Contents

1 DIS 0B **3**

1.1 Intro . . . 3

1.1.1 Some CS70 advice . . . 3

1.2 Propositional Logic . . . 3

1.3 Proofs . . . 3

1.3.1 Direct proof . . . 4

1.3.2 Contraposition . . . 4

1.3.3 Contradiction . . . 4

1.3.4 Cases . . . 4

2 DIS 1A **5**

2.1 Induction . . . 5

2.1.1 (Weak) Induction . . . 5

2.1.2 Strengthening the Hypothesis . . . 5

2.1.3 Strong Induction . . . 5

2.1.4 Weak vs Strong . . . 5

3 DIS 1B **6**

3.1 Stable Matching . . . 6

3.1.1 The Propose and Reject Algorithm . . . 6

3.1.2 Stability . . . 6

3.1.3 Optimality . . . 6

3.1.4 Potpourri . . . 7

4 DIS 2A **8**

4.1 Graphs . . . 8

4.1.1 Notation . . . 8

4.1.2 Vocabulary . . . 8

4.1.3 The holy grail for graph proofs . . . 9

4.1.4 Relevant Potpourri . . . 9

5 DIS 2B **11**

5.1 Trees . . . 11

5.2 Planarity . . . 11

5.3 Coloring . . . 11

5.4 Hypercubes . . . 12

6 DIS 3A **13**

6.1 Modular Arithmetic . . . 13

7 DIS 3B **14**

7.1 Modular inverse 14

7.2 Chinese Remainder Theorem (CRT) 14

8 DIS 4A **16**

8.1 Fermat's Little Theorem 16

8.2 RSA 16

8.2.1 The algorithm 16

8.3 Why does RSA work? 16

1 DIS 0B

1.1 Intro

- My OH is Monday 1-2 and Tuesday 3-4 in Cory 212.
- Email is first.last@
- 3rd year cs + math major
- hobbies?

1.1.1 Some CS70 advice

- Goal: enhance problem solving techniques/approach
- Don't fall behind on content, catching up will not be fun
- problems, problems, more problems
- Ask lots of questions (imperative for strong foundation)
- Don't stress, we're in this ride together

1.2 Propositional Logic

Relevant notation:

- $\wedge$ = and
- $\vee$ = or
- $\neg$ = not
- $\implies$ = implies
- $\exists$ = there exists
- $\forall$ = forall
- $\mathbb{N}$ = natural numbers $\{0, 1, \ldots\}$
- $a|b = a$ divides b

$P \implies Q$ is an example of an implication. We can read this as "If P, then Q." An implication is false only when P is true and Q is false. If P is false, the implication is vacuously true.

Definition 1.1 (Contrapositive)
If $P \implies Q$ is an implication, then the implication $\neg Q \implies \neg P$ is known as the **contrapositve**.

An important identity is that $P \implies Q \equiv \neg Q \implies \neg P$.

1.3 Proofs

Induction will be in its own section.

Different methods.

1.3.1 Direct proof

Want to show $P \implies Q$ by assuming P and logically concluding Q.

1.3.2 Contraposition

Want to show $P \implies Q$ by equivalently proving $\neg Q \implies \neg P$.

1.3.3 Contradiction

Want to show P. We do this by assuming $\neg P$ and concluding $R \wedge \neg R$.

Why? Idea is that if we can show the implication $\neg P \implies (R \wedge \neg R)$ is True, this is the same as showing $\neg P \implies F$ is True. The contraposition gives $T \implies P$.

1.3.4 Cases

Break up a problem into multiple cases i.e. odd vs even.

2 DIS 1A

2.1 Induction

Goal of induction is to show $\forall n P(n)$.

2.1.1 (Weak) Induction

- Prove $P(0)$ is true (or relevant base cases), then $\forall n \in \mathbb{N}\,(P(n) \implies P(n+1))$.
- Induction dominoes analogy!
- Sometimes you might have multiple base cases (Problem about $4x + 5y$ in Notes 3)

2.1.2 Strengthening the Hypothesis

Sometimes proving $P(n) \implies P(n+1)$ is not straightforward with induction. In such a scenario, we can try to introduce a (stronger) statement $Q(n)$. We want to construct Q such that $Q(n) \implies P(n)$. Inducting on Q proves P.

2.1.3 Strong Induction

- Prove $P(0)$ is true (or relevant base cases), then $\forall n\,((P(0) \wedge P(1) \wedge \cdots \wedge P(n)) \implies P(n+1))$.
- Dominoes analogy, but emphasis on the difference between weak and strong induction (assuming middle domino works vs everything from start to middle).

