Syllabus

Syllabus: CS 3305.HON Syllabus (Spring 2021)

Textbook: Mathematics for Computer Science

Meeting times: T/Th 11:30am-12:45pm

Last day of lecture: Thursday, May 6

Grading Criteria:

Note: Honorlock will be used for all exams, 24 hour window available.

1. Midterm 1: 20%
2. Midterm 2: 20%
3. Final exam: 20%
4. Homework: 25%
5. Paper review: 10%
6. Class participation: 5%

See syllabus (linked above) for more information.

Day 1: 1/19

Why Discrete?

Rigorous understanding of algorithms: Proving the correctness of algorithms and verify that their performance is as expected.

Time complexity and space complexity: space complexity is also important because RAM can only hold so much information before slowing down (due to accessing hard drive for swapping)

Asymptotic Bound

1. **Big-O**: Upper bound of complexity
2. **Big-Omega**: Lower bound
3. **Big-Theta**: Tight bound, set intersection of **Big-O** and **Big-Theta**

Modular Arithmetic

$a \equiv b \,(\text{mod}\, m) \iff a \mod m = b \mod m \iff (a-b) \equiv 0 \,(\text{mod}\, m)$

Representation of Numbers

- **Decimal**: 0, 1, 2, …, 9
- **Binary**: 0, 1
- **Hexadecimal**: 0, 1, 2, …, 9, A, B, C, D, E, F

Binary multiplying by power of 2

$5_{10} = 101_2 \implies 10_{10} = 1010_2 \implies 20_{10} = 10100_2$

Bit shift left for multiplying by 2, bit shift right for dividing (truncate) by 2.

Prime and Composite Numbers

Prime: only 2 factors, 1 and itself

Composite: more than 2 factors

Sieve of Eratosthenes

Discard multiples of prime numbers to *sieve* through set of all numbers less than n.

```python
def primes_less_than_n(n):
    primes = set([i for i in range(2, n + 1)])
    for i in range(int(sqrt(n))):
        if i in primes:
            j = 2
            while i * j <= n:
                primes.discard(i * j)
                j += 1
    return primes
```

Induction and Recursion

- **Recursion**: express problem as smaller instances
- **Induction**: used to prove recursion solutions
- **Strong Induction**: induction based on all previous steps rather than just the one previous step

Day 2: 1/21

Counting

Counting is a classic problem in discrete mathematics. Some counting problems are difficult at first, but we can map the original problem P into another problem P', count for P', then map back to P.

Pigeonhole Principle: if we have n pigeons and m pigeonholes, given $n > m$, there must be *at least one* pigeonhole with more than one pigeon.

Permutation and Combinations

Permutation: arrange k elements from a set of n elements (order matters) $= \frac{n!}{(n-k)!}$

Combination: number of ways to select k out of n: $\binom{n}{k} = \frac{n!}{(n-k)\cdot!k!}$

Binomial Theorem: $(x+a)^n = \Sigma_{k=0}^{n}\binom{n}{k}x^k a^{n-k}$

Fun fact: $(1+1)^n = 1 + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{n}$, which means that the row sum of the n-th row of the Pascal's Triangle is 2^n.

Another fun fact: $\binom{n}{k} = \binom{n}{n-k}$, which can be used to simplify calculations. The intuition is that we can select the people to *not* select, which is the same result as the original problem (mapping idea from above).

Fibonacci numbers

Recurrence relation: $n_i = n_{i-1} + n_{i-2}$, with $n_0 = 1,\ n_1 = 1$

Note that the ratio between two consecutive recurrence relations approaches ϕ.

Stairs Problem

Problem: You have a flight of n stairs, and you can take 1 or 2 steps at a time. How many ways are there to climb the n steps?

Let $f(n)$ be the number of ways to climb n steps. Then $f(n) = f(n-1) + f(n-2)$ because you can either take one step and have to take $n-1$ more steps, or take two steps and have to take $n-2$ more steps. This leads to two smaller problems that can be further reduced. One thing to note that this is the same as the Fibonacci sequence.

Relations

- **Reflective**: aRa
- **Symmetric**: aRb *implies* bRa, example is "sibling of"
- **Transitive**: aRb, bRc, then aRc, example is "less than"
- **Equivalence Relation**: reflective, symmetric, and transitive

Example

$a \equiv b \pmod{m}$

This yields m equivalence classes where each class is equal modulo m. For example, if $m = 5$, then $1, 6, 11, 16, 11, \ldots$ are all part of the same equivalence class where $a \equiv 1 \pmod{m}$.

Graphs

Graphs consist of a set of **vertices** and a set of **edges** connecting pairs of vertices. **Directed graphs** have edges that point in one direction, while **undirected graphs** have edges that do not have directions (goes both ways, equivalent to two opposite directed edges).

There are many examples of graphs, such as highway networks, the internet, or social networks. One example of a **Directed Acyclic Graph** (DAG) is a prerequisite graph, where vertices are courses and edges point from a prerequisite of a course to the course itself.

