

CSE21_Final_Review

Day 1 - Counting
Day 2 - Counting
Day 3 - Permutation & Combination
Day 4 - Identities & Complement & Recurrence
Day 5 - Counting with Recurrence
Day 6 - Asymptotic notation(Big O stuff)
Day 7 - Sorting
Day 8 - Loop invariant for sorting
Day 9 - Loop invariant for iterative algorithm
Day 10 - Time analysis for sorting and searching
Day 11 - Time analysis of iterative algorithm
MT1

Day 12 - Correctness of Recurrence
Day 13 -
Day 14 -
...
Day 25
Day 26

HW0: weak/strong induction, loop algorithm.

Counting

- Product Rule: sequences of tasks.
- Sum Rule: disjoint set.
- Templates:
 - Ex: 4-letter-strings with 1 vowels and 3 consonants.
- Inclusion/Exclusion Principle
 - $|A \cup B| = |A| + |B| - |A \cap B|$. Extend to n finite sets.
 - Ex: binary string start with 1 or end with 00.
- **Permutation**: rearrangement/**ordered** r objects of n , each appears exactly once.
 - $P(n, r) = n!/(n-r)!$
 - Ex: traveling to 7 cities. Start in NY, end in SW, SD after LA $\rightarrow 4!$.
 - Ex: pick 4 different roles out of 20
- Categories:
 - # categories = # objects / (size of each category)
 - Ex1: 3 pipe cleaners with 3 colors. $3!/(3*2) \rightarrow$ count rotation.
 - Ex2: length n binary string with k ones.
 - # objects: $n!$, n distinct numbers
 - Size of each category: $k!(n-k)!$, k ones are the same, $(n-k)$ zeros are the same
- **Combination**: r elements from n distinct objects is an **unordered selection** of them.

$$C(n, r) = n! / (r! (n-r)!)$$

Ex: subset & binary string. AKA, choose k out of n positions to be ones.

Ex: pick 4 out of 20 for a conference.

Note: $C(n, r) = P(n, r) / P(r)$

· Identity:

Symmetry: $C(n, k) = C(n, n-k)$


Pascal: $C(n+1, k) = C(n, k-1)$ [length $n+1$ string start with 1] + $C(n, k)$ [length $n+1$ string start with 0].

Binomial Theorem:

THEOREM 1

THE BINOMIAL THEOREM Let x and y be variables, and let n be a nonnegative integer. Then

$$(x + y)^n = \sum_{j=0}^n \binom{n}{j} x^{n-j} y^j = \binom{n}{0} x^n + \binom{n}{1} x^{n-1} y + \cdots + \binom{n}{n-1} x y^{n-1} + \binom{n}{n} y^n.$$

Proof: We use a combinatorial proof. The terms in the product when it is expanded are of the form $x^{n-j} y^j$ for $j = 0, 1, 2, \dots, n$. To count the number of terms of the form $x^{n-j} y^j$, note that to obtain such a term it is necessary to choose $n-j$ x s from the n sums (so that the other j terms in the product are y s). Therefore, the coefficient of $x^{n-j} y^j$ is $\binom{n}{n-j}$, which is equal to $\binom{n}{j}$. This proves the theorem. 

Sum identity: $\sum C(n, k)$ for k from 0 to $n = 2^n$. Binary strings with any number of 1s.

· Complement:

4 digit strings with 0-9 have at least one zero.

WRONG: $4 \cdot 10^3$.

CORRECT: $10^4 - 9^4$

Ex: at least 2 consecutive are the same. $10^4 - 10 \cdot 9^3$ [consecutive are different]

HW1indiv:

Red7(1-7, 7 colors): # hands with same color 7 colors, # hands one card of each color: 7^7

HW1Grp:

2n players, 4n players tennis problem: $(2n)! / (2^n \cdot n!)$; $(4n)! / (2^{2n} \cdot 2^n \cdot n!)$

7 card-hand with at least one 6 and at least one 3: inclusion exclusion principle; at least 2 7s: complement.

