CSE525 Lec21

After NP-completeness …

Debajyoti Bera (M21)

NP-Complete

SetCover (Universe $U = \{x_1 \dots x_n\}$, $S = \{S_1 \dots S_m\}$, k) : return true if $S_{i_1} \dots S_{i_k}$ s.t. $\bigcup_{j=1}^{k} S_{i_j} = U$

each $S_i \subseteq U$

VerifierSC((U, S, k), $S' \subseteq S$):

S' is a solution of (U, S, k) // verify that all $x \in U$ is also present in some $S_j \in S'$

① Algorithm for reduction

→ Signature

→ Correctness of logic (Reduction lemma)

→ Time complexity

② Algorithm for verification of PROB

→ Signature (Input, output)

Input: (i) instance (x) of PROB

(ii) (P) proof / certificate / witness (designed by you)

Output: Boolean

→ Correctness of logic (verification lemma)

↳ x is a Yes-instance of PROB iff there exists a proof P s.t. Verifier(x, P) → true

→ Time complexity

complexity of Verifier in terms of $|x|$

Verifier for ~~NP~~ Ham Path

~~L is a HamPath of G~~

of HamPath

def VerifyHamPath(instance (G,s,t), proof L): // L is a list of vertices (explain proof)

1. return false if L uses a vertex not in G, or does not use every vertex in G, or does not start with s, or does not end with t

exactly once

2. For every pair (u,v) of subsequent vertices in L:

   If (u,v) is not an edge: return false

3. Return true

Correctness claim: G has a Hampath from s to t iff there exists a proof L for which VerifyHamPath returns true.

⇒ Suppose G has a HamPath ($s - v_1 - v_2 - \cdots - v_{n-2} - t$). Consider the following $L = [s, v_1, \cdots, v_{n-2}, t]$.
Line 1 doesn't return false. Loop in ② doesn't return false. ∴ True is returned.

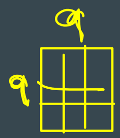
⇐ Suppose VerifyHP returns true. ∴ $L \subseteq V$, $L = s \cdots t$ and uses all vertices of G exactly once. ... ∴ L must a HamPath

Complexity claim: VerifyHamPath runs in time poly(|G|).

$O(|V|)$

**!! Prove that $\{x : x \text{ is a prime number}\}$ is in NP.

Verify for SUBSETSUM



def VerifySS(instance (A,T), proof B): B is a set of indices from {1...n}

    return false if B is not a subset of {1 ... n}

    return false if the elements of A at the indices given in B do not sum to T

    return true otherwise

Correctness claim: A has a subarray that sums to T <u>iff</u> there exists a proof B for which VerifySS returns true.

Sudoku(partially Board) → Yes if the board can be filled as per rules

VerifierSudoku(partially filled board, fully filled board):

① check if fully filled board extends partially filled board

② rules are ok.

n × n

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Complexity claim: VerifySS runs in time poly(|A|,|T|) = poly(nk, |T|) if A consists of k-bit integers.

Non-decision problems

For NP-completeness, need decision problems.

Problems that are not decision problems can be ...

- Function problems (<u>Find</u> a colouring of a graph using at most 3 colours)
- Counting problems (<u>Count</u> the number of 3-colourings of a graph)
- Optimization problems (<u>Optimize</u> the number of colours needed to colour a graph)

Finding Satisfiable Assignment

SolveSAT(F) := Output a satisfying assignment of F if one exists, NULL o/w

If **SolveSAT** can be solved in polytime, then **SAT** can be solved in polytime.
Show: If **SAT** can be solved in polytime, then **SolveSAT** can be solved in polytime.

