Chapter 9

Computational Complexity and Intractability: Introduction to the Theory of NP

- 복잡한 정도에 따른 문제들의 분류

Theory of NP

- Polynomial-time algorithm
  - if W(n) ∈ O(p(n)) where p(n) is a polynomial
- Find an efficient algorithm (Polynomial-time Alg.)
  - no efficient algorithm is known
  - no proof showing that efficient algorithm is impossible ➔ use some approximation solution
- 3 Categories of problems
  - (1) a polynomial-time algorithm is known
  - (2) proved that no polynomial-time algorithm exists (intractable problem)
  - (3) no polynomial-time algorithm is known, but not proved to be intractable
  - *Note) Most problems are category (1) or (3).*

Pseudo-polynomial Time Algorithm

- Is n input or input size ?
  - sorting, 0-1 Knapsack : input size *(amount)*
  - prime(n), fib(n) : input *(magnitude)*
- Definition of input size (입력 양의 크기)
  - the number of characters to write the input
- Example: prime(n)
  - $W(n) \in \Theta(n^{1/2})$
    (polynomial in terms of the magnitude)
    n is the magnitude (입력 값의 크기) in the input
  - input size $s = \lg n$ that is, $n = 2^s$
    ⇨ $W(s) \in \Theta(2^{s/2})$ *(exponential in terms of the size)*

✓ **Pseudo-polynomial Time Algorithm**
- ✓the worst-case time complexity is polynomial in terms of size and magnitude

Decision Problems

✓ Decision Problem
- ✓Problems simplified from optimization problems
- ✓The output is yes or no

✓ Examples
- ✓Traveling salesperson problem + given tour length
- ✓0-1 Knapsack problem + given profit
- ✓Graph coloring problem + given number of colors

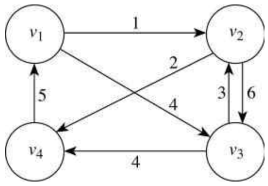
✓ Relationship
- ✓A polynomial-time algorithm of an optimization problem
- ✓→ a polynomial-time algorithm of the corresponding decision problem *(obvious)*
- ✓For many problems, the reverse relationship is also proven

P and NP

- Definition
  - **P**: the set of all **decision problems** that can be solved by **P**olynomial-time algorithms
  - **NP**: the set of all **decision problems** that can be solved by ***N****on-deterministic* **P**olynomial-time algorithms
- Non-deterministic algorithm
  - **Non-deterministic guessing**: non-deterministically produce some string S
  - **Deterministic verification**: deterministically verify S whether S is the answer or not
  - *Note) non-deterministic guessing is not realistic*
- Non-deterministic Polynomial-Time Algorithm:
  - non-deterministic algorithm whose verification stage is a polynomial-time algorithm
  - *Note) Any polynomial time algorithm is non-deterministic polynomial time algorithm*

Non-deterministic Decision Algorithm

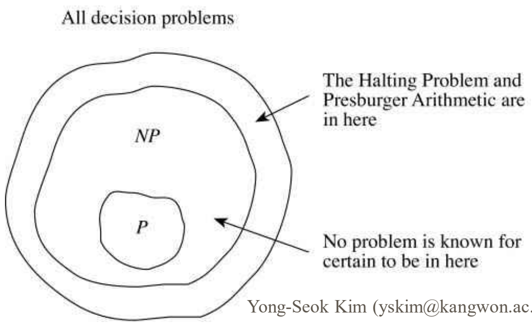
- Non-deterministic decision algorithm
  - \- the answer is "yes" if **there is some verified S**
  - \- the answer is "no" if **no such S exists**
- Traveling salesperson decision problem
  - Problem: There exists a tour of length less than 15?
  - Guessing: select any string S
  - Verification: "yes" if S is a tour whose length is
  - less than the given length, "no" otherwise
  - Fig. 9.2 → "yes"



Guessing	Verification	Reason
[v1, v2, v3, v4, v1]	No	length > 15
[v1, v4, v2, v3, v1]	No	not a tour
#$@ABX12	No	not a tour
[v1, v3, v2, v4, v1]	Yes	

