

더 좋은 알고리즘을 부탁해!

드디어 최종편입니다. 이야기가 점점 절정으로 치달고 있습니다. 이번 호는 여러 가지 면에서 매우 흥미 있는 내용일 것입니다. '알고리즘과 비밀의 방' 을 기억하고 있나요? 이번 호를 끝으로 드디어 그 방의 모든 비밀이 밝혀집니다. 바로 그 방에는 정말 무시무시한 용이 한 마리 살고 있다는 것입니다. 그 용의 이름은 'NP-complete' 입니다. 그 용과 맞서 싸우는 것은 우리의 숙명적인 임무이기도 하답니다.

연 + 재 + 숲 + 서

- 1회 2003.11 알고리즘과 비밀의 방
- 2회 2003.21 알고리즘과 두 개의 탑
- 3회 2003.31 더 좋은 알고리즘을 부탁해!

지금까지 우리는 알고리즘에 대한 몇 가지 오해와 궁금증에 대하여 공부해 보았습니다. 첫 번째 이야기는 알고리즘과 계산 모델(computational model)에 대한 내용이었습니다. 계산 모델과 모델의 기본적인 연산들을 정의함으로써 여러분은 알고리즘을 설계한다는 것이 얼마나 구체적인 작업인지 잘 느꼈으리라 생각합니다. 더욱이 이러한 계산 모델의 등장은 알고리즘을 컴퓨터로부터 그리고 특정 프로그래밍 언어(예를 들어 C 언어)로부터 자유롭게 만들 것입니다. 물론 이렇게 독립한 알고리즘은 컴퓨터를 대신하는 또 다른 계산 모델에 종속될 것입니다. 하지만 너무 걱정하지는 마십시오. 이 계산 모델은 보편적인 기계(universal model)이니까요. 이 기계는 여러분의 컴퓨터들을 대신할 것입니다. 우리는 그러한 보편적인 추상 모델(혹은 추상 기계)을 공부했습니다. 그 모델의 이름은 튜링(튜링 머신)입니다.

두 번째 이야기는 알고리즘의 정확성(correctness)과 효율성(efficiency)에 대한 이야기였습니다. 이것은 기준에 대한 이야기입니다. 성적표에 여러분이 수강한 과목들의 학점이 찍히듯이, 설계한 알고리즘의 성적에 관한 이야기입니다. 이것은 매우 중요한 이야기입니다. 여러분 자신이 설계한 알고리즘이 어떤 성능을 갖는지 알 수 없다면(혹은 말할 수 없다면), 설계 그 자체는 어떠한 의미 없는 일인지도 모르니까요. 여러분이 애써서 설계한 알고리즘은 어쩌면 제대로 동작조차 하지 않는 것인지도 모릅니다. 우리가 상상할 수 없을 정도의 많은 메모리와 시간을 필요로 하는 것인지도 모르죠. 여러분이 설계한 알고리즘에 가격표를 달아주세요. 얼마나 썩지 가르쳐 주십시오.

놀랍게도(?) 이번 호에서는 알고리즘이 아닌 문제(problem)에 대해서 다루고자 합니다. 세상의 모든 문제들은 모두 제각기 고유한 난이도(complexity)를 가지고 있습니다. 그 난이도에 대한 몇 가지 중요한 사실들을 공부할 것입니다. 이른바 전산학에서 말하는 계산 이론(complexity theory)에 대한 이야기입니다. 평소에 계산 이론에 관하여 흥미

를 가지고 있었던 분들(혹은 컴퓨터의 놀라운 힘을 믿는 사람)이라면 매우 좋은 기회가 될 것입니다. 이 글이 여러분이 평소에 갖고 있었던 많은 의문들을 상당수 해소시켜 주리라고 믿어 의심치 않습니다.

각 문제의 난이도

먼저 지난 호의 이야기로 돌아가 볼까요? 튜링의 방송 출연 말입니다. 여러분 모두가 아시다시피 튜링의 덧셈 묘기는 순조로웠습니다. 적어도 사회자의 마지막 질문이 있기 전까지는 말입니다. 그 질문의 요지는 '덧셈은 얼마나 어려운 문제인가?' 라는 것이었습니다. 그 질문은 사전에 약속된 것이 아니었기 때문에 여러분은 쉽게 대답할 수 없었습니다. 결국 망설이다가 덧셈은 매우 어려운 문제라고 얼버무리는 수밖에 없었습니다. 사회자는 다시 질문을 합니다.

