Announcements

Announcements

- Final HW is out
- Exam 3 will be posted on Sunday
- Timeslot signup will be announced this week- keep an eye on your e-mail

The Theory of Computational Complexity

Search vs. Optimization

- Search problem: Find solution satisfying some requirement . User specifies requirement / constraint
  - Find spanning tree with weight at most *b*
  - Find bipartite matching with flow greater than *b*
  - Etc.
- Optimization problem: Find *best* solution
  - Find minimum spanning tree
  - Find heaviest bipartite matching
  - Etc.
- Why are these basically equivalent?

Search vs. Optimization

not algorithms!

P NP

- P (Polynomial): The class of all $\boxed{\text{problems}}$ that can be solved in polynomial running time *

- NP (Non-deterministic Polynomial Time): The class of search problems with solutions that can be *verified* in polynomial time.

checked

deterministic

* there exists an algorithm for which the worst-case running time is a polynomial function of input size

Is P = NP?

$(x \vee y \vee z) \cdots (w \vee \bar{a} \vee \bar{b})$

- Whether P = NP is an open question!
- Most computer scientists think that P ≠ NP.
- But we don't have a proof…
- One of the most important open questions in computer science

$\cancel{P = NP}$

verification: there is some problem and a black box algorithm that claims to solve it. We give the BB input and it produces output. Can we check whether the output is correct in polynomial time?

Reductions

dynamic programming

KP → longest path on DAG

input

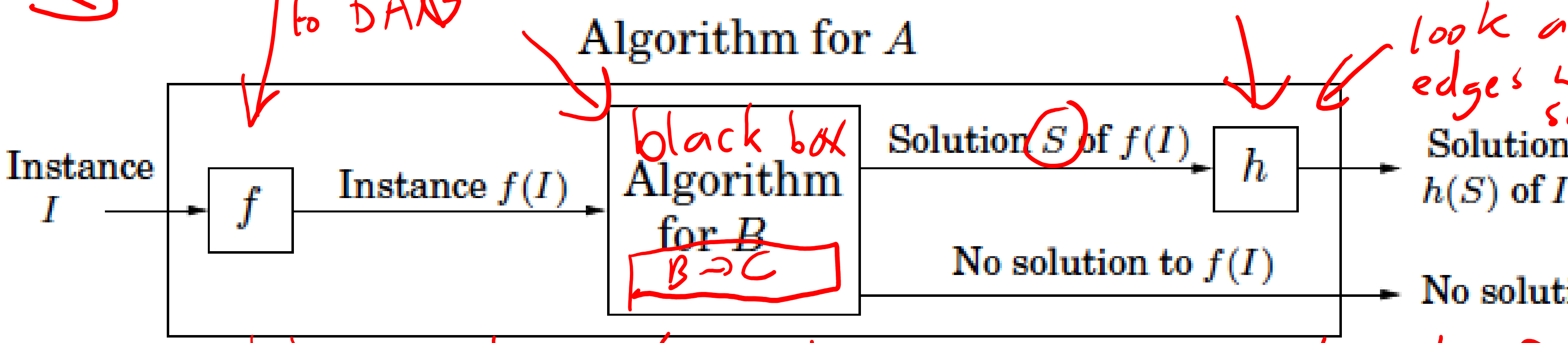
$P_A \rightarrow P_B$

- Problem A **reduces** to Problem B if you can convert every instance of Problem A to an instance of Problem B, and convert the solution to Problem B back to the original solution, in *polynomial time*.

convert KP to DAG

solution to Problem A

look at which edges were used, select corr. objects



~~bip~~ Factory / warehouses → network flow

- Is reduction transitive?

$A \to B \quad B \to C \quad A \to C$ yes

Reductions

0 1 2 . . . W-1 W

- Why are reductions useful?
- If we have ~~an~~ a polynomial-time algorithm for Problem B, and can convert back and forth in polynomial time, then we can solve Problem A in polynomial time!

f, h

- Example: Bipartite matching reduces to network flows

– many DP problems ⟶ longest/shortest path

NP-Completeness

f, h are polynomial-time

- Tricky use of reductions: If Problem A *does not* have a polynomial time solution, and it reduces to Problem B, then Problem B *also does not* have a polynomial time solution! $A \to B$
- A search problem is NP-complete if *every other problem in NP reduces to it*
- In other words: if you can solve an NP-complete problem quickly, you can solve anything in NP quickly! polynomial time $\Rightarrow P = NP$

Reduction Strategy

$A \rightarrow B \not\Leftrightarrow B \rightarrow A$

$A \rightarrow B$
Problem B is at least as hard as problem A

1. Direction of reduction depends on your goal:
   – If you have a polynomial algorithm for Problem B, and want to find one for Problem A, show that A reduces to B

     $A \rightarrow B$

   – If you want to show that Problem B has no polynomial solution, show that some NP-complete problem A reduces to Problem B

$(x \vee y)$
2SAT

$(x \vee y \vee z)$
3SAT

$\boxed{A \rightarrow B}$

by contradiction, means that B does not have poly-time algorithm

NP-complete

Suppose we try $B \rightarrow A$ instead
NP-C

$2SAT \rightarrow SAT$

A is hard! (no polynomial time alg for A)

