



알고리즘 설계 - 4장

계산복잡도와 다루기 힘든 정도:
NP 이론



목차

- 다루기 힘든 정도
- 3가지 일반적인 문제
- 비결정적 알고리즘
- NP-hard와 NP-complete





1. 다루기 힘든 정도

- 다차시간 알고리즘(Polynomial-time Algorithm)
 - 입력의 크기가 n 일 때, 최악의 경우 수행시간이 $W(n) \in O(p(n))$ 인 알고리즘.
 - 여기서 $p(n)$ 은 n 의 다차 함수(polynomial function)
- 보기
 - 시간 복잡도가 $2n, 3n^3 + 4n, 5n + n^{10}, n \log n$ 인 알고리즘은 다차시간 알고리즘이다.
 - 시간복잡도가 $2^n, 2^{0.01n}, 2^{\sqrt{n}}, n!$ 인 알고리즘은 다차시간 알고리즘이 아니다.



다루기 힘든 정도(Intractability)

- 다루기 힘든(**intractable**) 문제의 정의:
 - 다항식으로 시간복잡도가 표시되는 알고리즘을 찾을 수 없는 문제
- 다루기 힘든 문제의 예
 - 외판원 문제: $O(n^2 2^n)$
 - 0-1 배낭 채우기 문제: $O(2^n)$
- All-Pairs Shortest Path Problem
 - 무작정 알고리즘: $n!$
 - 동적 프로그래밍: n^3
 - 따라서 이 문제는 다루기 힘들지 않다 (not intractable)



2. 3가지 일반적인 문제

- 다차시간 알고리즘을 찾은 문제
- 다루기 힘들다고 증명된 문제
- 다루기 힘들다고 증명되지 않았고, 다차시간 알고리즘도 찾지 못한 문제



2.1 다차시간 알고리즘을 찾은 문제

- 모든 알고리즘들이 다차시간 알고리즘
 - 정렬 문제: $\Theta(n \log n)$
 - 정렬된 배열 검색 문제: $\Theta(\log n)$
 - 행렬곱셈 문제: $\Theta(n^{2.38})$
- 다차시간이 아닌 알고리즘도 있으나, 다차시간 알고리즘을 찾은 경우
 - All-Pairs Shortest Path Problem
 - Optimal Binary Search Tree Problem
 - Minimum Cost Spanning Tree Problem

2.2 다루기 힘들다고 증명된 문제

- 비다항식(non-polynomial) 크기의 결과를 요구하는 비현실적인 문제
 - 보기: 모든 해밀토니안 회로를 결정하는 문제
 - 만일 모든 정점들 간에 이음선이 있다면, $(n-1)!$ 개의 답을 얻어야 한다.
 - 이러한 문제는 필요이상으로 많은 답을 요구하므로 사실상 비현실적이고, 다루기 힘든 문제로 분류된다.
- 요구가 현실적이나, 다차시간에 풀 수 없다고 증명된 문제
 - 이런 부류에 속하는 문제는 상대적으로 별로 없다.
 - 결정불가능한 문제(Undecidable Problem)
 - 문제를 풀 수 있는 알고리즘이 존재할 수 없다고 증명
 - 예: 종료 문제(Halting Problem) 등
 - 프레스버거 산술(Presburger Arithmetic) 문제
 - Fischer와 Rabin에 의하여 증명됨(1974)

종료 문제(Halting Problem)

- 어떤 프로그램 **P**가 정상적으로 수행되어서 종료하는지를 결정하는 문제
 - 1936년 **Alan Turing**에 의해서 증명됨
- 정리: 종료 문제는 결정불가능하다.
- 증명: 이 문제를 풀 수 있는 알고리즘이 존재한다고 가정하자. 그 알고리즘은 어떤 프로그램을 입력으로 받아서 그 프로그램이 종료하면 “예”라는 답을 주고, 종료하지 않으면 “아니오”라는 답을 줄 것이다. 그 알고리즘을 **Halt**라고 하면, 다음과 같은 “말도안돼” 알고리즘을 만들 수 있다.

algorithm 말도안돼

```
if (Halt(말도안돼) == “예”) then  
    while true do print “야호”
```

- 만일 “말도안돼” 알고리즘이 정상적으로 종료하는 알고리즘이라고 한다면, **Halt(말도안돼)**는 “예”가 되고, 따라서 이 알고리즘은 절대로 종료하지 않는다. 이는 가정과 상반된다.
- 만일 “말도안돼” 알고리즘이 정상적으로 종료하지 않는 알고리즘이라고 한다면, **Halt(말도안돼)**는 “아니오”가 되고, 따라서 이 알고리즘은 종료하게 된다. 이도 마찬가지로 가정과 상반된다. 결론적으로, **Halt**라는 알고리즘은 존재할 수 없다.



