CSE 331 Final Exam Preparation

This is in no way a substitute for exam preperation, mearly a compilation of all the key talking points.

Chandra Neppalli

May 3, 2021

Counter Example

Best proof to use to *disprove* universally true propositions.

Counter Example

Best proof to use to *disprove* universally true propositions.

Ex: Every day is a Wednesday, where a counter example would be Monday is not Wednesday.

Contradiction

Best proof to use if you want to assert something is true.

Contradiction

Best proof to use if you want to assert something is true.

Assume what you want to prove is false, then show this leads to a contradiction.

Contradiction

Best proof to use if you want to assert something is true.

Assume what you want to prove is false, then show this leads to a contradiction.

Therefore, the original assumption has to be true.

Contraposition

Best proof for proving causality. Define two propositions E and F.

Contraposition

Best proof for proving causality. Define two propositions E and F.

If you want to prove that $E \rightarrow F$, it might be more doable to prove $\neg F \rightarrow \neg E$, as they are both logically equivalent.

Contraposition

Best proof for proving causality. Define two propositions E and F.

If you want to prove that $E \to F$, it might be more doable to prove $\neg F \to \neg E$, as they are both logically equivalent.

This is especially useful if the **scope** of F is smaller than the scope of E.

Direct Proof

If the proof is *simple*, consider directly proving it.

Direct Proof

If the proof is *simple*, consider directly proving it.

Remember though, that you must maintain $W.L.O.G$, that your proof can never be too specific and must be arbitrary.

Proof by Induction

Proof by Induction is a really nice proof technique when you reduce your proof to a known correct base case.

Proof by Induction

Proof by Induction is a really nice proof technique when you reduce your proof to a known correct base case.

If proof needs to be correct for all numbers $\in \mathbb{N}$, and each step is dependant on the previous step, then *every* step can be reduced to a definitive base case that is easy to directly prove.

Extra: Progress Measure

This is useful for proving an algorithm with a loop terminates.

Extra: Progress Measure

This is useful for proving an algorithm with a loop terminates.

Let $P(i)$ denote an integer such that:

Extra: Progress Measure

This is useful for proving an algorithm with a loop terminates.

Let $P(i)$ denote an integer such that:

- $P(0) = I$

Extra: Progress Measure

This is useful for proving an algorithm with a loop terminates.

Let $P(i)$ denote an integer such that:

- $P(0) = l$
- $P(i)$ is an accumulator. This means that $P(i+1) > P(i)$

Extra: Progress Measure

This is useful for proving an algorithm with a loop terminates.

Let $P(i)$ denote an integer such that:

- $P(0) = l$
- $P(i)$ is an accumulator. This means that $P(i + 1) > P(i)$
- $\forall i, \quad P(i) \leq k$

Extra: Progress Measure

This is useful for proving an algorithm with a loop terminates.

Let $P(i)$ denote an integer such that:

- $P(0) = l$
- $P(i)$ is an accumulator. This means that $P(i+1) > P(i)$
- $\forall i, \quad P(i) \leq k$

From these 3 properties, the number of iterations is bounded by $k - l + 1$

Extra: Progress Measure

This is useful for proving an algorithm with a loop terminates.

Let $P(i)$ denote an integer such that:

- $P(0) = l$
- $P(i)$ is an accumulator. This means that $P(i+1) > P(i)$
- $\forall i, \quad P(i) \leq k$

From these 3 properties, the number of iterations is bounded by $k - l + 1$

Note: This isn't a runtime analysis, rather a proof that the algorithm terminates.

Greedy Stays Ahead

This technique is used to prove that a greedy algorithm returns an optimal solution.

Greedy Stays Ahead

This technique is used to prove that a greedy algorithm returns an optimal solution.

At every step of a greedy algorithm, it will stay *at least* as far as the optimal solution at that step.

Greedy Stays Ahead

This technique is used to prove that a greedy algorithm returns an optimal solution.

At every step of a greedy algorithm, it will stay *at least* as far as the optimal solution at that step.

HW4 "Attack on Alarms" and Interval Scheduling are examples of problems with greedy solutions.

Introduction

Let's say there are two groups: Group A and Group B.

Introduction

Let's say there are two groups: Group A and Group B.

How do we generate a **stable** matching between each member of the two groups efficiently?

Introduction

Let's say there are two groups: Group A and Group B.

How do we generate a **stable** matching between each member of the two groups efficiently?

Moreover, what is a **stable** matching?

Perfect Matchings

A **perfect matching** is a bijective matching between A and B.

Perfect Matchings

A **perfect matching** is a bijective matching between A and B.

Every member in group A is matched with exactly one member in group B.

Perfect Matchings

A **perfect matching** is a bijective matching between A and B.

Every member in group A is matched with exactly one member in group B.

Conversely, every member in group B is matched with **exactly** one member in group A.

Perfect Matchings

A **perfect matching** is a bijective matching between A and B.

Every member in group A is matched with exactly one member in group B.

Conversely, every member in group B is matched with **exactly** one member in group A.

With n members in each group, there are $n!$ perfect matchings.

Instability

For a particular matching, define a member m from group A and n from group B such that (m, n) is not in the matching.

Instability

For a particular matching, define a member m from group A and n from group B such that (m, n) is not in the matching.

If m prefers n over their current matching **and** n prefers m over their current matching, the entire match is an instability.

Stable Matching

A stable matching **is** a perfect matching with no instability.

Stable Matching

A stable matching **is** a perfect matching with no instability.

It therefore follows that the number of stable matchings is *at least* the number of perfect matchings, or n!

Stable Matching

A stable matching **is** a perfect matching with no instability.

It therefore follows that the number of stable matchings is *at least* the number of perfect matchings, or $n!$

The Gale Shapely Algorithm is an $O(n^3)$ time algorithm that can output a stable matching.

Stable Matching

A stable matching **is** a perfect matching with no instability.

It therefore follows that the number of stable matchings is *at least* the number of perfect matchings, or $n!$

The Gale Shapely Algorithm is an $O(n^3)$ time algorithm that can output a stable matching.

With the right data structures, the runtime can be reduced to $O(n^2)$.

Stable Matching

A stable matching **is** a perfect matching with no instability.

It therefore follows that the number of stable matchings is *at least* the number of perfect matchings, or n!

The Gale Shapely Algorithm is an $O(n^3)$ time algorithm that can output a stable matching.

With the right data structures, the runtime can be reduced to $O(n^2)$.

Even though the runtime isn't linear, because the input size is $2n^2 \rightarrow \Theta(n^2)$[1], the runtime **with respect** to the input size is $O(N)$, or linear time.

[1] This comes from n Group A members and n Group B members with their $2n$ preference lists

Stable Matching

Code:

```
Initially all m ∈ M and w ∈ W are free
While there is a man m who is free and hasn't proposed to every
woman
      Choose such a man m
      Let w be the highest-ranked woman in m's
            preference list to whom m has not yet proposed
      If w is free then
            (m, w) become engaged
      Else w is currently engaged to m'
            If w prefers m' to m then
                  m remains free
            If w prefers m to m'
                  (m, w) become engaged
                  m' becomes free
            Endif
      Endif
EndWhile
Return the set S of engaged pairs.
```