12 gummi bears: unordered selection, symmetry, bears choose person + inclusion/exclusion.

Recurrence

Express $f(n)$ in terms of previous values.

- Guess and Check
- Unraveling

Ex: Tower of Hanoi. $T(n) = 2T(n-1) + 1$, $T(1) = 1$.

Ex: Binary strings avoiding 00. $B(n) = B(n-1) + B(n-2)$.

Ex: number of codewords: has even number of zeros. $C(n) = 10^{n-1} - C(n-1) + 9 \cdot C(n-1)$

Asymptotic Notation

BigO :

$f(n)$ is $O(g(n))$. $g(n)$ grows "just as fast or faster" than $f(n)$.

For functions $f(n) : \mathbb{N} \rightarrow \mathbb{R}$, $g(n) : \mathbb{N} \rightarrow \mathbb{R}$ we say *$f(n), g(n)$ are increasing functions*

$f(n) \in O(g(n))$

to mean there are constants, C and k such that $|f(n)| \leq C|g(n)|$ for all $n > k$.

The constants C and k are called *witnesses* to the relationship $f(n) \in O(g(n))$.

Rosen p. 205

Properties:

Domination: $f(n) \leq g(n)$ for all n , then $f(n)$ is $O(g(n))$

Transitivity: $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$

Additivity/Multiplicativity: $f(n)$ is $O(g(n))$, $h(n)$ is non-negative, $f(n) \cdot h(n)$ is $O(g(n) \cdot h(n))$

Sum is max: $f(n) + g(n)$ is $O(\max(f(n), g(n)))$.

Ignoring constants: $cf(n)$ is $O(f(n))$

Other Asymptotic notation

Theta: $f(n)$ is $O(g(n))$, $g(n)$ is $O(f(n))$

Omega: $g(n)$ is $O(f(n)) \rightarrow f(n)$ is $\Omega(g(n))$.

Limit Rule

If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c > 0$ and finite, then $f(n) \in \Theta(g(n))$

If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$ is finite, then $f(n) \in O(g(n))$

If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c > 0$ or infinite, then $f(n) \in \Omega(g(n))$

Ex: prove/disprove n^2 is $O(2^n)$. Apply L'Hospital's Rule.

Indi_HW2:

Explain recurrence (Remember Base Case!); guess/check, unraveling.

O notation: $\log(n) + \log(2n) + \log(3n) \rightarrow O(\log(n))$.

Grp_HW2:

· Recurrence:

Subset of no-three-consecutive integers: similar to binary strings avoid 000 \rightarrow case contain n , $n \& n-1$, n but no $n-1$; even run of 1's.

· O notation:

$\log(n^2) + \log(100n^{10})$ is $\omega(\log_2(n))$. **Property of logs.. IMP.**

$2^n < n!$ For $C = 1$, $k > 3$. Prove by induction.

Sorting

Selection Sort(min sort):

For $i := 1$ to n ,

Find the minimum am of a_i to a_n ,
interchange am with a_i .

Bubble Sort

Starting from the beginning of the list, compare every adjacent pair, swap their position if they are not in the right order (the latter one is smaller than the former one). After each [iteration](#), one less element (the last one) is needed to be compared until there are no more elements left to be compared.

Insertion Sort

For $j := 2$ to n
Find where a_j belongs in the list a_1 to a_{j-1}
Put a_j in its place.

Bucket Sort

Create Buckets that have ordering
Put elements into correct buckets and then sort
Concatenate the buckets

Merge Sort

Divide into 2 pieces
Merge the two lists

Loop invariants

State the property ->
Prove by induction it's true for any number of loop iterations ->
Use invariant to prove correctness by plugging in n into loop invariant

Ex: proof of binary search, min sort, and etc.

ITERATIVE ALGORITHM

Searching and Time analysis

Time: count comparisons.

