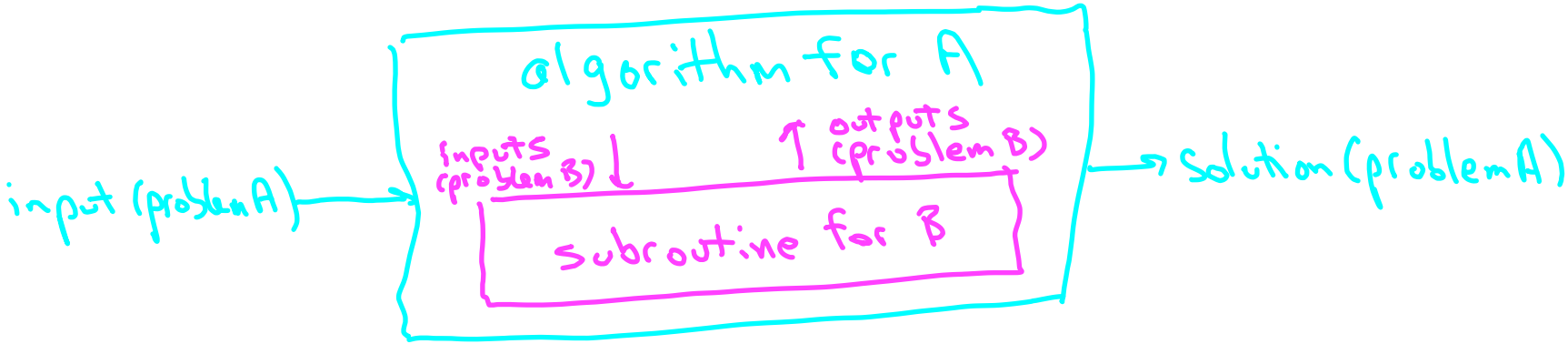
Section 19.5: Proving *NP*-Hardness: A Simple Recipe

Reductions

Definition: A problem A *reduces* to a problem B if an algorithm that solves B is easily translated to one that solves A. (polynomial # of subroutine calls, polynomial amt of additional work)



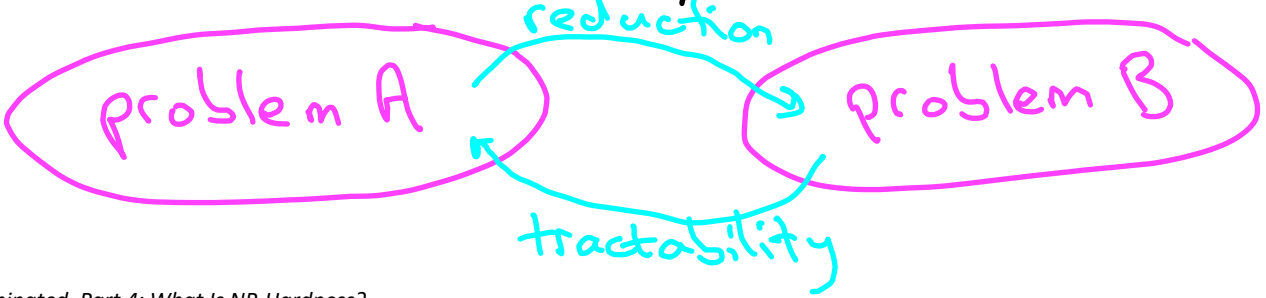
Using Reductions to Design Fast Algorithms

Examples: (1) median-finding reduces to sorting

(2) APSP reduces to SSSP.

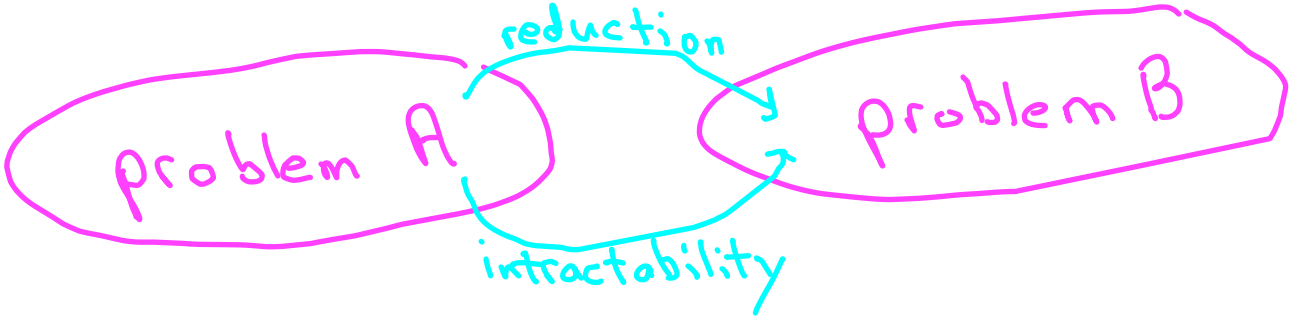
(3) Longest common subsequence reduces to sequence alignment.

