Lecture 34

CSE 331

Apr 26, 2021

Last day to give feedback!

note @1037

Feedback on CSE 331

Hi All,

I'm asking for your feedback about 331 and I prepared a form with custom questions. Please do give feedback via this anonymous form: https://forms.gle/zjC6JRwvLBKG92iQ7

Filling in this form is **completely optional** and **anonymous**.

I would love feedback even if it is critical. Also, after a week or so, I'll post my response to the feedback from y'all, though I might disagree with you on certain things. So at the ver
are in CSE 331. And then we can agree to disagree :)

Note that this is NOT the UB's course evaluation form; the results will be used to improve the class this semester and in future offerings.

logistics

edit · good note | 0

Course evaluations

note @1053

Incentive for filling course evals

I will release some questions on the final exam **depending on the level of response on the official course evals.**

- If **>=85%** students submit the course evals, I will release **Q1.a (worth 2 pts)**
- If **>=90%** students submit the course evals, I will release **Q1.a** and **Q1.b (worth 4 pts)**

Some other relevant comments:

- The deadline is May 9th.
- I will post the up-to-date response rate in the comments section below every 3 days until May 9th.
- The % is based on current student registered (248): i.e. it does not include students who have resigned
- This must the link to the course evals: https://sunyub.smartevals.com/
  - But double check the email you might have received on this.

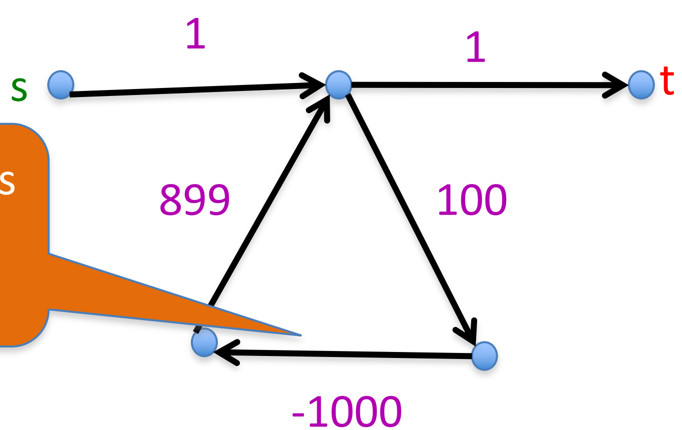
27% as of today at 12:34pm

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

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

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
to find optimal cost

Reading Assignment

Sec 6.8 of [KT]

Longest path problem

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

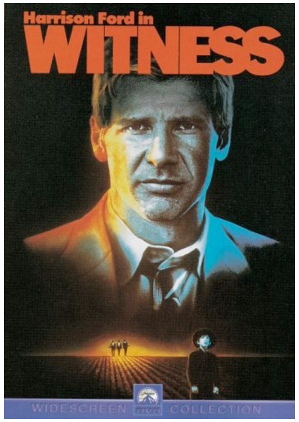
Longest vs Shortest Paths

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?