Q: Suppose there is a black-box B for solving **SAT** in polynomial-time. Design a polynomial-time algorithm (that uses B cleverly, maybe multiple times, maybe on cleverly constructed formulas) that can solve **SolveSAT** in polytime.

$$\phi_2 = (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \bar{x_2} \vee x_4) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_4}) \wedge \\ (x_1 \vee \bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_3 \vee x_4) \wedge \\ (\bar{x_1} \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_3 \vee \bar{x_4}) \wedge (x_2 \vee \bar{x_3} \vee \bar{x_4}) \wedge \\ (\bar{x_2} \vee \bar{x_3} \vee \bar{x_4})$$

```
def SolveSAT(F):
    r ← B(F)                          ← add a base case
    if r = false : return NULL
    // r is true ⟺ F is satisfiable
    x_1 : any variable in F
    F_{x_1=T} = copy of F with x_1 hardcoded to T
    r_1 ← B(F_{x_1=T})
    if r_1 = True :
        print("x_1 = T")
        SolveSAT(F_{x_1=T})
    else : // r_1 = false
        F_{x_1=F} = copy of F with x_1 hardcoded to F
        print("x_1 = F")
        SolveSAT(F_{x_1=F})
```

$$T(n) = 2\,\underbrace{\mathrm{Time}(B)}_{\text{poly}} + T(n-1)$$

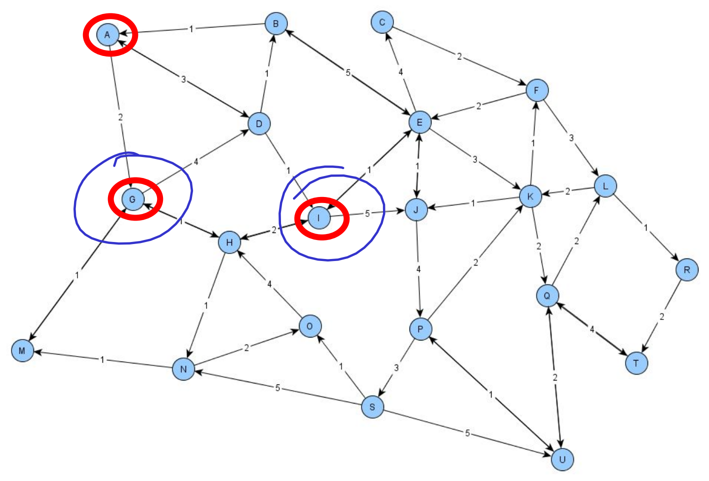
$$\leq 2 \cdot n \cdot \mathrm{Time}(B)$$

Ex. Find a solution to Sudoku from a solver for its decision version.

Finding optimal 3-colouring

decision

Q: Suppose there is a black-box B for solving 3COL in polynomial-time. Design a polynomial-time algorithm (that uses B cleverly, maybe multiple times, maybe on cleverly constructed graphs) that can find a valid 3-colouring of a graph, if one exists.



x & y : two vertices with an edge
Lemma: x and y must be differently coloured.

x & y : two vertices <u>without</u> an edge. How to colour x and y?
Gxy = merge x and y in G

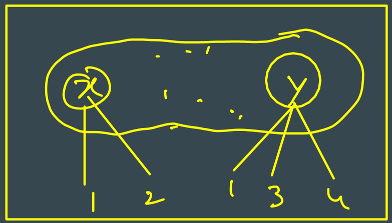
Prove that: G is 3-colourable <u>using same colours for x and y</u> iff Gxy is 3-colourable

Q: Show to compute a 3-colouring of G using black-box B.

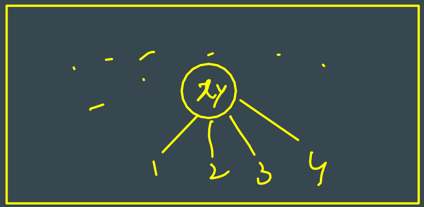
$r = B(G)$

if r = false: return NULL

// r = true $\Rightarrow$ G is 3colorable.



G



G_{xy}

if G_{xy} is 3 colorable, then the same coloring can be used for G.

If G has >3 vertices, it must have atleast 2 vertices that should be identically coloured.

do{ G: Take two vertices x & y without an edge

G_{xy}: merge x & y

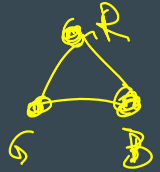
$r = B(G_{xy})$

if r = True:

recursively find a coloring for G_{xy}

else: repeat

} until (G is a triangle)



Verifier Knapsack(($V[1 \cdots n]$, $wt[1 \cdots n]$, W, V'), $B \subseteq \{1 \cdots n\}$):

verify that the items in B have total wt $\leq W$ & total value $\geq V'$