NPC

To show X is NPC

1. $X \in NPC$, show that there is an efficient verifier for X
2. X is NP-hard, show $Y \leq_q X$ for some NP-hard problm Y

Example. Subset-sum(SS) Given a finite set S of positive integers and a positive integer t, ask if there is a subset $S' \subseteq S$ such that $\sum_{x \in S} x = t$. Prove that subset sum problem is NP-complete

Proof. 2 steps

1. Prove SUBSET-SUM$\in NPC$ Given a subset sum problem $X = (S, t)$. Let certificate c be a subset $S' \subseteq S$. Let $B(X, c)$ be such that

$$B(X,c) = \begin{cases} yes & \text{if } \sum_{x \in c} x = t \\ no & \text{otherwise} \end{cases}$$

   Note B can be implemented by summing up all $x \in c$ and return the sum, which will finish in polynomial time. Hence B is an efficient verifier of X, hence SUBSET-SUM$\in NPC$

2. Prove SUBSET-SUM is NP-hard by proving $3-SAT \leq_q SS$. Start with some input to 3-SAT, and modify the input with function f and make it an input $f(x)$ to SS. Proving $3-SAT \leq_q SS$ is equivalent to prove that $3-SAT$ says yes on x if and only if SS says yes on $f(x)$ and f is a polynomial time operation. Let ϕ b input to 3-SAT

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge \cdots \wedge (a_r \vee b_r \vee c_r)$$

   where $a_i, b_i, c_i \in \{x_1, \cdots, x_s\}$. In summary, there are r clauses and s variables. Let integer $\hat{x}_j = 1$ followed by $s-j$ 0s followed by r digits where kth next digit equals 1 if x_j appears in clause c_k, 0 otherwise. and define $\neg\hat{x}_j$ be same as previous definition but when $\neg x_j$. For $j = 1 \to r$ integer $C_j = 1$ followed by $r-j$ 0s and integer $D_j = 2$ follwed by $r-j$ 0s $t = s$ 1s followed by r 4s
   Example of $r = 3$ clauses and $s = 4$ variables

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_4) \wedge (x_2 \cup \neg x_3 \cup x_1) \wedge (\neg x_3 \vee x_4 \vee \neg x_2)$$

Now we define S

$$
\begin{aligned}
S = \{ & \\
& \hat{x}_1 = 1000110, \neg\hat{x}_1 = 1000000, \\
& \hat{x}_2 = 100010, \neg\hat{x}_2 = 100101, \\
& \hat{x}_3 = 10000, \neg\hat{x}_3 = 10011, \\
& \hat{x}_4 = 1001, \neg\hat{x}_4 = 1100, \\
& C_1 = 100, C_2 = 10, C_3 = 1 \\
& D_1 = 200, D_2 = 20, D_3 = 2 \\
& t = 1111444\}
\end{aligned}
$$

Th operation takes polynomial time. now we show ϕ is satisfiable if and only if S has a subset whose sum is t

(a) => Assume ϕ satisfiable, implying each clause must have one true literal (variable). Let

$$S' = \{\text{numbers that correspond to the literals as constructed}\}$$

By construction, choose S' such that $\sum_{x \in S'} x = 1111 d_1 d_2 d_3$ where $1 \leq d_i \leq 3$ (Since at least 1 true varaible, and and at most 3 variables in a clause) Now we can select $S'' \subseteq \{C_i, D_i\}$, such that $\sum_{x \in S' \cup S''} x = t$

(b) <= There is some subset $S' \subseteq S$ such that $\sum_{x \in S'} x = t$ The truth assignment, $T(x_i)$ is true if $\hat{x}_i \in S'$ and $T(x_i)$ is false if $\neg\hat{x}_i \in S'$. The sum of all C_is and D_is is 333. So if $\sum_{x \in S'} x = t = 1111444$ which means we have some 1 in each of the 3 least significant bits. Then there is a truth literal in that clause, so every clause is true, and ϕ is true

