Applied Algorithms Lec 7: Mini-Midterm and Optimization/Code Review

Sam McCauley

October 21, 2021

Williams College

Admin

- Mini-midterm 1 due Wednesday at 10pm
- No TA office hours this week. (I'll hold my office hours as normal.)
- Assignment 2 back soon
- No class next Monday. Schedule slightly tight because of that.

Mini-Midterm Questions?

Quick Survey

- Who's seen bloom filters before?
- Who's seen cuckoo hashing before?
- Who's done any probabilistic algorithms analysis before?
- Who's heard of streaming algorithms before?
- Plan for this section of the course: I'm going to go through all of this in detail—but quickly. May be a tad slow/fast for some of you. I will try to post writeups, and I'm always available in office hours

Finishing up Assignment 1 Code Review

Calculating the height in constant time

- What's faster than calculating the height from scratch each time?
- Only adding on "new" items
- That's $O(1)$ on average, but it's a pain to implement
- Can we change our ordering to get improved performance?
- Next idea: fill in set *one item at a time*

Calculating the height in constant time

```
for(int64_t i = 0; i < f_half_length; i ++) {
        int64_t two_p_i = 1LL << i;
        for(int64_t j = 0; j < two_p_i; j ++){
                s2_heights[j + two_p_i].height = s2_heights[j].height + heights[i];
                s2_heights[j + two_p_i].mask = j + two_p_i;
        }
}
```

Sorting in linear time??

- Not possible in general, but our data has special structure

- Remember how we could more efficiently build up the heights. What would happen if we sorted the array at the same time?

Sorting in linear time (from best-performing solution)

```
for(int i=0; i<(inputSize-inputSize/2); i++){


        int64_t currID = s2t[i].id;
        double currH = sqrt(s2t[i].value);

        int maxq= pow(2,i);
        struct SubsetItem* newArr = (struct SubsetItem*)calloc(maxq,sizeof(struct SubsetItem));

        for(int q=0; q<maxq; q++){
                newArr[q].height=pArr[q].height+currH;
                newArr[q].key=pArr[q].key+currID;
        }

        sortedMerge(pArr,newArr,currPsize,maxq);
        currPsize+=maxq;
        free(newArr);
}
```

Binary search

- Really really costly

- Let's look at two attempts to engineer a more efficient binary search

Inlined, Unrolled binary search

```
 * Binary search for the predecessor in a sorted table
 * The loop is unrolled to enable slightly better performance
 */
__inline__ int unrolled_bin_search(entry* table, uint32_t high, double target) {
  uint32_t low = 0, mid;
  double mid_val;

  while (low < high) {
    mid = (low + high + 1) / 2;
    mid_val = table[mid].val;

    if (target > mid_val) {
      if (mid >= high) return mid;
      mid = (mid + high + 1) / 2;
      mid_val = table[mid].val;
      if (target > mid_val) {
        low = mid;
      } else if (target < mid_val) {
        high = mid - 1;
      } else {
        return mid;
      }

    } else if (target < mid_val) {
      if (low >= mid - 1) return low;
      mid = (low + mid) / 2;
      mid_val = table[mid].val;
      if (target > mid_val) {
        low = mid;
      } else if (target < mid_val) {
        high = mid - 1;
      } else {
        return mid;
      }
    } else {
      return mid;
    }
  }

  return low;
}
```

- From “best last year”
- Did much better than one would think—we’ll talk about why

Why is binary search really slow?

- Cache efficiency! Lots of cache misses
- Also branch mispredictions
- Can we avoid these?