"그럼 문제가 어렵다는 것은 정확히 어떤 의미죠? 어떤 근거로 그렇게 이야기할 수 있는 거죠?"

정말 대단한 사회자입니다. 혹시 전산학을 전공한 것은 아닌지 모르겠습니다. 여러분은 여기서 자신이 이러한 덧셈 방법을 생각하기 위해서 얼마나 고생했는지를 주저리주저리 설명하려고 합니다. 하지만 여러분과 사회자, 그리고 텔레비전 시청자에게 정작 필요한 것은 그러한 여러분의 고생스런 경험담이 아닐 것입니다. 여러분을 포함한 누구나 그 사실을 알고 있습니다. 그렇습니다. 여러분에게 필요한 것은 바로 증명입니다. '덧셈은 이 정도로 어려운 것이다' 라고 이야기할 수 있는 그 증명 말입니다(만약 덧셈이 별로 어려운 문제가 아니었다면 튜링의 이러한 묘기는 사실 대단한 것이 아닐지도 모르니까요).

세상의 많은 문제들은 제각기 고유한 난이도를 가지고 있습니다. 미분은 미분 나름대로의 난이도를 가지고 있고, 적분은 적분 나름대로의 난이도를 가지고 있습니다. 덧셈도 마찬가지입니다. 우리는 이러한 문제의 난이도에 대해서 두 가지 접근 방향을 가지고 있습니다. 그 한 가지 방향은 '이 문제는 얼마나 쉬운가?' 이고, 또 다른 한 가지 방향은 '이 문제는 얼마나 어려운가?' 입니다. 말장난하는 것 같다고요? 아닙니다. 절대 말장난이 아닙니다. 본격적인 문제의 난이도 이야기에 들어가기 앞서 일종의 몸풀기입니다.

덧셈은 얼마나 쉬운 문제인가?

다음의 <문장 1>은 수학적으로 잘 정의됐다고 말할 수 없습니다. 그러기에는 모호한 개념이 너무 많습니다. 어쨌든 중요한 것은 이 문장이 영어의 난이도에 대해서 부분적으로 이야기하고 있다는 사실입니다.

◆ <문장 1> 기적의 영어 공부 방법

- 누구나 A 공부 방법으로 영어를 매일 같이 2시간씩 6개월만 공부한다면 영어의 달인이 될 수 있습니다.

만약 이 문장이 참이라면, 우리는 다음과 같은 사실을 알게 될 것입니다. '영어의 달인이 되기 위해서는 A 방법으로 매일 같이 2시간씩 6개월만 투자하면 된다' 는 사실을 말입니다. 그것으로 충분합니다. 우리는 이제 영어가 얼마나 쉬운지에 대해서 감을 잡았습니다. 영어의 난이도는 최소한 이 정도(매일 같이 2시간씩 6개월간 공부하는 양 정도)로 쉽습니다. <문장 1>은 A 방법의 효율성과 함께 영어가 갖고 있는 난이도의 상한(upper bound)을 내포하고 있습니다.

◆ <문장 2> 또 다른 기적의 영어 공부 방법

- 누구나 B 공부 방법으로 영어를 매일 같이 2시간씩 3개월만 공부한다면 영어의 달인이 될 수 있습니다.

앞의 <문장 2> 역시 참이라고 가정해 봅시다. <문장 1>이 참인 동시에 <문장 2> 역시 참이라고 해서 전혀 이상할 것은 없습니다. 이 두 문장이 모두 참이라고 모순이 유도되는 것은 아니니까요. 단지 <문장 2>가 참인 사실이 추가적으로 증명된다면, 우리는 이제 '영어 공부를 위해서 6개월씩이나 시간을 투자할 필요가 없다' 는 사실을 알게 됩니다. 3개월이면 영어의 달인이 되기에 충분한 시간입니다. <문장 2>를 통하여 영어의 난이도에 대한 상한은 3개월로 낮추어 졌습니다.

지금 무슨 말을 하고 있는지 이해하겠습니까? 지금 우리는 임의의 문제가 얼마나 쉬운지에 대하여 이야기하고 있습니다. 그것을 어떤 방법으로 이야기하고 있습니까? 그 키는 바로 알고리즘이 쥐고 있습니다. 우리는 주어진 문제가 얼마나 쉬운지를 결정하기 위해서 그 문제를 해결하는 알고리즘을 이용하고 있습니다. 임의의

김형식 | morethan@jupiter.kaist.ac.kr

원저는 KAIST 전자전산학과 전산학 전공 박사과정으로 알고리즘을 연구하고 있다. 현실과 동떨어진 이론이 아닌 현실의 문제에 적용할 수 있는 실용적인 이론을 연구하는 것이 그의 희망이다. 최근에는 암호연구에 깊은 관심을 갖고 있다.



