Stony Brook University
Department of Computer Science

CSE 385 Section 01

Lecture 13-15

Probability and Combinatorics

Author: Yan Kun (Alex) Meng

Professor: Michael Bender

Date: 2/21/2023 - 2/28/2023

Probability Definitions:

- *Probability Sample Space (S, P)*
  Let S be a set of outcomes that is finite / countably infinite

  $$S = s_1, s_2, ..., s_n$$

  Let P be a probability function:

  $$P : S \to [0, 1] \quad s.t. \sum_i P(s_i) = 1$$

- *Event*
  Events A, B are a subsets of outcomes from Sample Space (S, P)

  $$Pr(A \cup B) \quad Pr(A \cap B) \quad \text{(set notation)}$$

- *Random Variable (r.v.)*
  A function:

  $$f : S \to \mathbb{R}^+$$

- *Expected Value (E[f]) of a Random Variable*
  The "average value" of the random variable

  $$E[f] = \sum Pr(s_i) f(s_i)$$

- *Linearity of Expectation*

  $$E[X + Y] = E[X] + E[Y]$$

Examples of Expected Value Problems:

1. I flip a coin until I get a head. You get a dollar for every tail that you see.
(a). What's the expected amount of money that you would get?
Sample Space: All possible sequences resulting from coin flips
Event of interest: All sequences that end in a head and 1 head only
Outcome Probability: Probability of each sequence occurring

Let r.v. X = amount of money you would get

Outcomes in Event	Prob	X
H	1/2	0
TH	1/4	1
TTH	1/8	2
TTTH	1/16	3
TTTTH	1/32	4

$$E[X] = \sum Pr(s_i)f(s_i) = 0(1/2) + 1(1/4) + 2(1/8) + 3(1/16) + ...$$

$$E[X] = \sum_{i=0}^{\infty} \frac{i}{2^{i+1}}$$

$$E[X] = 1$$

(b). Suppose that you get 2^t dollars if you lose on the t^{th} flip (instead of 1 dollar all the time). What's the expected value now?

Outcomes in Event	Prob	X
H	1/2	1
TH	1/4	2
TTH	1/8	4
TTTH	1/16	8
TTTTH	1/32	16

$$E[X] = \sum Pr(s_i)f(s_i) = 1(1/2) + 2(1/4) + 4(1/8) + 8(1/16) + ...$$

$$E[X] = 1/2 + 1/2 + 1/2 + 1/2 + ...$$

$$E[X] = \infty$$

2. Suppose that I have n envelopes and n letters. Each letter has its own envelope, but the letters are actually permuted randomly.
What is the expected number of envelopes that contain their correct letter?

Let (indicator r.v.) $x_i = \begin{cases} 1 & \text{if envelope contain correct letter} \\ 0 & \text{if envelope contain wrong letter} \end{cases}$

Let $X = \sum x_i$ = Total number envelopes that contain their correct letter.

$$E[X] = E\left[\sum_{i=1}^{n} x_i\right] = \sum_{i=1}^{n} E[x_i]$$

$$E[x_i] = \sum Pr(s_i)f(s_i) = Pr(x_i = 1) \cdot 1 + Pr(x_i = 0) \cdot 0$$
$$E[x_i] = Pr(x_i = 1)$$

What is the probability that after assigning letters to n envelopes randomly, an envelope contains the correct letter?

Probability correctly picking: $\frac{1}{n}$

$$E[X] = \sum_{i=1}^{n} E[x_i] = \sum_{i=1}^{n} \frac{1}{n} = \boxed{1}$$

After randomly assigning n envelopes to n letters, we expect 1 envelope to contain the right letter.

3. Suppose that I have n processors and n tasks. Each processor randomly chooses a task to execute.
(a) What's the expected number of tasks that don't get executed in first round?

Let (indicator r.v.) $x_i = \begin{cases} 1 & \text{if task didn't get executed} \\ 0 & \text{if task does get executed} \end{cases}$

Let $X = \sum x_i$ = Total number of tasks that didn't get executed.

$$E[X] = E\left[\sum_{i=1}^{n} x_i\right] = \sum_{i=1}^{n} E[x_i]$$

$$E[x_i] = \sum Pr(s_i)f(s_i) = Pr(x_i = 1) \cdot 1 + Pr(x_i = 0) \cdot 0$$

$$E[x_i] = Pr(x_i = 1)$$

What is the probability that after randomly choosing n times, a task doesn't get picked?

Probability getting picked: $\frac{1}{n}$
Probability doesn't get picked: $1 - \frac{1}{n}$

Probability doesn't get picked n times: $(1 - \frac{1}{n})^n$

$$E[x_i] = Pr(x_i = 1) = \left(1 - \frac{1}{n}\right)^n$$

Deathbed Formulas:

$$\left(1 + \frac{1}{n}\right)^n \approx e$$
$$\left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e}$$

Therefore:

$$E[x_i] = \left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e}$$
$$E[X] = \sum_{i=1}^{n} E[x_i] = \sum_{i=1}^{n} \frac{1}{e} = \boxed{\frac{n}{e}}$$

After randomly executing n tasks with n processors, we expect $\frac{n}{e}$ tasks does not get executed.

(b) What the expected number of tasks that survive round 2?

From part (a), we know that we expect $\frac{n}{e}$ tasks to remain unprocessed (survived).

Let (indicator r.v.) $x_i = \begin{cases} 1 & \text{if task still didn't get executed} \\ 0 & \text{if task does get executed} \end{cases}$

Let $X = \sum x_i$ = Total number of tasks that still didn't get executed.

What is the probability that the task still doesn't get executed?

Probability getting picked: $\frac{1}{\frac{n}{e}} = \frac{e}{n}$
Probability not getting picked: $1 - \frac{e}{n}$
Probability not getting picked n times: $\left(1 - \frac{e}{n}\right)^n$

$$Pr(x_i = 1) = \left(1 - \frac{e}{n}\right)^n = \left(1 - \frac{e}{n}\right)^{\frac{n}{e} \cdot e}$$

Deathbed Formulas:

$$\lim_{n \to \infty} (1 + \frac{x}{n})^n \approx e^x$$

Therefore, as n grows large (number of tasks to process is a lot):

$$Pr(x_i = 1) = \left(1 - \frac{e}{n}\right)^{\frac{n}{e} \cdot e} \approx \frac{1}{e^e}$$

$$E[X] = \sum_{i=1}^{n} E[x_i] = \sum_{i=1}^{n} \frac{1}{e^e} = \boxed{\frac{n}{e^e}}$$

After randomly executing n tasks with n processors again on $\frac{n}{e}$ tasks, we expect $\frac{n}{e^e}$ does not get executed.