2.3 다루기 힘들다고 증명되지 않았고, 다차시간 알고리즘도 찾지 못한 문제

- 많은 문제들이 이 카테고리에 속한다.
 - 0-1 배낭채우기 문제
 - 외판원 문제
 - 최대 파벌 문제
 - 해밀토니안 회로 문제 등등
- NP(Nondeterministic Polynomial) 문제에 속한다.



최적화 문제 vs. 결정 문제(1)

- 최적화 문제(optimization problem) - 최적의 해를 찾아야 한다.
- 결정 문제(decision problem) - 대답이 “예” 또는 “아니오”로 이루어지는 문제
⇒이론을 전개하고 이해하기 쉬움
- 최적화 문제와 결정 문제 간의 관계
 - 최적화 문제에 대한 해답을 구하면, 대응되는 결정 문제에 대한 해답은 저절로 얻는다.
 - 어떤 최적화 문제에 대해서 다차시간 알고리즘이 있다면, 그 알고리즘으로부터 그 문제에 해당하는 결정문제에 대한 다차시간 알고리즘을 쉽게 유추해 낼 수 있다.
- NP와 P를 다룰 때 결정 문제만 고려해도 충분하다.

최적화 문제 vs. 결정 문제(2)

■ 외판원 문제

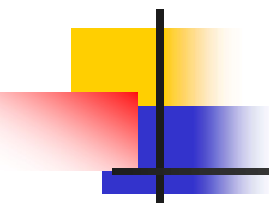
- 최적화 문제: 가중치가 있는 방향 그래프에서, 한 정점에서 출발하여 다른 모든 정점을 정확히 한번씩만 방문하면서 출발점으로 돌아오는 일주여행 경로 중에서 총 여행거리가 최소가 되는 경로를 구하시오.
- 결정 문제: 어떤 양수 d 가 주어지고, 총 일주여행거리가 d 를 넘지 않는 경로가 있는지 없는지를 결정하시오.

■ 0-1 배낭 채우기 문제

- 최적화 문제: 배낭에 넣을 아이템의 무게와 가치를 알고 있을 때, 용량이 W 가 되는 배낭에 아이템을 총 이익이 최대가 되도록 채우시오.
- 결정 문제: 용량이 W 가 되는 배낭에 아이템을 총 이익이 최소한 P 가 되도록 채울 수 있는지를 결정하시오.

3. 비결정적 알고리즘 (Nondeterministic Algorithm)

- 결정적 알고리즘(Deterministic Algorithm)
 - 모든 연산의 결과가 명확하게 정의
 - 우리가 이제까지 배운 대부분의 알고리즘들
- 비결정적 알고리즘(Nondeterministic Algorithm)
 - 연산의 결과가 정확하게 정의되지 않고, 여러 개의 가능한 값 중의 하나로 제한.
 - 다음과 같은 함수들을 가짐
 - Choice(S): 집합 S중의 하나를 선택(선택 규칙 X)
 - Failure(): 성공적이지 못한 완료
 - Success(): 성공적인 완료
 - 하나의 선택이라도 성공적인 완료에 도달할 수 있으면, 그 선택이 이루어져 비결정적 알고리즘은 성공적 완료.



비결정적 알고리즘의 예(1)

- Nondeterministic Search

```
int j = Choice(1, n);  
if (A[j] == x)    { cout << j; Success(); }  
else             { cout << '0'; Failure(); }
```

비결정적 알고리즘의 예(2)

- Nondeterministic Knapsack Algorithm

```
void DKP(int p[], int w[], int n, int m, int r, int x[])
{
    int W = 0, P = 0;
    for (int i = 1; i <= n; i++) {
        x[i] = Choice(0, 1);
        W += x[i] * w[i]; P += x[i] * p[i];
    }
    if ((W > m) || (P < r))    Failure();
    else    Success();
}
```

비결정적 알고리즘의 예(3)