2.1.4 Weak vs Strong

A common point of confusion is when one should use strong induction in lieu of weak induction. Strong induction **always** works whenever weak induction works. However, there may be scenarios in which the induction hypothesis to prove $n = k + 1$ requires more information than just $n = k$. A scenario like this requires strong induction.

3 DIS 1B

3.1 Stable Matching

Cool application of induction.

3.1.1 The Propose and Reject Algorithm

Suppose jobs proposes to candidates.

- both jobs and candidates have a list of preferences
- every day a job that doesn't have a deal with a candidate will propose to the next best candidate on its preference list
- every candidate will tentatively "waitlist" the offer from the job (put it on a string)
- if a candidate has multiple offers, they will choose the one they prefer the most
- the algorithm ends when every candidate has a job on their "waitlist" (all these WLs becomes acceptances)

(walk through q1 of dis as a class to visualize this)

3.1.2 Stability

Definition 3.1 (Rogue Couple)
A job-candidate pair (J,C) is denoted as a **rogue couple** if they prefer each other over their final assignment in a stable matching instance.

Definition 3.2 (Unstable)
A matching that has at least one rogue couple is considered **unstable**.

Conversely, a **stable** matching is one that has no rogue couples.

Some tricky vocab stuff like stable matching instance.

Lemma 1 (Improvement) *If a candidate has a job offer, then they will always have an offer from a job at least as good as the one they have right now.* □

Matchings produced by the algorithm are always **stable**.

3.1.3 Optimality

The propose and reject algorithm is proposer *optimal* and receiver *pessimal*.

Definition 3.3 (optimal)
A pairing is optimal for a group if each entity is paired with who it most prefers while maintaining stability.

Can be thought of a (well that's the best I could do) analogy.

Definition 3.4 (pessimal)
A pairing is pessimal for a group if each entity is paired with who it least prefers while maintaining stability.

Can be thought of a (well it can't get worse than this) analogy.

3.1.4 Potpourri

It is possible that there exists a stable matching instance that is neither job optimal nor candidate pessimal.

Consider the following preferences

Jobs	Preferences
A	$1 > 2 > 3$
B	$2 > 3 > 1$
C	$3 > 1 > 2$

Candidates	Preferences
1	$B > C > A$
2	$C > A > B$
3	$A > B > C$

The matching above can generate (at least) 3 stable matching instances

$$S = \{(A,1), (B,2), (C,3)\}$$
$$T = \{(A,3), (B,1), (C,2)\}$$
$$U = \{(A,2), (B,3), (C,1)\}.$$

We see

- S is job-optimal/candidate-pessimal (result of running propose and reject with jobs proposing to candidates)
- T is candidate-optimal/job-pessimal (result of running propose and reject with candidates proposing to jobs)
- U is neither optimal nor pessimal for both candidates and jobs (S and T) corroborate that.

Also some other important facts that can be seen (from discussion worksheet questions):

- There is at least one candidate that will receive only one proposal (that too on the last day)
- We can upper bound the number of days needed by P&R algorithm to $(n-1)^2 + 1 = n^2 - 2n + 2$ (think about why)
- As a consequence of above, we can upper bound the number of rejections needed by P&R algorithm to $(n-1)^2 = n^2 - 2n + 1$ rejections.

4 DIS 2A

4.1 Graphs

4.1.1 Notation

- V denotes set of vertices (points)
- E denotes set of edges (lines)
- $|V|$ denotes size of set of vertices i.e number of vertices; $|E|$ similarly
- Graph G with vertices V and edges E is denoted $G = (V, E)$.

4.1.2 Vocabulary

Definition 4.1 (Path)
A **path** is a sequence of edges. In CS70, we assume a path is *simple* which means no repeated vertices.

Definition 4.2 (Cycle)
A **cycle** is a simple path that starts and ends at the same vertex.

Definition 4.3 (Walk)
A **walk** is any arbitrary connected sequence of edges.

Definition 4.4 (Tour)
A **tour** is a walk that starts and end at the same vertex.

Definition 4.5 (Connected)
A graph is **connected** if there exists a path between any two distinct vertices.

Definition 4.6 (Eulerian Walk)
An **Eulerian walk** is a walk covering all edges without repeating any.

Definition 4.7 (Eulerian Tour)
An **Eulerian tour** is an Eulerian walk that starts and ends at the same vertex.

To summarize,

	no repeated vertices	no repeated edges	start = end	all edges	all vertices
Walk					
Path	✓	✓			
Tour			✓		
Cycle	✓*	✓	✓		
Eulerian Walk		✓		✓	
Eulerian Tour		✓	✓	✓	
Hamiltonian Tour	✓	✓	✓		✓

(*except for start and end vertices)

Theorem 4.1 (Euler's Theorem)
An undirected graph G has an Eulerian tour iff G is connected and all its vertices have even degree.