(c) What the expected number of tasks that survive round 3?

$$E[X] = \sum_{i=1}^{n} E[x_i] = \sum_{i=1}^{n} \frac{1}{e^{e^e}} = \boxed{\frac{n}{e^{e^e}}}$$

Since denominator is getting exponentially bigger, we expect all tasks to be executed in a couple of rounds even if we had a very large number of tasks to execute.

4. Suppose I have an unfair coin.

Tails with a Probability of p
Heads with a Probability of 1-p

(a) What's the expected number of flips until our first head?

Let r.v. X = number of flips until first head

$$E[X] = \sum_{i=1}^{n} Pr(s_i) f(s_i)$$

$$Pr(X = 1) = 1 - p$$

$$Pr(X = 2) = p(1 - p)$$

$$Pr(X = 3) = p^2(1 - p)$$

$$\vdots$$

$$Pr(X = n) = p^{n-1}(1 - p)$$

$$E[X] = (1 - p) + 2p(1 - p) + 3p^2(1 - p) + ... + np^{n-1}(1 - p)$$

$$E[X] = (1 - p)\left(1 + 2p + 3p^2 + ...np^{n-1}\right)$$

$$E[X] = (1 - p)\frac{d}{dp}\left(1 + p + p^2 + p^3 + ...p^n\right)$$

Sum of a geometric series:

$$\text{If } |r| < 1, \text{ then S} = \tfrac{1}{1-r}$$

$$E[X] = (1 - p)\frac{d}{dp}\left(\frac{1}{1 - p}\right)$$

$$E[X] = (1 - p)\left(\frac{1}{(1 - p)^2}\right)$$

$$E[X] = \boxed{\frac{1}{1 - p}}$$

If the coin is unfair and has a probability 1-p to getting a head, we expect to flip $\frac{1}{1-p}$ times until we see our first head.

(b) Suppose I have a coin with $1 - p = Pr(heads) = \frac{1}{100}$. If I flip the coin 100 times, what's the probability that I get heads at least once?

Negation: What's the probability that I don't get any heads in 100 times? (Meaning I get all tails in 100 flips)

$$Pr(\text{ all tails }) = \left(1 - \frac{1}{100}\right)^{100} \approx \frac{1}{e}$$

$$Pr(\text{ at least one head }) = 1 - Pr(\text{ all tails }) = \boxed{1 - \frac{1}{e}}$$

If the probability of heads is very little ($\frac{1}{100}$) and I flip it 100 times, then there is a $1 - \frac{1}{e}$ change, which is approximately 63%, I will see at least one head.

5. I roll a die until I see a 6.
What is the expected number of rolls until I see a 6?

Expected value is the reciprocal of the probability for a geometric distribution. The expected number of throws to getting a six is 6.

5. (Coupon Collector Problem)
Suppose that I have n bins.
I throw balls randomly at them until every bin has at least 1 ball.
What is the expected number of ball that I need to throw?

Recall: (Qs 3) When we had n processors and n tasks, in order to know the expected number of tasks that don't get executed, we needed "rounds" or "phases" of the experiment.

Divide this problem into phases: In phase i, there are i empty bins.

In the beginning, we have **n empty bins** , so we are in **phase n** .
At the end, we have **no more empty bins** , so we are in **phase 0** .

In phase i, we have n bins, and i of them are empty. So the probability that a ball lands in one of those empty bins is:

$$Pr(\text{ Ball lands in empty bin in phase i }) = \frac{i}{n}$$

Let r.v. x_i = number of balls until a ball lands in empty bin in phase i

$$E[x_i] = \frac{1}{p} = \frac{n}{i}$$

Let r.v. X = total number of balls needed to fill all bins

$$X = x_n + x_{n-1} + x_{n-1} + ... + x_2 + x_1$$

$$E[X] = E\left[\sum_{i=1}^{n} x_i\right] = \sum_{i=1}^{n} E[x_i] = \sum_{i=1}^{n} \frac{n}{i} = n\sum_{i=1}^{n} \frac{1}{i} = n\left(\frac{1}{n} + \frac{1}{n-1} + ... + 1\right)$$

Harmonic Series:

$$1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n} \approx \ln(n)$$

$$E[X] \approx n\ln(n)$$

In order for each of the n bins to fill with at least 1 ball, we expect to throw around $n \ln n$ balls.

Fun fact: If you are buying mcdonalds and want to collect all toys, if there are n toys, you can expect to buy around $n \ln n$ meals to collect all toys.

EXTRA PRACTICE PROBLEMS:

1. (**Pairwise and mutual independence**) A lot contains 50 defective and 50 not defective pens. Two pens are drawn at random, with replacement. The events A, B, and C are defined as:

A = (the first pen is defective)
B = (the second pen is not defective)
C = (the two pens are both defective or both non-defective)

Determine whether:

(i) A, B, C are pairwise independent
(ii) A, B, C are mutually independent

Answer:

Let D = defective, N = not defective.

$$Pr(A) = Pr(\text{ the first is defective }) = Pr(DN, DD) = \frac{1}{2}\cdot\frac{1}{2} + \frac{1}{2}\cdot\frac{1}{2} = \frac{1}{2}$$

$$Pr(B) = Pr(\text{ the second one is not defective }) = \frac{1}{2}$$

$$Pr(C) = Pr(\text{ the two pens are both defective or non-defective }) = \frac{1}{2}$$

$$Pr(A \cap B) = Pr(DN) = \frac{1}{2}$$
$$Pr(B \cap C) = Pr(NN) = \frac{1}{2}$$
$$Pr(A \cap B \cap C) = Pr(\{\}) = 0$$

Since $Pr(A \cap B) \neq Pr(A \cap B \cap C)$, A, B, C are pairwise independet and NOT mutually independent.

