NP-Complete Problems

Prem Nair

Class *P and class NP*

Loosely speaking:

A problem belong to class P if it can be solved in polynomial time.

A problem belong to class NP if it can be solved in polynomial time using a non-deterministic algorithm.

Which is same as saying

A problem belong to class NP if it can be verified in polynomial time.

Solving vs Verifying

Solve the equation:

$7x^2 - 12x - 352 = 0$

Verify $x = 8$ is a solution to the equation

$7x^2 - 12x - 352 = 0$

Verification takes less time!

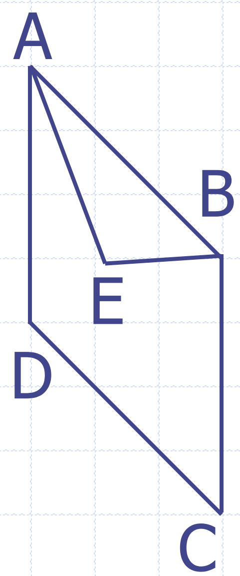
Example : A Non-deterministic Algorithm

```
IsPresent(A, x)
// A is an array of items. x is an item.
// Will return true if x is present and false otherwise.
        i <- guess(A, x)  // guess will return the correct index
                          // if x is present in the array A.
        if (A[i] == x)
                return true
        else
                return false
```

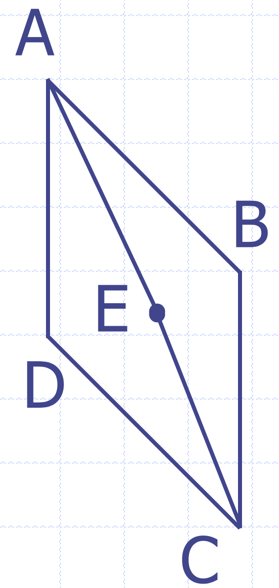
$P \subseteq NP$

Is $P = NP$ we do not know.