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

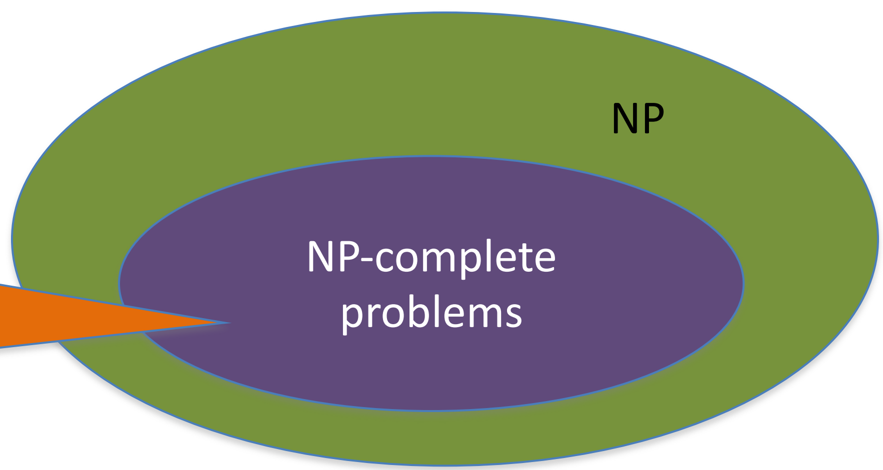
Proving P = NP

Will make cryptography collapse

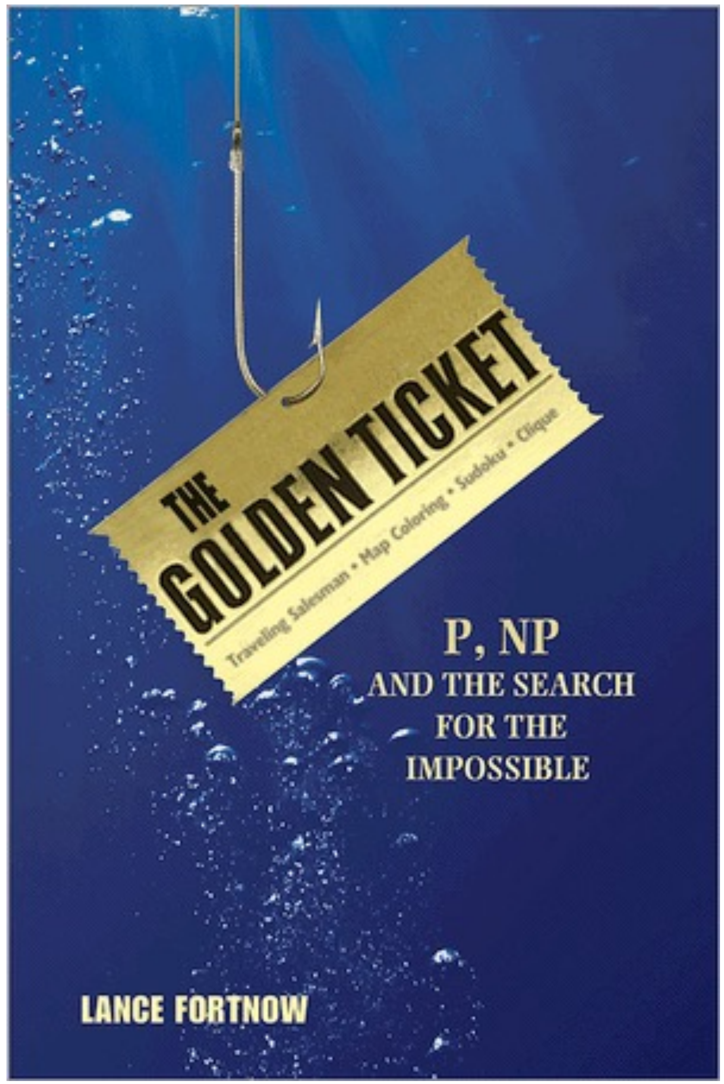
Compute the encryption key!

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"

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 mathematical argument) to change `Question 2` in a way so that you can map it to `Question 1`. Once you have the mapping, then you use the known `Answer 1` to `Question 1`

“Yes, we can” reductions (Example)

Question 2 (Crimson Hawks are in town) [25 points]

The Problem

The `Crimson Hawks` basketball team in the bay area thinks they are not doing enough to hire athletes from UB so they decide to do an exclusive recruitment drive for UB athletes. The `Crimson Hawks` decide to fly over n athletes from UB to the bay area during December for an on-site interview on a single day. The team sets up m slots in the day and arranges for n `Crimson Hawks` coaches to interview the n athletes. (You can and should assume that $m > n$.) The fabulous scheduling algorithms at the `Crimson Hawks` ' offices draw up a schedule for each of the n athletes so that the following conditions are satisfied:

- Each athlete talks with every `Crimson Hawks` coach exactly once;
- No two athletes meet the same `Crimson Hawks` coach in the same time slot; and
- No two `Crimson Hawks` coaches meet the same athlete in the same time slot.

In between the schedule being fixed and the athletes being flown over, the `Crimson Hawks` coaches were very impressed with the CVs of the athletes (including, ahem, their performance in CSE 331) and decide that the `Crimson Hawks` should hire all of the n athletes. They decide as a group that it would make sense to assign each athlete A to a `Crimson Hawks` coach C in such a way that after A meets C during her/his scheduled slot, all of A's and C's subsequent meetings are canceled. Given that this is December, the `Crimson Hawks` coaches figure that taking the athletes 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 athletes to the bay area.

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

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

To help you get a grasp of the problem, consider the following example for $n = 2$ and $m = 4$. Let the athletes be A_1 and A_2 and the `Crimson Hawks` coaches be C_1 and C_2. Suppose A_1 and A_2's original schedules are as follows:

Athlete	Slot 1	Slot 2	Slot 3	Slot 4
A_1	C_1	free	C_2	free
A_2	free	C_1	free	C_2

In the above schedules "free" means that the athlete is not meeting any coach.

