CSC373 Week 8 Notes

Turing Machines:

- Turing Machines (TM) are a mathematical model for what it means to perform a mechanical computation.

- It has the following:
  1. An infinitely long tape divided into cells.
  2. A pointer that can move around the tape. It always starts off at the head of the tape.
  3. A transition function that describes how to change states, move the pointer and read/write symbols on a tape.

  Note:
  - The input is given on the tape.
  - Intermediate computations can go there.
  - The output is also on the tape.

  TMs also maintain an internal state.

- The Church-Turing Thesis states that:
  Everything that is computable can be computed by a TM.

  It is widely accepted but cannot be proven. Also, there are some problems that a TM cannot solve, regardless of the time given. (E.g. Halting Problem)

Encoding:

- Let S be a set of finite symbols.
- Let S^* be a set of all finite strings using symbols from S.
  I.e. $S^* = \bigcup_{n \geq 0} S^n$

- Input:
  - $w \in S^*$
  - $|w|$ = length of input = length of w on the tape
    Each cell of the tape can contain at most 1 symbol.
    So, if w takes up 5 cells, its length is 5.

- Output:
  - $f(w) \in S^*$
  - Length of output $= |f(w)|$
  - Note: For decision problems, the output is always Yes or No.

- Going back to S, our set of finite symbols, we will let $S = \{0,1\}$.
  (I.e. S will be binary)
  Note: We could've used S = 3-ary or S = 18-ary, but the only difference is that the length of the encoding (input/output) would change by a constant factor.
  If we're using binary to encode everything, then the length of the input and output, say a_i and a_o will be $\log_2 a_i$ and $\log_2 a_o$, respectively.

  I.e.
  Input $\rightarrow a_i \xrightarrow{\text{length}} \log_2 a_i$
  Output $\rightarrow a_o \xrightarrow{\text{length}} \log_2 a_o$

If we were to use 3-ary or 18-ary, then the length would differ by a constant factor.

I.e.
Let a_i = input and a_o = output. Then:

base	Length of input	Length of output
2	$\log_2 a_i$	$\log_2 a_o$
3	$\log_3 a_i$	$\log_3 a_o$
18	$\log_{18} a_i$	$\log_{18} a_o$

However, if we let S be unary, 1-ary, then the length blows up exponentially.

E.g. Say the input is 2.
In binary → 10
In unary → 11 (Note: These are lines)

Both are length of 2

Now, say the input is 100.
In binary → 1100100 ← Length of 7
In unary → $\underbrace{11\ldots\ldots1}_{\text{100 lines}}$ ← Length of 100

Binary is good enough, but unary isn't.

Efficient Computability:

- A TM solves a problem in polynomial time if there is a polynomial p s.t. on every instance of n-bit input and m-bit output that the TM halts/stops in at most $p(n,m)$ steps.

  Polynomial: $n, n^2, 5n^{100}, n\log 100$
  Non-polynomial: $2^n, 2^{\sqrt{n}}, 2^{\log_2 n}$

- Extended Church-Turing Thesis: Everything that is efficiently computable is computable by a TM in poly time.

  Much less widely accepted, esp today.

Polynomial Time Problems:

- Denoted as P.
- Is the set of all decision problems computable by a TM in polynomial time.

Non-deterministic Polynomial Time Problems:

- Denoted as NP.
- Is the set of all decision problems where the answer is yes can be proved/verified in polynomial time by a TM.
- E.g. (Subset sum problem)
  Given an array/set of numbers, is there a subarray/subset of numbers that add up to 0?

  Can prove in polynomial time

  Consider the set $\{-7, -3, -2, 5, 8\}$.
  We can see that $\{-3, -2, 5\}$ is one such subset.
  However, it would take an exponentially long amount of time to answer the original problem. Have to iterate over each subset $\rightarrow O(2^n)$
  Hence, this is an example of a NP problem.

- Informally:
  - Whenever the answer is Yes, it can be proved/verified in polynomial time.
  - Whenever the answer is No, there may not be a short proof/it may not be proved/verified in poly time.

