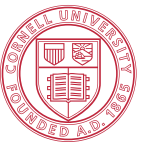
CS5112: Algorithms and Data Structures for Applications

Lecture 10: From hashing to machine learning

Ramin Zabih

Some figures from Wikipedia/Google image search



Administrivia

- HW comments and minor corrections on Slack
- HW3 coming soon
- Non-anonymous survey coming re: speed of course, etc.

Office hours

- Prof Zabih: after lecture or by appointment
- Tuesdays 11:30-12:30 in Bloomberg 277 with @Julia
- Wednesdays 2:30-3:30 in Bloomberg 277 with @irisz
- Wednesdays 3:30-4:30 in Bloomberg 277 with @Ishan
- Thursdays 10-12 in Bloomberg 267 with @Fei Li

Today

- Universal hashing
  - Recap: randomized quicksort
- From universal to perfect
- Learning to hash
- Nearest neighbor search

Quick impossibility proof

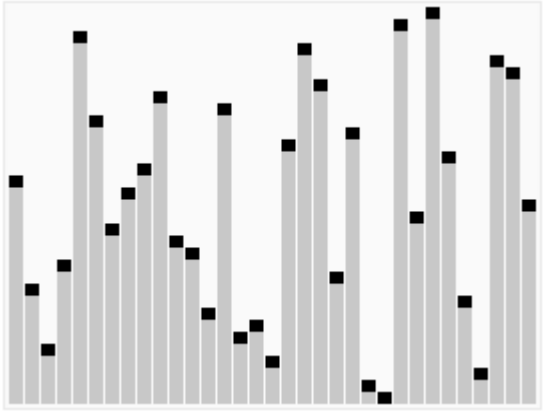
- Generally, proofs that there is no algorithm that can do X are quite complicated
- There is a simple one that follows naturally from Richard's lecture last Thursday on compression
- Do (lossless) compression algorithm always work?
  - I.e., reduce the size of a file?

Perfect & minimal hashing

- Choice of hash functions is data-dependent!
- Let's try to hash 4 English words into the buckets 0,1,2,3
  - E.g., to efficiently compress a sentence
- Words: {"banana", "glib", "epic", "food"}
  - Can efficiently say sentence like "epic glib banana food" = 3,2,1,0
- Can you construct a minimal perfect hash function that maps each of these to a different bucket?
  - Needs to be efficient, not (e.g.) a list of cases

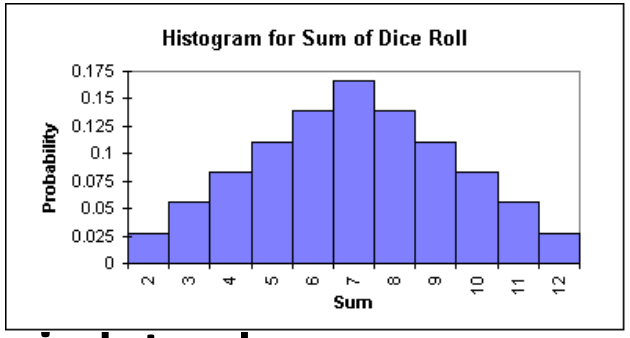
Recall: Quicksort

- Basic idea: pick a pivot element x of the array, use it to divide the array into the elements $\leq x$ or $> x$, then recurse
- Worst case: pivot is min or max element
  - Complexity is $O(n^2)$
  - This often happens in practice (why?)
- What is the average case?
  - In general, arguments about average case are much harder



Expectation of random variables

- Randomized quicksort: pick a random pivot
  - Equivalent to randomizing the input then running (usual) quicksort
- Recall: a (discrete) random variable has different possible values with different probabilities
  - Think of a coin, a die, or a roulette wheel
  - Visualize as a histogram



- The expectation of a random variable is the weighted sum, where each value is weighted by its probability
  - Example: expectation of a 6 sided die is 3.5

Average case quicksort

- The number of comparisons performed by randomized quicksort on an input of size n is a random variable
  - Small chance of $O(n)$ comparisons (great luck with pivot!) or $O(n^2)$ comparisons (terrible luck!)
  - With some effort you can show that the expectation of this random variable is $O(n \log n)$

Universal hashing

- We can randomly generate a hash function h
  - This is NOT the same as the hash function being random
  - Hash function is deterministic!
  - Can re-do this if it turns out to have lots of collisions
- Assume input keys of fixed size (e.g., 32 bit numbers)
- Ideally h will spread out the keys uniformly

$$P[h(x) = h(y) | \, x \neq y] \leq \frac{1}{2^{32}}$$

  - Think of this as fixing $x, y | x \neq y$ and then picking h randomly
- If we had such an h, the expected number of collisions when we hash N numbers is $\frac{N}{2^{32}}$

Universal hashing by matrix multiplication

- This would be of merely theoretical interest if we could not generate such an h
- There's a simple technique, not efficient enough to be practical
  - More practical versions follow the same idea
- Now assume the inputs/outputs are 4 bit numbers/3 bit numbers respectively, i.e. inputs: 0-15, outputs: 0-7
- We will randomly generate a 3x4 array of bits, and hash by 'multiplying' the input by this array

Universal hashing example

- We multiply using AND, and we add using parity
  - Technically this is mod 2 arithmetic

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Balls into bins

- There is an important underlying idea here
  - Shows up surprisingly often
- Suppose we throw m balls into n bins
  - Where for each ball we pick a bin at random
  - How big do we need to make n so that with probability $> \frac{1}{2}$ there are no collisions?
  - This is the opposite of the birthday paradox
- Answer: need $n \approx m^2$
- So to avoid collisions with probability ½ we need our hash table to be about the square of the number of elements

Perfect hashing from universal hashing

- We can use this to create a perfect hash function
- Generate a random hash function h
  - Technically, from a universal family (like binary matrices)
- Use a "big enough" hash table, from before
  - I.e., size is square of the number of elements
- Then the chance of a collision is < ½

Nearest neighbor search

- Fundamental problem in machine learning/AI: find something in the data similar to a query
  - Selected examples: predicting purchases (Amazon/Netflix), avoiding fraudulent credit card transactions, finding undervalued stocks, etc.
- Easiest way to do classification (map items to labels)
- Important application: density estimation
- Exact versus approximate algorithms
- Techniques are often classical CS, including hashing

Problem definitions

- Input: data items $X = \{x_1, \ldots, x_N\}$, query q, distance function $dist(q, x)$
  - Typically in a vector space under Euclidean (l_2) norm
- Nearest neighbor (NN) search: $\arg\min_{x \in X} dist(q, x)$
- K-nearest neighbor (KNN): find the closest K data items
- R-near neighbor search: $\{x \mid dist(q, x) \leq R\}$
- Approximate NN: if x^* is the NN, find

  $\{x | dist(q, x) \leq (1 + \epsilon) dist(q, x^*)\}$

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 rerank the bucket elements
- Most famous technique is Locality Sensitive Hashing (LSH)

NN Density estimation

- Lots of practical questions boil down to density estimation
  - Even if you don't explicitly say you're doing it!
    - "What do typical currency fluctuations look like?"
- Your classes generally have some density in feature space
  - Hopefully they are compact and well-separated
- Given a new data point, which class does it belong to?
  - One hypothesis per class, maximize $P(data|class)$,
    - This requires the density