In this case the (only) valid truncation is for A_1 to get assigned to C_2 in the third slot and for A_2 to get assigned to C_1 in the second slot. So the truncated schedule will look like this:

Athlete	Slot 1	Slot 2	Slot 3	Slot 4
A_1	C_1	free	C_2 (truncate here)	
A_2	free	C_1 (truncate here)		

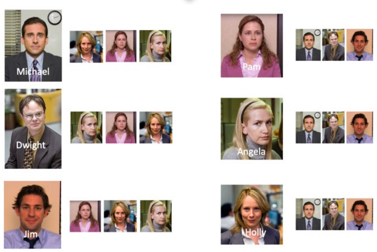
Overview of the reduction

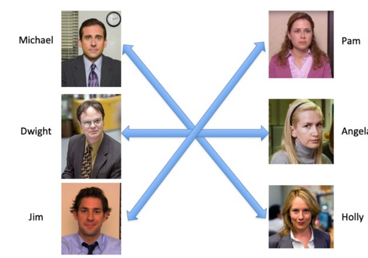
Question 2 (Crimson Hawks are in town)

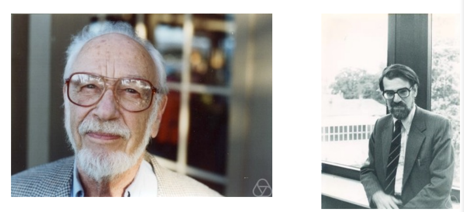
NRMP
National Resident Matching Program

Athlete	Slot 1	Slot 2	Slot 3	Slot 4
A_1	C_1	free	C_2	free
A_2	free	C_1	free	C_2

Athlete	Slot 1	Slot 2	Slot 3	Slot 4
A_1	C_1	free	C_2 (truncate here)	
A_2	free	C_1 (truncate here)		







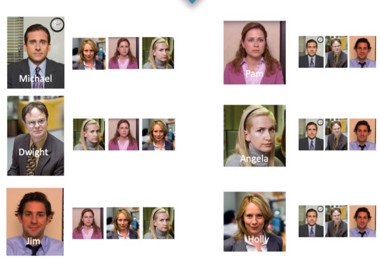
Nothing special about GS algo

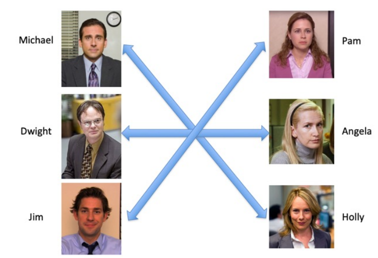
Question 2 (Crimson Hawks are in town)

NRMP
National Resident Matching Program

Athlete	Slot 1	Slot 2	Slot 3	Slot 4
A_1	C_1	free	C_2	free
A_2	free	C_1	free	C_2

Athlete	Slot 1	Slot 2	Slot 3	Slot 4
A_1	C_1	free	C_2 (truncate here)	
A_2	free	C_1 (truncate here)		





ANY algo for stable matching problem works!

Another observation

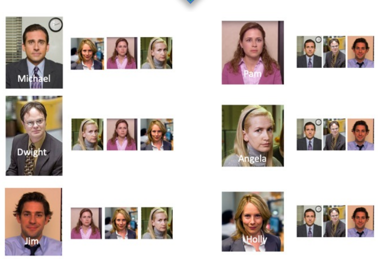
Question 2 (Crimson Hawks are in town)

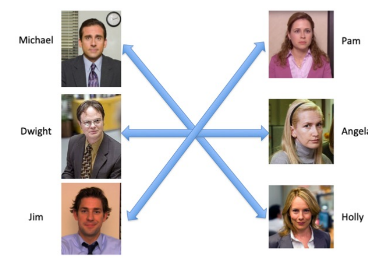
NRMP
National Resident Matching Program

Athlete	Slot 1	Slot 2	Slot 3	Slot 4
A_1	C_1	free	C_2	free
A_2	free	C_1	free	C_2

Athlete	Slot 1	Slot 2	Slot 3	Slot 4
A_1	C_1	free	C_2 (truncate here)	
A_2	free	C_1 (truncate here)		

Poly time steps





ANY algo for stable matching problem works!