[정렬 문제와 상한, 그리고 점근적인 표시법]

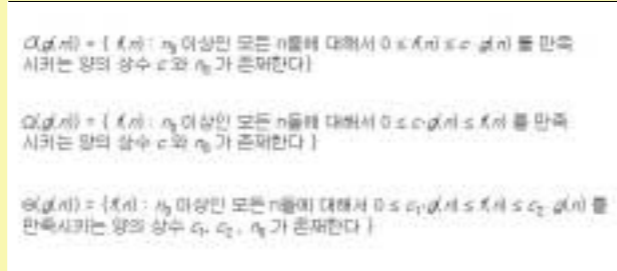
정렬 문제(sorting problem)는 임의의 순서가 정렬되지 않은 n개의 수들이 주어졌을 때, 오름차순(ascending order)으로 수들을 나열하는 문제를 말합니다. 이 문제는 전산학의 대표적인 문제 중 하나입니다. 이 정렬 문제를 해결하기 위하여 매우 간단한 알고리즘을 하나 설계해 보겠습니다. 그 알고리즘은 먼저 주어진 n개의 수들에 대하여 모든 순열(permutation)들을 차례로 나열합니다(물론 유효성을 획득하기 위해서는 모든 순열들을 나열하는 구체적인 방법을 제시해야 할 것입니다). 그리고 나열된 모든 순열들에 대하여 각각 오름차순인지를 테스트합니다(역시 유효성을 획득하기 위해서 추가적인 작업이 필요할 것입니다). 이 테스트를 성공적으로 통과하는 순열이 있다면 바로 그 순열을 출력하고 알고리즘을 종료합니다. 물론 이러한 알고리즘은 항상 올바른 결과를 출력할 것입니다(정확성에 대한 증명은 여러분에게 맡기겠습니다).

그렇다면 이 간단한 알고리즘의 성능을 한번 분석해 보도록 할까요. 알고리즘의 성능을 본격적으로 분석하기에 앞서 우리는 분석을 보다 쉽게 하도록 도와줄 표현 방법을 공부해야 할 것 같습니다. 우리는 지난 호에서 성장의 차수(order of growth)라는 개념을 공부하였습니다. 성장의 차수는 수행 시간 중 가장 중요한 부분만을 고려하고 나머지 부분들을 무시하는 것을 의미합니다. 성장의 차수는 우리의 분석 작업을 보다 용이하게 도와줄 것입니다. 일반적으로 성장의 차수가 의미를 가질 수 있는 충분한 크기의 입력을 고려할 때, 점근적인 효율성(asymptotic efficiency)을 사용합니다.

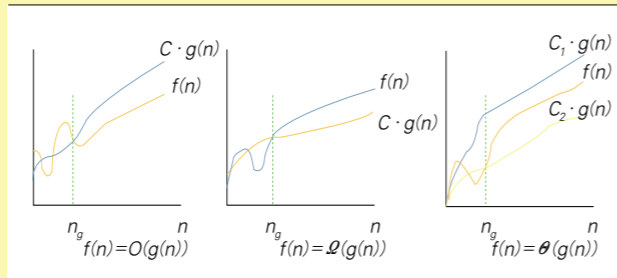
점근적인 효율성이란 무한대의 크기를 갖는 입력에 대해서 알고리즘의 수행 시간이 얼마나 걸리는지를 분석하는 방법을 말합니다. 우리는 이러한 점근적인 효율성을 표현하는 방법으로 점근적인 표시법(asymptotic notation)을 사용할 것입니다. 점근적인 표시법은 도메인(domain)을 자연수 집합으로 갖는 함수들로부터 정의됩니다. 그러한 대표적인 표시법으로는 여러분이 잘 아는 O , Ω , Θ 등이 있습니다. 각 표시법이 의미하는 바를 정확하게 살펴볼까요? 주어진 함수 $g(n)$ 에 대해서 각각의 $O(g(n))$, $\Omega(g(n))$, $\Theta(g(n))$ 은 <그림 1>과 같은 함수들의 집합으로 정의할 수 있습니다.