Reduction Strategy

2. When reducing Problem A reduces to Problem B, show the following: 0. Find the reduction (f, h)
   1. Any instance of Problem A can be converted to an instance of Problem B in polynomial time (f runs in poly-time)
   2. A solution to the converted instance can be converted back to a solution for Problem A in polynomial time (h)
   3. If Algorithm B finds a solution to the converted instance, it corresponds to an actual solution to the Problem A instance → solution is correct
   4. ~~If the Problem A instance has a solution, then Algorithm B is able to find a solution to the converted instance~~

If Algorithm B says "no solution", then there is no solution to prob. A input

In-Class Exercise: Hamiltonian Cycles and Paths

- Hamiltonian Cycle: in an undirected graph, ~~is there~~ Find a cycle that passes through every node exactly once?

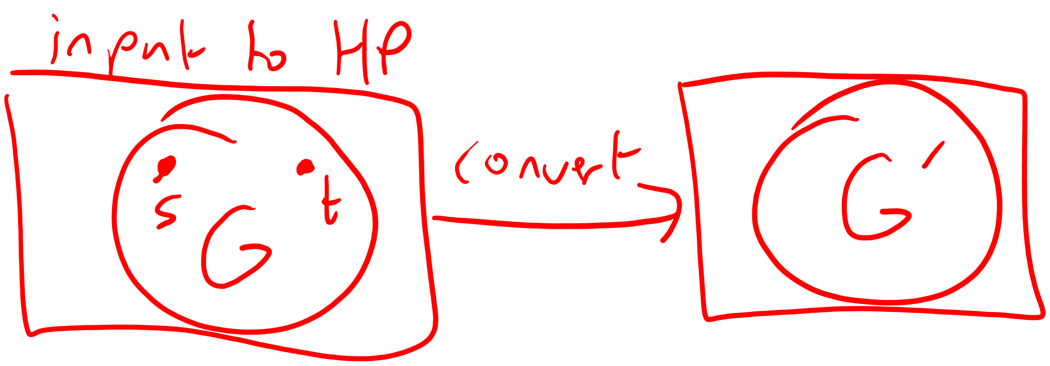
$s \neq t$ in undirected graph

- Hamiltonian Path: Given vertices *s* and *t*, ~~is there~~ Find a path from *s* to *t* that passes through every node exactly once?
- Show how to reduce Hamiltonian Path to Hamiltonian Cycle. (i.e., if we have an algorithm for the Hamiltonian Cycle problem, can we use that to solve the Hamiltonian Path problem?)

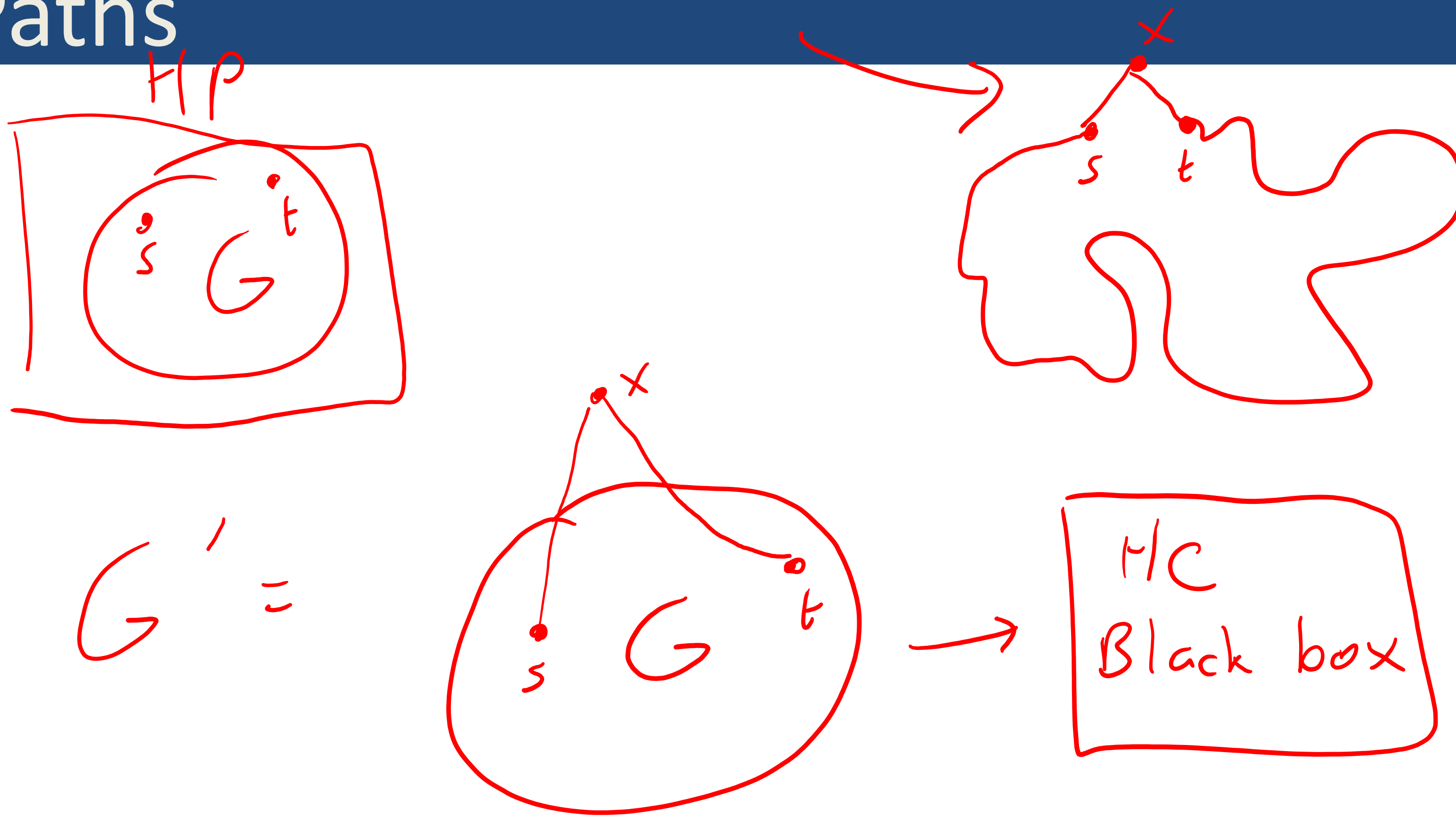
In-Class Exercise: Hamiltonian Cycles and Paths

