Lecture 35

CSE 331

Last two weeks will be rough…

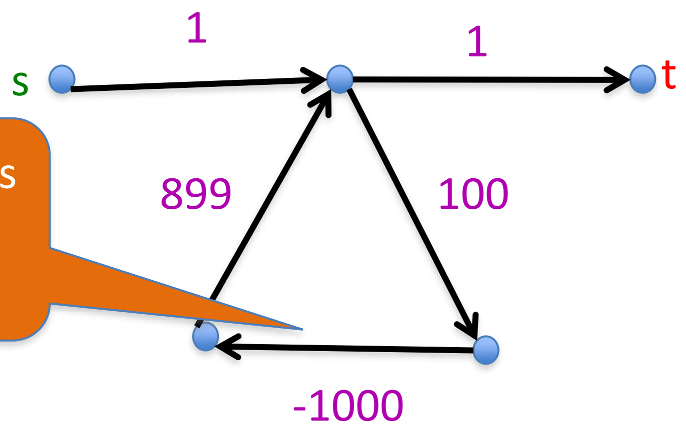
Week 14 Mon, May 1	The P vs. NP problem ▶F22 ▶F21 ▶S21 ▶S20	[KT, Sec 8.1]
Wed, May 3	More on reductions ▶F22 ▶F21 ▶S21 ▶S20	[KT, Sec 8.2]
Fri, May 5	The SAT problem ▶F22 ▶F21 ▶S21 ▶S20	[KT, Sec. 8.3, 8.4] (**HW 8 out**) (**Project (Problem 3** `Coding` **) in**)
Week 15 Mon, May 8	NP-Completeness ▶F22 ▶F21 ▶S21 ▶S20	[KT, Sec 8.7] (**Quiz 2**) (**Project (Problem 3** `Reflection`**) in**)
Wed, May 10	*k*-coloring problem ▶F22 ▶F21 ▶S21 ▶S20	[KT, Sec 8.7]
Fri, May 12	*k*-coloring is NP-complete ▶F22 ▶F21 ▶S21 ▶S20	(**HW 8 in**) (**Project (Problems 4 & 5** `Coding` **) in**)
Mon, May 15		(**Project (Problems 4 & 5** `Reflection`**) in**) (**Project Survey in**)
Wed, May 17	**Final Exam**	(**11:45am-2:45pm**)

Shortest Path Problem

Input: (Directed) Graph G=(V,E) and for every edge e has a cost c_e (can be <0)

t in V

Output: Shortest path from every s to t



Shortest path has cost negative infinity

Assume that G has no negative cycle

The recurrence

$OPT(u,i)$ = shortest path from u to t with at most i edges

$$OPT(u,i) = \min \{ OPT(u,i-1), \min_{(u,w) \text{ in } E} \{ c_{u,w} + OPT(w, i-1)\} \}$$

Some consequences

OPT(u,i) = cost of shortest path from u to t with at most i edges

$$OPT(u,i) = \min \left\{ OPT(u, i-1), \min_{(u,w) \text{ in } E} \{ c_{u,w} + OPT(w,i-1) \} \right\}$$

OPT(u,n-1) is shortest path cost between u and t

Can compute the shortest path between s and t given all OPT(u,i) values

Bellman-Ford Algorithm

Runs in $O(n(m+n))$ time

Only needs $O(n)$ additional space

Reading Assignment

Sec 6.8 of [KT]

Longest path problem

Given G, does there exist a simple path of length n-1 ?

Longest vs Shortest Paths

Two sides of the “same” coin

Shortest Path problem

Can be solved by a polynomial time algorithm

Is there a longest path of length n-1?

Given a path can verify in polynomial time if the answer is yes

Poly time algo for longest path?

Clay Mathematics Institute

Dedicated to increasing and disseminating mathematical knowledge

HOME | ABOUT CMI | PROGRAMS | NEWS & EVENTS | AWARDS | SCHOLARS | PUBLICATIONS

First Clay Mathematics Institute Millennium Prize Announced

Prize for Resolution of the Poincaré Conjecture Awarded to Dr. Grigoriy Perelman

March 18, 2010. The Clay Mathematics Institute (CMI) announces today that Dr.

- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- P vs NP
- Poincaré Conjecture
- Riemann Hypothesis

P vs NP question

CMI

ABOUT PROGRAMS PEOPLE MILLENNIUM PROBLEMS PUBLICATIONS EVENTS NEWS

Millennium Problems

Yang–Mills and Mass Gap

Experiment and computer simulations suggest the existence of a "mass gap" in the solution to the quantum versions of the Yang-Mills equations. But no proof of this property is known.

Riemann Hypothesis

The prime number theorem determines the average distribution of the primes. The Riemann hypothesis tells us about the deviation from the average. Formulated in Riemann's 1859 paper, it asserts that all the 'non-obvious' zeros of the zeta function are complex numbers with real part 1/2.

P vs NP Problem

If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem? This is the essence of the P vs NP question. Typical of the NP problems is that of the Hamiltonian Path Problem: given N cities to visit, how can one do this without visiting a city twice? If you give me a solution, I can easily check that it is correct. But I cannot so easily find a solution.

P vs NP question

P: problems that can be solved by poly time algorithms

Is P=NP?

NP: problems that have polynomial time verifiable witness to optimal solution

Alternate NP definition: Guess witness and verify!

Proving P ≠ NP

Pick any one problem in NP and show it cannot be solved in poly time

Pretty much all known
proof techniques
provably will not work

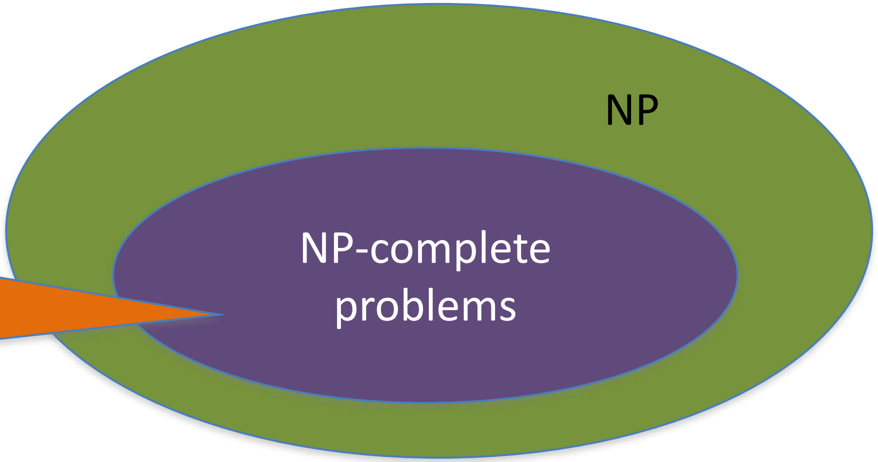
Proving P = NP

Will make cryptography collapse

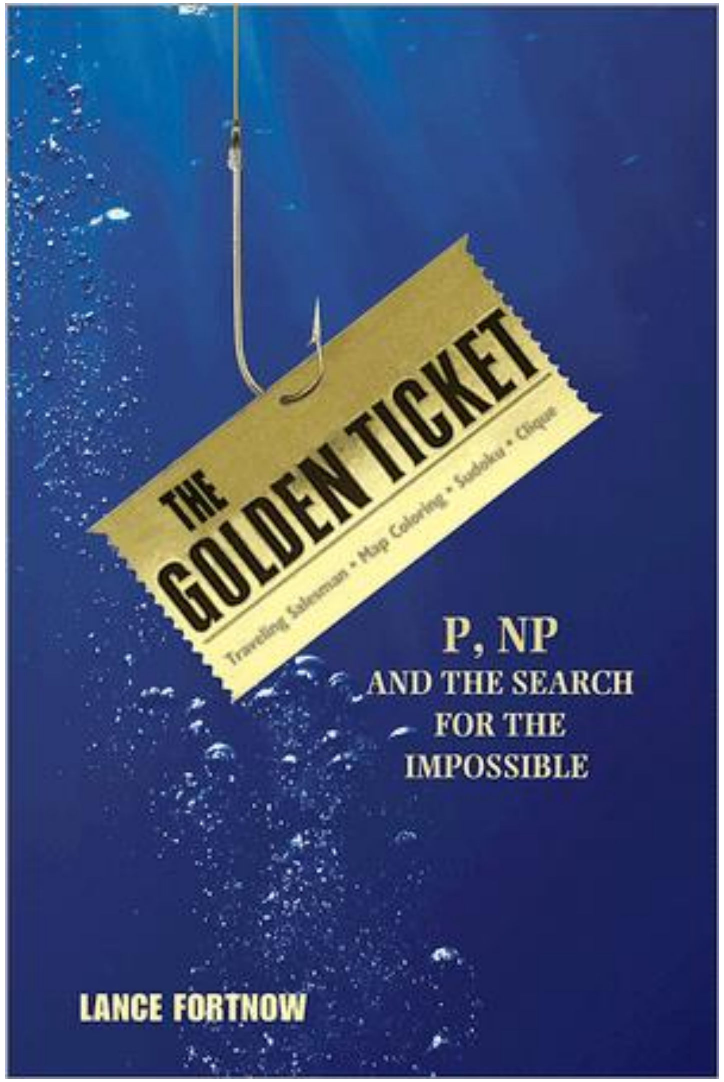
Compute the encryption key!