이 표시법의 직관적인 의미는 무엇일까요? n_0 보다 크거나 같은 모든 n 들에 대해서 함수 $f(n)$ 이 또 다른 함수 $g(n)$ 에(O , Ω , Θ 에 따라서) 어떤 식으로 한정되는 것을 의미한다는 것을 이해할 수 있나요? 이러한 점근적인 표시법은 충분히 큰 n 에 대해서 낮은 차수의 항(lower order term)들과 상수 계수를 무시할 수 있게 만들 것입니다. 결과적으로 이러한 함수에 대한 간략화(simple characterization)는 알고리즘의 분석을 보다 손쉽게 할 수 있도록 도와줍니다(물론 경우에 따라서 알고리즘의 수행 시간을 정확히 분석해야 하는 경우도 있습니다. 하지만 대부분의 경우에 있어 이러한 분석은 매우 까다로운 작업일 것입니다. 그리고 어느 정도의 큰 입력들에 대하여 이러한 정확한 분석은 실제로 큰 의미를 갖지 못하는 경우가 많습니다). <그림 2>는 각 표시법들의 예제를 차트로 나타낸 것입니다.

<그림 1> O , Ω , Θ 표시법의 정의



<그림 2> O , Ω , Θ 표시법의 차트로 나타낸 예제들



이제 점근적인 표시법을 이용해 앞에서 기술한 간단한 정렬 알고리즘을 분석해 봅시다. 알고리즘을 세 개의 커다란 부분으로 나눌 수 있겠군요. 첫 번째 단계는 모든 순열(permutation)들을 차례로 나열하는 단계입니다. 모든 순열의 개수가 $n!$ 개 존재하고, 순열 하나를 나열하는데 대략적으로 $O(n)$ 의 비용이 소요되기 때문에, 우리는 첫 번째 단계에서 알고리즘의 비용은 $O(n!)$ 이라는 것을 알 수 있습니다. 두 번째 단계는 이렇게 나열된 각 순열들에 대해서 오름차순인지를 검사하는 단계입니다. 이 단계 역시 유사한 방법으로 $O(n!)$ 의 비용을 갖는다는 사실을 알 수 있습니다. 마지막 단계는 테스트를 통과한 순열의 n 개의 수를 차례로 출력하는 단계입니다. 우리는 쉽게 이 비용이 $O(n)$ 이라는 것을 분석할 수 있습니다. 이러한 각 단계의 비용들을 모두 합산하면 이 알고리즘의 전체 성능은(점근적인 의미에서) $O(n!)$ 이라고 결론지을 수 있습니다. 그리고 역시 이러한 분석으로부터 정렬 문제의 난이도에 대한(점근적인 의미에서) 상한은 $O(n!)$ 라고 자신 있게 이야기할 수 있습니다(비록 이러한 결과가 그리 탐탁해 보이지 않더라도 말입니다).

여러분은 정렬 문제의 난이도에 대한 상한을 얼마까지 낮출 수 있습니까? 문제의 난이도에 대한 상한을 낮추기 위해서 알고리즘을 설계해야 합니다. 수업 시간에 배운 기억을 한번 떠올려 보십시오. 만약 $O(n \log n)$ 까지 낮출 수 없다면, 다시 주변 어딘가에 꽂혀있는 알고리즘 책을 한번 꺼내보기 바랍니다. 병합 정렬(merge sorting) 부분을 찾으셨나요? 이 알고리즘의 성능이 바로 $O(n \log n)$ 이군요. 정렬 문제의 난이도에 대한 상한이 $O(n!)$ 에서 $O(n \log n)$ 으로 극적으로 낮춰지는 순간이군요(더 낮출 수 있는 분 있가요?).

병합 정렬 알고리즘에 대하여 보다 상세한 내용이나 구현하는 방법에 대해서 알고 싶은 독자 분은 다음과 같은 참고 서적들을 참고하기 바랍니다. 크누쓰(Donald E. Knuth)의 'The Art of Computer Programming' 시리즈나 혹은 세지윅(Robert Sedgwick)의 'Algorithms'을 추천합니다. 콜먼(Thomas H. Cormen) 등이 쓴 'Introduction to Algorithms'나 고넷(G. H. Gonnet) 등이 쓴 'Handbook of Algorithms and Data Structures'도 좋은 참고가 될 것입니다.

문제(영어)에 대해서 현재까지 알려진 최상의 해결 방법(B 공부 방법)이 있다면, 우리는 다음과 같이 묻고 대답할 수 있을 것입니다.

“나는 영어가 제일 걱정이야. 그래서 말인데 영어를 잘하려면 어느 정도 열심히 공부해야 하는 걸까?”
 “글쎄.... (영어는) 3달이면 충분할 걸. B 방법 써봤어? 매일 같이 2시간씩만 공부해봐.”