Hamiltonian Cycle And Vertex Covers



HC
Minimum VC = {A, C, E}



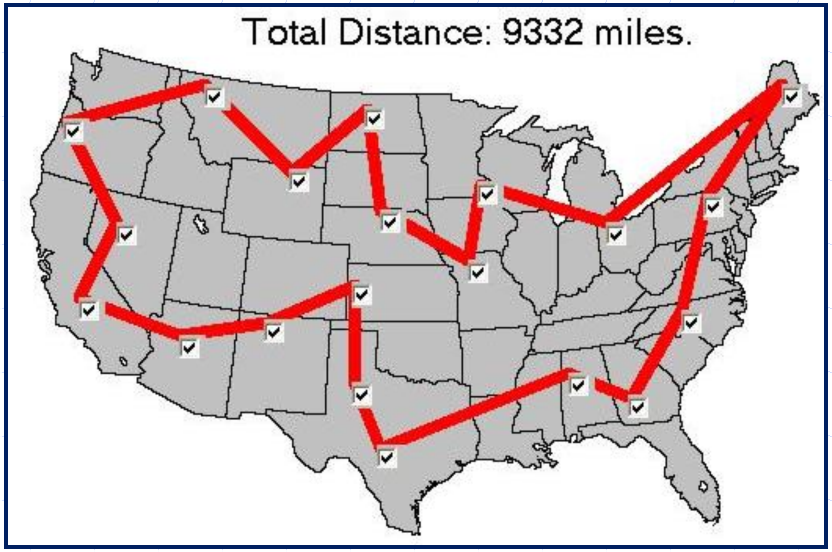
Not an HC
Minimum VC = {A, C}

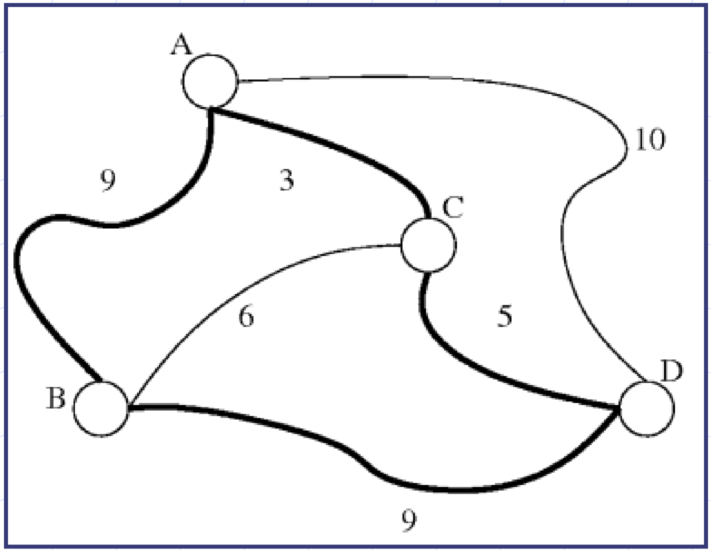
Hamiltonian Cycle And Vertex Covers

- A Hamiltonian cycle in a graph G is a simple cycle that contains every vertex of G. A graph is a Hamiltonian graph if it contains a Hamiltonian cycle.
- Examples. The Herschel graph is not a Hamiltonian graph.
- If $G = (V,E)$ is a graph, a vertex cover for G is a set $C \subseteq V$ such that for every $e \in E$, at least one end of e lies in C.
- Fact. The known algorithms for determining whether a graph is Hamiltonian, and for computing the smallest size of a vertex cover, run in exponential time.

Traveling Salesperson Problem

- *Traveling Saleperson Problem* (TSP): Given a complete graph G with cost function c: E $\rightarrow$ N and a positive integer k, is there a Hamiltonian cycle C in G so that the sum of the costs of the edges in C is at most k? Solution data: a subset of E.





Problems in NP

- HC
- VC
- TSP
- Are in NP

Reducibility (informal 1)

Let Q denote the problem of finding the area of a square.

Let R denote the problem of finding the area of rectangle.

Given an instance I_Q of Q, we can transform into an instance I_R of R.

I_Q has a solution iff I_R has a solution

- We write $Q \xrightarrow{\text{poly}} R$
- Note that R is "harder" than Q

```
Algorithm areaSquare(double side)
    return (areaRectangle(side, side)
```

Reducibility (informal 2)

Let Q denote the problem computing the distance between two points in 2D.

Let R denote the problem computing the distance between two points in 3D.

Given an instance I_Q of Q, we can transform into an instance I_R of R.

I_Q has a solution iff I_R has a solution

- We write $Q \xrightarrow{\text{poly}} R$
- Note that R is "harder" than Q

```
Algorithm distance2D((x1, y1), (x2, y2))
    return distance3D((x1, y1, 0), (x2, y2, 0))
```

HamiltonianCycle → TSP

Let Q denote the Subset sum problem

Let R denote the Knapsack problem

Given an instance I_Q of Q, we can transform into an instance I_R of R.

I_Q has a solution iff I_R has a solution

- We write $Q \xrightarrow{\text{poly}} R$
- Note that R is "harder" than Q

Algorithm SubsetSum(S, k) //subset and k
    return Knapsack(S, S, k) //values are same as weights

SubsetSum $\xrightarrow{\text{poly}}$ Knapsack

Let Q denote the Subset sum problem

Let R denote the Knapsack problem

Given an instance I_Q of Q, we can transform into an instance I_R of R.

I_Q has a solution iff I_R has a solution

We write Q $\xrightarrow{\text{poly}}$ R

Note that R is "harder" than Q

```
Algorithm SubsetSum(S, k)    //subset and k
    return Knapsack(S, S, k)   //values are same as weights
```

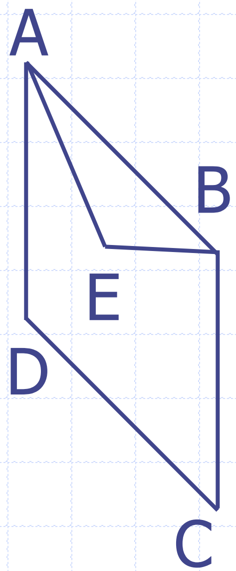
HamiltonianCycle $\xrightarrow{\text{poly}}$ TSP

We show HamiltonianCycle is reducible to TSP

- Given a graph G = (V,E) on n vertices (input for HamiltonianCycle) – notice G is a subgraph of K_n. Obtain an instance H, c, k of TSP as follows: Let H be the complete graph on n vertices (i.e. H is K_n), obtained by adding the missing edges to G. Let
c(e) = 0 if e $\in$ E, else c(e) = 1. Let k = 0.
- Need to show: G has a Hamiltonian cycle if and only if H, c, k has a Hamiltonian cycle with edge cost $\leq$ k
- If G has Hamiltonian cycle C, C is Hamiltonian in H also. Since each edge e of C is in G, c(e) = 0. So cost sum $\leq$ k. Converse: A solution C for H,c,k implies all edges of C have weight 0; therefore, every edge of C also is an edge in G. Therefore C is an HC in G.