Code that avoids these

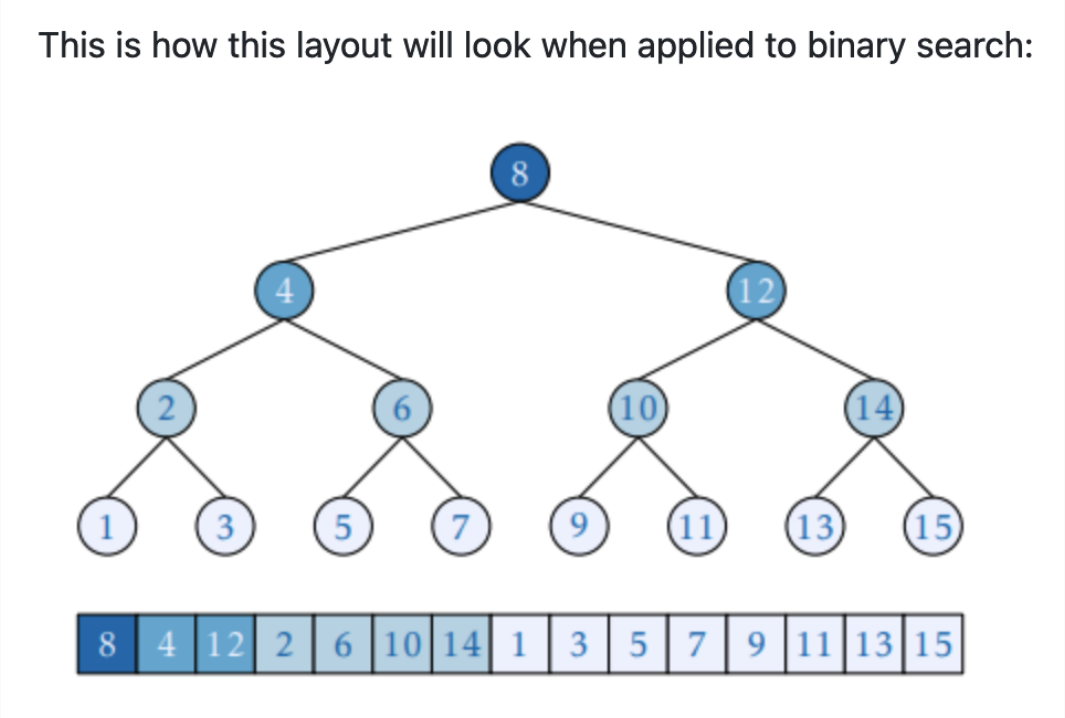
```
int64_t eytzinger(int64_t i, int64_t k, double* a, double* b) {
  if (k <= n) {
      i = eytzinger(i, 2 * k + 1,a,b);
      b[k] = a[i++];
      i = eytzinger(i, 2 * k,a,b);
  }
  return i;
}

__inline__ __attribute__((always_inline)) int64_t search(double x, double* b) {
    int64_t k = 1;
    while (k <= n) {
      __builtin_prefetch(b + k * block_size);
      k = 2 * k + (b[k] > x);
    }
    k >>= __builtin_ffs(~k);
    return k;
}
```

- What the heck?

Eytzinger layout

- Comment in code led to github that referenced a paper about a way to rearrange arrays to lead to better binary search performance