기억하십시오. 문제가 얼마나 쉬운지를 표현하는 방법은 바로 문제를 직접 해결하는 것입니다. 해결된 문제는 최소한 그 풀이만큼은 쉽습니다. 이전에 우리가 설계한 튜링의 덧셈 알고리즘으로 되돌아가 볼까요. 만약 그 덧셈 알고리즘이 크기가 n 인 입력에 대해서 $f(n)$ 의 성능을 가진다면, 우리는 아마도 이렇게 이야기할 수 있을 것입니다.

“어떤 (입력의 크기가 n 인) 두 수를 더하는 일이라도 $f(n)$ 시간 이내에는 해결할 수 있지.”

누군가가 $f(n)$ 보다 더 작은 크기의 $g(n)$ 의 성능을 가지는 새로운 덧셈 알고리즘을 개발한다면, 물론 이러한 덧셈 문제의 난이도에 대한 상한은 $f(n)$ 에서 $g(n)$ 으로 낮추어질 것입니다. 즉, 현재까지 알려진 덧셈 문제에 대한 가장 우수한 알고리즘의 성능이 덧셈 문제에 대한 난이도의 상한이 됩니다.

우리는 주어진 문제에 대하여 항상 더 좋은 성능의 알고리즘을 제안하기 위하여 노력합니다. 그리고 이러한 알고리즘의 제안은 동시에 주어진 문제의 난이도에 대한 상한을 끊임없이 끌어내리고 있습니다. 우리는 언제까지 이러한 노력을 계속해야 하는 것일까요? 미래에는 일주일만 공부하면 영어의 달인이 되는 혁신적인 공

부 방법이 개발될 수 있을까요?

덧셈은 얼마나 어려운 문제인가?

다음의 <문장 3>과 같은 ‘그리 놀랍지 않은 소식’이 저명한 영문학자에 의하여 밝혀졌다고 가정하여 봅시다.

◆ <문장 3> 그리 놀랍지 않은 소식

- 외국에 한 번도 나간 적이 없는 어떤 한국인의 경우에는 영어의 달인이 되기 위해서 최소한 12시간 이상은 공부해야 한다.

이 문장이 의미하고 있는 것은 무엇입니까? 영어가 얼마나 어려운지에 대하여 설명하고 있다면 여러분은 납득하겠습니까? <문장 3>은 다음의 내용을 이야기하고 있습니다. 이 세상에는 12시간 미만의 공부량으로는 절대로 영어의 달인이 될 수 없는 사람이 존재한다는 사실을 말합니다. 그 어떠한(혁신적인) 공부 방법도 소용이 없습니다. 12시간 미만으로는 역부족입니다. 모든 사람에게 일반적으로 적용할 수 있는 12시간 미만의 영어 공부 방법은 존재하지 않습니다. 즉, <문장 3>은 영어의 달인이 되기 위해서는 최소한 12시간은 공부해야 한다는 것을 말하고 있습니다. 이를 계산 이론 관점에서 살펴보면, 영어의 난이도에 대한 하한(lower bound)으로 생각할 수 있습니다.

◆ <문장 4> 조금 놀라운 소식

- 나이 60세 이상의 영어를 전혀 못하는 어떤 한국인의 경우에는 영어의 달인이 되기 위해서 최소한 178시간 이상은 공부해야 한다.

<문장 4>와 같은 ‘조금 놀라운 소식’이 사실로 밝혀진다면, 우리는 이 문장으로부터 어떠한 결론을 이끌어낼 수 있을까요? 그렇습니다. 여러분 모두가 예상할 수 있듯이 영어의 난이도에 대한 하한은 이제 178시간으로 바뀌게 됩니다. 모든 사람을 영어의 달인이 되게 만드는 그러한 178시간 미만의 공부 방법은 존재할 수 없다는 이야기입니다(현재까지 존재하지 않는다는 것이 아니라 아예 존재할 수 없다는 것이 중요합니다). 영어의 달인이 된다는 것은 예상보다 훨씬 더 쉽지 않은 일인가 봅니다.