- Nondeterministic Satisfiability

```
void Eval(cnf E, int n)    // E = n개 변수의 논리식
{                          // E를 참으로 하는 n의 조합을 구하는 문제
    int x[SIZE];
    // 진리값을 n개의 변수에 할당
    for (int i = 1; i <= n; i++)
        x[i] = Choice(0, 1);
    if (E(x, n))    Success();
    else    Failure();
}
```


4. NP-hard와 NP-complete

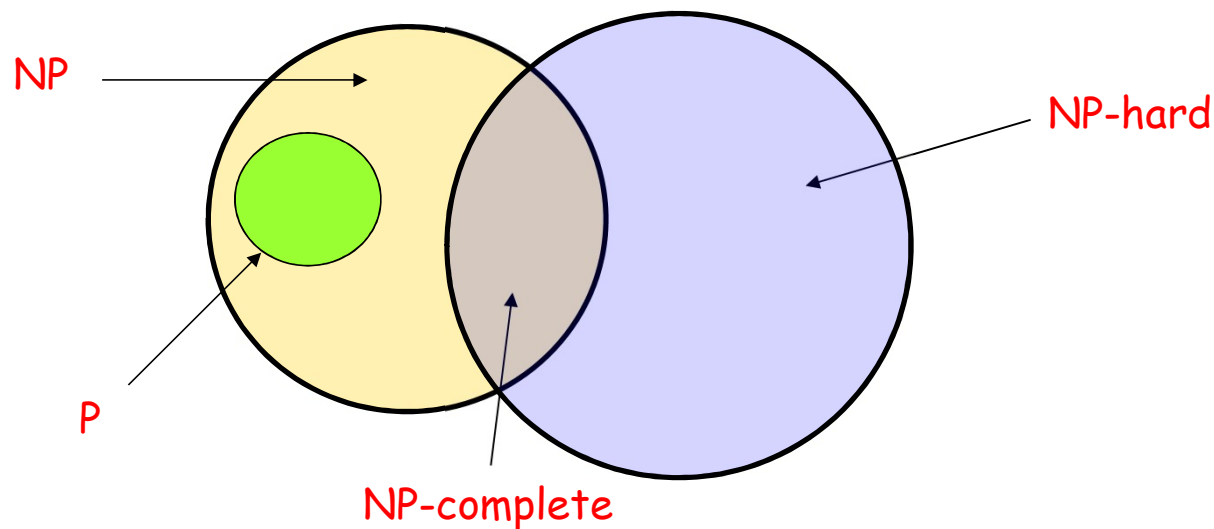
- P와 NP의 정의
 - P: 결정적 알고리즘을 이용하여 다차시간에 풀 수 있는 문제들의 집합
 - NP: 비결정적 알고리즘을 이용하여 다차시간에 풀 수 있는 문제들의 집합
- P와 NP의 관계
 - $P \subseteq NP$ (OK)
 - Problem: $P = NP$ or $P \neq NP$?
 - NP에 속한 문제 중에서 P에 속하지 않는다고 증명이 된 문제는 하나도 없다. 따라서 아마도 $NP - P = 0$ 일지도 모른다.
 - 그럼 $P = NP$? 이것이 사실이라면 거의 모든 문제가 NP에 속하기 때문에 우리가 아는 거의 모든 문제의 다차시간 알고리즘이 있다는 의미!
 - 사실 많은 사람들이 $P \neq NP$ 일 것 이라고 생각하고 있기는 하지만 아무도 이것을 증명하지 못하고 있는 것이다.

축소 관계 (\propto)

- 변환(transform) 알고리즘
 - 풀고 싶은 어떤 문제를 **A**라고 하고, 이미 알고리즘을 알고 있는 어떤 문제를 **B**라고 하자.
 - 변환 알고리즘 $f()$ 의 정의:
 - $\forall i, B(f(i)) == \text{true} \rightarrow A(i) == \text{true}$
 - $\forall i, B(f(i)) == \text{false} \rightarrow A(i) == \text{false}$
 - 변환 알고리즘과 **B**에 대한 알고리즘을 합하면, **A**에 대한 알고리즘 생성 가능.
- 정의: **A**는 **B**로 축소 가능하다. ($A \propto B$)
 - **B**에 대한 다차 시간 결정 알고리즘이 존재하고 ($B \in P$),
 - **A**에서 **B**로 다차시간 변환 알고리즘이 존재할 경우,
 - **A**를 다차시간에 해결하는 결정 알고리즘이 존재 ($A \in P$).
- 정리 **1**: 결정 문제 **B**가 **P**에 속하고 $A \propto B$ 이면 결정 문제 **A**는 **P**에 속한다.