Stony Brook University
Department of Computer Science

CSE 385 Spring 2023

Lecture 15-17

WHP, Quicksort, Quickselect

Author: Yan Kun (Alex) Meng

Professor: Michael Bender

Date: March 2023

Definitions

- *Conditional Probability*
  When an event B has happened then A happens:

$$Pr(A|B) = \frac{Pr(A \cap B)}{Pr(B)}, P(B) \neq 0$$

- *Independence*
  Two events A and B are independent iff:

$$Pr(A \cap B) = Pr(A) \cdot Pr(B)$$

$$Pr(A|B) = Pr(A)$$

- *Inclusion/Exclusion*
  When we want to find the probability of both events (with overlap):

$$Pr(A \cup B) = Pr(A) + Pr(B) - Pr(A \cap B)$$

  When we want to find the probability of three events (with overlap):

$$Pr(A \cup B \cup C) = Pr(A) + Pr(B) + Pr(C) - Pr(A \cap B) - Pr(B \cap C) - Pr(A \cap C) + Pr(A \cap B \cap C)$$

- *Union Bound*
  The probability of unions of events must be less than the sum of their probabilities (at most the sum if events are disjoint):

$$Pr(A \cup B \cup C) \leq Pr(A) + Pr(B) + Pr(C)$$

- *With High Probability*
  An event A occurs w.h.p. with respect to parameter n if:

$$Pr(A) \geq 1 - \frac{1}{n^c}, \text{ for some constant c} \qquad Pr(\bar{A}) < \frac{1}{poly(c)}$$

- *Rank in an Array*
  The rank of an element in an array is the position that element would be in if we sorted the elements. In other words, if your rank is k, you are the **k-th smallest element**

With-High-Probability

1. How many balls do we need to throw into n bins until all the bins have at least 1 ball with high probability?

Start by guessing a number of balls (expected), then calculate **Pr(error)**.

$$Pr(\text{ error }) = Pr(\text{ there exist bin } i \text{ that does not satisfy requirement})$$

Bin i does not satisfy requirement means it has no balls.
Bin i could be bin #1 or bin #2 or bin #n.

$$Pr(\text{ error }) < Pr(\text{ bin 1 empty }) + Pr(\text{ bin 2 empty }) + ... + Pr(\text{ bin n empty })$$

Previously, we calculated (Coupon Collectors Problem):

$$E[\text{ number of balls need to throw until all n bins have } \geq 1 \text{ ball}] \approx n \ln n$$

Using that as our "guess", $Pr(\text{bin 1 empty})$ after $c \cdot n \ln n$ (because $\theta(n \log n)$) throws is:

$$Pr(\text{bin 1 empty}) = (1 - \frac{1}{n})^{c \cdot n \ln n} \approx \frac{1}{e^{c \ln n}} = \frac{1}{n^c}$$

$$Pr(\text{ error }) \leq Pr(\text{ bin 1 empty }) + Pr(\text{ bin 2 empty }) + ... + Pr(\text{ bin n empty })$$

$$Pr(\text{ error }) \leq \frac{1}{n^c} + \frac{1}{n^c} + ... + \frac{1}{n^c}$$

$$Pr(\text{ error }) \leq \frac{n}{n^c}$$

$$Pr(\text{ error }) \leq \frac{1}{n^{c-1}} = \frac{1}{poly(c)}$$

Therefore, throwing $\theta(n \log n)$ balls with large enough constant will fill all bins WHP.

2. **WHP is a bound. It assures that error won't happen that often.**
Is WHP always O(expected number)? How would we know to get a tighter bound?

It depends on how tight we want the bound to be, but usually the answer to WHP is based on the expectation.

3. **(a) How many times do I need to flip a coin to get a head with high probability in n?**

Calculate the Pr(error) given L flips.

$$Pr(error) = Pr(\text{ no heads in L flips }) = \frac{1}{2^L}$$

What is L: the number of flips to get at least 1 head with high probability.

$$\frac{1}{2^L} = \frac{1}{n^c}$$
$$2^L = n^c$$
$$L = c \lg n$$

Therefore, $c \lg n$ flips results in at least 1 head with high probability.

(b) How many flips do I need to get $c \log n$ *heads* **with high probability in n?**

Let's always develop some intuition first.
We just said lg(n) flips results in 1 head with high probability. Now we are interested in the number of flips to result not in 1 head, but $c \log n$ heads, so it should be $\log n \cdot c \log n = c(\log n)^2$ flips, right?

It takes $\theta(\log n)$ flips to get at least 1 head, another $\theta(\log n)$ flips to get $\theta \log n$ heads, so let's try picking an arbitrary constant ($c = 10$) as an example and calculate.

What is the probability we get exactly $\underline{c \log n}$ *heads in* $\underline{10c \log n}$ *flips?*

$$prob = \binom{10c \log n}{c \log n} \left(\frac{1}{2}\right)^{c \log n} \left(\frac{1}{2}\right)^{9c \log n}$$

What is the probability we get exactly $\underline{c \log n}$ *heads in* $\underline{10c \log n}$ *flips with high probability?*

$$Pr(\text{ error }) = Pr(\leq c \log n \text{ heads in } 10c \log n \text{ flips })$$
$$= Pr(\text{ } 9c \log n \text{ tails in } 10c \log n \text{ flips }) = \binom{10c \log n}{9c \log n} (1/2)^{9c \log n} \leq \left(\frac{1}{2}\right)^{c \log n}$$

Therefore, $10c \log n$ flips result in c log n heads with high probability.

Quicksort

Pseudocode:
Note: This pseudocode does not handle duplicates in array correctly.

```
Quicksort(A[1 ... n])
if n = 1 then return
p = choosePivot(A[1 ... n])
partition(A[1...n], p)
Quicksort(A[1 ... rank(p) - 1])
Quicksort(A[rank(p) + 1 ... n])
```

1. **What is the recurrence when all pivots are *good*? And what is the total cost?**

Definition(A good pivot):
A pivot is good if its rank is $\geq \frac{n}{4}$ and $\leq \frac{3n}{4}$

We want a pivot whose rank is in the middle.
We can choose the first element, median of first/last/middle, randomly, something else.

Recurrence when all pivots are good:
$T(n) \leq T(\frac{n}{4}) + T(\frac{3n}{4}) + n = O(n \log n)$
If we choose a random pivot, $Pr($ pivot is good $) = \frac{1}{2}$.

Calculating total cost:
At any level of the recursion tree, the work is O(n).
If all pivots are good, the depth of the recursion tree is:

$$h(n) \leq h\left(\frac{3n}{4}\right) + 1$$
$$\leq log_{\frac{4}{3}} n = O(\lg n)$$

Therefore, the total amount of work (cost) (AKA time complexity of the algorithm) is $\boxed{\theta(n \log n)}$.

1. **On any path in the recursion tree, there can be good pivots and bad pivots.**
(a) What is the maximum possible number of good pivots on any path?

$$log_{4/3} n$$

On every good pivot, we have a remaining problem of size $\leq \frac{3n}{4}$. So $log_{4/3} n$ is the number of times we can reduce n by a factor of $\frac{3}{4}$.