덧셈은 어떨까요? 덧셈의 난이도에 대한 하한을 이야기하기 위해선 <문장 3>이나 <문장 4>와 같은 덧셈에 대한 성질들을 증명해

야 할 것입니다. 이러한 증명은 알고리즘의 분석에 매우 중요한 부분입니다. 왜냐하면 알고리즘의 효율성을 평가하는데 있어서 매우 중요한 기준을 제공하기 때문입니다. 임의의 문제의 하한이 알려졌을 때, 이 하한에 근접하는 성능을 가지는 알고리즘을 설계한다면, 우리는 설계된 알고리즘의 성능이 매우 우수하다고 이야기할 수 있습니다. 그리고 만약 문제의 하한과 일치하는 성능을 가지는 알고리즘을 설계한다면, 우리는 그러한 알고리즘을 최적의 알고리즘(optimal algorithm)이라고 부릅니다. 계산 복잡도의 관점에서 이러한 최적의 알고리즘은 문제에 대한 최상의 해결책이 됩니다. 왜냐하면 문제의 하한보다 더 우수한 성능을 가지는 알고리즘은 절대 존재할 수 없기 때문입니다. 하지만 문제의 하한을 증명한다는 것은 매우 어려운 일입니다. 그래서 이용하는 증명 방법 중 하나가 바로 기존에 난이도가 알려진 문제에 대해서 연관시키는 방

법입니다. 즉, 영어의 어려운 정도를 보여 주기 위하여 다음과 같은 식으로 이야기하는 것입니다.

“영어의 최소한 수백만개는 어렵지 않습니까? 영어를 잘하려면 수학 공부하는 것만큼은 투자해야 된다고 생각합니다.”

이러한 테크닉을 ‘문제의 전형(transformation of problems)’이라고 부릅니다(일부 텍스트에서는 ‘문제의 변형(reduction of problems)’이라는 용어를 사용하기도 합니다). 물론 우리는 이러한 테크닉을 사용하지 않고, 문제의 고유한 성질들을 잘 고려하여 난이도에 대한 하한을 증명할 수도 있습니다. 사실 덧셈 문제는 직접 하한을 증명하는 것이 ‘문제의 전형’을 사용한 증명보다 더 쉬울지 모르겠습니다. 하지만 이 글에서는 하한을 분석하는 가장 중요한 방법 중의 하나인 ‘문제의 전형’을 설명하기 위해서 ‘문제의 전형’을 이용하여 덧셈의 하한을 증명하도록 하겠습니다.

문제의 전형

먼저 다음과 같은 과정을 갖는 문제 A와 문제 B가 존재한다고 가정해 봅시다.

- 1 문제 A에 대한 입력을 문제 B에 대한 적합한 입력으로 변환합니다.
- 2 문제 B를 풉니다.
- 3 문제 B에 대한 출력을 문제 A의 출력으로 전형합니다.

우리는 이러한 과정을 ‘문제 A가 문제 B로 전형되었다(problem A has been transformed to problem B)’고 이야기합니다. 문제 A에 대한 입력의 크기가 n 이었을 때, 단계 1과 3이 모두 $O(\tau(n))$ 시간 내에 수행된다면, 문제 A는 문제 B로 $\tau(n)$ -전형되었다(problem A is $\tau(n)$ -transformable to problem B)고 이야기합니다. 다음은 ‘문제의 전형’을 이용한 매우 중요한 정리입니다.

- ◆ (하한 정리) 문제 A가 적어도 $T(n)$ 의 수행 시간을 필요로 하고 문제 A가 문제 B로 $\tau(n)$ -전형된다면, 문제 B는 적어도 $T(n) - O(\tau(n))$ 의 수행 시간을 필요로 한다.

역시 상한에 대해서도 앞과 유사한 정리를 전개시킬 수 있습니

다. 이러한 내용을 <그림 3>과 같이 요약할 수 있습니다. 즉, ‘문제의 전형’으로부터 문제 A는 문제 B에 대한 하한을, 문제 B는 문제 A에 대한 상한을 제공합니다. 이러한 상한과 하한에 대한 관계는 $\tau(n)$ 이 $O(T(n))$ 의 함수에 포함될 때만 유지될 수 있습니다.

자, 그러면 이제 덧셈 문제의 하한에 대해서 생각해 볼까요. 덧셈의 하한을 증명하기에 앞서 우리는 덧셈 문제를 보다 일반적인 형태로 확장해야 할 필요가 있습니다. 이전에 정확하게 언급하지는 않았지만, 우리가 풀었던 덧셈 문제는 오직 양의 정수만을 입력으로 고려한 제한된 형태의 문제였습니다. 이러한 제약을 보다 완화하여 음수도 입력될 수 있는 새로운 덧셈 문제를 생각하여 봅시다. 이 문제의 하한은 얼마나 될까요? 얼마나 어려운 문제일까요?

