e. $h_{min}(S) = \arg\min_{s \in S} h(s)$
  - This uses hashing to compute a set's "signature"
- Probability that $h_{min}(A) = h_{min}(B)$ is $J(A, B)$
- Do this with a bunch of different hash functions

MinHash applications

- Copying detection from articles
- Collaborative filtering!
  - Amazon, NetFlix, etc.

Curse of dimensionality

- High dimensions are not intuitive, and data becomes sparse
  – Volume goes up so fast that an exponential amount of data needed
- “Most of the volume is in the corners” theorem
  – Think of a 3D sphere inscribed in a box
- Particularly problematic for NN search
- k-d trees requires exponential time

Approximate NN via hashing

- Normally collisions make a hash function bad
  - In this application, certain collisions are good!
- Main idea: hash the data points so that nearby items end up in the same bucket
  - At query time, hash the query and check the bucket elements
- Most famous technique is Locality Sensitive Hashing (LSH)

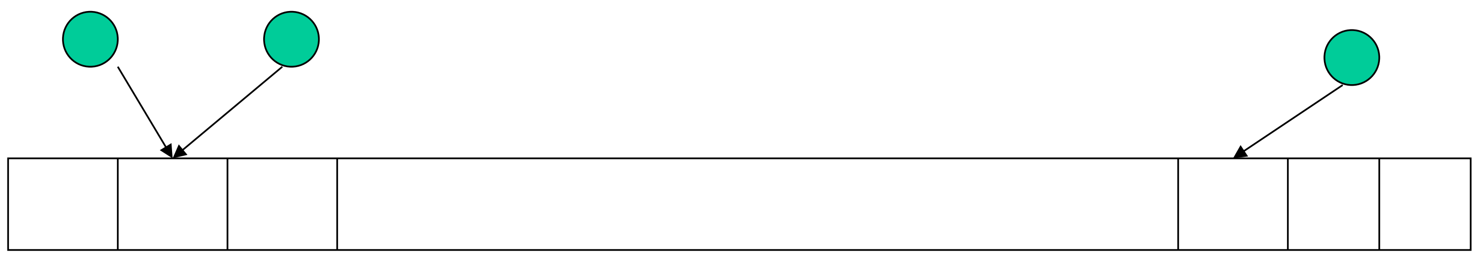
Locality sensitive hashing

- We can use hashing to solve ‘near neighbors’
  - And thus nearest neighbors, if we really need this
- Ensuring collisions is the key idea!
- Make nearby points hash to the same bucket
  - In a probabilistic manner
- By giving up on exactness we can overcome the curse of dimensionality
  - This is an extremely important technique in practice

CORNELL TECH

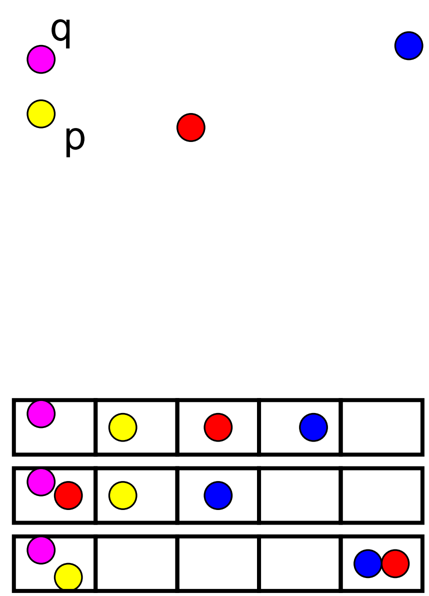
Basic idea of LSH

- A family of hash functions is locality sensitive if, when we pick a random element h of that family, for any 2 points p, q
  - $P[h(p) = h(q)]$ is 'high' if p is 'close' to q
  - $P[h(p) = h(q)]$ is 'low' if p is 'far' from q



- Cool idea, assuming such hash functions exist! (they do)

Visualizing LSH



How to use LSH for NN

- Pick l different hash functions at random, $h_1, \dots, h_l$
- Put each data point p into the l buckets $h_1(p), \dots, h_l(p)$
- Now for a query point q we look in the corresponding buckets $h_1(q), \dots, h_l(q)$
  - Return the closest point to q
- How does the running time scale up with the dimension d?
  - Nothing is exponential in d, which is awesome!
  - Can prove worst case is slightly better than $O(nd)$

Do such functions exist ?