In general: If A reduces to B and B is poly-time solvable, then A also poly-time solvable.



Using Reductions to Spread *NP*-Hardness

Nefarious use: If A reduces to B and A is NP-hard, then B is also NP-hard.



How to prove a problem B is NP-hard:

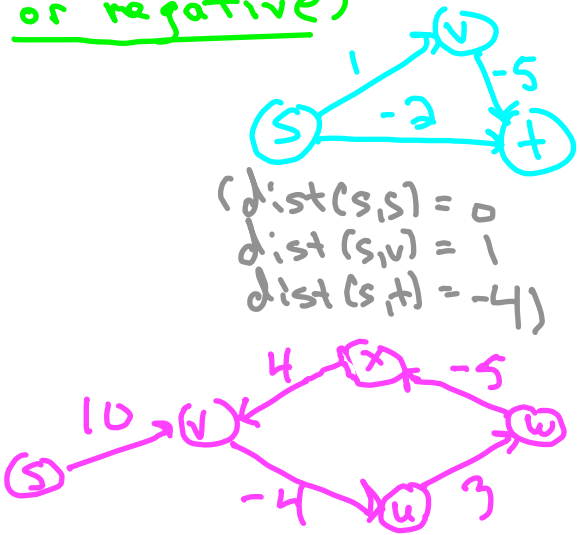
1. Choose an NP-hard problem A.
2. Prove that A reduces to B.

Cycle-Free Shortest Paths

Input: directed graph $G = (V, E)$, starting vertex s, length ℓ_e for each $e \in E$. (can be positive or negative)

Output: For every $v \in V$, the minimum length of a cycle-free s-v path (or $+\infty$, if G has no s-v paths)

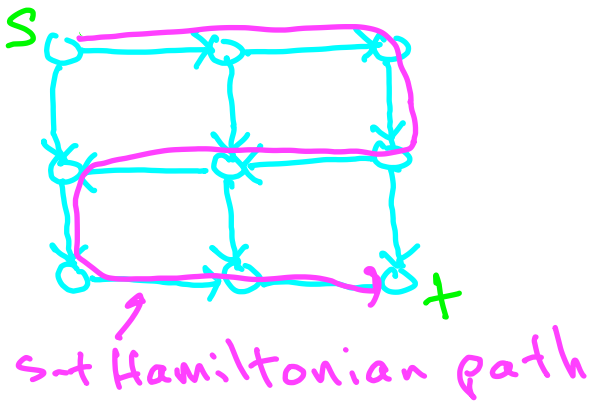
Fact: NP-hard.

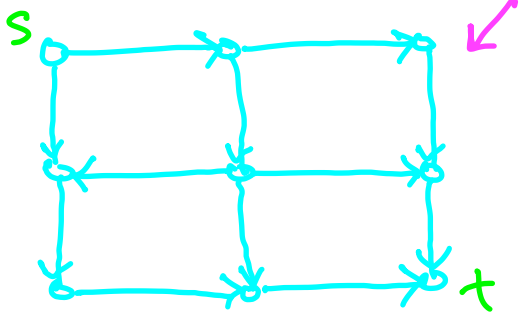


The Directed Hamiltonian Path Problem

Input: Directed graph $G=(V,E)$, starting vertex s, ending vertex t.

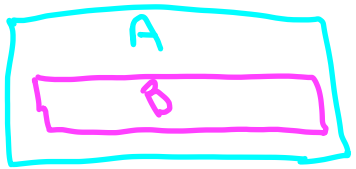
Output: "Yes" if G contains an s-t Hamiltonian path (visits every vertex once), "no" otherwise.





Fact: DHP is NP-hard.

DHP Reduces to CFSP



Given: instance of DHP: $G=(V,E)$, s, t.

1. Set $l_e = -1$ $\forall e \in E$.
2. Invoke CFSP subroutine.
3. If $dist(s,t) = -(n-1)$ return "yes." $(n=|V|)$
   else return "no."