The requires condition for an Eulerian walk is that we have exactly 2 vertices of odd degree. (Of course, the case of 0 odd vertices trivially works since we claim from Euler's Theorem that we can find an Eulerian tour which is a stronger statement than an Eulerian walk)

Definition 4.8 (Bipartite)
A graph is considered bipartite if V can be partitioned into two sets L and R where $V = L \cup R$ such that there are no edges between vertices in L and no edges between vertices in R.

4.1.3 The holy grail for graph proofs

Induct, induct, induct, and induct.

- Think about what you want to induct on (edges or vertices???)
- Base case (read the problem carefully!)
- Prove for n by going from $n \to n-1 \to I.H. \to n$.
  - **DO NOT** go from $n-1 \to n$ directly.
  - Why? Build-up error!
  - Good example of build-up error when trying to prove "if every vertex of a graph has degree at least 2, then there exists a cycle of length 3." Any attempt at induction will give us a false proof but we cannot make square from triangle!
  - It's also a logistical nightmare lol (in the times it might accidentally work). Try generating all 5-vertex trees from all 4-vertex trees yikes.

4.1.4 Relevant Potpourri

Some other relevant information.

Definition 4.9 (Degree)
The **degree** of a vertex v denoted $\deg(v)$ is defined to be the number of incident edges to v.

Lemma 2 (Handshake)

$$\sum_{v \in V} \deg(v) = 2|E|.$$

□

The idea of a degree (with no adjective) is only well-defined for undirected graphs. We see for directed graphs it's a little funky; we need to introduce the concept of indegree and outdegree.

In a directed graph, the number of outgoing edges equals the number of ingoing edges.

We will discuss trees, planarity, coloring, and hypercubes in the next discussion.

5 DIS 2B

5.1 Trees

A graph $G = (V,E)$ is a Tree if any of the statements below is true. TFAE (The following are equivalent):

- G is connected and has no cycles
- G is connected and $|E| = |V| - 1$
- G is connected and removing a single edge disconnects G
- G has no cycles and adding a single edge creates a cycle

Definition 5.1
A leaf is a node of degree 1.

A consequence of above is that every tree has at least 2 leaves.

5.2 Planarity

Definition 5.2 (planar)
A graph is **planar** if it can be drawn without any edge crossings.

Theorem 5.1 (Euler)
For every connected planar graph, $f + v = e + 2$.

Corollary 1 *If a graph is planar, then $e \leq 3v - 6$.* □

Theorem 5.2 (Kuratowski)
A graph is non-planar iff it contains K_5 or $K_{3,3}$.

(draw the two above graphs on the board)

The notation K_x denotes a complete graph with x vertices.

Definition 5.3 (complete graph)
A **complete graph** is a graph where all possible edges exist. Formally, in graph $G = (V,E)$, for any distinct $u,v \in V$, then $\{u,v\} \in E$.

5.3 Coloring

Two types: edge and vertex

- edge: color edges so that no two adjacent edges have the same color
- vertices: color vertices so that no two adjacent vertices have the same color

Theorem 5.3 (4 color theorem)
If a graph is planar, then it can be colored with 4 (or less) colors.

5.4 Hypercubes

A hypercube of dimension n is a graph whose vertices are bitstrings of length n. An edge between two vertices exists iff the two vertices differ at exactly 1 bit.

(draw $n = 1, 2, 3$ on the board)

We can see that $|V| = 2^n$ and $|E| = n2^{n-1}$.

Give some motivation on induction on hypercubes.

6 DIS 3A

Definition 6.1 (Greatest Common Divisor)
The **greatest common divisor** (gcd) of two integers a, b is the greatest $d \in \mathbb{Z}$ such that $d|a$ and $d|b$.

How does one efficiently calculate the GCD?

Algorithm 6.1 (Euclidean Algorithm)

```
function GCD(a,b)
    if b = 0 then
        return a
    return GCD(b, a mod b)
```

6.1 Modular Arithmetic

The relevant notation we'll be using for this section is expressions of the form

$$a \equiv b \pmod{x}$$

reads "a is equivalent to b mod x". It means that the remainder of a when divided by x equals the remainder of b when divided by x.

An important identity is that

$$a \equiv b \pmod{x} \iff (\exists k \in \mathbb{Z})(a = b + kx).$$

Talk about the "clock analogy".