- Consider the hypercube, i.e.,
  - Points from $\{0,1\}^d$
  - Hamming distance $D(p, q) =$ # positions on which p and q differ
- Define hash function h by choosing a set S of k random coordinates, and setting

$$h(p) = \text{projection of } p \text{ on } S$$

Example

- Take

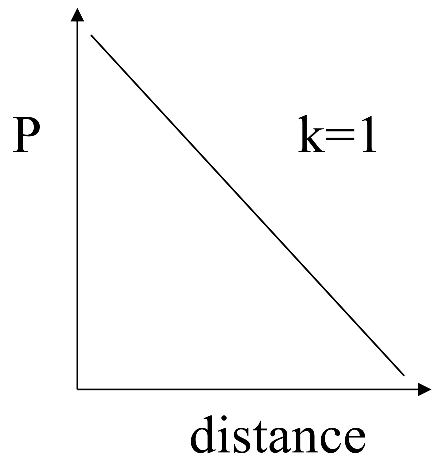
$$d = 10, p = 0101110010$$
$$k = 2, S = \{2,5\}$$

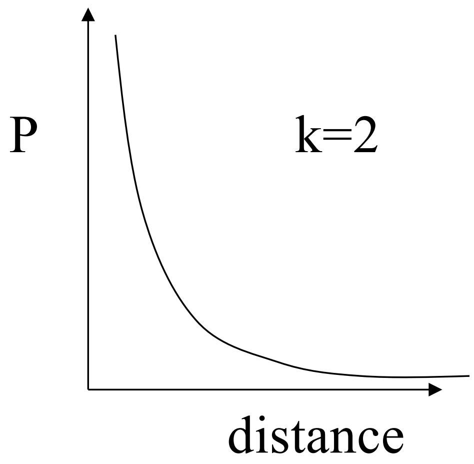
- Then $h_{2,5}(p) = 11$
- The hash function $h_{2,5}$ just looks at bit 2 and 5 of its input
- The family of hash functions include, e.g. $h_{1,7}$
  - Here we took $k = 2$

These hash functions are locality-sensitive

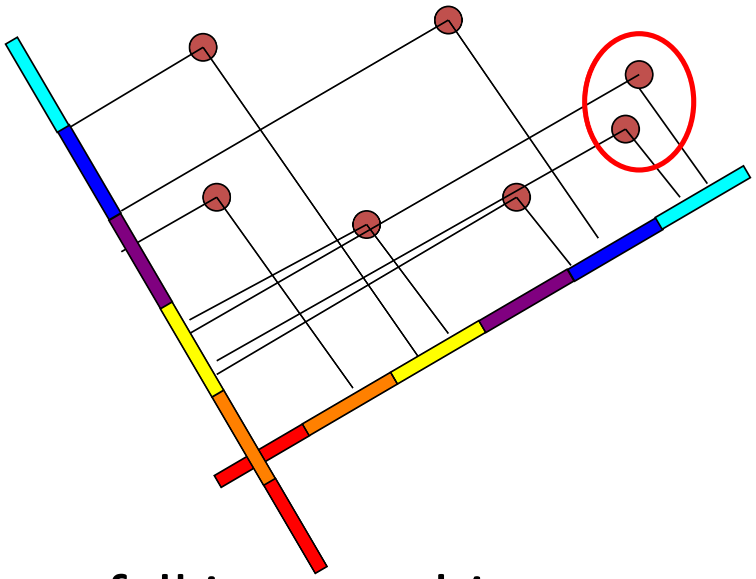
$$P[h(p) = h(q)] = (1 - \frac{D(p,q)}{d})^k$$

- We can vary the probability by changing *k*





Random Projections + Binning



- Nearby points will often fall in same bin
- Faraway points will rarely fall in same bin

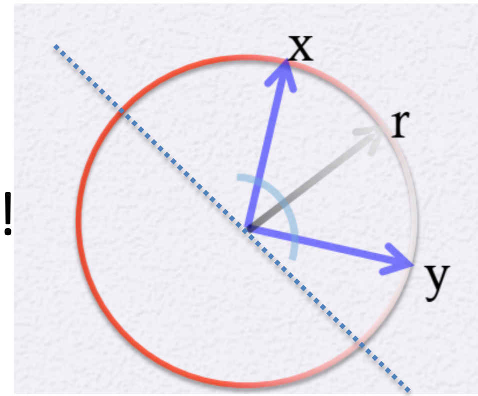
Other important LSH families

- We looked at Hamming distance via random subset projection
- Angle similarity via projection onto random vector

CORNELL TECH

Angle similarity via SimHash

- Angle similarity via projection onto random vector
  - VERY important for machine learning, etc.
- Pick a random unit vector r, and check if the two inputs x, y are on the same side of the half-space it defines
- Compute the dot products $\langle x, r \rangle, \langle y, r \rangle$
  - Do they have the same sign?
  - This gives us 1 bit
  - Nearby documents tend to be the same!



CORNELL TECH