Prove that all problems in NP can be solved by polynomial time algorithms

Solving any ONE problem in here in poly time will prove P=NP!



A book on P vs. NP



The course so far…

https://www.teepublic.com/sticker/1100935-obama-yes-we-can

The rest of the course…

https://www.madduckposters.com/products/megamind-no-you-cant?variant=13565168320556

No, you can't– what does it mean?

NO algorithm will be able to solve a problem in polynomial time

Still for worst-case runtime

No, you can't take- 1

Adversarial Lower Bounds

Some notes on proving Ω lower bound on runtime of *all* algorithms that solve a given problem.

The setup

We have seen earlier how we can argue an Ω lower bound on the run time of a *specific* algorithm. In this page, we will aim higher

The main aim

Given a problem, prove an Ω lower bound on the runtime on *any* (correct) algorithm that solves the problem.

What is the best lower bound you can prove?

$\Omega(N)$

No, you can't take- 2

Lower bounds based on output size

Lower Bound based on Output Size

Any algorithm that for inputs of size N has a worst-case output size of $f(N)$ needs to have a runtime of $\Omega(f(N))$ (since it has to output all the $f(N)$ elements of the output in the worst-case).

Question 2 (Listing Triangles) [25 points]

The Problem

A `triangle` in a graph $G = (V, E)$ is a 3-cycle; i.e. a set of three vertices $\{u, v, w\}$ such that $(u, v), (v, w), (u, w) \in E$. (Note that G is undirected.) In this problem you will design a series of algorithms that given a *connected* graph G as input, lists **all** the triangles in G. (It is fine to list one triangle more than once.) We call this the `triangle listing problem` (duh!). You can assume that as input you are given G in *both* the adjacency matrix and adjacency list format. *For this problem you can also assume that G is connected.*

2. Present an $O(m^{3/2})$ algorithm to solve the triangle listing problem.

Exists graphs with $m^{3/2}$ triangles

No, you can't take- 2

Lower bounds based on output size

On input n, output 2^n many ones

Every algo takes (doubly) exponential time

But at heart problem is "trivial"

From now on, output size is always O(N) and could even be binary.

No, you can’t take -3

Argue that a given problem is AS HARD AS

a “known” hard problem

How can we argue something like this?

Reductions

So far: “Yes, we can” reductions

https://www.teepublic.com/sticker/1100935-obama-yes-we-can

Reduce Y to X where X is "easy"

Reduction

Reduction are to algorithms what using libraries are to programming. You might not have seen reduction formally before but it is an important tool that you will need in CSE 331.

Background

This is a trick that you might not have seen explicitly before. However, this is one trick that you have used many times: it is one of the pillars of computer science. In a nutshell, reduction is a process where you change the problem you want to solve to a problem that you already know how to solve and then use the known solution. Let us begin with a concrete non-proof examples.

Example of a Reduction

We begin with an elephant joke. There are many variants of this joke. The following one is adapted from this one.

- `Question 1` How do you stop a rampaging blue elephant?
- `Answer 1` You shoot it with a blue-elephant tranquilizer gun.

- `Question 2` How do you stop a rampaging red elephant?
- `Answer 2` You hold the red elephant's trunk till it turns blue. Then apply Answer 1.

- `Question 3` How do you stop a rampaging yellow elephant?
- `Answer 3` Make sure you run faster than the elephant long enough so that it turns red. Then Apply Answer 2.