In a prerequisite graph, the longest chain is smallest number of semesters required for the student to graduate. Thus, the student should choose to postpone other courses that are *not* on this **critical path** so that they do not delay graduation.

Another example is the internet, where the web servers must find the shortest path to get information (web pages, videos, images) to your browser. This would minimize the amount of delay between sending the request and receiving the information.

There are many examples of how graphs can be used, such as social network or the Bacon number. The key takeaway is that because there are so many applications, designing efficient (both time and space) algorithms is very important to solving problems that would otherwise be very time-consuming.

Representing Graphs

Adjacency Matrix
$n \cdot n$ matrix M where $M[i, j] = 1$ means there is an edge between vertex i and j.
Note: M^2 represents connection between vertices separated by length 2.

Trees

Definition: Graph with no cycles.

Examples of trees include computer file systems, tournament brackets, dictionary search trees, etc.

Tournament Example
Problem: In an elimination tournament bracket, how many games are played before the winner emerges?
Answer: There must be $n - 1$ games because each game eliminates one team, and the last team can not be eliminated (it is the winner).
Note: this is the *easier* way to solve the problem, you can also solve the problem using geometric series or by noticing that a tree has $n - 1$ edges.

Day 3: 1/26

Graphs

Graph Terminology

- **Graphs** are composed of a set of vertices and a set of edges.
- The **in-degree** of v is the number of edges ending at v
- The **out-degree** of v is the number of edges starting at v

Note that the sum of all in-degrees is the same as the sum of all out-degrees.

Directed Multigraph have multiple directed edges for the same pair of vertices. The **multiplicity** of edge (u, v) is the number of directed edges from u to v.

Walk is an alternating sequence of vertices and edges, basically a path from one vertex to another vertex where each consecutive pairs of vertices are connected by an edge. Vertices and edges can appear more than once, and the **length** of the walk is the number of edges in the walk.

Paths are a *subset* of walks, where no vertex is visited more than once.

Closed walks start and end at the same vertex

Cycle is a closed walk where all vertices are distinct except for the first and last vertex.

Theorem

A shortest walk from one vertex to another is a path.

Proof by contradiction: if there is a cycle in the path, we can remove the cycle and obtain a shorter walk from the source to the destination vertex.

Articulation point is a vertex where removing it would split the graph into two disconnected components.

Cut-set is the minimum subset of edge that you should remove to split the graph into two.

Flow networks: graph where edge weights represent the flow from one vertex to another

Handshaking Theorem

If $G = (V, E)$ is a undirected graph with m edges, then

$$2m = \sum \deg(v)$$

Degree of Vertices Theorem

An undirected graph has an even number of vertices of odd degree.

Proof 1:

Whenever we're adding an addition vertex, we can add 0 to n edges to the existing graph (where n is the number of vertices in the original graph). If we add an edge to an old vertex, then the number of vertices with odd degree can be changed in any of the 3 ways:

1. $+2$ if the new vertex was even degree (now odd) and old vertex was also even degree (now odd)
2. 0 if either the new vertex was even (now odd) **or** the old vertex was even (now odd)
3. -2 if the new vertex was odd (now even) and old vertex was also odd (now even)

Thus, the number of odd vertices always increases by an even number. Since the number of odd vertices starts at 0 (when there are no vertices), the number of odd vertices is always even.

Proof 2:

From the **Handshaking Theorem**, we have the following equation:

$$2m = \sum \deg(v) = \sum_{v \in v_1} \deg(v) + \sum_{v \notin v_1} \deg(v)$$

Where v_1 is the set of vertices with odd degree.

Since the second term in the equation consists of any number of even degree, it has to be even as well. Since $2m$ (on the left side) is always even, we know that the middle term (sum of degrees of odd vertices) must be even. Since the degree for all of the vertices in v_1 is odd, the number of these vertices must be even for the product to be even.

Day 4: 1/28

Adjacency Matrix

Adjacency Matrix uses a $n \times n$ matrix, where the value at (i, j) represents if there is a connection between i and j in the graph.

1. If the graph is undirected, then the matrix is symmetric.
2. Multiplying the adjacency matrix by itself represents the connections between vertices that are 2 edges apart
3. To find a shortest path between two vertices, we can keep multiplying A by itself until (i, j) is non-zero, which must be the shortest path. Note that this has a $O(N^4)$ runtime.

4. Bellman-Ford finds shortest paths between all pairs of vertices

Directed Acyclic Graph (DAG)

DAG is a directed graph with no cycles, commonly used to represent scheduling constraints. For example, edge (u, v) could mean that task u must happen before v (which is transitive). If we can assign two threads for example, what is the best way to schedule the tasks so we can finish earliest?

Topological Sort: any total ordering that is consistent with the *partial ordering* in the DAG

Uniprocessor Scheduling: any top sort

Parallel Scheduling: tasks with no constraints between them can be scheduled in parallel

Critical Path: the longest chain in the DAG

Note that if the DAG is your course sequence, the critical path dictates the earliest you can graduate.

To route in a binary tree between two nodes x and y, we can find the **lowest common ancestor** z, and find the route from x to z and y to z, finally their sum will be the shortest path between x and y.