Co-NP:

- Same as NP except whenever the answer is no, it can be proved in polynomial time.

- A decision problem X is co-NP iff its complement $\bar{X}$ is NP.

Cook's Conjecture:

- P is likely not equal to NP

Reductions:

- Problem A is p-reducible to problem B, denoted as $A \leq_p B$, if an oracle/subroutine for B can be used to efficiently solve A.

  I.e. You can solve A by making polynomially many calls to the oracle for B and doing additional polynomial-time computation.

- If $A \leq_p B$ and B can be solved efficiently, then so can A.

- If $A \leq_p B$ and A can't be solved efficiently, then neither can B.

- Note: The statement "If $A \leq_p B$ and B can't be solved efficiently then neither can A" is wrong.
  A could be an "easy" problem that can be reduced to B, a "hard" problem.

- If you want to prove that a problem is hard, you should reduce a known hard problem to it.

Note: Reducing your problem to a hard problem doesn't work. Your problem could be easy and be reducible to a hard problem.

NP- Completeness:

- Denoted as NPC.
- A problem B is NPC if it is in NP and every problem A in NP is p-reducible to B.
- Hardest problems in NP.
- If one of them can be solved efficiently, then every problem in NP can be solved efficiently, implying P=NP.
- If A is in NP and some NPC problem B is p-reducible to A, then A is NPC too.
  - Every problem in $NP \leq_p B \leq A$

CNF Formulas:

- Conjunctive Normal Form (CNF)
- Let $x_1, x_2, \dots, x_n$ be boolean vars.
- Let $\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n$ be their negations.
- Let L be a literal.
  A literal is a var or its negation.
- A clause is a disjunction of literals.
  E.g. $C = \ell_1 \vee \ell_2 \vee \dots \vee \ell_n$
- A CNF formula is a conjunction of clauses.
  E.g. $F = C_1 \wedge C_2 \wedge \dots \wedge C_m$
- kCNF: Each clause has at most k literals.
- Exact kCNF: Each clause has exactly k literals.

- E.g. $(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_1 \vee x_2)$ is 3CNF
  $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$ is exact 3CNF

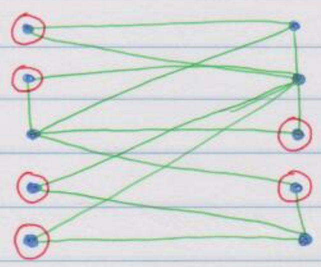
SAT and Exact 3SAT:

- A CNF formula F is satisfiable if there is an assignment of truth values (T/F) to the vars s.t. the formula evaluates to True.
  I.e. In each clause, there must be at least 1 literal that is True.
- SAT: Given a CNF formula, is it satisfiable?
- Exact 3SAT: Given an exact 3CNF formula, is it satisfiable?

Independent Set:

- Problem: Given an undirected graph $G=(V,E)$ and an int k, does there exist a subset of vertices $S \subseteq V$ with $|S|=k$ s.t. for each edge, at most one of its endpoints is in S?

- E.g.



There exists an indep set of size 6, but not 7.

- Claim: Indep set is in NP.

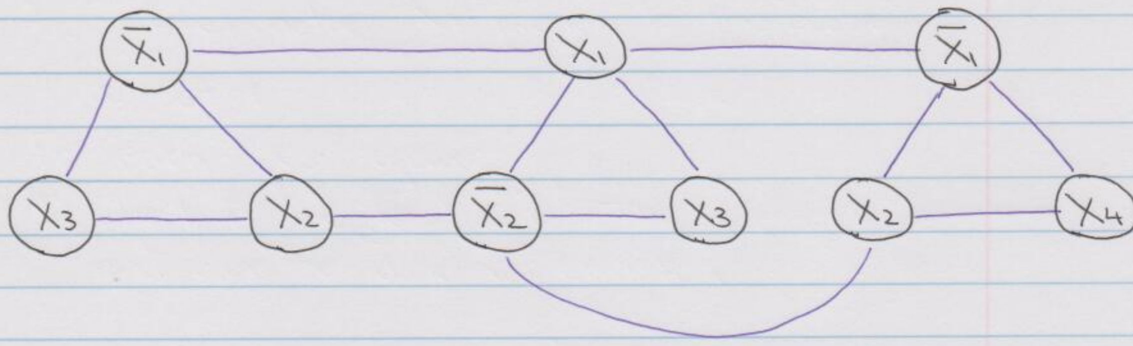
- To prove that a problem is in NP, you must do the following:
  1. Prove that a "Yes" solution works in poly time.
  2. Reduce another NP problem to this problem.