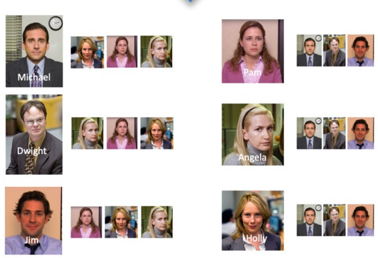
Poly time reductions

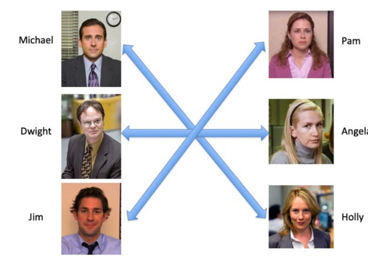
Question 2 (Crimson Hawks are in town) $\leq_P$ NRMP National Resident Matching Program

Athlete	Slot 1	Slot 2	Slot 3	Slot 4
A_1	C_1	free	C_2	free
A_2	free	C_1	free	C_2

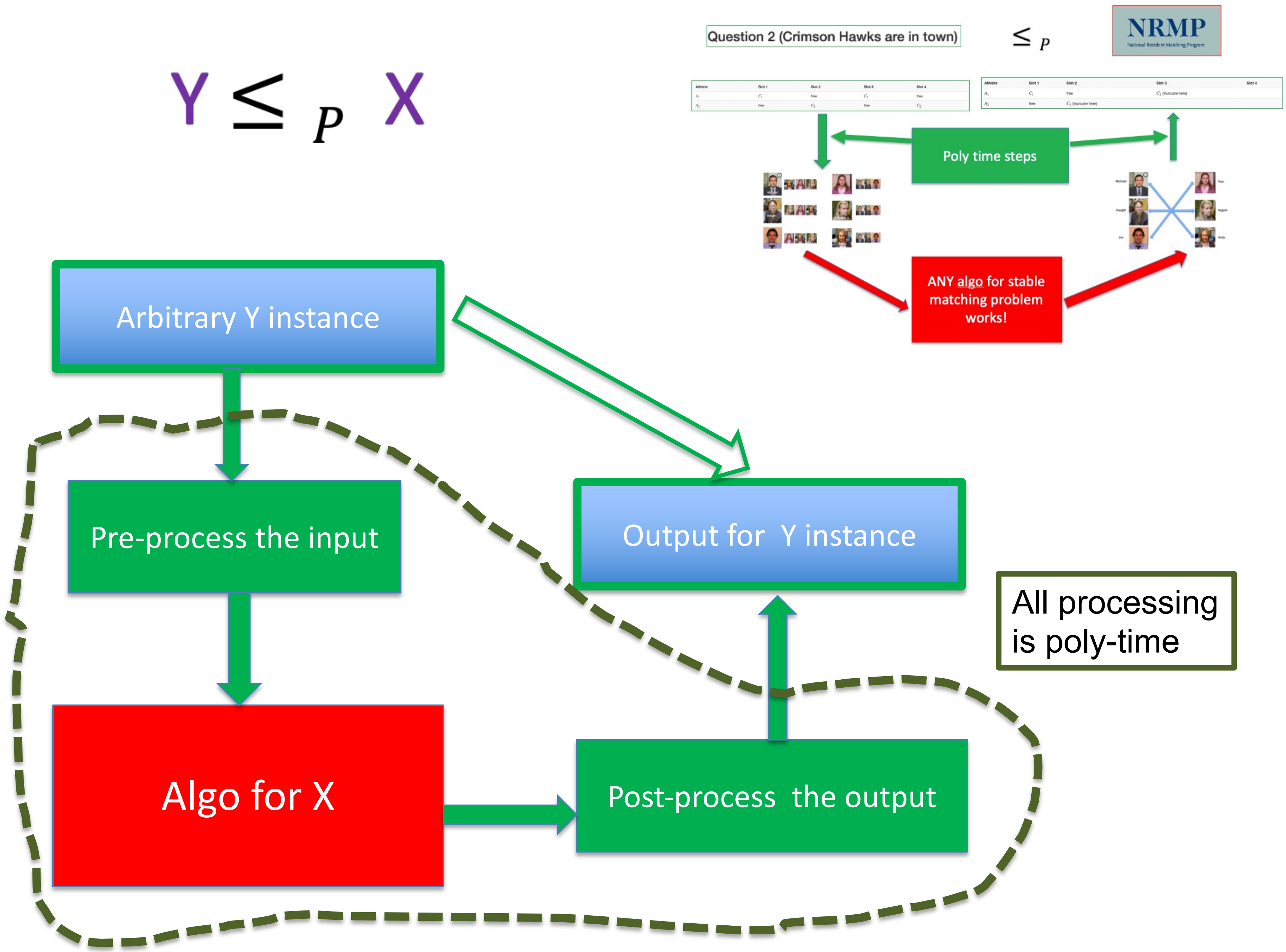
Athlete	Slot 1	Slot 2	Slot 3	Slot 4
A_1	C_1	free	C_2 (truncate here)	
A_2	free	C_1 (truncate here)		