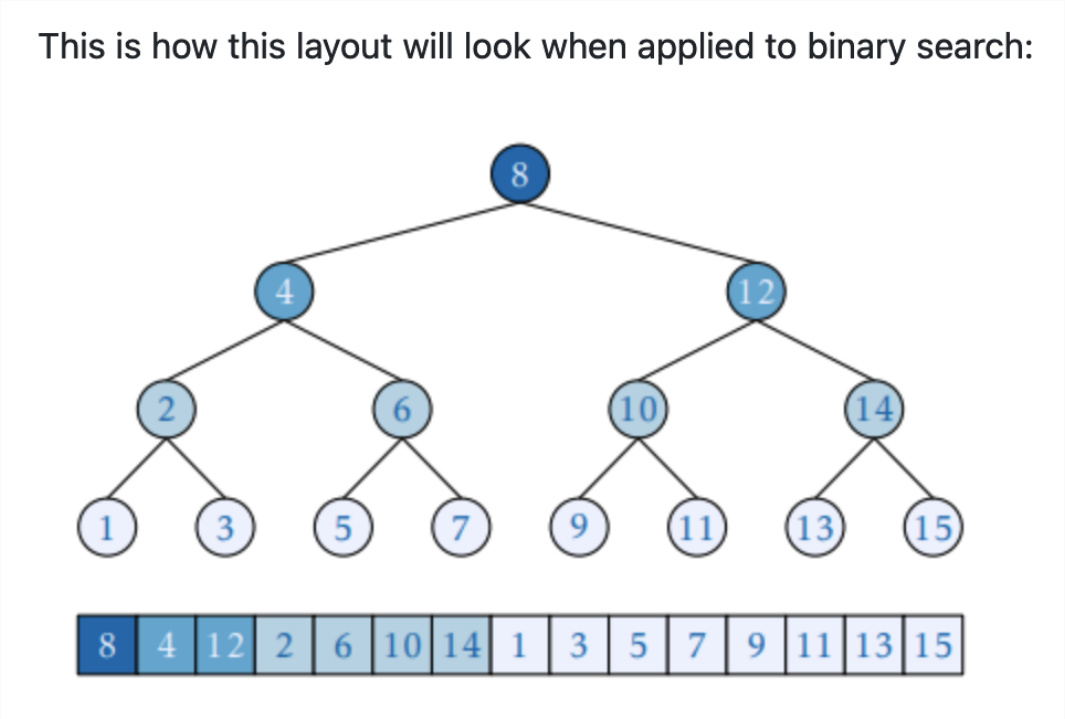


Analysis

- Recall: normal binary search is $O(\log_2 N/B)$ cache misses
- What about with this layout?
- Answer: $O(\log_2 N/B)$ cache misses!
- What gives?
- By the way: there IS a layout that gives $O(\log_B N/B)$ cache misses. But it has issues with constants.

Analysis

- Let's say I'm at a given branch in my binary search. What can I say about where I'll be 2 branches from now?



Analysis

- Let's say I'm at a given branch in my binary search. What can I say about where I'll be 2 branches from now?
- Idea: with this layout, we know where we're going ahead of time. While doing previous operations, can *prefetch* future cache accesses so that they're already available by the time we get there.
- Prefetching is very rare as an optimization technique. But this is a cool example of it

Getting rid of the binary search

- That said, even these highly optimized binary searches are costly. Can we avoid them entirely?

- Hint: the high cost of binary search is that we're jumping all over the table. Can we group entries so that we don't need to jump all over the table?

- Stronger hint: if I have two similar heights for the first half of my blocks, I'm going to be doing very similar searches in the second half...

Use two tables

- Idea: make a table for *both* halves of the input; sort each. $O(2^{n/2})$ time with the optimizations from before.
  - This does double our space usage!
- Now, can do a merge-like operation to determine, for each set in the first half, find the optimal set in the second half
- Same idea as 3SUM
- $O(2^{n/2})$ total time.
- Cache efficiency? $O(2^{n/2}/B)$.
- Best last year did this reordering, but did a full binary search instead of a merge

(Pretty much) rest of best solution this year

```
int currPindex = pSize-1;
for(int i=0; i<qSize; i++){

        while(qArr[i].height+pArr[currPindex].height>totalHeight/2){
                currPindex--;
        }
        if(maxSubset.height<qArr[i].height+pArr[currPindex].height){
                maxSubset.height=qArr[i].height+pArr[currPindex].height;
                maxSubset.key=qArr[i].key+pArr[currPindex].key;
        }
}
```

Lessons

- Cache efficiency is king
- In this case, most optimizations depended on the problem itself. gcc can't help with that
    - I think Assignment 2 is more optimization-heavy once it fits in cache.
- Looks like I need to increase the input size a bit next time to make sure the final times are macroscopic

Problem Set Questions

- Out of time to do them in class (sorry!)

- I'm happy to go over them in office hours

Probability

Probability Takeaways

What I want you to know:

1. Definition of probability/basic calculations
2. Determine if two events are independent
3. Calculate expectation
4. Linearity of expectation
5. Difference between "concentration bounds" vs expected performance

Useful Formulas (use $e = 2.71\ldots$)

Two useful approximations for simplifying exponents (presented as inequalities, but really quite tight even for moderate n):

$$(1 + 1/n)^n \leq e \qquad\qquad (1 - 1/n)^n \leq 1/e$$

Example: $(1.1)^{10} = 2.593\ldots$

With probability we often use choose (a.k.a. binomial) notation, but it's unweidly. Here's a good way to approximate it:

$$\left(\frac{x}{y}\right)^y \leq \binom{x}{y} \leq \left(\frac{ex}{y}\right)^y$$

Example: $\binom{n}{10} = \Theta(n^{10})$

Definition of Probability

- Defined over a set of possible *outcomes* (often called the *sample space*)
- An *event* is a subset of the outcomes
- $$\Pr[\text{Event } E] = \frac{\#\text{ outcomes in the event}}{\text{Total }\#\text{ of outcomes}}$$
- Formal definition probability generally applies weights to the events (in which case the definition of probability is the weight of outcomes in the event, divided by total weight of all events). We will usually have equal-weight events.