Selection Sort (Min sort)

$O(C(n, 2)) \rightarrow n-1 + n-2 + \dots + 1 = n(n-1)/2$

Linear search - unsorted:

Worst case: $O(n)$
Best case: $O(1)$

Binary search - sorted:

Best case: $O(1)$ or $O(\log(n))$
Worst case: $\log(n)$ [take integer part] + 1 comparisons: $O(\log(n))$.
Sort first, then search: $O(n \log(n)) + O(\log(n))$ is $O(n \log(n))$

Ex: Summing triples

Case1: Loop through every possible i, j and $k \rightarrow O(n^3)$
Case2: For each candidate ($O(n^2)$), do linear search $O(n)$.
Case3: For each candidate ($O(n^2)$), do binary search $O(\log(n)) \rightarrow O(n^2 * \log(n))$

Fastest known: n^2

Intersecting sorted list:

Use linear search to search for b_1

Start search for b_j where b_{j-1} left off.

Indi_HW3:

State loop invariant, linear search/binary search condition, iterative algorithm runtime.

Grp_HW3:

InsertionSort: A: beginning, B: end, C: binary.

Loop invariant: range M problem, state 8 cases clearly

Best case and worst case for algorithm.

RECURSIVE ALGORITHM

Ex: count how many times substring 00 occurs in the string

Proof by induction

Base case: base case of recursion

Induction: Assume algorithm is true for ALL input size $n - 1$, Show the algorithm is correct on ALL input size n . NOTE **Condition: $k > 0$**

Time analysis:

Ex: substring 00: $T(n) = T(n-1) + c$, $T(0) = T(1) = d$. Unravel $\rightarrow T(n) = c(n-1) + d \rightarrow O(n)$

Ex: Merge sort: refer below. $T(n) = 2 * T(n/2) + cn$.

Divide and Conquer - Merge Sort

Divide size n into a subproblems of size n/b ,

Recursively sort each sublist,

Conquer by combining solutions of subproblems

Master Theorem

$T(n) = a * T(n/b) + g(n)$.

Master Theorem

Theorem: If $T(n) = aT(n/b) + O(n^d)$ for some constants $a > 0, b > 1, d \geq 0$,

*non recursive part
polynomial time.*

Then

$$T(n) \in \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log n) & \text{if } a = b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

MergeSort = $O(n \log(n))$

BinarySearch = $O(\log(n))$

?? Multiplication... From $O(n^2)$ to $O(n^{1.58})$, by replacing

Indi_HW4

Recurrence relationship, Master Theorem (Note: geometric theorems), Recursive proof, Best case, worst case analysis.

Grp_HW4

Master Theorem, unraveling to compute Runtime.

Compute decimal expansion of $2^n \rightarrow n^{1.56}$

Induction proof on recursive algorithm (Find 01), analyze best/worst case, apply master theorem to find run time.

Probability

Sample space, S : (finite or countable) set of possible outcomes

Probability distribution, p : assignment of probabilities to outcomes in S so that $0 \leq p(s) \leq 1$.

Event, E : subset of possible outcomes

· Uniform distribution

$$P(E) = |E| / |S|$$

$$P(k \text{ Hs}) = C(n, k) / 2^n$$

Binomial distribution: probability of exactly k successes in n independent trials

$$C(n, k) p^k (1-p)^{(n-k)}$$

· Conditional probability

Probability of E given F: $P(E | F) = P(E \& F) / P(F)$

Ex:

girls, boys.

2 siblings are girl if the oldest is a girl

/ 2 siblings are boys if one of them is a boy

??? Simpson Paradox

· Bayes' Theorem:

Independent event: $P(E|F) = P(E) * P(F)$

$$P(F|E) = \{P(E|F) * P(F)\} / \{P(E|F)*P(F) + P(E|\text{not } F)* P(\text{not } F)\}$$

· **Random Variables / Expected Value**

Random variable: function from sample space to real numbers.

Distribution: possible values to [0, 1]

$$\text{Expectation: } E(X) = \sum P(s) X(s) = \sum P(X=r)*r$$

Ex: expected number of boys in a family with 2 children:

$$\text{Linearity: } E(X_1 + \dots + X_n) = E(X_1) + E(X_2) + \dots E(X_n)$$

Ex: consecutive heads. $X_i = 1, 0$; $E(X_i) = \frac{1}{4} + 0$.