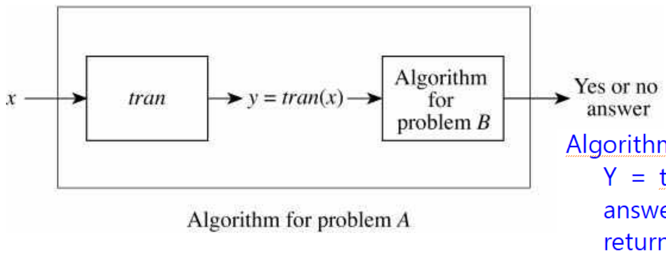
Set of Decision Problems

- ✓Traveling salesperson decision problem ∈ NP
  - ✓← exist a polynomial time verification algorithm
- ✓Thousands of known NP decision problems
- ✓Relationship Between P and NP
  - ✓P ⊆ NP ? (true)
  - ✓P = NP ? (not proven until now)



Problem Transformation

- ✓Problem transformation (Fig. 9.4)




```
AlgorithmA (X) {
    Y = tran(X);
    answer = AlgorithmB(Y);
    return answer;
}
```

➔ $T_A(n) = T_{tran}(n) + T_B(n)$

➔ *tran() 이 polynomial time 이고, AlgorithmB() 도 polynomial time이면 AlgorithmA 도 polynomial time*

➔ *tran() 이 polynomial time 이고 problem B 가 P problem 이면 A 도 P problem (Thm 9.1)*

➔ *tran() 이 polynomial time 이고 problem B 가 NP problem 이면 A 도 NP problem*

Transformation Example

- Example
  - A: At least one of given n variables is true?
    OR(B1, B2, ..., Bn) = TRUE ?
  - B: The largest of given n integers is positive?
    max(I1, I2, ..., In) > 0 ?
- Algorithm A

```
AlgorithmA(n, B[1..n])
{   index k; int I[1..n]; boolean answer;
    // transform
    for (k=1; k <= n; k++)
        if (B[k] == TRUE) I[k] = 1;
        else              I[k] = 0;
    // solve A using B
    answer = AlgorithmB(n, I[]);

    return answer;
}
```

- $T_A(n) = T_{tran}(n) + T_B(n)$

문제 B에 대한 polynomial time 알고리즘이 존재하면
문제 A에 대한 polynomial time 알고리즘도 존재한다.

Polynomial-Time Many-One Reducible

- Definition
  - A is polynomial-time many-one reducible to B if there exists a *polynomial time transformation* algorithm from decision problem A to decision problem B
  - Notation: A ∝ B, say A reduces to B
  - *Note) many-one: many instances of A can be mapped to one instance of B*
- Theorem 9.1: If decision problem B is in P and A∝B then A is in P

Example

✓ Decision problem A and B

A: At least one of given n variables is true?

OR(B1, B2, ..., Bn) = TRUE ?

B: The largest of given n integers is positive?

max(I1, I2, ..., In) > 0 ?

✓ Proof of A $\propto$ B

- ✓ Algorithm of A
- ✓ $T_{tran}(n) = \Theta(n)$

✓ $T_A(n) = T_{tran}(n) + T_B(n)$

```
AlgorithmA(n, B[1..n])
{   index k; int I[1..n]; boolean answer;
    // transform
    for (k=1; k <= n; k++)
        if (B[k] == TRUE) I[k] = 1;
        else              I[k] = 0;
    // solve A using B
    answer = AlgorithmB(n, I[]);

    return answer;
}
```


Example

✓ Decision problem A and B

A: At least one of given n variables is true?

OR(B1, B2, ..., Bn) = TRUE ?

B: The largest of given n integers is positive?

max(I1, I2, ..., In) > 0 ?

✓ B $\propto$ A ?

✓ Algorithm of B ?

✓ $T_{tran}(n)$ = ?

```
AlgorithmB(n, I[1..n])
{ index k; boolean B[1..n]; boolean answer;
  for (k=1; k <= n; k++)
      if (I[k] > 0) B[k] = TRUE;
      else          B[k] = FALSE;
  answer = AlgorithmA(n, B[]);
  return answer;
}
```

✓ Many-one ?

