

계산복잡도
Computational Complexity

크리스토스 H. 파파디미트리우

Chistos H. Papadimitriou

캘리포니아 대학교 샌디에이고

©1994 Addison-Wesley Publishing Company, Inc.

번역 김태원

2023년 8월 8일

차 례

차 례	i
제 I 편 알고리즘	1
제 1장 문제와 알고리즘	3
1.1 그래프 도달가능성	3

제 I 편

알고리즘

알고리즘 책은 계산복잡도를 다루는 장으로 끝나기 마련이니, 알고리즘에 관한 기본 사실 몇 가지 돌이키며 본고를 시작하는 편이 적절하겠다. 이어지는 세 장에서 우리의 목표는 간단하지만 중요한 요점을 조금 지적하는 것이다. 바로 계산 $computational$ 문제란 해결되어야 하는 것일 뿐만 아니라, 탐구할 가치가 있는 객체이기도 하다는 점이다. 문제와 알고리즘은 수학적으로 형식화되고 분석될 수 있다. 차례로 이를테면 언어 $languages$ 나 튜링장치 $Turing machines$ 가 그렇다. 그리고 정확한 형식주의는 별 상관 없다. 다항 시간 계산가능성 $Polynomial-time computability$ 은 계산 문제에서 바라는 성질로, 실용적인 해결가능성 $practical solvability$ 의 직관적인 개념과 동질이다. 여러 상이한 계산 모델 $models$ 은 효율성의 다항 손실 $polynomial loss$ 로 또 다른 모델을 시뮬레이션할 수 있다—비결정론 $nondeterminism$ 이라는 예외, 즉 제 시뮬레이션에 지수시간 $exponential time$ 을 요구하는 것으로 보이는 예외를 제외하면 말이다. 그리고 알고리즘을 아예 지니지 않는 문제가 존재하는데, 아무리 비효율적인 것조차 지니지 않는다.

제 1 장

문제와 알고리즘

알고리즘은 문제를 풀기 위한 자세한 스텝별 *step-by-step* 방법론이다. 다만 문제 *problem*란 뭔가? 우리는 이 장에서 중요한 예시 세 개를 소개한다

1.1 그래프 도달가능성

그래프 $G = (V, E)$ 는 노드 *nodes* V 와 선 *edges* E , 즉 노드 쌍의 집합이다 (이른다면 그림 1.1을 보라. 우리의 그래프는 모두 유한하고 유향 *directed* 이겠다). 여러 계산 문제는 그래프를 다룬다. 그래프에 관해 가장 기본적인 문제는 이것이다. 그래프 G 와 두 노드 $1, n \in V$ 가 주어질 때 1에서 n 까지 경로 *path*는 존재하는가? 우리는 이 문제를 도달가능성 *REACHABILITY*[†]이라고 부른다. 가령, 그림 1.1에는 분명 노드 1에서 $n = 5$ 까지 경로, $(1, 4, 3, 5)$ 가 존재한다. 대신 만약 선 $(4, 3)$ 의 방향을 역전하면, 그런 경로는 존재하지 않는다.

흥미로운 문제가 대다수 그렇듯, 도달가능성은 가능한 일례 *instances*의 무한 집합을 지닌다. 각 일례는 수학적 객체 (우리의 경우, 그래프와 그 두 노드로), 곧 우리가 질문을 묻고 답을 기대하는 대상이다. 이때 질문이 속한 특정 종류가 문제를 특징짓는다. 도달가능성은 “예” 혹은 “아니오” 가운데 하나의 답안을 요구하는 질문이라는 점에 유의하라. 이런 문제는 결정문제 *decision problems*라고 부른다. 복잡도 이론에서 보통 우리는 온갖 상이한 답안을 요구하는 문제보단 결정문제만 다루는 편이 편리하게 통합적이고 단순하다고 본다. 그러니 결정문제는 본고에서 중요한 역할을 맡겠다.

우리는 우리의 문제를 푸는 알고리즘에 관심이 있다. 다음 장에서 우리는 튜링장치 *Turing machine*를 다루겠는데, 이는 임의의 알고리즘을 표현하기 위한 형식 모델 *formal*

[†]복잡도 이론에서, 계산 문제는 단지 풀어야 하는 것일 뿐 아니라, 다만 그 자체로 흥미로운 수학적 객체이기도 하다. 문제가 수학적 객체로 다뤄질 때, 그 이름을 대문자로 표기하겠다. [역자: 대문자 표기를 드러냄표(*circumph*)로 대체한다.]

