Contents

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
Walk				
Path	✓	✓		
Tour			✓	
Cycle	✓*	✓	✓	
Eulerian Walk		✓		✓
Eulerian 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.

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