Poly time steps

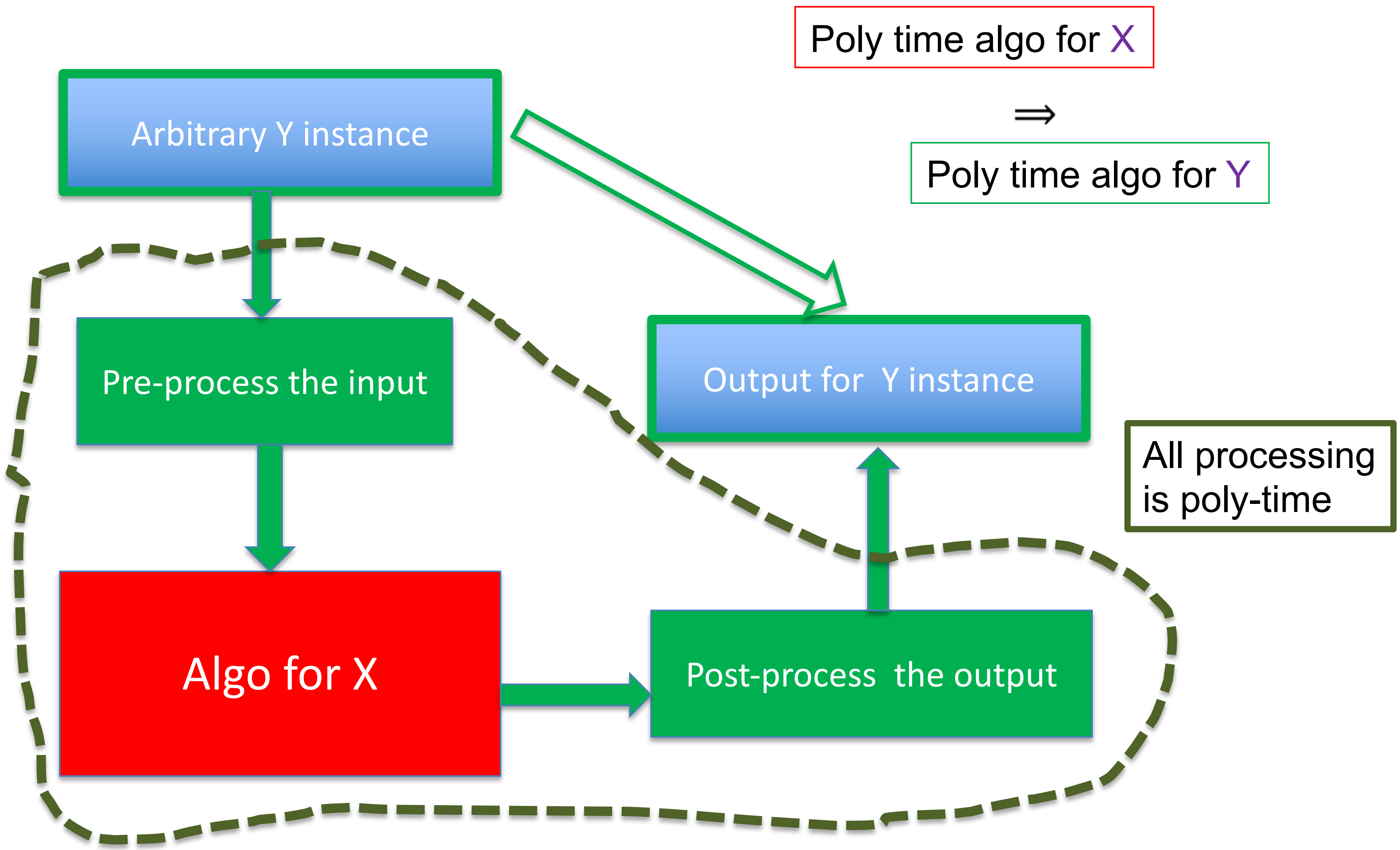




ANY algo for stable matching problem works!