- Proof of 1:
  For each "Yes" answer,
  1. Check if the length $= k$. This is easy to do and can be done in $O(n)$ time.

  2. Check to make sure that no 2 vertices have an edge connecting them. To do this, for each node, check if there's an edge with every other node in the subset. This can be done in $O(n^2)$ time.

- Proof of 2:
  - Claim: Exact 3Sat reduces to Indep Set
  - This means that an oracle that solves Indep Set can be used to solve Exact 3Sat.
  - Construct the graph like so:
    1. Given a formula F with k clauses, we want to solve Indep Set with input (G,k).
    2. Let each clause be a triangle s.t. all 3 literals in each clause are connected with each other.
    3. Connect a literal with all of its negations in the other clauses/triangles.

- E.g.
  Formula: $(\overline{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_4)$



- Exact 3Sat → Indep Set:
  - If there is an assignment s.t. the formula evaluates to True, we know that at least 1 literal in each clause must be True.

    Choose 1 true literal from each clause. That is the indep set of size k.

- Indep Set → Exact 3Sat:
  - Set each literal to True and their negations to False.
  - We know that we can never pick both x_i and $\overline{x}_i$ because of the edge connecting them.
  - If there is an indep set of size k, then we know exactly 1 node from each clause/triangle will be selected. These will be the literals that make the entire formula True.

Different Types of Reductions:

$A \le B \rightarrow$ - Karp Reductions: Take any arbitrary instance of A and in polynomial time, construct a single instance of B with the same answer.

  Is a very restricted type of reduction.

- Turing/Cook Reductions: Take an arbitrary instance of A and solve it by making multiple polynomial calls to an oracle for solving B and some polynomial-time extra computation.

  Is a very general type of reduction.

Subset Sum:

- Problem: Given a set of integers $S = \{w_1, w_2, \dots, w_n\}$ and int w, is there a $S' \subseteq S$ s.t. the numbers in S' adds exactly to w?

- Claim: Subset sum is in NP.

  Proof:

  Given any subset of S, we can easily check if it adds up to w or not.

- Claim: Exact 3SAT $\le_p$ Subset Sum

  Given a formula F of Exact 3SAT, we want to construct (S, w) of Subset Sum with the same answer.

Consider a 3CNF formula with vars $x_1, x_2, \ldots, x_n$ and clauses $C_1, C_2, \ldots, C_m$.

We will construct a table s.t.

1. The vars and clauses make up the columns.
   (Think of each number of a column as a digit)

2. Each var x_i will have 2 numbers y_i and z_i.
   $y_i = x_i$
   $z_i = \overline{x_i}$

3. Treat each row as a number in S, represented in decimal.

4. Each y_i will have a 1 for its corresponding var and for every clause that has x_i.
   Each z_i will have a 1 for its corresponding var and for every clause that has $\overline{x_i}$.

5. There will be dummy rows to help pad the clause columns to add up to the digit in w iff at least 1 literal is set to True.