I1[] = {2, 3, 0, -1} ⇨ B[] = {T, T, F, F}

I2[] = {5, 4, -1, 0} ⇨ B[] = {T, T, F, F}

NP-Complete

- Definition

  Problem A is **NP-complete** if

  (1) A is in NP and (2) for every other problem B ∈ NP, B ∝ A
- (Cook's Theorem) CNF-Satisfiability is NP-Complete
  - *proof)* by Stephen Cook in 1971
- (참고) CNF-Satisfiability Decision Problem
  - 주어진 CNF 가 참이 되는 경우가 있으면 yes 아니면 no
  - CNF (Conjunctive Normal Form) example

    $$A = (x_1 \vee x_2) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_2})$$

    A is true if x_1 = *true*, x_2 = *false*, x_3 = *false*

    ⇨ A is satisfiable
- NP = P ?

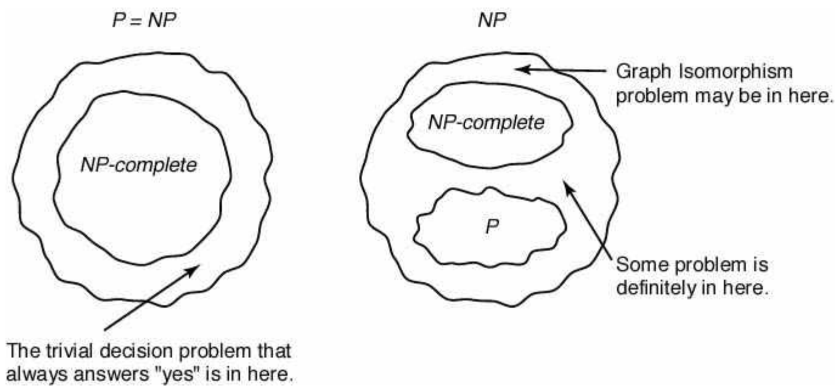
  If an NP-complete problem is proven to be in P, all NP problems are in P

Proof of NP-Completeness

- (Thm 9.3) Problem A is NP-Complete if
  - 1. A is in NP and
  - 2. for some other NP-complete problem B, B ∝ A
- Proof Examples
  - (Ex. 9.9) CNF-Satisfiability ∝ Clique Decision problem
  - (Horowitz and Sahni 1978) CNF-Satisfiability ∝ Hamiltonian Circuit Decision problem
  - (Ex. 9.10) Hamiltonian Circuit Decision problem ∝ Traveling Salesperson (Undirected) Decision problem
  - (Ex. 9.11) Traveling Salesperson (Undirected) Decision problem ∝ Traveling Salesperson Decision problem
- NP-Complete includes
  - …

P, NP, and NP-Complete

- Fig. 9.7 (Decision problem 들에 대하여)
  - *Note) If any NP-complete problem is proven to be in P, NP = P*



NP-Hard Problem

- Def) A is polynomial-time Turing reducible to B
  - if problem A can be solved in polynomial time using a hypothetical polynomial-time algorithm for problem B
  - (Notation: $A \propto_T B$, say A Turing reduces to B)
- Def) A problem B is **NP-hard**
  - if for some NP-complete problem A, $A \propto_T B$
  - *Note) Every NP-complete problem is NP-hard*
- TSP is NP-Hard (Ex. 9.15)
  - *proof)* Traveling Salesperson Decision problem $\propto_T$ Traveling Salesperson Optimization problem
  - *Note) The optimization problem corresponding to any NP-complete problem is NP-hard*

Example of Turing reduction

✓ Problems

A: At least one of given n variables is true? *(Decision problem)*

OR(B1, B2, ..., Bn) = TRUE ?

B: Find the largest of given n integers *(General problem)*

max(I1, I2, ..., In)

✓ A $\propto_T$ B ?

```
AlgorithmA(n, B[1..n])
{   index k;  int I[1..n], max; boolean answer;
    // transform
    for (k=1; k <= n; k++)
        if (B[k] == TRUE) I[k] = 1;
        else I[k] = 0;
    // solve A using B
    max = AlgorithmB(n, I[1..n]);
    if (max == 1) answer = "yes";
    else answer = "no";
    return answer;
}
```


Traveling Salesperson Problem is NP-hard

✓ Traveling salesperson decision problem ∈ NP-complete

✓ TSPD $\propto_T$ Traveling salesperson problem ?

```
AlgorithmTSPD(G, len)
{
      opttour = AlgorithmTSP(G);
      if (length(opttour) < len)
        return "yes"
      else
        return "no"
}
```

NP-Easy, NP-Equivalent

- Def) A problem A is NP-easy
  - if for some NP problem B, $A \propto_T B$
  - *Note) Every P and NP problems are NP-easy*
- Def) A problem is NP-equivalent
  - if it is NP-hard and NP-easy
- NP-Easy problem example

A: 입력 중에 절대값이 a인 것이 몇 개인가?