덧셈의 하한을 보이기 위해 앞서 뺄셈 문제의 하한이 $\Omega(n)$ 으로 기존에 알려져 있다고 가정해 봅시다. 이러한 가정과 <하한 정리>로부터 덧셈 문제의 하한을 증명할 수 있습니다. <하한 정리>를 사용하기 위해서 뺄셈 문제를 덧셈 문제로 $\tau(n)$ -전형해야 할 것입니다 (물론 $\tau(n)$ 이 $O(n)$ 에 포함되도록 말입니다). 만약 $\tau(n)$ -전형할 수 있다면, 덧셈 문제의 하한은 $\Omega(n) - O(\tau(n))$ 이라고 이야기할 수 있습니다. 이제 전형 과정을 살펴보도록 합시다.

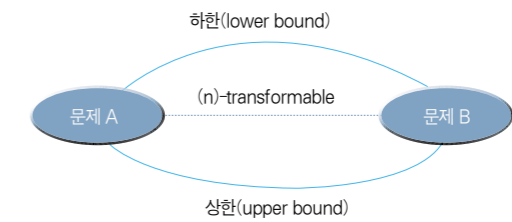
- 1 뺄셈 문제의 입력 a, b를 덧셈 문제에 대한 입력 a와 -b로 변환합니다.
- 2 입력 a와 -b에 대해서 임의의 덧셈 알고리즘을 이용해 덧셈을 합니다.
- 3 더해진 결과를 그대로 뺄셈 문제에 대한 출력으로 사용합니다.

이러한 전형 과정이 적절하다는 것을 우리는 쉽게 증명할 수 있습니다(구체적인 증명은 생략하겠습니다). **1**과 **2**의 수행 시간은 각각 $O(1)$ 의 시간이 소요됩니다(역시 증명은 생략합니다. 입력에 대한 인코딩이 매우 중요한 요소가 될 것입니다). 우리는 이러한 $O(1)$ 전형으로부터 덧셈 문제에 대한 하한이 $\Omega(n)$ 이라고 결론을 내릴 수 있습니다. 여기서 중요한 것은 **2**의 임의의 알고리즘이 실제로 존재할 필요가 없다는 사실입니다. 이러한 알고리즘이 덧셈 문제를 해결하는 가상의 알고리즘이라도 덧셈 문제의 하한을 이야기하기에는 충분한 것입니다.

튜링과 다루기 힘든 문제

여러분은 지금까지 튜링을 이용해 덧셈 문제를 푸는 알고리즘을 설계하고 그 성능을 분석해 보았습니다. 매우 당연한 이야기이지

<그림 3> 전형된 문제들 간의 상한과 하한



만 덧셈은 그리 어려운 문제가 아닙니다. 사실 튜링이 덧셈을 풀다 고 해서 박수를 치고 환호성을 지를 사람들은 그리 많지 않을 것입니다. 하지만 튜링은 아니 튜링 머신(Turing machine)은 실제로 엄청난 수의 문제들을 해결할 수 있습니다. 적어도 여러분의 컴퓨터만큼이나 많은 수의 문제를 해결할 수 있을 것입니다. 이들 사이의 계산가능성(computability)은 사실 동등합니다(엄밀하게 말하면 이 문장은 거짓말입니다. 여러분의 컴퓨터는 무한대의 메모리를 갖고 있지 않으니까요).

물론 컴퓨터가 풀 수 없는 문제들도 존재합니다. ‘컴퓨터라면 뭐든지 해결할 수 있어’라고 믿었던 사람이 있다면, 지금 당장 그 생각을 바꾸길 바랍니다. 실망스럽게도 컴퓨터가 풀 수 없는 문제들은 무수히 많습니다(대표적인 문제 중의 하나가 바로 그 유명한 정지 문제(halting problem)입니다). ‘풀 수 없다’는 이야기는 ‘알고리즘이 존재하지 않는다’는 이야기입니다. 여러분은 그러한 문제들에 대해서 결코 알고리즘을 설계할 수 없습니다.

그래도 여러분은 ‘컴퓨터가 풀 수 있는 문제들의 집합이 상당히 크다’는 사실에 그나마 안도의 한숨을 내쉬고 있을지 모르겠습니다. 하지만 진짜 끔찍한 비극은 이제부터 시작입니다. 비록 컴퓨터가 해결할 수 있더라도 그 풀이를 실용적으로 사용할 수 없는 문제들이 존재한다는 사실이 증명되었기 때문입니다. 이렇게 원칙적으로 풀 수는 있지만, 그 풀이가 실제로 사용하기에는 엄청나게 긴 시간이나 많은 공간을 필요로 하는 문제들을 ‘다루기 힘든 문제’라고 부릅니다(문제의 시간 비용에 대한 하한이 지수(exponential) 함수로 표현되는 문제들입니다).

