Introduction

- The algorithms we studied so far are examples of *polynomial-time algorithms*.
  - If their inputs are of size n, their running time is $O(n^r)$ for some positive integer r.
- Not every algorithm is of polynomial-time type.
  - There are algorithms of exponential-time type.
  - For instance, recursive algorithms for solving Towers of Hanoi problem.

Definition

The class P of problems is a class of problems that can be solved in polynomial time, that is, if input data size is n, then there exists at least one algorithm for which worst-case running time is $O(n^r)$. Such problems are considered as *tractable*.

Introduction (cont)

- If for instance $r = 100$ then such a problem is difficult to solve in a reasonable time for larger n.
  - But very few practical problems require time on the order of such high-degree polynomial.
  - Usually $r < 10$.
- The class of polynomial-time solvable problems has closure properties, since polynomials are closed under addition, multiplication and composition.
  - For example, if the output of one polynomial-time algorithm is an input of another, the composite algorithm is still polynomial-time.

NP and NP-Complete Classes

- Sometimes we are given data which we need to verify (check) whether they are a solution of a problem.
  - Usually verification time is *smaller* than solution time for a given problem.
- We are mostly interested in problems for which there exist *verification algorithm* of polynomial time with respect to the number of input data.
  - Often such problems have *non-polynomial solution time*.

Definition

The class NP of problems is a class of problems whose solutions can be verified in polynomial time.

NP and NP-Complete Classes (cont)

- The name "NP" stands for "*nondeterministic polynomial time*."
  - Such problems were (theoretically) solved in polynomial time on (also theoretical) nondeterministic computers.
  - The computers we currently know are only deterministic.

NP and NP-Complete Classes (cont)

- As an example of a simple verification algorithm we can consider an algorithm which verifies if a given input data is sorted in ascending order.

```
/* return: false = if not sorted, true = if sorted */
int verify_sort(int *a, int n)
{
  for (int i = 0; i < n-1; i++)
    if (a[i] > a[i+1]) return false;
  return true;
}
```

- As we see this verification algorithm is simpler than any sorting algorithm
- Its running time is only $O(n)$.
- The class NP contains much more difficult problems than this above.

NP and NP-Complete Classes (cont)

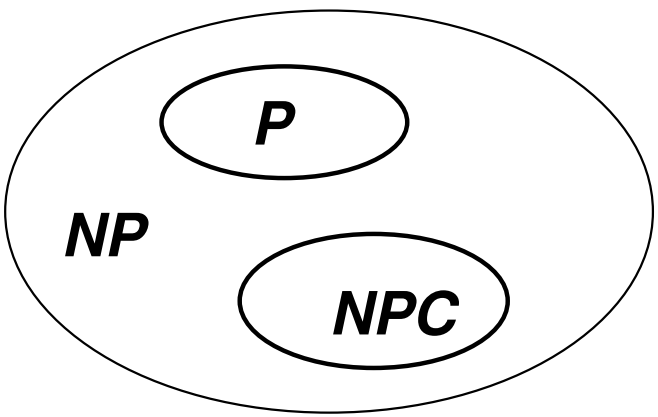
Definition (Informal)

The class NP-Complete (NPC) is a class of "hardest" problems in the NP class having property that if **any** NPC problem can be solved in polynomial time then **all** problems in the NP class could be solved in polynomial time.

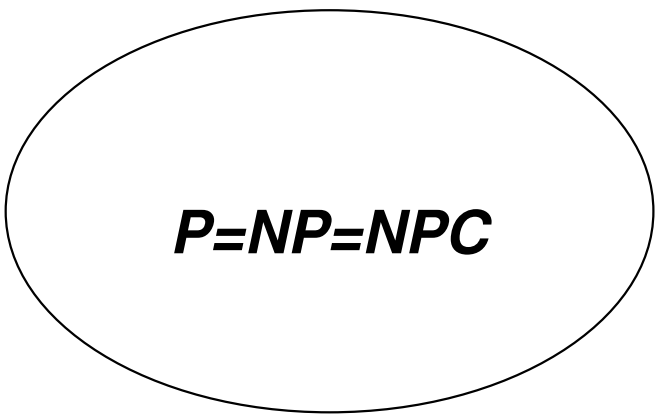
- We will try to give a more formal definition of the NPC class later.

Relation Between P and NP Classes

- The most important (and interesting) question in computer science is: *what is a relation between the P class and NP class of problems*?
- We know that $P \subseteq NP$. And from the definition of NPC we know that $NPC \subseteq NP$. But we still do not know if $NP = P$ or $NP \neq P$.