Probability Calculation Examples

- Let's say I roll a 20-sided die. What is the probability that an even number comes up?
- Answer: $10/20 = 1/2$.
- Let's say I flip a coin 10 times. What is the probability of getting exactly 5 heads?
- $\binom{10}{5}/2^{10}$

Conditional Probability

- Sometimes we want to calculate the probability of an event, when we already have some partial information about the outcome
- Specifically: want to calculate the probability of event E_1, already knowing that the outcome is in E_2. Denoted $\Pr[E_1|E_2]$.
- Example: let's say I'm playing cards with a 52-card deck. I have already drawn three cards; all three were clubs. What is the probability that the fourth card is a club?
- Pr[draw 4 clubs | first three cards were a club]
- How many outcomes are there for the fourth card? How many of them are a club?
- 49 outcomes. 10 of them are clubs. Probability: 10/49.

Conditional Probability Second Example

Conditional probability can be a bit unintuitive at times! Break it down to be sure you're getting the right answer.

- Let's say that I have two children. One of them is a boy. What is the probability that both of them have boys?[1]
- Outcomes for my children: BB BG GB GG
- Outcomes consistent with "one of them is a boy": BB BG GB
- Probability that both of them are boys: 1/3
- Rephrasing the question: Let's say I have two children. They are not both girls. What is the probability that they are both boys?

[1] Assume an over-simplified world where a given child is a "girl" or "boy" with probability 1/2

Independence

- Idea: two events are independent if one does not have any impact on the other
- Example: let's say I flip a (fair) coin twice. Let E_1 be the event that the first flip is heads, and E_2 be the event that the second flip is heads. E_1 and E_2 are independent.
- Formal definition: E_1 and E_2 are independent if $\Pr[E_1 \mid E_2] = \Pr[E_1]$ and $\Pr[E_2 \mid E_1] = \Pr[E_2]$.
- I am not going to ask you to prove that two events are independent formally. But, let's look at one example.

Proving Independence

Example: let's say I flip a (fair) coin twice. Let E_1 be the event that the first flip is heads, and E_2 be the event that the second flip is heads. E_1 and E_2 are independent.

Definition 1

E_1 and E_2 are independent if $\Pr[E_1 \mid E_2] = \Pr[E_1]$ and $\Pr[E_2 \mid E_1] = \Pr[E_2]$.

All possible outcomes: HH HT TH TT

Proving Independence

Example: let's say I flip a (fair) coin twice. Let E_1 be the event that the first flip is heads, and E_2 be the event that the second flip is heads. E_1 and E_2 are independent.

Definition 2

E_1 and E_2 are independent if $\Pr[E_1 \mid E_2] = \Pr[E_1]$ and $\Pr[E_2 \mid E_1] = \Pr[E_2]$.

All possible outcomes: HH HT TH TT

$$\Pr[E_2] = 1/2$$

Proving Independence

Example: let's say I flip a (fair) coin twice. Let E_1 be the event that the first flip is heads, and E_2 be the event that the second flip is heads. E_1 and E_2 are independent.

Definition 3

E_1 and E_2 are independent if $\Pr[E_1 \mid E_2] = \Pr[E_1]$ and $\Pr[E_2 \mid E_1] = \Pr[E_2]$.

All possible outcomes: $\boxed{HH}$ $\boxed{HT}$ TH TT

$$\Pr[E_2 \mid E_1]$$

Proving Independence

Example: let's say I flip a (fair) coin twice. Let E_1 be the event that the first flip is heads, and E_2 be the event that the second flip is heads. E_1 and E_2 are independent.

Definition 4

E_1 and E_2 are independent if $\Pr[E_1 \mid E_2] = \Pr[E_1]$ and $\Pr[E_2 \mid E_1] = \Pr[E_2]$.