(b) What is the maximum depth of any path with high probability? $\theta(\log n)$

Linear Time Rank Finding (Quickselect)

Goal: Find k-th smallest element in an array
Pseudocode:

Select(A[1 ... n], k)
if sizeof(A) <= 32 then: sort A; return A[k] (cheesy base case)
p := choosePivot(A)
rank(p) := partition(A, p)
if rank(p) = k then return p
if rank(p) < k then return Select(A[rank(p) + 1 ... n], k - rank(p))
if rank(p) > k then return Select(A[1 ... rank(p)], k)

1. Analyze the best and worst case of linear rank finding.

If the pivot p is always the median, the runtime is:

$$T(n) = T\left(\frac{n}{2}\right) + n = O(n)$$
$$T(1) = 1$$

If the pivot p is always on the ends in sorted order, the runtime is:

$$T(n) = T(n-1) + n = O(n^2)$$
$$T(1) = 1$$

2. Show the runtime of Quickselect is O(n) WHP

A good pivot range: $\frac{n}{4} \leq k \leq \frac{3n}{4}$

The probability of selecting a good pivot: $\frac{1}{2}$

There's four cases after we select a good pivot:

Case(1): Discard $\frac{3n}{4}$ elements to left of p **Case(2):** Discard $\frac{3n}{4}$ elements to right of p **Case(3):** Discard $\frac{n}{4}$ elements to left of p **Case(4):** Discard $\frac{n}{4}$ elements to right of p

In all cases, we can discard at least $\frac{n}{4}$ elements.

$$T(n) \leq T(\frac{3n}{4}) + n = O(n), \qquad T(1) = 1$$

To show this happens WHP, let's apply coin flip logic. We need $log_{4/3}n$ good pivots, so it would take $\theta(\log n)$ "flips" to get these pivots (shown on page 3).

Stony Brook University
Department of Computer Science

CSE 385 Spring 2023

CSE 385 Final Review

With High Probability

Author: Yan Kun (Alex) Meng

Professor: Michael Bender

Date: 2/28/2023

The Union Bound

$$Pr[A \cup B \cup C] \leq Pr[A] + Pr[B] + Pr[C]$$

This is the **tightest**, when A and B and C are disjoint. Vice versa,
$Pr[$ event A or event B or event C happens] could be much smaller than the probability of event A happening plus the probability of event B happening plus the probability event C happening.

This is usually used for calculating small errors. Who cares if there's small deviation.

W.H.P.

An event, **A**, occurs with high probability (WHP) with respect to a parameter, **n**, if:

$$Pr[\text{A not happening}] = Pr[\ \underline{\text{Error}}\ ] < \frac{1}{poly(n)}$$

"Probability that A doesn't occur is polynomially small"

$$Pr[A] \geq 1 - \frac{1}{n^C}$$

"Probability that A occurs is very close to 100%"
Where **C** is as large as possible. Greater than 1.

Think about it. A is happening with a very high probability, which means it is probably going to happen. So the Pr[A] doesn't happen should be very small.

When we presented the concept of "expectation", we just said it was an average over the total possible outcomes. It is what we should expect to get in any outcomes, and it is what we get the most if we did an infinitely number of trials. But I don't feel comfortable!! If we expect to get into 5 colleges in a round of college apps (I can't apply to colleges an infinitely number of times in my life), will we actually get into 5 colleges? What is the probability of actually getting into 5 colleges?

When any mental trauma like this occurs, we can try to use the reasoning of "getting into 5 colleges" is **with high probability** to gain some confidence. We need to **prove** that **it is really really likely** that we will "get into 5 colleges".

*Note: With high probability is tied very closely with **expectations** and the **union bound** for calculating errors.*

Example 1:

Suppose I have n bins. I randomly throw balls at the n bins. The expected number of balls we need to throw was calculated to be $n \ln(n)$ balls. Now we ask the question:

Does throwing $n \ln n$ balls at the n bins gets at least 1 ball into each bin WHP?

Does our expectation happen with high probability?

Strategy:

To prove something is going to happen, we need to show that the total error is small.

Explanation: Remember union bound. We are going to calculate the probability that ***all possible errors*** *that could happen, add them all up, and show that* ***it is still polynomially small****.*

What is the probability that after throwing $n \ln n$ balls, there exists a bin that's still empty?

What do you think the number of balls we need to throw is? Note that we are going to think asymptotically here. Meaning that we can add a constant C. $\rightarrow C \cdot nlnn$

1. Add up all possibilities of error

Let's calculate the probability that one bin is not filled = Probability that after cnln(n) throws, bin 1 is still empty.

$$Pr[\text{ bin 1 is not filled }] = \left(1 - \frac{1}{n}\right)^{c \cdot n \ln n} \approx \frac{1}{e^{c \cdot \ln n}} = \frac{1}{n^c}$$

2. By union bound:

$$Pr[\text{ there exists a bin that is not filled }] \leq n \cdot \frac{1}{n^c} = \frac{1}{n^{c-1}}$$

So that there exists a bin that is not filled is very not probable. Now we are confident.

Another way to ask the question:

Qs: What is the expected number of balls we need to throw into n bins so that no bin is empty WHP with respect to the number of bins n?

$\theta(n \log n)$ balls with a large enough constant

Example 2:

(a) How many times do I need to flip a fair coin to get a head, with high probability, w.r.t n?

Note: It does not matter what n is.

Say we flip L times:

Event: Flip a coin L times and get a head

Error: Flip a coin L times and don't get a head (Prove unlikely)

$$Pr[\text{ no head in 1 flip }] = \frac{1}{2}$$

$$Pr[\text{ no head in L flips }] = \frac{1}{2^L} = Pr[\text{ error }]$$

$$Pr[error] < \frac{1}{poly(n)} = \frac{1}{n^c} = \frac{1}{2^L}$$

$$L = c \lg n$$

If we want to get a head in some number of flips, with high probability, we need $c \lg n$ flips with respect to some parameter n.

What is n? n is just parameter. n is given by the problem. n means n. It might not have any meaning now. But it will have meaning later. Now we need n to say our error is polynomially small in n. That's true no matter what n is.

(b) How many times do I need to flip a fair coin to get some number of heads, say $\log n$ heads, with high probability with respect to n?

Event: Flip a coin L times and get $\log n$ heads

Error: Flip a coin L times and don't get $\log n$ heads (Prove poly small)

Bound the error probability, and show that the error probability is poly small or smaller.

$$Pr[\text{ there are less than } c \log n \text{ heads in } 10c \log n \text{ flips}]$$

Simplify:

$$Pr[\ c \log n \text{ heads in } 10c \log n \text{ flips }]$$
$$= \binom{10c \log n}{c \log n} (1/2)^{c \log n} (1/2)^{9c \log n}$$

$$Pr[\text{ less than } c \log n \text{ heads in } 10c \log n \text{ flips }]$$
$$\leq \binom{10c \log n}{9c \log n} (1/2)^{9c \log n}$$

Intuition: the probability that there are less than $c \log n$ heads in $10c \log n$ flips means there are more than, or at least, $9c \log n$ tails. So the probability that there are at least $9c \log n$ tails in $10c \log n$ flips, not caring the rest $c \log n$ flips is $\binom{10c \log n}{9c \log n}(1/2)^{9c \log n}$(no constraint).