If P not equal to NP



If P equal to NP

Relation Between P and NP Classes (cont)

- Our intuition tells us that $NP \neq P$, but no one has proved it yet.
- Is it important to know whether a problem is NP-Complete?
  - Yes, since if a problem is NP-Complete, then we cannot expect to find a polynomial-time algorithm to solve it.
  - And then we look for an efficient *approximation algorithm* which can find a solution close to the optimal (or exact) solution.
  - Or we can look for a *heuristic algorithm*, which can solve the problem for many (input data) cases, but for some cases it gives up.

Decision Problems

- The theory of NP-completeness restricts attention to *decision problems*: those having a yes/no solution.
- Any general problem can be replaced by an equivalent decision problem,but sometimes we need to add an additional parameter (such as k below).

Decision Problems (cont)

Example

- Consider the shortest-path problem. The general problem can be formulated as: *given a graph $G = (V, E)$, and two vertices $u, v \in V$, what is the length of a shortest path from u to v*?
- The decision problem (equivalent to the general one) is formulated as follows: *given a graph $G = (V, E)$, two vertices $u, v \in V$, and a nonnegative integer k, does a path exist in G between u and v whose length is at most k*?
  - k here is necessary to formulate the decision problem

- Further we restrict our attention only to decision problems. This is not restrictive, since solving general problems is at least as hard as solving equivalent decision problems.

Traveling Salesman Problem (TSP)

- Suppose that we have an undirected graph with weighted edges.
- We will think of such a graph as having vertices that represent cities, and having edges connecting pairs of cities whose weights give the cost of traveling between those cities.
- The problem is to find a circuit of least cost that visits all the cities in the graph exactly once and returns to its starting point.
  - This is the route that a traveling salesperson might follow to visit all of the cities at the least possible expense.
- To formulate this as a decision problem we need to add a parameter k and ask the following question: *Is there a route that visits every city exactly once, returns to the start, and has the total distance* $\leq k$*?*

Traveling Salesman Problem (TSP) (cont)

- To check if TSP $\in$ NP we need to prove that a given route can be verified in a polynomial time. Assume that we have n cities. We need to check that
  - the route visits every city exactly once
  - the route returns to the start
  - the total distance $\leq k$
- Since all of these can be done in $O(n)$ time, so TSP $\in$ NP.

Traveling Salesman Problem (TSP) (cont)

- To be sure that the decision problem is equivalent to the general TSP problem we show how to get the solution of the general problem using the TSP decision problem solution (which is true or false).
- Denote this decision problem solution by D-TSP(k).
  1. set $k = c$ (c is a small constant);
  2. call the TSP decision problem;
  3. repeat
     1. if (D-TSP(k))
     2. then decrease k and call the TSP decision problem;
     3. else increase k and call the TSP decision problem;
  4. until the problem converges to the actual minimum value (given by the current value of k).

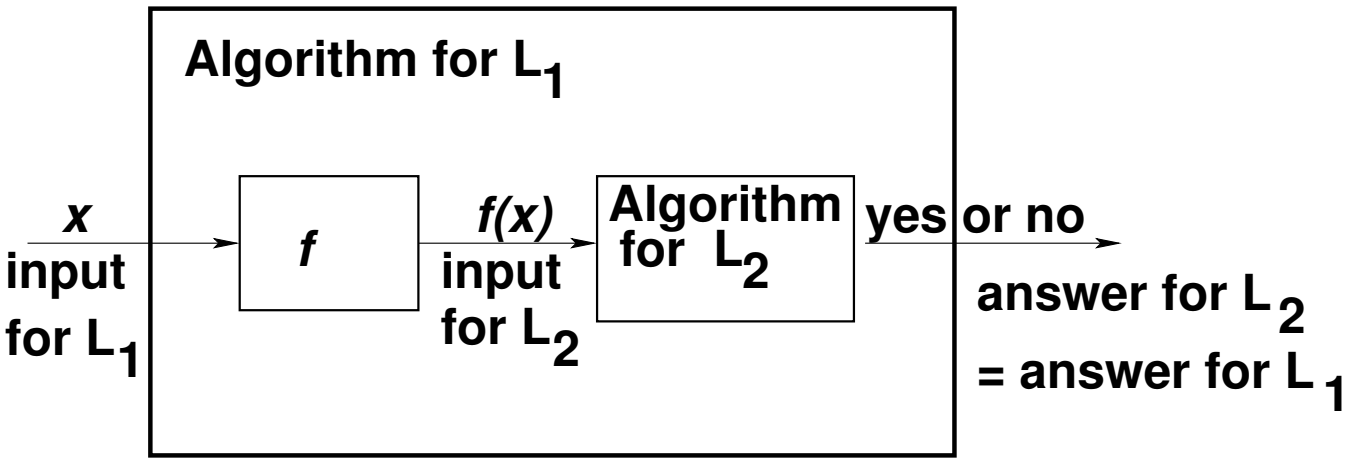
Reducibility

Let L_1 and L_2 be two decision problems.