Ex: find max.

Case analysis:

$$E(X) = P(A) * E(X | A) + P(A^c) * E(X | A^c)$$

Ex: the number of pairs of consecutive Hs when we flip three times.

Ex: ending condition: Let event A be success at the first try.

Properties

$E(X)$, $E(X^2)$ -> every random variable takes square, $(E(X))^2$ -> value square

Independent: $E(XY) = E(X) * E(Y)$

Ex: E randomly generated bitstring of length 4 starts with a 1

F this string contains even number of 1s

$$P(E) = \frac{1}{2}$$

$$P(F) = C(4,0) + C(4, 2) + C(4, 4) = \frac{1}{2}$$

Concentrations

Concentration

How close (on average) will we be to the average / expected value?
Let **X** be a random variable with $E(X) = E$.

The **unexpectedness** of X is the random variable
 $U = |X - E|$

The **average unexpectedness** of X is
 $AU(X) = E(|X - E|) = E(U)$

The **variance** of X is
 $V(X) = E(|X - E|^2) = E(U^2)$

The **standard deviation** of X is
 $\sigma(X) = (E(|X - E|^2))^{1/2} = V(X)^{1/2}$

Weight all differences from mean equally

Weight large differences from mean more

Theorem

Concentration

How close (on average) will we be to the average / expected value?
Let **X** be a random variable with $E(X) = E$.

The **variance** of X is
 $V(X) = E(|X - E|^2) = E(U^2)$

Theorem: $V(X) = E(X^2) - (E(X))^2$

Proof: $V(X) = E((X-E)^2) = E(X^2 - 2XE + E^2) = E(X^2) - 2E E(X) + E^2$

$E(2XE) = 2E E(X) = 2E^2$

$E(X^2) - 2E^2 + E^2 = E(X^2) - (E(X))^2$

Linearity of expectation

Additive Property of Variance

Markov's Inequality....

Element distinctness..这个好难

Indiv_HW5:

Data Compression/Encoding

Best possible: $\log_2(S)$.

Encoding: fixed density strings. Optimal bits: $\log_2(C(n,k))$

Fixed windows size, marker, position of 1s.

Encoding/Decoding: Fixed Density Strings

Output size?

Assume n/k is a power of two. Consider s a binary string of length n with k 1s. Given $|E(s)| \leq k \log_2(n/k) + 2k$, we need at most $k \log_2(n/k) + 2k$ bits to represent all length n binary strings with k 1s. Hence, there are at most 2^{\dots} many such strings.

$$\binom{n}{k} \leq 2^{k \log_2(n/k) + 2k} = \left(\frac{4n}{k}\right)^k$$

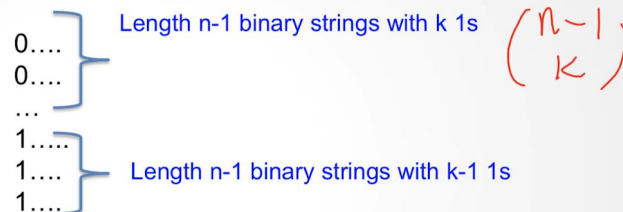
$$\binom{n}{k} = O(n^k)$$

Hard details

Lex-order Encoding

Lex Order: Algorithm?

For $E(s, n, k)$:



- Any string that starts with 0 must have position before $\binom{n-1}{k}$
- Any string that starts with 1 must have position at or after $\binom{n-1}{k}$

GRAPHS

Type of graphs

Variants of graphs

Undirected graph:

Rosen p. 644

No arrows on edges.

An edge e_{vw} connects the unordered pair of vertices $\{v, w\}$.

Directed graph:

Arrows on edges.

An edge e_{vw} connects the ordered pair of vertices (v, w) .

Multigraph:

Undirected graph that may have several edges between a pair of nodes. Such edges are sometimes called *parallel* edges.

Simple graph:

Undirected graph with no self-loops (edge from v to v) and no parallel edges.