Since $\binom{10c \log n}{9c \log n} = \binom{10c \log n}{c \log n}$

$$Pr[\text{ less than } c \log n \text{ heads in } 10c \log n \text{ flips }] \leq \binom{10c \log n}{c \log n}(1/2)^{9c \log n}$$

By Deathbed Formula:

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

$$\binom{10c \log n}{c \log n} \leq (10e)^{(c \log n)}$$

$$Pr[\text{ less than } c \log n \text{ heads in } 10c \log n \text{ flips }] \leq (10e)^{(c \log n)}\,(1/2)^{9c \log n} = \left(\frac{10e}{2^9}\right)^{c \log n}$$

What the heck? what is this $\frac{10e}{2^9}$ bs, all I see is some constant less than 1, so let's make it $\frac{1}{2}$.

$$Pr[\text{ less than } c \log n \text{ heads in } 10c \log n \text{ flips }] \leq \left(\frac{10e}{2^9}\right)^{c \log n} = \left(\frac{1}{2}\right)^{c \log n} = \frac{1}{2^{c \log n}} = \frac{1}{poly(n)}$$

Therefore, $10c \log n$ flips results in $c \log n$ heads with high probability.

Extra Problem 1:
If I throw n balls in n bins, the fullest bin has $\theta(\frac{\log(n)}{\log(\log(n))})$ balls. Prove that, with high probability, $\theta(n)$ bins will be full, and $\theta(n)$ bins will be empty.

Extra Problem 2:
If I throw n balls into m bins, what's the smallest value of m such that the expected number of bins with more than 1 ball is less than 1.

Note: In other words, what's the smallest number of bins, so that we don't expect any bins to have more than 1 ball in it. (Hint: Birthday Paradox)

Stony Brook University
Department of Computer Science

CSE 385 Spring 2023

Lectures 16-22

Optimization Problems

Author: Yan Kun (Alex) Meng

Professor: Michael Bender

Date: 5/16/2023

Algorithms

- Repeated Doubling
- Shortest Path in a DAG
- Longest Path in a DAG
- Bottleneck Path in a DAG
- Floyd-Warshall Algorithm
- Edit Distance
- Knapsack Problem

Repeated Doubling (Optimization Introduction)

There's a hole in the fence distance N away from a cow. How can the cow walk to the hole in the minimum number of steps.
You are blind-folded and there's a book somewhere near you. How can you touch the book in the minimum number of steps.
There's a car that needs gas. There's a gas station somewhere near it. How can you reach that gas station in the least amount of time.

How is this an optimization problem?

Minimize the number of steps taken to reach a fixed goal.

Recurrence

$$T(n) = T(\frac{n}{2}) + n = \theta(n)$$

Fibonacci Numbers (Dynamic Programming Introduction)

The simplest dynamic programming problem you can have. Initialize. Loop Assign. Finalize.

Dynamic Programming: Remember past computations in an array

Directed Acyclic Graph

$$G = (V, E)$$
$$|V| = n = \text{number of vertices}$$
$$|E| = m = \text{number of edges}$$

Adjacency Matrix (2D Array)

Query Connection time: $O(1)$
Space: $O(n^2)$
Time required to find all neighbors: $O(n)$

Incidence List (Array of lists)

Query Connection time: $O(d(i))$
Space: $O(n + m)$
Time required to find all neighbors: $O(d(i))$

Degree of node d(v) (number of touching edges)

$$\sum_{u \in V} d(u) = 2|E| \qquad (\text{ undirected graph })$$
$$\sum_{u \in V} din(u) = \sum_{u \in V} din(u) = |E| \qquad (\text{ directed graph })$$

Directed Acyclic Graph (DAG)

Def(DAG): A directed graph with no cycles
Model: One comes before the other. One can never go back to itself.

Topological Sort on DAG

Def(topological sort): Order nodes such that all edges go from left to right
Time required: $O(n + m)$

Shortest Path in a DAG (Modeling DP Problems)

w_{ij} =the weight of edge ij

How is this an optimization problem?

Minimizing the distance from one node to the n-th node.

$$min(D_{1n})$$

(After topological sort, this is the shortest distance from first node to the n-th node)

Loop Assignment:

$$D_{1j} = min_{k=1,...,j-1}(D_{1k} + w_{kj})$$

Problem 1 (2022)
T F The shortest path in a DAG necessarily stays the same when you add 1 to the weight of every edge

False. Proof by Counterexample:

If ($a \to b$, 1) and ($b- > c$, 1) and ($a \to c$, 2.5), then $a \to b \to c$ is the current shortest path. But if you add 1 to every edge, then $a \to c$ will be the shortest path.

Problem 2 (2022)
In this problem we show how to find the **lexicographically smallest path** in a directed acyclic graph (DAG).

Definition: Consider two paths A and B through a DAG. Let $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ be the **sorted edge weights in descending order** for A. Let $\beta = (\beta_1, ..., \beta_n)$ be the **sorted edge weights in descending order** for B.

Assumptions:

- All edge weights in the given DAG are **positive integers**
- All edge weights are **unique**

Strategy:

- You are given a DAG G = (V,E, ω_G). Construct a new DAG H = (V, E, ω_H)
- Graphs G and H share the same vertices and edges, but have different edge weights. Specifically, for all $e \in E$ with a weight $\omega_G(e)$ in G, the edge will have weight $\omega_H(e) = 2^{\omega_G(e)}$ in H. Thus, if an edge in G has weight 3, then that edge in H has weight $2^3 = 8$.

- Then find the shortest path through H (using the algorithm taught in class)

Notation:

- Let A_G be some path in G
- Let A_H be the same path, but in H. Thus, the weight for each edge in A_G is $\omega_H(e)$ instead of $\omega_G(e)$
- α_G is the corresponding sequence of sorted edge weights for A_G
- α_H is the corresponding sequence of sorted edge weights for A_H

(1) Show that if path A_G has a **smaller bottleneck edge** than path B_G, then A_H is shorter than B_H by calculating:

- lower Bound and Upper bound on length of A_H (in terms of x, the heaviest edge weight in A_G)
- lower Bound and Upper bound on length of B_H (in terms of Y, the heaviest edge weight in B_G)
- now use the fact that y > x to show that A_H is shorter than B_H

The transformation from G to H by setting $\omega_H(e) = 2^{\omega_G(e)}$ ensures that the weights in H are greater than or equal to the corresponding weights in G. As a result, finding the shortest path in H will yield the path in G with the minimum sum of edge weights.

Since the edge weights in G are positive, the sum of edge weights along a path in G is directly related to the maximum (bottleneck) edge weight on that path. In other words, the bottleneck weight of a path in G is the largest weight among all the edges on that path.