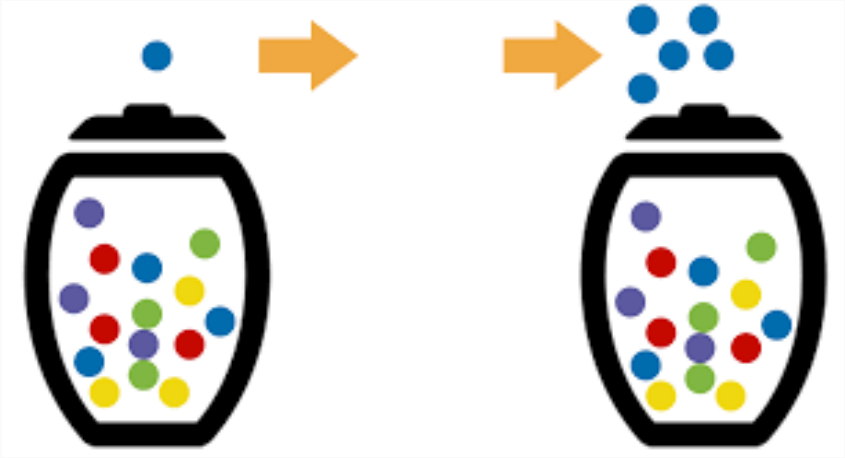
All possible outcomes: $\boxed{HH}$ $\boxed{HT}$ TH TT

$$\Pr[E_2 \mid E_1] = 1/2$$

Again: we'll normally be looking at this intuitively.

Independence: Examples

- Let's say I have a bag of balls, half of which are black, and half of which are white. I take a ball out of the bag and look at what color it is. Then I take another ball out of the bag and look at what color it is.
  - Event 1: The first ball is white.
  - Event 2: The second ball is black
  - Are these events independent?



Independence: Examples

- Let's say I have a bag of balls, half of which are black, and half of which are white. I take a ball out of the bag and look at what color it is. Then I take another ball out of the bag and look at what color it is.
  - Event 1: The first ball is white.
  - Event 2: The second ball is black
  - Are these events independent?

- No! If the first ball is white, there will be more black balls then white balls remaining in the bag for the next draw.

Independence: Examples

- Let's say I shuffle a deck of cards and look at the top card. I replace the card and shuffle the deck again and look at the top card. Is the event that the first card is red, and the event that the second card is red, independent?
- Yes
- Let's say I roll a 20-sided die. Is the event that the resulting number is a multiple of 3 independent of the event that the result is even?
    - Yes. (Multiples of 3 are: 3 6 9 12 15 18; half of these are even.)

Why independence is useful

- If A and B are independent, then $\Pr[A \text{ and } B] = \Pr(A) \cdot \Pr(B)$.

- What is the probability of flipping 10 heads in a row? All 10 are independent, so $1/2^{10}$.

Why independence is useful

- Let's say you're in a class of n students. Every day the professor asks a student to explain the previous night's reading (the student is chosen by rolling an n-sided die). What is the probability that you won't be chosen after all k lectures in the course?
- Probability (not being chosen on one day) is $(1 - 1/n)$
- Probability (not being chosen after k days) is $(1 - 1/n)^k$
- Side note: we can put this in a more readable form. $(1 - 1/n)^k = ((1 - 1/n)^n)^{k/n} \approx 1/e^{k/n}$

Cuckoo Hashing

A randomized algorithm

- Before we finish talking about probability, let's look at an example of a randomized algorithm
- Hashing: way to implement a dictionary with constant-time insert, delete, lookup
- Hashing is randomized, so performance can be bad sometimes
- Cuckoo hashing: $O(1)$ *worst-case* lookup. (Inserts are usually constant-time, but can be expensive sometimes.)

Reminder: Dictionary

- You've seen in 136 and/or 256
- Idea: want to store n key/value pairs
- Can insert new key/value pair
- Query: given a key, get the associated value stored in the dictionary
- How fast can we do inserts and queries? How much space do we need?
    - Can get $O(1)$ expected time for both operations using $O(n)$ space.

Assumptions on hash

- Let's assume we have access to a *uniform random* hash h that hashes any item to a value in $\{0, \ldots, M\}$.
- For any x and any $i \in \{0, \ldots, M\}$, $\Pr(h(x) = i) = 1/M$
- We assume h is independent: so even if we know that $h(y) = a$ and $h(z) = b$ (and so on), then we still have "For any x and any $i \in \{0, \ldots, M\}$, $\Pr(h(x) = i) = 1/M$"
- Assume that M is much bigger than the number of items in our dataset. (Like $M = 2^{64}$)

Building a Dictionary (doesn't quite work yet)

To store n items:

- Allocate an array A with cn slots for some c. Each slot must be large enough to store a key/value pair
- Insert x: store item x at position $h(x)\%cn$
- Query q: look at position $h(q)\%cn$ and see if the key is stored there
- If we can do this: $O(1)$ worst-case query, insert; always correct.
- What's the problem with this approach?