Definition

We say that the problem L_1 is *polynomial-time reducible* (or simply reducible) to a problem L_2, written $L_1 \leq_P L_2$, if there exists a polynomial-time computable function f transforming input for L_1 into an input for L_2 such that the transformed input for L_2 generates a yes (true) result if and only if the original input generates a yes (true) result for L_1.

Reducibility



Theorem

If $L_1 \leq_P L_2$ and $L_2 \in P$, then $L_1 \in P$ (if L_1 is reducible to L_2 and L_2 can be solved in polynomial time, then L_1 also can be solved in polynomial time).

Hamiltonian Circuit (HC)

- A *Hamiltonian circuit* (*HC*) is a cycle (closed path) in an undirected graph $G = (V, E)$ of n vertices that travels through each of the vertices exactly once and returns to its starting vertex.
- We can prove that HC $\leq_P$ TSP.
- To do this we need to define a transformation f mapping inputs to HC into inputs to TSP and show that it can be computed in polynomial time.
  - To be more specific, f must map the input $G = (V, E)$ for HC into a list of cities, distances and a bound k for input to TSP.
  - Moreover, f must be computable in polynomial time in size of HC input graph.

Hamiltonian Circuit (HC) (cont)

- We also need to show that the transformation f is correct:
  - show that f transforms inputs for HC generating "yes" answer into inputs for TSP generating also "yes" answer
  - show that f transforms inputs for HC generating "no" answer into inputs for TSP generating "no" answer
- We can say that G has a Hamiltonian circuit if and only if transformed input has a TSP route with length $\leq k$.

Reducing HC To TSP

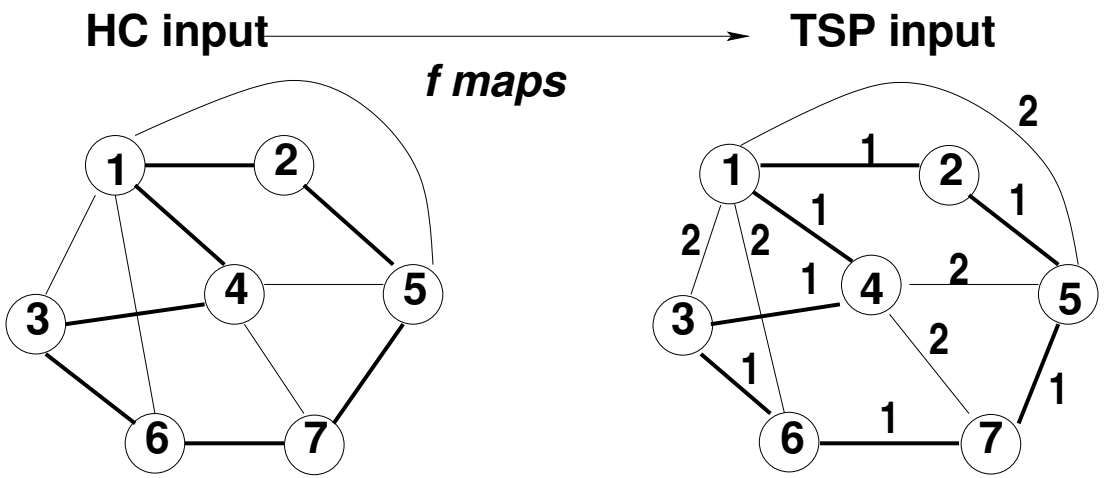
We need to define a transformation f which transforms HC input (input graph $G = (V, E)$) into TSP input.

- create a set of n cities labeled with names in V
- set intercity distances $d(u, v)$:

$$d(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \text{ and } (u,v) \in HC \\ 2 & \text{if } (u,v) \in E - HC \end{cases}$$

- set bound $k = n$

The above f transformation is computed in $O(n^2)$ time



Correctness of the Transformation

We need to show that the transformation f for HC $\leq_P$ TSP is correct.

1. We show that if $x \in$ HC then $f(x) \in$ TSP.
   - $x \in$ HC means that the HC input $G = (V, E)$ has a Hamiltonian circuit ("yes" answer)
   - Suppose that it is $(v_1, v_2, \ldots, v_n, v_1)$. It is also a route of the cities in $f(x)$ (TSP input).
   - The distance of the route is $n = k$, since each consecutive pair (v_i, v_{i+1}) belongs to Hamiltonian circuit in E and has distance 1 in $f(x)$.
   - The answer for TSP with input $f(x)$ is "yes." So $f(x) \in$ TSP.

Correctness of the Transformation (cont)

2. We show that if $f(x) \in$ TSP then $x \in$ HC.
   - $f(x) \in$ TSP means that there exists a route in TSP input that has total distance $\leq n = k$.
   - Assume that it is $(v_1, v_2, \ldots, v_n, v_1)$.
   - Since all intercity distances are either 1 or 2 in $f(x)$, and the total distance is n, then each distance in the route must be equal to 1.
   - The route edges must belong to E, and therefore $(v_1, v_2, \ldots, v_n, v_1)$ is a Hamiltonian circuit in G.
   - Thus $x \in$ HC.