In the above both `Answers 2` and `3` are reductions. For example, in `Answer 2`, you do some work (in this case holding the elephant's trunk: in this course this work will be a

“Yes, we can” reductions (Example)

Question 2 (Big G is in town) [25 points]

The Problem

The `Big G` company in the bay area decides it has not been doing enough to hire CSE grads from UB so it decides to do an exclusive recruitment drive for UB students. The `Big G` decides to fly over n CSE majors from UB to the bay area during December for on-site interview on a single day. The company sets up m slots in the day and arranges for n `Big G` engineers to interview the n UB CSE majors. (You can and should assume that $m > n$.) The fabulous scheduling algorithms at `Big G`'s offices draw up a schedule for each of the n majors so that the following conditions are satisfied:

- Each CSE major talks with every `Big G` engineer exactly once;
- No two CSE majors meet the same `Big G` engineer in the same time slot; and
- No two `Big G` engineers meet the same CSE major in the same time slot.

In between the schedule being fixed and the CSE majors being flown over, the `Big G` engineers were very impressed with the CVs of the CSE majors (including, ahem, their performance in CSE 331) and decide that `Big G` should hire all of the n UB CSE majors. They decide as a group that it would make sense to assign each CSE major S to a `Big G` engineer E in such a way that after S meets E during her/his scheduled slot, all of S's and E's subsequent meetings are canceled. Given that this is December, the `Big G` engineers figure that taking the CSE majors out to the nice farmer market at the ferry building in San Francisco during a sunny December day would be a good way to entice the CSE majors to the bay area.

In other words, the goal for each engineer E and the major S who gets assigned to her/him, is to **truncate** both of their schedules after their meeting and cancel all subsequent meeting, so that no major gets **stood-up**. A major S is stood-up if when S arrives to meet with E on her/his truncated schedule and E has already left for the day with some other major S'.

Your goal in this problem is to design an algorithm that always finds a valid truncation of the original schedules so that no CSE major gets stood-up.

To help you get a grasp of the problem, consider the following example for $n = 2$ and $m = 4$. Let the majors be S_1 and S_2 and the `Big G` engineers be E_1 and E_2. Suppose S_1 and S_2's original schedules are as follows:

CSE Major	Slot 1	Slot 2	Slot 3	Slot 4
S_1	E_1	free	E_2	free

Overview of the reduction

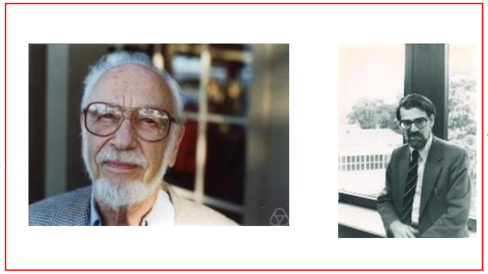
Question 2 (Big G is in town)

NRMP

National Resident Matching Program

CSE Major	Slot 1	Slot 2	Slot 3	Slot 4
S_1	E_1	free	E_2	free
S_2	free	E_1	free	E_2

CSE Major	Slot 1	Slot 2	Slot 3	Slot 4
S_1	E_1	free	E_2 (truncate here)	
S_2	free	E_1 (truncate here)		



Nothing special about GS algo

Question 2 (Big G is in town)

NRMP

National Resident Matching Program

CSE Major	Slot 1	Slot 2	Slot 3	Slot 4
S_1	E_1	free	E_2	free
S_2	free	E_1	free	E_2

CSE Major	Slot 1	Slot 2	Slot 3	Slot 4
S_1	E_1	free	E_2 (truncate here)	
S_2	free	E_1 (truncate here)		

ANY algo for stable matching problem works!

Another observation

Question 2 (Big G is in town)

NRMP
National Resident Matching Program

CSE Major	Slot 1	Slot 2	Slot 3	Slot 4
S_1	E_1	free	E_2	free
S_2	free	E_1	free	E_2

CSE Major	Slot 1	Slot 2	Slot 3	Slot 4
S_1	E_1	free	E_2 (truncate here)	
S_2	free	E_1 (truncate here)		

Poly time steps

ANY algo for stable matching problem works!