Collisions

- Several items might hash to the same location. How can we resolve this?
  - Chaining
  - Linear Probing

Chaining

- Each entry in our array A is the head of a singly-linked list
- Insert: add item to linked list
- Query: find item in linked list
- Advantages?
  - Space-efficient (just need pointers for linked list)
  - Simple
  - Good worst-case insert time
- Disadvantages?
  - Cache inefficient

Linear Probing

- Set $c > 1$ (often have $1.5n$ or $2n$ slots in practice)
- Insert: attempt to insert x into $h(x)\%cn$. If slot is full, keep moving down the table to find the next empty slot.
- Query: start at $h(x)\%cn$. Need to keep checking until find the item, or find an empty slot
- Advantages?
  - Somewhat space-efficient, good average insert and query time
  - Cache-efficient!
- Disadvantages?
  - Not that efficient; performance is terrible if table fills up

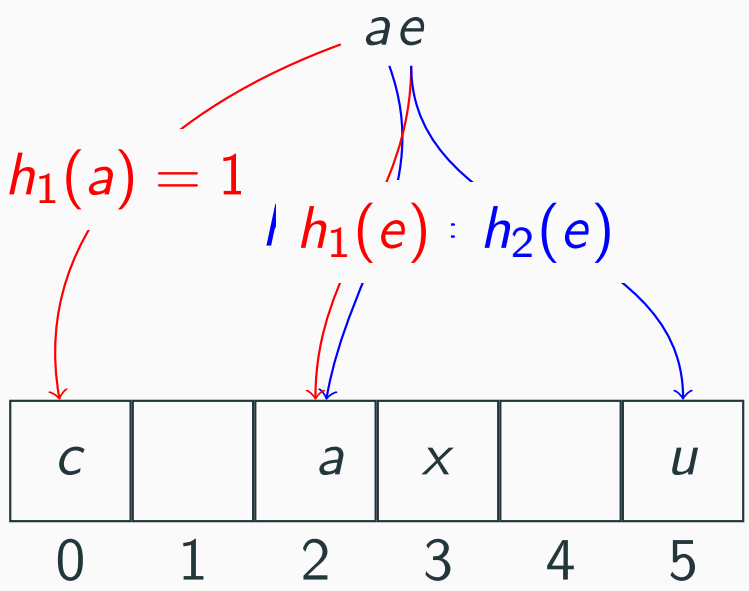
Cuckoo Hashing [Pagh, Rodler 2005]

- A third method of resolving collisions
- Queries are $O(1)$ *worst case*
- Insert is still $O(1)$ on average

Cuckoo Hashing Invariant

- Have two hash functions h_1, h_2
- Table of size cn with $c = 2$ (for now)
- Invariant: item x is either stored at $h_1(x)\%cn$, or at slot $h_2(x)\%cn$.
- We'll come back to inserts. But how can we query? How much time does a query take?