Thus, when you find the shortest path through H, which corresponds to a path in G, it will also be the path with the minimum maximum edge weight (bottleneck weight) among all paths in G. Hence, the shortest path through H is equivalent to the bottleneck shortest path through G.

Longest Path in a DAG

How is this an optimization problem?

Maximize the distance between two nodes.

Problem 1 (Longest Increasing Subsequence)
Given an integer array nums, return the length of the longest strictly increasing subseqeuence.
Def(subsequence): An array derived from another array by deleting 0 or more elements

Bottleneck Shortest Path in a DAG

I am going from New York to Seattle. Path 1 I go to Chicago then go to Seattle. Path 2 I go to UIUC then go to Seattle. In path 1, the maximum distance is the distance I have to travel from chicago to Seattle. In path 2, the maximum distance is the distance I have to travel from UIUC to Seattle. That maximum distance is smaller in path 1 comparing to path 2, so I will go path 1. However, the total distance in path 2 might be shorter than path 1.

How is this an optimization problem?

Minimize the maximum distance between different paths between two nodes.

Problem 1 (2022)
T F This Algorithm necessarily stays the same when you add 1 to the weight of every edge

Floyd-Warshall's All-Pairs Shortest Path

Find the shortest distance between every pair of nodes.

How is this an optimization problem?

Minimize the total distance from every node to 1 node.

Note: Floyd-Warshall's all-pair shortest path cannot handle negative cycles, but can handle positive cycles.

Problem 1 (2022)
T F This Algorithm works correctly on a DAG with negative weights

Edit Distance

The Knapsack Problem

Given a set of integers S = $s_1, s_2, s_3, ...s_n$ and a target value T. Find a subset S' of S such that

$$\sum_{s_i \in S'} s_i = T$$

How is this an optimization problem?

Maximizing the sum of a subset (under the constraint of some value).

Problem 3 (2022)

You run a fledgling tech company and your hiring manager comes to you with n potential employees and their salary requirements. Unfortunately You have a hard limit of d dollars in salary that you cannot go over. Assume that the salaries are perfectly set, and that they are directly proportional to how much production you get from each employee. Thus, you maximize productivity by maximizing the total salary (while staying under your limit of d dollars). What algorithmic problem explains how to choose what set of employees to hire?

d is the target size
n is the total number of elements in the set
time: $O(nd)$
space: $O(nd)$

Why is this not a polynomial time algorithm? (why is knapsack NP-Complete)

What's the size of the knapsack problem in bits? We can assume all $s_i < d$

$$O(n \log d) \quad (\text{ bits })$$

Recall: a word of is $\theta(\log n)$ *bits.*

The running time is linear in n but exponential in d.

Suppose every word is 64-bits. Since $64 = \log d$, $2^{64} = d$. So the actual running time would be $O(n \cdot 2^{64})$

Problem 1 (2022)

T F The dynamic-programming solution to the Knapsack problem is pseudo-polynomial because it runs in polynomial time with respect to the input size n and the maximum size of the knapsack, but exponential time with respect to the number of bits required to represent the maximum size of the knapsack

Stony Brook University
Department of Computer Science

CSE 385 Spring 2023

Lectures 23-24

Skip Lists

Author: Yan Kun (Alex) Meng

Professor: Michael Bender

Date: 5/15/2023

Lecture 23 – Skip Lists Introduction

Why do we need Balanced Binary Search Trees?

To maintain some type of **order**, so we can implement successor(x) and predecessor(x).

We defined dictionaries as data structures that maintains a set of elements S.

Successor(x): Minimum y that is bigger than x.

Predecessor(x): Maximum y that is smaller than x.

Skip Lists

Goal: Implement a Dictionary.

For a sorted linked list, the time it takes to search the list is:

$$search : O(n)$$

$$14 \to 18 \to 23 \to 28 \to 34 \to 42 \to 50 \to 59 \to 66 \to 72 \to 79$$

How fast can you search with two linked lists?

$$time : O(\sqrt{n}) \leftarrow 2\sqrt{n}$$

In the express lane, we traverse at most $\sqrt{n}$ nodes, and then in the normal lane, we traverse another $\sqrt{n}$ nodes. This is because the express lane nodes divides the normal lane nodes to $\sqrt{n}$ sections, so each section must have $\sqrt{n}$ nodes for the normal lane to have n nodes in total.

What's the total amount of time needed for search if we have k levels in a skip list?

We have to go from the top-most list all the way down to the "normal lane", and each lane we have to traverse $O(n^{\frac{1}{k}})$ elements. So the time it needs to find a node in a skip list:

$$time : k \cdot n^{\frac{1}{k}} = O(n^{\frac{1}{k}})$$

How many nodes does the topmost level have if we have n lanes?

$$space : \theta(n^{\frac{1}{n}})$$

The goal of skip lists is to minimize search time. How many levels do we need to minimize the search time using skip lists?

The optimal number of levels is $\log(n)$. time $= \log(n) \cdot n^{\frac{1}{\log n}}$

$$\log(n) \cdot n^{\frac{1}{\log n}} = \log(n) \cdot (2^{\log n})^{\frac{1}{\log n}} = 2\log(n) = \boxed{O(\log n)}$$

Lecture 24 – Skip Lists Analysis

Goal: How does **coin flip proof** relate to **skip lists**?

With High Probability Review:

Event E occurs WHP wrt n if

$$Pr[\text{Error}] \leq \frac{1}{poly(n)} = \frac{1}{n^c}$$

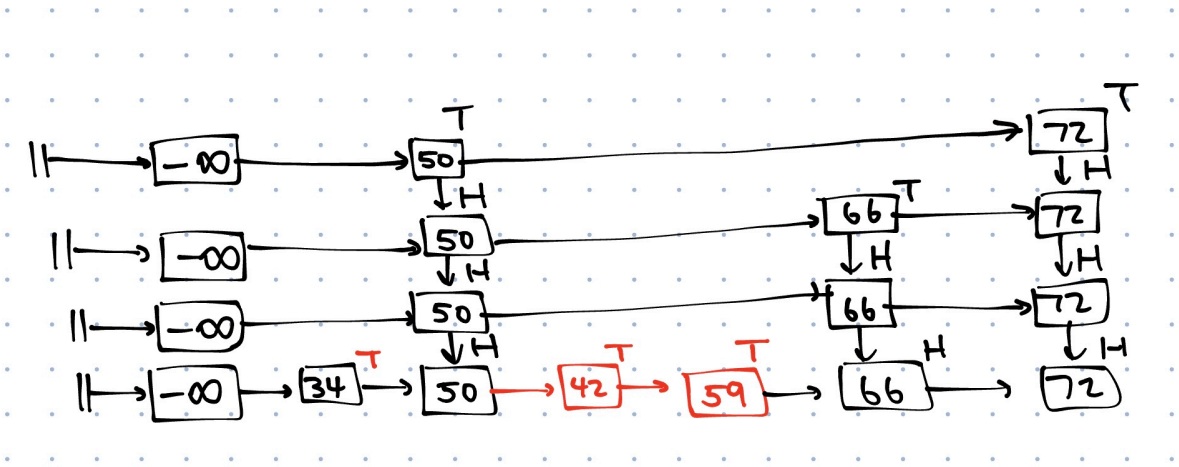
Claim 1: If we flip a coin $c \log n$ times, we will get at least 1 head WHP wrt n.