B: 입력 x의 절대값이 주어진 값 a와 일치하는가?

A is NP-Easy since

- $B \in NP$
- $A \propto_T B$ ←
- $T_A(n)$ = ?

```
ProblemA(n, I[], a) {
    count = 0;
    for (k=0; k < n; k++)
      if (ProblemB(I[k], a) == "yes")
        count++;
    return count;
}
```

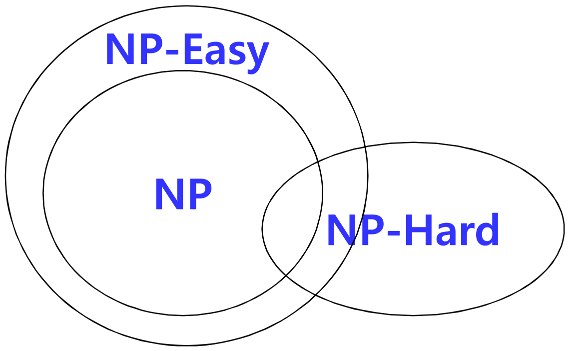

Another Example

- Cost:
  - 주어진 그래프에 대해 여비를 결정하시오. 최단 tour의 길이가 단거리 기준(100km) 미만은 10만원, 중거리 기준(200km) 미만은 20만원, 그 이상이면 30만원의 여비를 지급한다.
- Cost is NP-Easy since
  - TSPD ∈ NP
  - Algorithm of Cost
  - $T_{Cost}(n)$ = ?
  - → Cost $\propto_T$ TSPD
  - Therefore Cost ∈ NP-easy

```
Cost(G, ShortLen, MidLen) {
   if (TSPD(G, ShortLen) == "yes")
     cost = 10;
   else if (TSPD(G, MidLen) == "yes")
     cost = 20;
   else
     cost = 30;
   return cost;
}
```

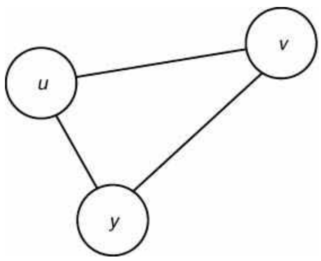
- Cost is NP-Hard since
  - TSPD $\propto_T$ Cost ?
- Cost is NP-Equivalent

NP, NP-Complete, NP-Hard, NP-Easy



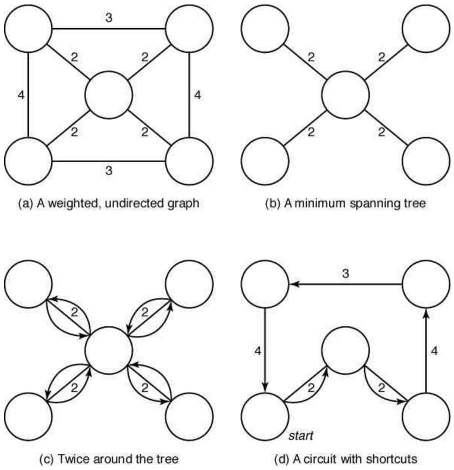
Handling of NP-Hard Problems

- ✓Approximation algorithms
- ✓Probabilistic algorithms
  - ✓Genetic algorithm
  - ✓Simulated annealing algorithm
- ✓A simple approximation algorithm for traveling salesperson problem
  - ✓Assume triangular property
  - ✓$W(u,v) \leq W(u,y) + W(y,v)$
  - ✓Utilize minimum spanning tree



Example of Approximation Algorithm

- ✓1. find minimum spanning tree
- ✓2. make a tour with edges of forward and backward
- ✓3. apply triangular property
- ✓Thm. 9.6) distance ≤ 2 x optimal-solution



Summary

- ✓Pseudo-polynomial time algorithm
- ✓Decision problem
- ✓P problem, NP problem
- ✓Polynomial time reducing and NP-Complete
- ✓Polynomial time Turing reducing and NP-Hard
- ✓NP-Easy and NP-Equivalent
- ✓Approximation algorithm