E.g. 1

$$F = \overset{C_1}{(\bar{x} \vee y \vee z)} \wedge \overset{C_2}{(x \vee \bar{y} \vee z)} \wedge \overset{C_3}{(\bar{x} \vee \bar{y} \vee \bar{z})}$$

Table:

	x	y	z	C_1	C_2	C_3
x	1	0	0	0	1	0
$\bar{x}$	1	0	0	1	0	1
y	0	1	0	1	0	0
$\bar{y}$	0	1	0	0	1	1
z	0	0	1	1	1	0
$\bar{z}$	0	0	1	0	0	1
	0	0	0	1	0	0
	0	0	0	2	0	0
	0	0	0	0	1	0
	0	0	0	0	2	0
	0	0	0	0	0	1
	0	0	0	0	0	2
ω	1	1	1	4	4	4

Notice that each row var is 1 if either the corresponding col var is the same or if it's used in a clause.

Dummy rows

E.g. 2

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$$

Table:

	x_1	x_2	x_3	C_1	C_2
y_1	1	0	0	1	0
z_1	1	0	0	0	1
y_2	0	1	0	1	1
z_2	0	1	0	0	0
y_3	0	0	1	0	0
z_3	0	0	1	1	1
	0	0	0	1	0
	0	0	0	1	0
	0	0	0	0	1
	0	0	0	0	1
ω	1	1	1	3	3

← Each row is a number in S represented by decimal / base 10.

Dummy row

Proof (Subset Sum = Yes → Exact 3SAT = Yes):
If there is a subset of numbers that add up to w, then we know the following:

1. Each y_i and z_i pair must have exactly one True value / have exactly one of them be chosen. This is because both (x_i, y_i) and (x_i, z_i) have 1, so if both y_i and z_i are chosen, then we wouldn't add up to w who has 1's for the var columns.

2. For the clause columns, they only add up to their respective w digit iff at least one literal has a value of 1.

3-Coloring:
- Problem: Given an undirected graph $G=(V,E)$, can we color each vertex of G using at most 3 colors s.t. no 2 adjacent vertices have the same color?

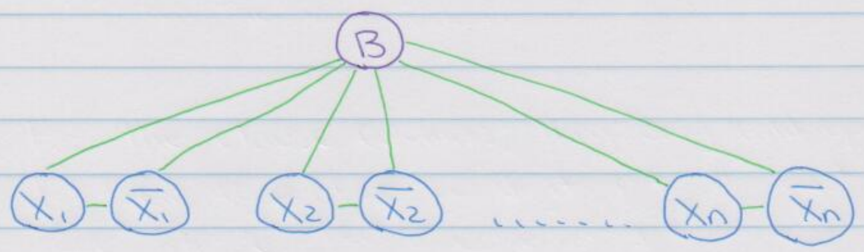
- Claim: 3-Coloring is in NP.
  Proof:
  Given any colored graph G, we can traverse it to see if any 2 adjacent nodes have the same color. If none do, return yes. Otherwise, return no.
  Worst-case time complexity: $O(n^2)$

- Claim: Exact 3SAT $\leq_p$ 3-Coloring

Proof:

Part 1:

- We'll have 3 colors, T, F and Base.
- For each var x_i, we'll create 2 nodes: x_i and $\overline{x_i}$. We'll also have a 3rd node to represent Base and we'll connect them like shown below