Pr[no head] $(c = 1) = \frac{1}{n}$
Pr[no head] $(c = \frac{1}{2}) = \frac{1}{n^{\frac{1}{2}}}$

Claim 2: If we flip a coin $c \log n$ times, we will get $\theta(\log n)$ heads WHP wrt n.

Insert an element:

Note: The probability of getting a hundred heads in a row is similar to two meteours hit you at the same time. It could happen. With high probability it won't.



Claim 1: With high probability in n, the skip list has at most $\theta(\log n)$ levels . EVERY. SINGLE. NODE. has at most $\theta(\log n)$ levels.

Proof 1:
Error: some node have greater than $\log n$ levels.

$$Pr[\text{ node 1 has greater than } \log n \text{ levels }] \leq \frac{1}{n^c}$$

"We need to flip coin $\theta(\log n)$ times to get a tail WHP, which is when we stop going up the skip list."

$$Pr[\text{ any node has greater than } \log n \text{ levels }]$$
$$\leq Pr[\text{ node 1 error }] + Pr[\text{ node 2 error }] + ... + Pr[\text{ node n error }]$$
$$= n \cdot \frac{1}{n^c} = \frac{1}{n^{c-1}}$$

This is true by **union bound**.

Therefore, claim 1 is true.

Claim 2: With high probability in n, a search path spends $O(\log n)$ time on any level.

Proof 2:
The reason why we spend a lot of time on a level is because **all of the nodes we are searching are tails.** If a node is a head, we go down to the next level, and so on.

We need $\theta(\log n)$ flips to get 1 head, which is when we leave the level. So there's at most $\theta(\log n)$ tails on each level.

Combining Previous Claims

Ultimate Claim: With High Probability, a search for a node takes time $O(log^2 n)$ in a skip list.
Intuition: You spend $\log n$ amount of time on each level, for $\log n$ levels.

The catch is, it is $O(\log n)$ on one level, but **you will not spend** $\log n$ time on **every level**. So we can improve things **overall**. How do we prove:

You will not spend $O(\log n)$ time on every level

Backwards Analysis – Building Search Path

If we go backwards on a search path:

Search path comes from the left when the coin flip is tails. Search path comes from above when the coin flip is heads.

After

We know there are at most $\theta(\log n)$ levels WHP. After we flip $\theta(\log n)$ times, we know we have $c \log n$ heads WHP by Claim 2 in the beginning. So the search path has at most $\theta(\log n)$ heads. So this sequence of head bounds the search path.

Claim: the length of the search path is bounded by **the number of coin flips that you do until you get $\theta(\log n)$ heads**. Because we know that if you got $\theta(\log n)$ heads, you went down $\theta(\log n)$ levels and have found the target.

Space of Skip List WHP: $O(n)$
A randomized process that gives you the space of a skip list:

$$\textit{The number of coin flips until we get } n \textit{ tails} = 2n = O(n)$$

Stony Brook University
Department of Computer Science

CSE 385 Spring 2023

Lectures 25-27

Disk Access Model

Author: Yan Kun (Alex) Meng

Professor: Michael Bender

Date: 5/16/2023

Summary

Memory = Cache, and we are reading from or writing to **Disk Cache Oblivious: Parameterized by B and/or M**

- Linear Scan $[O\left(\frac{N}{B}+1\right)]$ **B might be bigger than N**

  Cache Oblivious: **No** Optimal: **YES**

- Binary Search $[O\left(\log\left(\frac{N}{B}\right)\right)]$ **Inefficient**

$$T(N) = T\left(\frac{N}{2}\right) + 1 \quad T(B) = 1$$

  Cache Oblivious: **No** Optimal: **No**

- Binary MergeSort $[O\left(\frac{N}{B}\log\frac{N}{M}\right)]$ **Recursion Tree Analysis**

$$T(N) = 2T\left(\frac{N}{2}\right) + \frac{N}{B} \quad T(M) = M/B$$

  Cache Oblivious: **No** Optimal: **No**

- K-Way MergeSort $[O\left(\frac{N}{B}log_{\frac{M}{B}}\left(\frac{N}{B}\right)\right)]$ **Better memory utilization with Parallel Merge**

$$T(n) = \frac{M}{B}T\left(\frac{N}{\frac{M}{B}}\right) + \frac{N}{B} \quad T(M) = \frac{M}{B}$$

  Cache Oblivious: **YES** Optimal: **YES**

Matrix Multiplication:

- Blocking (size B) $[O\left((\frac{N}{\sqrt{B}})^3\right)]$ **Each multiplication take constant time**

  Cache Oblivious: **YES** Optimal: **no**

- Blocking (size M) $[O\left(\frac{N^3}{B\sqrt{M}}\right)]$ **Each multiplication take $\frac{M}{B}$ time**

  Cache Oblivious: **YES** Optimal: **YES**

- Divide and Conquer (Space-Filling Curves; Row-major Order with Tall-cache) $[O\left(\frac{N^3}{B\sqrt{M}}\right)]$ **Automatically optimal**

$$T(N) = 8T\left(\frac{N}{2}\right) + \frac{N^2}{B} \quad T(\sqrt{M}) = \frac{M}{B}$$

  Cache Oblivious: **No** (analysis depends on B and M, algorithm doesn't) Optimal: **YES**

Lecture 24 – I/O Cost Mindset

Consider a two level hierarchy, where there is the cache and a disk. The cache is size N, and B is the block size, so it can store $\frac{N}{B}$ blocks.

Cost:

- 1 for transfer of block between cache and disk
- 0 for computation

What is the cost to scan an array?

Here it is not the number of elements in the array, but the number of blocks in an array. Which is $O(\frac{N}{B})$

Binary Search in an array of size N

Since there are N/B blocks, the cost is $\log(\frac{N}{B})$. You can organize the disk more efficiently than storing data in sorted order.

Recurrence:

$$T(n) = T\left(\frac{n}{2}\right) + 1$$
$$T(B) = 1$$

Look in middle, look to left and right.