HamiltonianCycle $\xrightarrow{\text{poly}}$ TSP (Yes case)

G

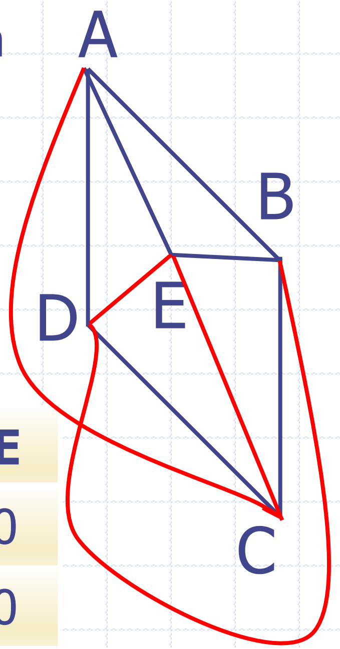


	A	**B**	**C**	**D**	**E**
A	0	1	0	1	1
B	1	0	1	0	1
C	0	1	0	1	0
D	1	0	1	0	0
E	1	1	0	0	0

Does G has a HC?

H

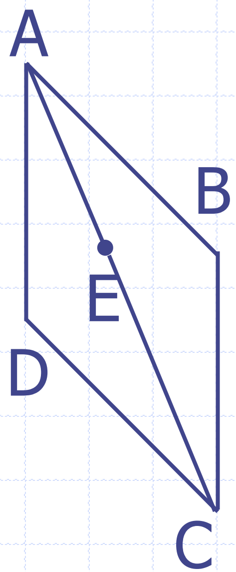
Can TSP visit with k = 0?



	A	**B**	**C**	**D**	**E**
A	0	0	1	0	0
B	0	0	0	1	0
C	1	0	0	0	1
D	0	1	0	0	1
E	0	0	1	1	0

HamiltonianCycle $\xrightarrow{\text{poly}}$ TSP (No case)

G

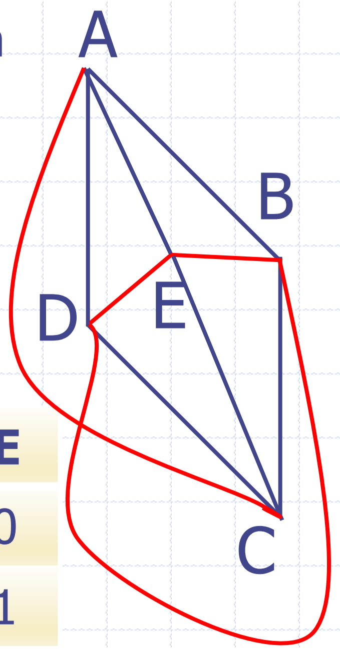


	A	**B**	**C**	**D**	**E**
A	0	1	0	1	1
B	1	0	1	0	0
C	0	1	0	1	1
D	1	0	1	0	0
E	1	0	1	0	0

Does G has a HC?

Can TSP visit with k = 0?

H



	A	**B**	**C**	**D**	**E**
A	0	0	1	0	0
B	0	0	0	1	1
C	1	0	0	0	0
D	0	1	0	0	1
E	0	1	0	1	0

HamiltonianCycle $\xrightarrow{\text{poly}}$ TSP

Note: From a practical point of view, all we have to do is to change all non-diagonal values of 1's and 0's in the adjacency matrix of an instance of HC to 0's and 1's to obtain an instance of TSP.

Algorithm HCInstanceToTSPInstance(H)
Input : H an instance of HC as an adjacency matrix.
Output : T an instance of TSP as an adjacency matrix.

```
for i = 0 to n – 1 do
    for j = i + 1 to n – 1 do
        T[i, j ] <- (H[i, j] + 1) % 2
        T[j, i ] <- (H[i, j] + 1) % 2
//Arrays starts with index 0. T initialized with 0 during creation.
//Time complexity O(n²).
```

HamiltonianCycle $\xrightarrow{\text{poly}}$ TSP

Algorithm isHC(H)
Input : H an instance of HC as an adjacency matrix.
Output : **true** if H is Hamiltonian. **false** otherwise.

T <- HCInstanceToTSPInstance(H)
return isTSP(T, 0)

//isTSP(T, k)
//T an instance of TSP (adjacency matix)
//k a non-negative integer.
//isTSP(T, k) returns **true** if TSP can visit all cities at the cost of k and come back to home city. **false** otherwise.