Cuckoo Hashing Inserts

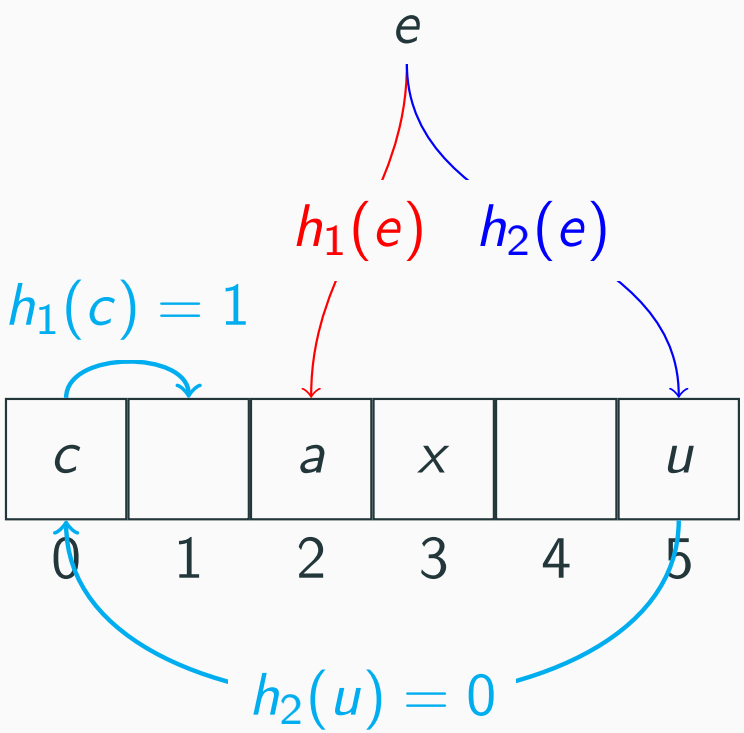


- Let's say we want to insert a new item a. How can we do that?
- Easy case: if $h_1(a)\%cn$ or $h_2(a)\%cn$ is free, can just store a immediately.
- What do we do if both are full?

Cuckooing!

Cuckoos kick other bird's eggs out of the nest, replacing them with their own.

Cuckoo Hashing Inserts



- Let's say we want to insert a new item a. How can we do that?
- Easy case: if $h_1(a)\%cn$ or $h_2(a)\%cn$ is free, can just store a immediately.
- What do we do if both are full?
- Move one of the items in the way to its other slot!
- If there's an item THERE, recurse

Cuckoo Hashing Example on Board

Does this always work?

- Recall our invariant: every item x is stored at $h_1(x)\%cn$ or $h_2(x)\%cn$
- Is there a simple example where this is impossible?
- One option: three items x, y, and z all have the same two hashes
  - What is the probability that this happens?
  - $1 - O(1/n)$ (outside the scope of the course; may give a simplified explanation Thursday)

Cuckoo Hashing Performance

- Queries: $O(1)$ worst case
- Cache performance?
  - Two cache misses per query. Is that good?
  - Kind of! Probably better than chaining. But linear probing has only $\approx$one cache miss on any query, so long as $\log n$ items fit in a cache line
- Insert: Still $O(1)$ on average (we'll come back to this)
- Cache performance?
  - One cache miss per "cuckoo"–OK but not great
  - In practice, inserts are really pretty bad for cuckoo hashing due to poor constants
- Idea: cuckoo hashing does great on queries (though with potentially worse cache efficiency than linear probing), but pays for it with expensive inserts

Expectation

Random Variable

- A variable whose values depend on the outcome of a random process
- Let's say I draw four cards from a deck of cards. Let S be a random variable indicating the number of clubs I draw.
- What can we say about S?
  - S is at least 0 and at most 4
  - What is the probability that S is 0?
  - $13/52 \cdot 13/51 \cdot 13/50 \cdot 13/49 \approx .0043$
  - What is the probability that S is 4?
  - $13/52 \cdot 12/51 \cdot 11/50 \cdot 10/49 \approx .00264$
- Since each card is a club with probability (about) $1/4$, and we draw 4 cards, it seems like S should generally be around 1. Can we formalize this intuition?

Expectation

- When we make random decisions, we often care about the *average* outcome
- Example: let's say I flip a fair coin until I get a heads. How long will it take me on average?
- 2 flips
- Another example: Consider a quicksort implementation that chooses each pivot at random. This algorithm takes $O(n \log n)$ time on average.

Expectation

- Let's say a random variable X takes values $\{1, \ldots k\}$

- Then $\mathsf{E}[X] = \sum_{i=1}^{k} i \cdot \Pr[X = i]$

Expectation example

Let's say I roll a 20-sided die, and I give you money equal to the number that shows up on top. I charge \$10 to play this game. Should you play it?

Let's look at what you win on average

- Random variable X to represent how much you win
- $\mathrm{E}[X] = \sum_{i=1}^{20} i/20$
- $\mathrm{E}[X] = \frac{20 \cdot 21}{2 \cdot 20} = 10.5$
- So yes, you'll win \$.50 on average

Useful expectation fact

- If X is a $0/1$ random variable, then $\mathrm{E}[X] = \Pr[X = 1]$

Independence of Random Variables

- Two random variables are independent if the value of 1 does not depend on the other.
- Example: let X_1 denote the number of heads on my first coin flip, and X_2 denote the number of heads on my second coin flip. These are independent.
- But: let X_H denote the number of heads I flip over k coin flips, and X_T denote the number of tails. These are not independent.