Nahh. B-trees do much faster search.

Sorting in DAM (Binary Mergesort)

Memory of size M. Two sorted arrays, each of size $\frac{N}{2}$. Output a single array of size N. Why is it $\frac{N}{B}$? It's the number of blocks we need to touch to merge.

$$T(N) = 2T\left(\frac{N}{2}\right) + \frac{N}{B}$$

MergeSort: Bring in two blocks ($\frac{N}{2B}$ blocks in each) into memory. Merge them ($\frac{N}{B}$). Spit them out of memory (0). $\theta(\frac{N}{B})$

What's the base case of the recurrence? It's when the problem is small enough to fit inside memory entirely (when N becomes M) (**M/B because is the cost to read it in**, cost 0 to sort, cost 0 to write)

$$T(M) = M/B$$

Once the sorting problem is **small enough to fit into memory**, the only cost is reading the data.

Analysis of Binary MergeSort:

Recursion Tree: $1 + \log N = (\log N - \log M) + (1 + \log M) = (\log \frac{N}{M}) + (1 + \log M)$
Top Level: N **Cost:** $\frac{N}{B}$
Second Level: N/2; N/2 **Cost:** $\frac{N}{B}$
Third level: N/4; N/4; N/4; N/4 **Cost:** $\frac{N}{B}$
Some Level: M; M; M; .... M; **Cost:** $\frac{N}{B}$
Last Level: 1; 1; 1; 1; 1; 1; 1; 1 **Cost: 0**

$$\boxed{TotalCost : O\left(\frac{N}{B}\log\frac{N}{M}\right)}$$

Note: As soon as we have a problem of size M, we are using memory efficiently. Note: We are using all memory for base case, but only $2B$ while merging.

Lecture 25 - K-Way MergeSort

More Efficient MergeSort (using all memory for merging):

Recursively divide into subarrays of size $b = \frac{N}{\frac{M}{B}}$, merge all together AT THE SAME TIME in memory $\frac{M}{B}$, and output a merged array of size N.

What is the recurrence relation? Divide each problem of size n into problems of size M/B

$$T(n) = \frac{M}{B}T\left(\frac{N}{\frac{M}{B}}\right) + \frac{N}{B}$$

Base Case:

$$T(M) = \frac{M}{B}$$

If we were dividing into K problems (K = number of blocks fit in memory at once):

$$T(n) = KT\left(\frac{N}{K}\right) + \frac{N}{B}$$
$$T(M) = K$$

Total Cost = $O\left(\frac{N}{B}log_{M/B}\left(\frac{N}{B}\right)\right)$

We say that $\frac{M}{B} = K$, the number of blocks we fit in memory, and it's name is the "fan out".

If you are actually implementing this in a Database:

- Don't use full memory. Only use a part of it.
  –there are other programs going on –Smaller K $< \frac{M}{B}$(the fan-out value - how many blocks we are merging in the memory **at once**)
  –If you $\sqrt{K}$, then that's going to double your cost

Space is more important than time

- Pay attention to CPU Cost too
  –Maintain blocks in a priority queue
  –Use heap to keep track of min values to merge quickly

Matrix Multiplication:

How do people make Matrix Multiplication fast?
–parallel programming
–optimize the base case
–break up matrix multiplication to fit in memory
–**blocking**

Blocking:

In the context of the DAM model, what's a good block size for Matrix Mult.?

Analysis with chunks of size **B** ($\sqrt{B} \times \sqrt{B}$ matrices)

Cost to multiply two chunks: **O(1)** (in memory)
The matrix is divided into: $\frac{N}{\sqrt{B}}$ chunks
Total Matrix Multiplication Cost (w/ blocking B): $O\left((\frac{N}{\sqrt{B}})^3\right)$

What if we chose the chunk size to be M ($\sqrt{M} \times \sqrt{M}$), really big chunks, the size of the cache?

Cost to multiply 2 chunks together: $O(\frac{M}{B})$
How many chunk multiplications: $(\frac{N}{\sqrt{M}})^3$
Total Cost: $O\left(\frac{M}{B}(\frac{N}{\sqrt{M}})^3\right) = O\left(\frac{N^3}{B\sqrt{M}}\right)$

Lecture 26 - D&C Matrix Multiplication in DAM Model

We need to start with a memory **layout**:

- Row-Major Order (left-to-right)
- Column-Major Order (top-to-bottom)
- Blocking Order (stuff inside doesn't matter in terms of the **DAM model**)
- Space-filling Curve (Morton Order/Z-Order, Four chunks recursive)

D&C with Space-Filling Curve:

We can do sequential scan of memory because space-filling curve is sequential in nature:

Recurrence for I/O Cost:

$$T(N) = 8T\left(\frac{N}{2}\right) + \frac{N^2}{B}$$

$$T(\sqrt{M}) = \frac{M}{B}$$

Note: Matrix of size N utilizes memory of N^2, if we want to fit into memory M, we have to remember to square-root it.

With analysis from recurrence Tree, **Total Cost** $= O\left(\frac{N^3}{B\sqrt{M}}\right)$

D&C with Row Major Order:

What's the space to store a $\sqrt{m} \times \sqrt{m}$ recursive chunk?
If contiguous: $O(m)$
But not contiguous now: $M + 2B\sqrt{M}$

Claim: Tall-Cache Assumption ($M > B^2$)

If M and b satisfy the tall-cache assumption, then $M + 2B\sqrt{M} < M + 2\sqrt{M}\sqrt{M} = O(M)$

D&C is not cache oblivious but it is **optimal**.

Cache Oblivious Analysis:

Oblivious - "don't care"

Write an algorithm that is not parameterized by B and M.
Analyze in DAM Model using B and M.
If the algorithm is optimal in DAM model, then **it is optimal for all values of B and M** (works anywhere). Which means that it is optimal on a multilayered memory hierarchy or a hierarchy with unknown parameterization.

Stony Brook University
Department of Computer Science

CSE 385 Spring 2023

Lectures 25-27

B-Trees in DAM Model

Author: Yan Kun (Alex) Meng

Professor: Michael Bender

Date: 5/17/2023

Lecture 26, 27 - B-Trees

Optimal for searches (expensive), not for insertions and deletions (cheap).
Rebalancing is a very very minor cost.

Search:

Pivots in nodes, all leaf nodes are at same depth.

Search Tree with Fan-out **B (width of a node)**

Cost to search (height of tree): $O(log_B(\frac{N}{B}))$

$$T(N) = T\left(\frac{N}{B}\right) + 1$$
$$T(B) = 1$$

Insertions and Deletions:

Too big ($<$ B): split
Too small ($>$ B/4): merge