Poly time reductions

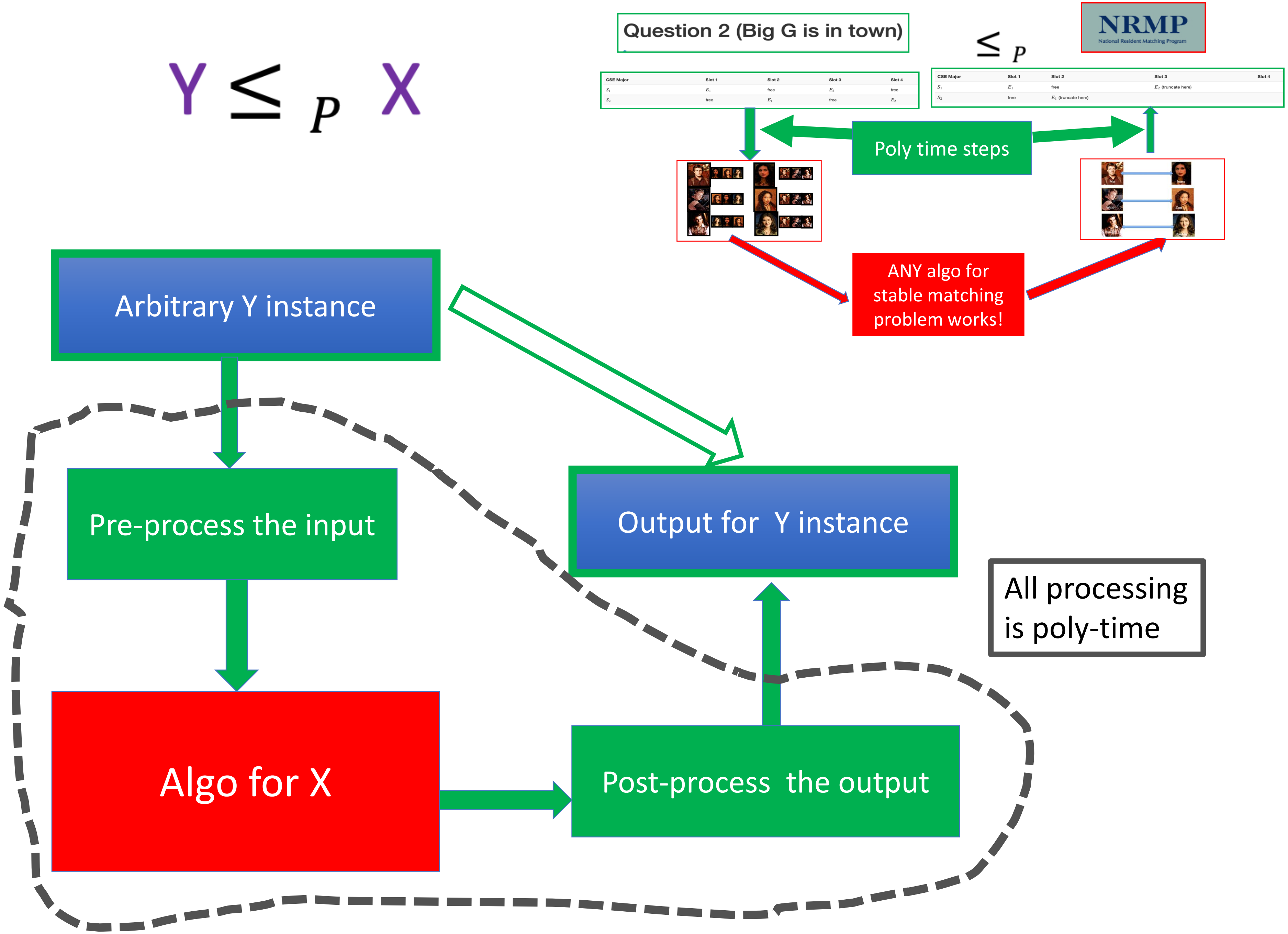
Question 2 (Big G is in town) $\leq_P$ NRMP National Resident Matching Program

CSE Major	Slot 1	Slot 2	Slot 3	Slot 4
S_1	E_1	free	E_2	free
S_2	free	E_1	free	E_2

CSE Major	Slot 1	Slot 2	Slot 3	Slot 4
S_1	E_1	free	E_2 (truncate here)	
S_2	free	E_1 (truncate here)		

Poly time steps

ANY algo for stable matching problem works!


Y ≤ P X
Question 2 (Big G is in town)
≤ P
NRMP
National Resident Matching Program
Poly time steps
ANY algo for stable matching problem works!
Arbitrary Y instance
Pre-process the input
Output for Y instance
All processing is poly-time
Algo for X
Post-process the output