□

Theorem. *If D is NPC, then $D \in P \iff P = NP$*

Proof. prove if and only if

1. => $D \in P$ let $D' \in NP$ Show $D' \in P$ D is NPC, implies $D' \leq_q D$, implies $D' \in P$
2. <= D is NPC implies $D \in NP = P$ implies $D \in P$

□

Categories of NPC problems

1. **Packing problem**
   (a) independent set
   (b) set packing
2. **Covering problem**
   (a) vertex cover
   (b) set cover
3. **Partition problem**
   (a) 3-dimensinoal matching
   (b) graph coloring
4. **Sequencing problem**
   (a) Hamiltonian cycle
   (b) Hamiltonian paht
   (c) traveling salesman
5. **Numerical problems**
   (a) subset sum
6. **Constraint satisfaction problems**
   (a) 3-SAT
   (b) CNF-SAT
   (c) SAT
   (d) Circuit-SAT

Self-Reducible Problems

Search problem, find a solution
optimization problem, find the best prossible solution

Definition. ***Search problem*** *A search problem is self-reducible if any efficient solution to the corresponding decision problem can be used to solve the search problem efficiently. Search problems are typically harder than the corresponding decision problem.*

Definition. ***Independent-Set-Decision***(G, k)

Example. Hamiltonian cycle

1. problem is self-reducible Let HCS be the search problem and HCD be the decision problem.

2. **Correctness** $HCD(G')$ remains true for every state. So at the end, E contains every edge in hamiltonian cycle of G. Moreover, we remove every other edge so we are left with only a hamiltonian cycle at the end.

3. **Complexity** Assume the time complexity of decision problem is $t(V, E)$. Then the complexity of the search problem is $O(t(V, E) + E(1 + t(V, E))) = O((E + 1)t(V, E))$

```
1  Function Hamiltonian-cycle-search(G)
2      if ! Haniltonian-cycle-decision(G) then
3          return
4      NIL
5  for e ∈ G.E do
6      E' ← E \ {e}
7      G' ← (V, E')
8      if Haniltonian-cycle-decision(G') then
9          E ← E'
10 return E
```

Example. Vertex-Cover

1. **Complexity** $O((V + 1)t(V, E) + V \times (V + E))$

```
1  Function Vertex-Cover-Search(G, k)
2      if ! Vertex-Cover-Decision(G, k) then
3          return NIL
4      C ← {}
5      for v ∈ V do
6          if k ≤ 0 then
7              break
8          V' ← V \ {v}
9          E' ← E \ {(v, w)|(v, w) ∈ E}
10         G' ← (V', G')
11         if Vertex-Cover-Decision(G', k − 1) then
12             v ← v'
13             E ← E'
14             C ← C ∪ {v}
15             k ← k − 1
```

Optimization Problem

Independent set, given G find an independent set of maximum size. use binary search to find left/right partition with independent-set-decision

handling intractable problems

It is not possible to solve NP-hard problems exactly and in polynomial time (unless $P = NP$)

Definition. ***Approaches***

1. ***Sacrifice efficiency*** *NP-hard problems, generic problems that accounts for worst case problems. In reality, the inputs may be much better. For example find independent set on a tree*
2. ***Sacrifice exactness*** *For optimization problems, it is okay to find a good enough solution compared to the best possible solution. We can use approximation algorithms.*

Example. Vertex-cover

1. Find a set of vertices of minimum possible sizse of every edge in G passes through one of these vertices.
2. claim $|S| \leq 2 \times OPT$

   Proof. Any vertex cover of G must include at least one of the two vertices connecting an edge in S. Also, in S, all edges are disjoint, i.e. with no end points in common. so $OPT \geq |S|/2$ □

3. In general, how can we comput approximation ratio if we dont know what the optimal solution is? use a lower bound instead. For other value LB that is easy to compute, show that $LB \leq OPT$ and solution $A \leq ratio \times LB$

```
1 Function
2     S ← {}
3     for e = (u, v) ∈ E do
4         S ← S ∪ {v, u}
5         E' ← {(u, w)|(u, w) ∈ E} ∪ {(v, w)|(v, w) ∈ E}
6         E ← E'
7     return S
```