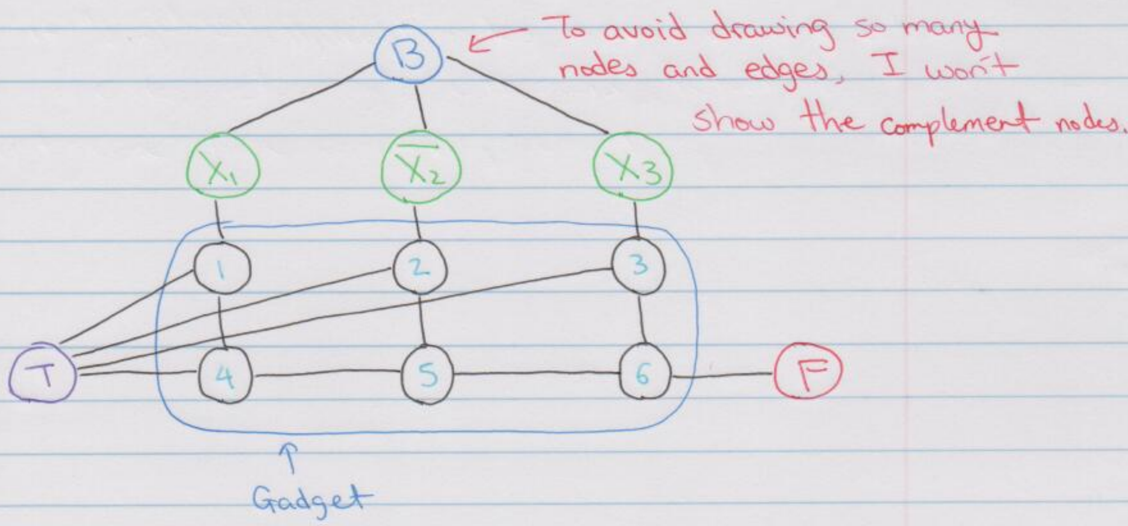


This way, exactly one of either x_i and $\overline{x_i}$ must be True and the other must be False.

Part 2:

- For each clause, we'll create the gadget shown in the example below.

- E.g. 1 $C = (x_1 \vee \overline{x_2} \vee x_3)$



Suppose $X_1 = \bar{X}_2 = X_3 = F$. Then:

1. The nodes 1, 2 and 3 can only be set to Base as they're connected to both True and False nodes.

2. Node 4 must be False as it's connected to True and Node 1, who's value is Base.

3. Node 5 must be True as it's connected to Node 4 (False) and Node 2 (Base)

4. Node 6 does not have a valid option.
   It is connected to:
   Node 3 - Base
   Node 5 - True
   False node

Now, suppose X_3 is True and X_1 and X_2 are False. Then:

1. Node 1 can be either F or B. Wlog, say it's F.
2. Nodes 2 and 3 must be B.
3. Node 4 must be B.
4. Node 5 must be F.
   If it's T, then node 6 will be connected to a T node, a B node and a F node.
5. Node 6 must be T.

Binary Integer Linear Programming (BLIP):

- Problem: Given $c \in R^n$, $b \in R^m$, $A \in R^{m \times n}$, $k \in R$ does there exist $x \in \{0,1\}^n$ s.t. $c^T x \geq k$ and $Ax \leq b$?

  Note: For simplicity, let $c = k = 0$ so that $c^T x \geq k$ is always True.

- Claim: BLIP is in NP.

  Proof:

  Given c, b, A, k, let the advice be x. Then, we can easily verify if $c^T x \geq k$ and $Ax \leq b$.

- Claim: Exact 3SAT $\leq_p$ BLIP

  Proof:

  Take any Exact 3SAT formula F and convert each var x_i in F to a binary var x_i and each negated var $\overline{x_i}$ in F to binary $(1-x)$.

  I.e.

Var in F	Binary Var in BLIP
x_i	x_i
$\overline{x_i}$	$(1-x_i)$ ← Use $(1-x)$ to rep $\overline{x}$

Use addition for "OR" →

For each clause C_i, to make sure that at least 1 of its vars is True, we check to see if the sum of its vars is at least 1.

I.e. $x_i + (1-x_2) + x_3 \geq 1$ ← Doesn't have to be in standard form.

Use multiplication for "And" →

To make sure all clauses are True, we multiply them and see if their product ≥ 1.

Easy to see that F is True iff the system has a feasible soln.

Integer Linear Programming:

- Problem: Given $b \in R^m$ and $A \in R^{m \times n}$, does there exist $x \in Z^n$ s.t. $Ax \leq b$?

- We can easily see that we can reduce BLIP to ILP. This means ILP is NP-hard.
  I.e. BLIP $\leq_p$ ILP

- However, it is not clear if for every YES instance that there's a polynomial-length advice vector x satisfying $Ax \leq b$.