NP-hard Problems

A problem Q is ***NP**-hard* if for *every* problem R in ***NP***, R is polynomial reducible to Q.

$$R \xrightarrow{\text{poly}} Q$$

You can think of them as problems harder than all problems in NP.

NP-Complete Problems

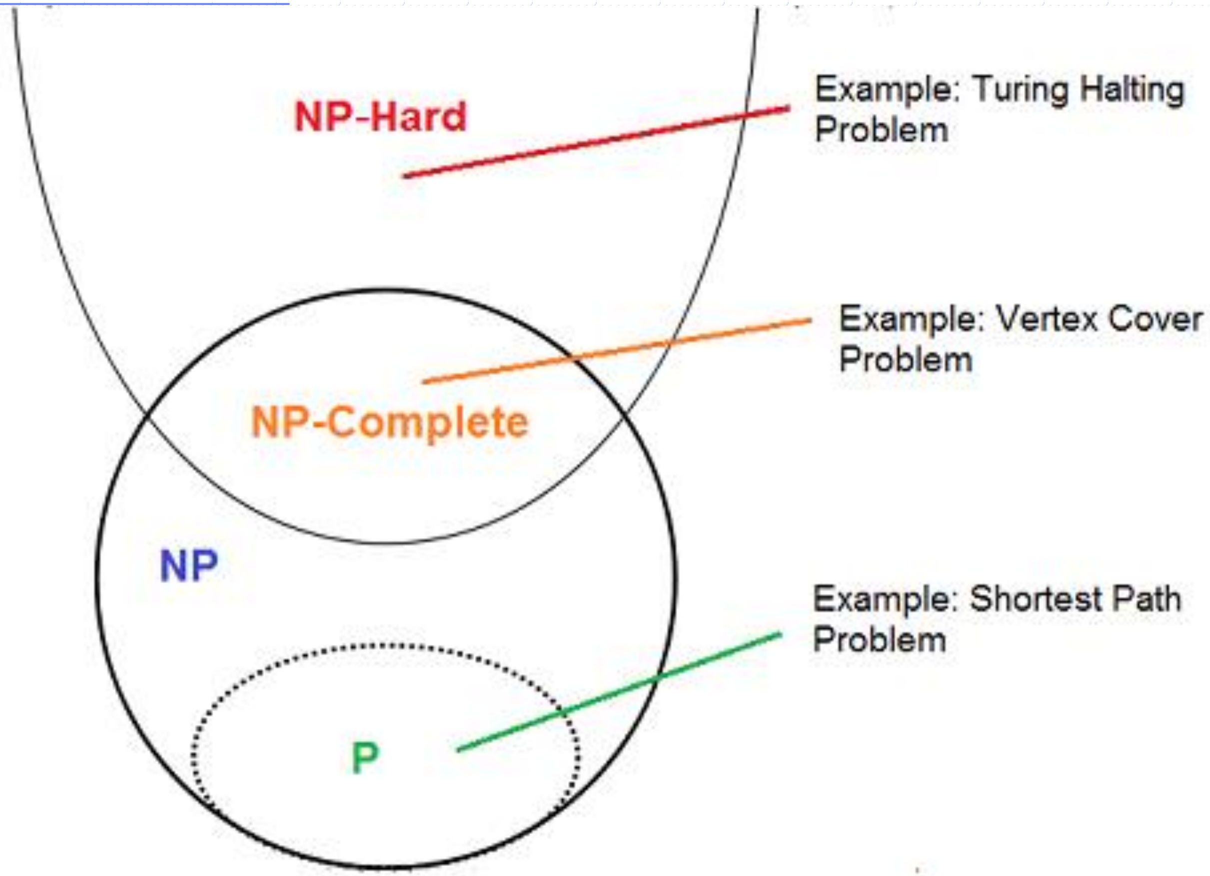
A problem Q is ***NP**-complete* if

Q belongs to *NP*,

and

Q is NP-hard.

NP-Complete Problems



This diagram assumes that P != NP

HamiltonianCycle is NP-Complete

This is an outline of a proof that HamiltonianCycle is NP-Complete under the assumption that VertexCover is NP-complete:

- Show HC is in NP.
- Pick VC as the known NP-complete Problem
- Show VC is polynomial reducible to HC (see Slide 20)

Summary: To show Y is NP-Complete

- Show Y is in NP.
- Pick X. A known NP-complete Problem
- Show X is polynomial reducible to Y