Definition of NP-Completeness

Definition

A decision problem L is *NP-Complete* (NPC) if

1. $L \in NP$, and
2. for every $L' \in NP$ we have $L' \leq_P L$ (that is, every problem L' in NP can be reduced to L, and therefore L is at least as hard as every problem in NP).

- If L satisfies only condition 2, we say that this problem is *NP-Hard*.

NP-Completeness

Theorem

Let $L \in NPC$. Then

- *if there exists a polynomial-time algorithm for L, then there exist polynomial-time algorithms for every $L' \in NP$, that is, $P = NP$.*
- *if there does not exist a polynomial-time algorithm for L, then there does not exist a polynomial-time algorithm for any $L' \in NPC$, that is, $P \neq NP$.*

How To Show a Problem Is NP-Complete

- To show that a problem L is NP-Complete we need to
  1. show that $L \in$ NP
  2. select a known NP-Complete problem L' and show that $L' \leq_P L$.
- We prove that the above is sufficient for $L \in$ NPC.
  - Since $L' \in$ NPC, then for all $L'' \in$ NP we have $L'' \leq_P L'$ (by definition of NPC).
  - Since $L' \leq_P L$, then transitivity of the relation $\leq_P$ gives $L'' \leq_P L' \leq_P L$.
  - Then $L'' \leq_P L$ for all $L'' \in$ NP.
  - By definition of NPC we get that $L \in$ NPC.

How To Show a Problem Is NP-Complete (cont)

- To use this proof technique we need to know at least one problem which is NP-Complete. And the proof for such a particular problem cannot use the above technique.
- Fortunately, we know such NPC problems.

Satisfiability Problem (SAT)

- One of such problems is the satisfiability problem (SAT), which can be formulated as follows: Consider a boolean expression, for instance,

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

  Does there exist a set of boolean values (0 or 1) for (x_1, x_2, x_3, x_4) such that this expression is true (evaluated to 1)?
- Such a boolean set (if exists) is called *satisfying truth assignment*, and the above expression is an example of a *satisfiable formula*.

Satisfiability Problem (SAT) (cont)

- For this formula we can show that it has the satisfying assignment $(0,0,1,1)$, since

$$\begin{aligned}\phi &= ((0 \to 0) \vee \neg((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0 \\ &= (1 \vee \neg(1 \vee 1)) \wedge 1 \\ &= (1 \vee 0) \wedge 1 \\ &= 1.\end{aligned}$$

Theorem

The Satisfiability problem (SAT) is NP-Complete.

Classic NP-Complete Problems

We know many NP-Complete problems (more than a hundred). There are some NP-Complete problems which are now considered as classic.

- **The 3-Satisfiability Problem (3-SAT):** it is a boolean expression of the following form:

$$(x_1 \vee x_2 \vee \neg x_1) \wedge (x_3 \vee x_4 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

  - This formula is in conjunctive normal form (CNF), since it is expressed as an AND of clauses, each of which is the OR of one or more literals (a literal = a variable or its negation).
  - In this case each clause has exactly three distinct literals, therefore it is called 3-conjunctive normal form (3-CNF).
  - The question for this problem is: *does there exist a satisfying truth assignment for a given 3-CNF formula?*

Classic NP-Complete Problems (cont)

- **The vertex cover problem (VC):** consider an undirected graph $G = (V, E)$ and a positive integer $k \leq |V|$.
  - A vertex cover of G is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ (or both).
  - That is, each vertex "covers" its edges (edges having this vertex as the beginning or end), and a vertex cover for G is a set of vertices that covers all the edges in E.
  - The question for this problem is: *Is there a subset $V' \subseteq V$ of size $\leq k$ such that each edge in E has at least one endpoint in V'?*
  - Or in other words, *is there a vertex cover of size $\leq k$ in G?*

Classic NP-Complete Problems (cont)

- **The Clique Problem (Clique):** consider an undirected graph $G = (V, E)$ and a positive integer $k \leq |V|$.
  - A *clique* in G is a subset $V' \subseteq V$ of vertices, **each** pair of which is connected by an edge in E (a complete subgraph of G).
  - The size of a clique is the number of vertices it contains.
  - The question in this problem is: *Does G have a clique of size* $\geq k$*?*
- **The Independent Set Problem (IS):** consider a graph $G = (V, E)$ and a positive integer $k \leq |V|$.
  - An *independent set* of G is a subset $V' \subseteq V$ of vertices such that each edge in E has at most one vertex in V'.
  - In other words, for each pair of vertices from V' there is no edge from E.
  - The question for this problem is: *Does G have an independent set of size* $\geq k$*?*