Vertex Cover:

- Problem: Given an undirected graph $G=(V,E)$ and int k, does there exist $S \subseteq V$ with $|S|=k$ s.t. every edge is connected to at least 1 vertex in S.

- Claim: Vertex Cover is in NP.
  Proof:
  Let the advice be S. Then, we can easily verify if $|S|=k$ and if every edge is connected to a node in S.

- Claim: G has a VC of size k iff G has an Indep Set of set $n-k$. $n=|V|$
  Proof:
  #1 (VC $\Rightarrow$ Indep Set):
  - Suppose G has a VC of size k. Call this set S.
  - This means that every edge is connected to a node in S.
    - This means that each edge is connected to any of the remaining $n-k$ nodes at most once.
      I.e. No edge can be connected to 2 nodes in the $n-k$ or $V \setminus S$ set.
    - This is the def of indep set. The num of nodes is $n-k$.

  #2 (Indep Set $\Rightarrow$ VC):
  - Suppose G has an indep set of size $n-k$.
  - We know each edge must be connected to those $n-k$ nodes at most once.
  - Hence, all edges will be connected to the remaining k nodes at least once.
  - This is the def of VC. The num of nodes is k.

- Claim: S is a VC iff $V \setminus S$ is an indep set.
  Proof is similar to last page's.

- Claim: Indep Set $\le_p$ VC
  Proof:
  Given $G=(V,E)$ and int k for Indep set, try to create an instance of VC.
  Let $(G, n-k)$ be the instance for VC.
  We've already shown the proof.

Set Cover:

- Problem: Given a universe of elements U, a family of subsets S and an int k, does there exist k sets from S whose union is U?

- E.g.
  $U = \{1, 2, 3, 4, 5, 6, 7\}$
  $S = \{\{1,3,7\}, \{2,4,6\}, \{4,5\}, \{1\}, \{1,2,6\}\}$
  $k = 3 \rightarrow$ Yes $(\{1,3,7\}, \{4,5\}, \{1,2,6\})$
  $k = 2 \rightarrow$ No

- Claim: Set Cover is in NP.
  Proof:
  Let the advice be the k sets. Checking to see if all k sets are in S and if their union is U can be done in quadratic time at worst.

- Claim: VC $\le_p$ Set Cover

Proof:

Given an instance of VC, (G,k), we want to create an instance of Set Cover that solves the problem.

Let $U = E$

For each $v \in V$, S contain set S_v that has all the edges connected to v.

See if the union of k sets $= U$.

Set Cover $\rightarrow$ VC Proof:

- If we can find k sets whose union is U from the creation above, then that means there are k nodes in S s.t. every edge is connected to at least one of them.

SAT $\le_p$ Exact 3SAT:

Proof:

Given a CNF formula F for SAT, we want to create an Exact 3CNF formula F' that solves it.

Let $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$

For each clause C_i,

1. If C_i has 1 literal, add 2 new vars z_1 and z_2 and replace C_i with $(l_1 \vee z_1 \vee z_2) \wedge (l_1 \vee \bar{z}_1 \vee z_2) \wedge (l_1 \vee z_1 \vee \bar{z}_2) \wedge (l_1 \vee \bar{z}_1 \vee \bar{z}_2)$

2. If C_i has 2 literals, add 1 new var z_1 and replace C_i with $(l_1 \vee l_2 \vee z_1) \wedge (l_1 \vee l_2 \vee \bar{z}_1)$

3. If C_i has 3 literals, we do nothing.

4. If C_i has more than 3 literals then add vars $z_1, \ldots, z_{k-3}$ and replace C_i with

$$(\ell_1 \vee \ell_2 \vee z_1) \wedge (\ell_3 \vee \overline{z_1} \vee z_2) \wedge \ldots \wedge (\ell_{k-2}, \overline{z_{k-4}}, z_{k-3}) \wedge (\ell_{k-1} \vee \ell_k \vee \overline{z_{k-3}})$$