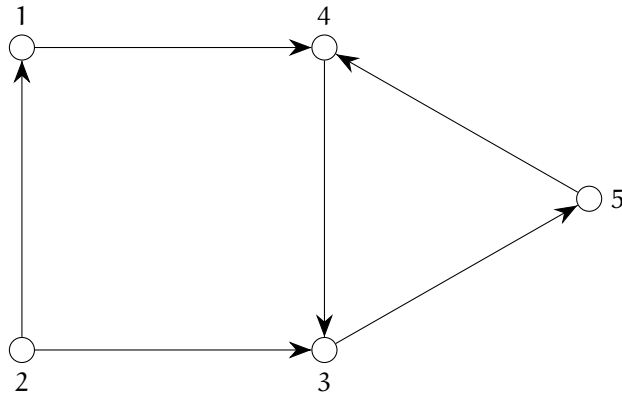


그림 1.1: 그래프.

model이다. 지금으로서는 우리의 알고리즘을 비형식적으로 기술하겠다. 가령 도달가능성은 이른바 탐색 알고리즘 *search algorithm*으로 풀 수 있다. 이 알고리즘은 이렇게 작동한다. 알고리즘에 걸쳐 우리는 노드의 집합을 지니는데, 이를 S 라고 표기한다. 처음에는, $S = \{1\}$ 이다. 각 노드는 표시되거나 *marked* 표시되지 않는다 *unmarked*. 노드 i 가 표시된다는 말은 i 가 과거 어느 지점 S 에 있었다는 (혹은, 지금 S 에 있다는) 뜻이다. 처음에는, 오직 1만 표시된다. 알고리즘의 각 반복 *iteration*에서, 우리는 노드 $i \in S$ 를 선택해 S 에서 제거한다. 우리는 그런 다음 i 로부터 하나씩 모든 선 (i, j) 를 처리한다. 노드 j 가 표시되지 않았다면, 표시하고, S 에 추가한다. 이 과정은 S 가 빌 때까지 계속된다. 이 지점에서, 우리는 노드 n 이 표시되었다면 “예”, 표시되지 않았다면 “아니오”라고 답한다.

이 친숙한 알고리즘이 도달가능성을 해결한다는 사실은 분명할 것이다. 증명은 1에서 노드까지 경로가 존재하는 경우 그리고 오직 그 경우 *if and only if* 노드가 표시된다고 확증할 것이다. 또 분명한 것은, 다만, 알고리즘에 대한 우리의 기술에는 중요한 디테일이 생략되었다는 사실이다. 가령, 그래프는 어떻게 알고리즘에 입력으로 표현되는가? 이런 적절한 표현은 우리가 사용하는 알고리즘의 특정 모델에 의존하기에, 이는 우리가 특정 모델을 지니기 전까지는 미룰 사안이다. 이 논의의 요점은 (2.2절을 보라) 바로 정확한 표현이 별로 상관 없다는 것이다. 그동안 그래프가 인접행렬 *adjacency matrix*로 (그림 1.2로) 주어진다고 가정할 수 있는데, 이는 모든 요소가 랜덤 액세스 *random access* 느낌으로 알고리즘에 의해 접근될 수 있는 행렬이다.[†]

알고리즘 자체에도 불분명한 지점이 있다. 원소 $i \in S$ 는 어떻게 S 의 모든 원소가 운데 선택될 수 있는가? 이때 선택은 탐색 스타일에 막대한 영향을 미칠 것이다. 가령, 우리가 S 에 가장 오래 머무른 노드를 항상 선택한다면, (달리 말해, 우리가 S 를 큐 *queue*

[†]실은, 7장에서 복잡도 이론에 대한 도달가능성의 중요한 쓰임새를 살필 예정인데, 거기서는 그래프가 암시적으로 *implicitly* 주어진다. 즉, 입력데이터로 그 인접행렬의 각 요소를 계산할 수 있다.

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

그림 1.2: 인접 행렬.

로 구현한다면) 탐색으로는 너비 우선 *breadth-first*이 이어지고, 최소 경로가 확인될 것이다. (가장 최근에 추가된 노드를 선택하여) S 가 스택 *stack*으로 정비되었다면, 우리는 일종의 깊이 우선 *depth-first* 탐색을 지니게 된다. S 를 간수하는 다른 방법들은 완전히 상이한 종류의 탐색으로 이어질 것이다. 그러나 알고리즘은 이들 선택 모두에 대해 올바르게 작동한다.