What does it mean to reduce HP → HC?

- Assume we have a black box algo for HC
- Construct an algorithm for HP
  - input is graph G, two nodes s & t



In-Class Exercise: Hamiltonian Cycles and Paths



a HC on G' touches x exactly once.
So it must enter x from s or t and leave by the other edge.

In-Class Exercise: Hamiltonian Cycles and Paths

f: add node x, add ~~and~~ edges (s,x) (t,x)

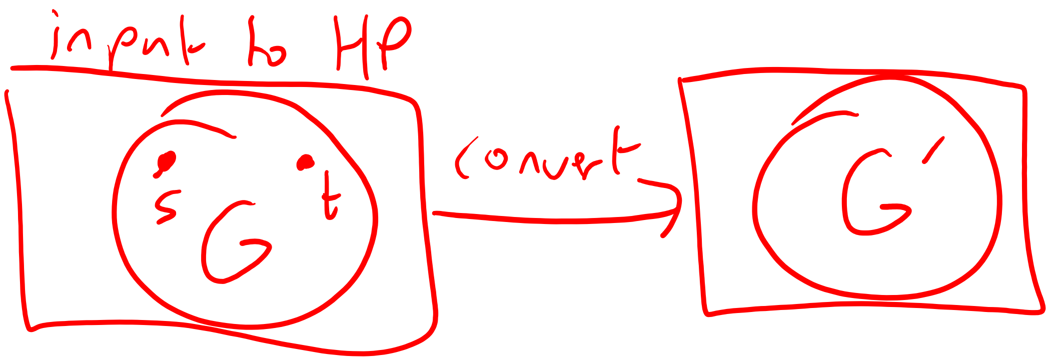
h: ~~so~~ remove x, (s,x), (t,x) from the HC output by black box, what's left will be a HP in G from s → t

1. Show that f runs in poly-time.
   All we do is add a row/column, change 4 0s to 1 in adj. matrix.
   Obv. poly-time.
2. Show that h runs in poly time, Obvious, all we do is remove 2 edges from HC.

In-Class Exercise: Hamiltonian Cycles and Paths

What does it mean to reduce HP → HC?

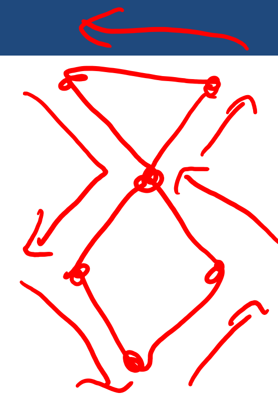
- Assume we have a black box algo for HC
- Construct an algorithm for HP
  - input is graph G, two nodes s & t



In-Class Exercise: Hamiltonian Cycle and Traveling Salesman Problem

- Hamiltonian Cycle: Does the graph have a cycle that visits every node?
- Traveling Salesman Problem: Given a graph with integer weights on the edges, and a starting node *s*, and an integer budget *b*, is it possible to find a cycle beginning and ending at *s* that touches every node such that the total weight of edges in the cycle is less than *b*?
- Reduce Hamiltonian Cycle to TSP

In-Class Exercise: Hamiltonian Cycle and Traveling Salesman Problem



unweighted G

input to HC

weighted G'

nodes

budget b

f: adds weights to G
sets s = some node
sets b = # nodes
h: returns the same output

cycle visits every node at least once, but total # edges/nodes in the cycle $\leq b$, so each node seen exactly once

s = arbitrary node
b = # nodes
weight in G' = 1