Linearity of Expectation

- Let's say a random variable can be represented as the sum of other random variables
- $X = X_1 + X_2 + \ldots + X_n$
- Then $\mathrm{E}[X] = \mathrm{E}[X_1] + \mathrm{E}[X_2] + \ldots + \mathrm{E}[X_n]$
- True even if the X_i are not independent!!!

Using Linearity of Expectation

- Let's say I flip a coin 100 times. How many heads will I see on average?
- $X =$ number of heads I see in 100 flips.
- $$X_i = \begin{cases} 1 & \text{if the } i\text{th flip is heads} \\ 0 & \text{otherwise} \end{cases}$$
- We can see that $\mathrm{E}[X_i] = 1/2$.
- $X = X_1 + X_2 + \ldots X_n$
- $\mathrm{E}[X] = 50$ by linearity of expectation

Linearity of Expectation Example 2

- Hashing with chaining. Let's say we hash n items to to n slots. What's the expected length of each chain?
- $X^j =$ length of a chain j
- $X_i^j = 1$ if the ith item hashes to slot j
- $\mathrm{E}[X_i^j] = 1/n$
- $\mathrm{E}[X^j] = \sum_{i=1}^{n} \mathrm{E}[X_i^j] = 1$

Linearity of Expectation Example 3

- Let's say we want to Bubble Sort an array A, where A is randomly permuted
- How many swaps does Bubble Sort perform?
- Fact: number of swaps performed by Bubble Sort is exactly the number of *inversions* in A (pairs i,j such that $i < j$ but $A[i] > A[j]$)
- Rephrasing: how many inversions are there in a randomly-permuted array?

Linearity of Expectation Example 3

- $X =$ number of inversions in A. What is E$[X]$?
- Let $X_{ij} = 1$ if i, j represent an inversion; 0 otherwise
- Are the X_{ij} independent?
  - No! If a, b is an inversion, and b, c is an inversion, then a, c is an inversion.
  - Reason: know $a < b < c$, but $A[a] > A[b]$ and $A[b] > A[c]$, so $A[a] > A[c]$.

Linearity of Expectation Example 3

- $X =$ number of inversions in A. What is $\mathrm{E}[X]$?
- Let $X_{ij} = 1$ if i, j represent an inversion; 0 otherwise
- $X = \sum_{i,j} X_{ij}$
- $\mathrm{E}[X] = \sum_{i,j} \mathrm{E}[X_{ij}]$
- $\mathrm{E}[X_{ij}] = 1/2$
- $\mathrm{E}[X] = n(n-1)/4$

Limits of Expectation

- Let's say I charge you \$1000 to play a game. With probability 1 in 1 million, I give you \$10 billion. Otherwise, I give you \$0.
- Would you play this game?
- Answer: probably not. You're just going to lose \$1000.
- But expectation is good! You expect to win \$9000.

Concentration bounds

- Rather than giving the *average* performance, bound the probability of bad performance.
- Let's say I flip a coin k times. On average, I see $k/2$ heads. But what is the probability I never see a heads?
- Answer: $1/2^k$
- Quicksort has expected runtime $O(n \log n)$. What is the probability that the running time is more than $O(n \log n)$?
- Answer: $O(1/n)$

With High Probability

- An event happens *with high probability* (with respect to n) if it happens with probability $1 - O(1/n)$
- So: quicksort is $O(n \log n)$ with high probability
- Cuckoo hashing can maintain its invariant with high probability
- Cuckoo hashing inserts require $O(\log n)$ swaps with high probability
- Linear probing queries require $O(\log n)$ time with high probability. (Contrast to $O(1)$ in expectation!)
- With high probability is always with respect to a variable. Assume that it's with respect to n unless stated otherwise.

WHP example

- How many coins do I need to flip before I see a heads with high probability? (With respect to some variable n)
- If I flip k times, I see a heads with probability $1 - 1/2^k$.
- So I need $1/2^k = O(1/n)$. Solving, $k = \Theta(\log n)$.

Expectation vs Concentration (WHP)

- We'll usually use "with high probability" for concentration bounds
- Expectation states how well the algorithm does on average. Could be much better or worse sometimes!
- With high probability gives a guarantee that will almost always be met: if n is large it becomes vanishingly unlikely that the bound will be violated.

Hashing Analysis

If we have time

- What is the probability that the *first* slot is empty when inserting using linear probing?
- What is the expected length of a chain when using chaining?
- Quicksort analysis