NP-Hard and NP-Complete

- Cook's Theorem:
 - Satisfiability is in P if and only if $P = NP$.
- 문제 L은 **NP-hard**, if and only if,
 - satisfiability $\propto L$.
- 문제 L은 **NP-complete**, if and only if,
 - L이 NP-hard이면서 $L \in NP$
- P, NP, NP-complete, and NP-hard 문제와의 관계(믿음)



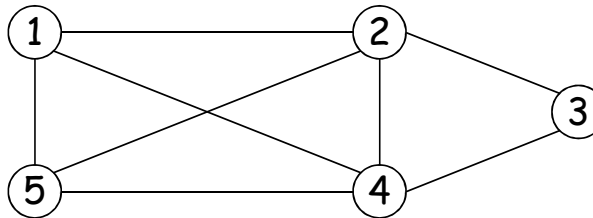
NP-Hard 임을 증명하는 방법

- 문제 L2가 NP-Hard임을 증명
 - 1) 현재까지 NP-Hard임이 증명된 문제 L1을 선정
 - CNF Satisfiability \propto L1
 - 2) L1의 사례 I에서 L2의 사례 I'로 다차시간 변환이 가능하며, I'의 해에서 I의 해를 다차시간에 변환할 수 있음을 증명
 - 3) 단계 2)의 결과로 L1 \propto L2 관계를 유도
 - 4) 단계 1)과 3)의 결과로 L2는 NP-hard임이 증명됨.
 - CNF Satisfiability \propto L1 \propto L2 \rightarrow
 - CNF Satisfiability \propto L2

대표적인 NP-Hard 문제들(1)

■ Max Clique Decision Problem

- $G = (V, E)$ 의 maximal complete subgraph를 발견
- Clique: $\{2, 3, 4\}$, $\{1, 2, 4, 5\}$
- Max Clique = $\{1, 2, 4, 5\}$

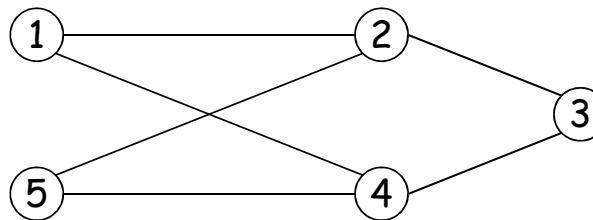


■ Directed/Undirected Hamiltonian Cycle Problem

대표적인 NP-Hard 문제들(2)

■ Node Cover Decision Problem

- Node cover $S \subseteq V$: E 의 모든 에지가 S 에 incident
- 예: $S = \{2, 4\}$ or $S = \{1, 3, 5\}$
- Does an undirected graph G has a node cover of size at most k ?



■ Chromatic Number Decision Problem

- Coloring of a graph $G = (V, E)$
 - $f: V \rightarrow \{1, 2, \dots, k\}$. If $(u, v) \in E$, then $f(u) \neq f(v)$
- Whether a graph G has a coloring for a given k ?
- Possible 2-coloring: $f(1) = f(3) = f(5) = 1, f(2) = f(4) = 2$

대표적인 NP-Hard 문제들(3)

■ Partition Problem

- $A = \{a_1, a_2, \dots, a_n\}$, a_i : 양의 정수
- $\sum_{i \in P} a_i = \sum_{i \notin P} a_i$ 인 P 를 찾는 문제

■ Scheduling Problem

- P_i , $1 \leq i \leq m$, be m identical processors
- J_i , $1 \leq i \leq n$, be n identical jobs
 - t_i : J_i 의 처리 시간, f_i : J_i 의 완료 시간
 - T_i : P_i 가 할당된 모든 job을 처리 완료한 시간
- Schedule S : job을 processor에 할당하는 문제
 - Non-preemptive schedule, Preemptive schedule
 - Finish time of S , $FT(S) = \max_{1 \leq i \leq m} \{T_i\}$
- NP-hard: Minimum finish time non-preemptive schedule
- Example: P_1 and P_2
 - $t_1 = 2, t_2 = 5, t_3 = 6, t_4 = 7, t_5 = 10$
 - $P_1: t_2 \rightarrow t_5, P_2: t_1 \rightarrow t_3 \rightarrow t_4$ $FT(S) = 15$