Complete: undirected edges between any two vertices

Edgeless: zero edge

Complement: reverse the edges

Bipartite: two sets of vertices. No two edge in the same set are connected by an edge

Tree(undirected): an undirected with no cycles

Tree(directed): AKA, rooted tree, every vertex has one edge pointing to it except for the root

Hypercube: $Q_n \rightarrow 2^n$ vertices. Length- n bit string with an edge between two bit strings if they differ by exactly one bit.

Connectedness(undirected): any pair of v and w there is a path from v to w .

Directed graph: strongly connected path for any ordered pairs. Weakly connected path if ignoring the underlying order.

Number of ordered pairs:

Directed: $C(n, 2) * 2 = n(n-1)$

Undirected: $C(n, 2) = n(n-1) / 2$

Self-loop = $n + n + \dots + n = n^2$

Adjacency Matrix

Entry in row i and column j is the number of edges from vertex i to vertex j .

Definition:

Degree: total number of edges incident with it. Loop contributes twice

Ex: simple, undirected graph with n vertices and each degree d . $2|E| = n * d$. (Sum of all degrees)

Path: route from one vertex to another

Length of path: number of edges

Simple path: path that doesn't repeat

Circuit: path that starts and ends at the same vertex, length greater than zero

Loop(self-loop): edge from a vertex to itself

Eulerian path:

A path where each edge occurs exactly once.

Circuit: Eulerian path that starts and ends at the same vertex

TWO Cases:

A. all vertices have even degree. [a circuit]

B. the **starting and ending vertices** have odd degree; all other have even degree [not a circuit]

Theorem: if G has an Eulerian path, G has at most two odd-degree vertices.

Converse is also true.

Prove the algorithm.

Hamilton:

A path where each vertex occurs exactly once.

NOT KNOWN HOW TO FIND

Ex: DNA reconstruction

Ex: Three levers, \rightarrow find Hamiltonian graph

Try all 2^n possible combinations with only $2^{(n-1)}$

Graph Search:

Ex: Tartaglia's pouring problem.

Number of possible configuration, configurations actually possible

BFS: breadth first search; DFS: depth-first search

Directed Acyclic Graphs:

directed graphs with no cycles

Topological ordering:

an (ordered) list of all its vertices such that for each (directed) edge (v, w) in the graph, v comes before w in the list.

Source:

no incoming edges

A. Every DAG has at **least one source**: prove by Pigeonhole principle \rightarrow keeping going to find repeated vertices \rightarrow not acyclic

B. Let v be a source vertex of G , G is a DAG if and only if $G - v$ is a DAG

(if G is not a GAG, $G - v$ is not a DAG)
Find topological ordering, **algorithm**

Sinks: no outgoing edges

Tree

Rooted tree:

connected **DAG**, one root has **no incoming** edges, every other vertex has **exactly one** incoming edge.

Binary tree:

Rooted tree, where each vertex has no more than 2 children

Full binary tree:

Number of vertices: $2^{(h+1)} - 1 = \theta(2^h)$

$N - 1 \geq \text{Height} \geq \log(n+1) - 1 = \theta(\log(n))$

Binary search tree:

Maintain sorted order

Unrooted Tree:

Connected, undirected graphs with no cycles.

Equivalence between rooted and unrooted trees.

Application:

Another application of counting ... lower bounds

What's the **fastest possible worst case** for any sorting algorithm? $\log_2(n!)$

How big is that?

$$\begin{aligned}\log n! &= \sum_{k=1}^n \log k > \int_1^n \log x \, dx \\ \int_1^n \log x \, dx &= x \log x - x \Big|_1^n = (n \log n - n) - (0 - 1) = n \log n - n + 1 \\ \log n! &= \Omega(n \log n)\end{aligned}$$

Another application of counting ... lower bounds

What's the **fastest possible worst case** for any sorting algorithm? $\log_2(n!)$

Therefore,

the best sorting algorithms will need $\Omega(n \log n)$ comparisons in the worst case.

It's impossible to have a comparison-based algorithm that does better than **Merge Sort** (in the worst case).