Y ≤ p X
Question 2 (Crimson Hawks are 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
Algo for X
Post-process the output
Output for Y instance
All processing is poly-time

Implications of Y $\leq_p$ X



$$A \implies B$$

$$!B \implies !A$$

Implications of Y $\leq_p$ X

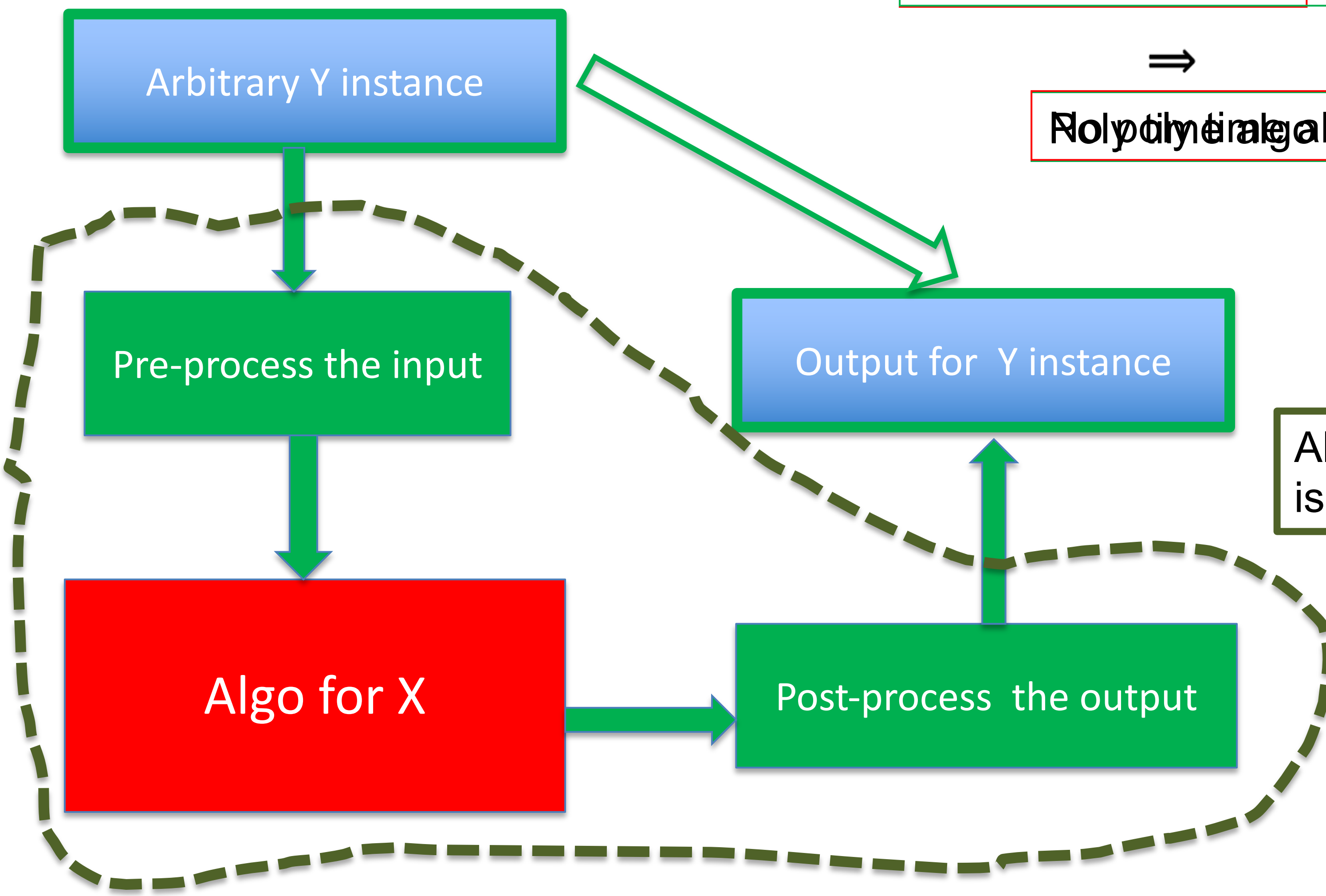
Polytime algo for X

No polytime algo for Y

$\Rightarrow$

Polytime algo for Y

No polytime algo for X



All processing is poly-time