뿐만 아니라, 효율적으로 작동한다. 그렇다고 확인하기 위해, 인접행렬의 각 요소는 단 한 번, 행에 대응하는 꼭지점이 선택될 때 방문된다는 점에 유의하라. 그리하여, 우리는 선택된 노드로부터 선을 처리하며 대략 n^2 연산을 사용한다 (결국, 그래프에는 최대 n^2 개의 선이 있을 따름이다). 요구되는 다른 간단한 (집합 S 에서 원소 선택하기, 꼭지점 표시하기, 꼭지점이 표시됐는지 전하기 같은) 연산은 각각 어떻게든 상수시간 *constant time*에 끝날 수 있으니, 우리는 탐색알고리즘이 n 개의 노드를 지닌 그래프 상의 두 노드가 연결되느냐 마느냐 하는 결정을 최대 n^2 에 비례하는 시간 안에, 혹은 $\mathcal{O}(n^2)$ 안에 내린다고 결론지을 수 있다.

막 사용한 \mathcal{O} 표기와 그 동족들은 복잡도 이론에서 아주 유용하니, 간략한 삽입구를 열어 이들을 형식적으로 정의한다.

정의 1.1. 우리는 \mathbb{N} 으로 모든 음이 아닌 정수의 집합을 표기한다. 복잡도 이론에서 우리는 \mathbb{N} 에서 \mathbb{N} 으로 가는 함수를 다루는데, 이를테면 $n^2, 2^n, n^3 - 2n + 5$ 가 그렇다. 우리는 글자 n 을 그런 함수의 표준 인수 *argument*로 사용하겠다. 정수가 아닌 것들이나 음의 값—가령 $\sqrt{n}, \log n, \sqrt{n} - 4 \log^2 n$ —을 용인하는 방식으로 우리의 함수를 표기할 때라도, 우리는 언제나 그 값을 음이 아닌 정수로 생각할 것이다. 즉, 이들 예제처럼, $f(n)$ 으로 표기되는 모든 함수가 우리에게 정말 뜻하는 바는 $\max\{f(n), 0\}$ 이다.

그리하여, f 와 g 는 \mathbb{N} 에서 \mathbb{N} 으로 가는 함수라고 하자. 우리는 양의 정수 c 와 n 이 존재하여 모든 $n \geq n_0$ 에 대해 $f(n) \leq c \cdot g(n)$ 인 경우 $f(n) = \mathcal{O}(g(n))$ 이라고 쓴다 (“ $f(n)$ 은 $g(n)$ 의 빅오 *big oh*다”, 혹은 “ f 는 g 의 차수 *order*다”라고 발음한다). $f(n) = \mathcal{O}(g(n))$ 은 f 가 g 만큼 자라거나 더 느리다는 것을 비형식적으로 뜻한다. 그 반대가 일어나면, 즉, $g(n) = \mathcal{O}(f(n))$ 이라면, 우리는 $f(n) = \Omega(g(n))$ 이라고 쓴다. 끝으로, $f(n) = \theta(g(n))$ 은 $f(n) = \mathcal{O}(g(n))$ 이고 $f(n) = \Omega(g(n))$ 임을 뜻한다. 후자는 f 와 g 가 정확히 똑같은 증가율 *rate of growth*를 지닌다는 것을 뜻한다.

보이기 쉬운 예를 들면, 수학적 $p(n)$ 이 계수degree d 의 다항식인 경우, $p(n) = \theta(n^d)$ 다. 곧, 다항식의 증가율은 다항식의 첫 0이 아닌 항non-zero term으로 포착된다. 이어지는 바는 아마 복잡도 이론에 대한 함수의 증가에 관해 가장 중요하고 유용한 사실이다. $c > 1$ 가 정수이고 $p(n)$ 이 모든 다항식이라면, $p(n) = \mathcal{O}(c^n)$ 이지만, 이는 $p(n) = \Omega(c^n)$ 인 경우가 아니다. 곧, 모든 다항식은 모든 지수식보다 엄격하게*strictly* 느리게 자란다 (증명으로는 문제 1.4.9를 보라). 지수가 하나 낮아지면, 같은 성질은 $\log n = \mathcal{O}(n)$ 이라는 것을 암시한다 —실은, 모든 k 승에 대해 $\log^k n = \mathcal{O}(n)$ 이라는 것을 암시한다. \square