Example 6.1
We can see a display of some of the properties:

- Addition: $7 + 4 \equiv 1 \pmod{5}$
- Subtraction: $7 - 4 \equiv 1 \pmod{2}$
- Multiplication: $2 \cdot 3 \equiv 0 \pmod{6}$.
- Division??

In modular arithmetic, division is not well-defined. The opposite of multiplication is multiplying by the modular inverse.

Definition 6.2 (modular inverse)
The value a is the **modular inverse** of x with respect to mod m if

$$ax \equiv 1 \pmod{m}.$$

Does an inverse always exist? No.

Theorem 6.1
Let x and m be positive integers. Then $x^{-1} \pmod{m}$ exists and is unique only if $\gcd(x,m) = 1$.

7 DIS 3B

More mods.

7.1 Modular inverse

Lemma 3 (Bézout) *For integers* x, y *such that* $\gcd(x,y) = d$*, there exist integers* a *and* b *that obey*

$$ax + by = d.$$

□

We care about the case when $\gcd(x,y) = d = 1$.

Why? This is how we can find the modular inverse.

If $ax + by = 1$, taking $\bmod x$ gives us

$$by \equiv 1 \pmod{x} \implies b \equiv y^{-1} \pmod{x}.$$

Similarly, taking $\bmod y$ gives us

$$ax \equiv 1 \pmod{y} \implies a \equiv x^{-1} \pmod{y}.$$

Takeaway: the values of a and b we will solve for (Q1 on discussion) give us the inverse of x with respect to y and vice versa.

7.2 Chinese Remainder Theorem (CRT)

Theorem 7.1 (CRT)
For pairwise relatively prime integers $m_1, m_2, \ldots, m_n$, the modular system

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_n \pmod{m_n} \end{aligned}$$

has a unique solution $x \pmod{m_1 m_2 \cdots m_n}$.
To clarify, the term pairwise relatively prime means for any distinct i, j, it follows $\gcd(m_i, m_j) = 1$.

How do we solve the system above? Discussion Q2...

...or we can solve them a faster way (not taught in the course lol)

Example 7.1
Suppose we take the first two systems from Q2 on discussion.

$$\begin{aligned} x &\equiv 1 \pmod{3} \\ x &\equiv 3 \pmod{7}. \end{aligned}$$

Since $\gcd(3,7) = 1$, CRT tells us x has a unique solution mod 21. The first equation tells us there exists some integer k such that $x = 1 + 3k$. Plugging this into the second equation we have

$$1 + 3k \equiv 3 \pmod 7 \implies k \equiv 3 \pmod 7.$$

Plugging in $k = 3$ gives $x \equiv 10 \pmod{21}$.

If we wanted to solve entirety of Q2 this way, we then apply the same trick above to the systems

$$\begin{aligned} x &\equiv 10 \pmod{21} \\ x &\equiv 4 \pmod{11}. \end{aligned}$$

8 DIS 4A

8.1 Fermat's Little Theorem

A relevant theorem in modular arithmetic that will help us with RSA is Fermat's Little Theorem (FLT).

Theorem 8.1 (Fermat's Little Theorem (FLT))
For prime p and $a \in \{1, 2, \ldots, p-1\}$, it follows

$$a^p \equiv a \pmod{p}.$$

Special case, if a is not divisible by p, then

$$a^{p-1} \equiv 1 \pmod{p}.$$

8.2 RSA

Objective: Alice transfers info to Bob without Eve cracking it.

8.2.1 The algorithm

Here's a detailed outline of how the scheme works for RSA with 2 primes:

1. Entire world knows about a public key (N,e) where $N = pq$ for primes p and q such that $\gcd(e, (p-1)(q-1)) = 1$.
2. Alice and Bob meet in private, and Alice tells Bob what p and q are.
3. On his own time, Bob computes $(p-1)(q-1)$ and then calculates
$$d = e^{-1} \pmod{(p-1)(q-1)}.$$
(Think about why we know such a d must exist)
4. To encrypt her message x, Alice sends $E(x)$ to Bob where
$$E(x) = x^e \pmod{N}.$$
5. To decrypt the message received y, Bob calculate $D(y)$ where
$$D(y) = y^d \pmod{N}.$$
High level idea of why this works:
$$\begin{aligned} D(E(x)) &= D(x^e) \pmod{N} \\ &= x^{ed} \pmod{N} \\ &= x \pmod{N}. \end{aligned}$$
More detailed proof by cases in page 3? of Note 7.

8.3 Why does RSA work?

- N is too large to brute force solve x where $y = x^e \pmod{N}$.
- N is too large to factor into $p \cdot q$. Factorization is an intractable problem!