풀 수 있는 문제들 중에서도 다항식 시간(polynomial time) 내에 풀 수 있는 문제들과 다항식 시간으로는 풀 수 없고 오직 지수 시간(exponential time)으로만 풀 수 있는 문제들로 구분되기 시작한 것입니다. 이에 대한 정의는 <그림 4>와 같습니다.

◁그림 4> P와 EXPTIME

$$P = \bigcup_i \text{Time}(n^i)$$

$$\text{EXPTIME} = \bigcup_i \text{Time}(2^{n^i})$$

◁그림 5> NP

$$NP = \bigcup_i \text{NTIME}(n^i)$$

이때, TIME(t(n))은 'L : L은 튜링 머신에 의하여 O(t(n)) 시간으로 결정될 수 있는 언어' 와 같은 집합을 말합니다.

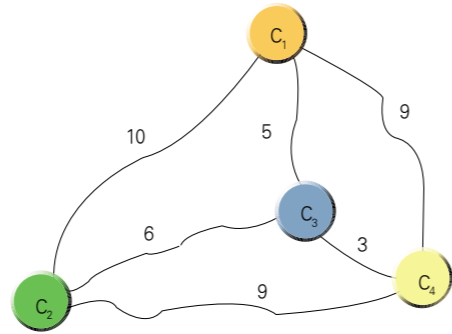
명백하게 P는 EXPTIME에 포함됩니다. 즉, 다항식 시간으로는 풀 수 없고 오직 지수 시간(exponential time)으로 풀 수 있는 문제들은 EXPTIME - P가 되는 것입니다. 한편 또 다른 의미에서 매우 중요한 문제들의 집합이 있습니다. 그 집합의 이름은 NP입니다. NP라는 용어는 비결정적인 다항식 시간(nondeterministic polynomial time)으로부터 유래되었습니다.

NTIME(t(n))은 TIME(t(n))과 유사한 방법으로 정의됩니다. 차이가 있다면 언어를 결정하는 기계가 비결정적이라는 것이 다릅니다. 즉, NTIME(t(n))은 'L : L은 비결정적 튜링 머신에 의하여 O(t(n)) 시간으로 결정될 수 있는 언어' 라는 집합을 가리킵니다 (비결정적 튜링 머신이란 추측을 통하여 기계의 다음 단계를 정확히 예측하는 튜링 머신을 말합니다).

그렇다면 NP는 어떠한 의미를 가질까요? NP에 속하는 문제들은 사실 다항식 시간 내에 검증할 수 있는 문제들입니다. 다항식 시간 내에 검증한다는 의미는 문제의 해답에 대해서 다항식 시간 내에 옳고 그른 것을 판별할 수 있다는 이야기입니다.

주어진 문제에 대하여 P가 빠르게 정답을 찾을 수 있는 문제들의 집합이라면, NP는 빠르게 주어진 해답이 정답인지를 검증할 수 있는 문제들의 집합입니다. 역시 명백하게 P는 NP에 속합니다. NP에 속하지만 아직 P에 속한다고 증명되지 않은 대표적인 문제 중의 하나가 바로 다음의 문제입니다.

◁그림 6> 장물뱅이의 여행 문제에 대한 입력예



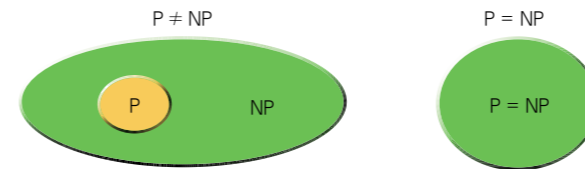
- ◆ 문제 : 장물뱅이의 여행(traveling salesman)
- ◆ 입력 : 무방향성 그래프(undirected graph) $G = (V, E, C(e))$ ($0 \leq e \in E$)
- ◆ 출력 : 각 도시(그래프의 정점(vertex))를 정확히 한 번 방문하는 최소 비용을 갖는 여행 경로를 발견하라(여기서 C(e)는 예지 e의 비용을 의미합니다).

이 문제의 풀이 방법은 매우 쉽습니다. 그래프의 모든 정점들에 대해서 나열하고 그 중 문제의 조건을 만족하면서 최소의 비용을 갖는 여행 경로를 출력하면 됩니다.

하지만 불행히도 제한한 이 알고리즘의 성능은 그렇게 우수하지 않습니다. 입력의 크기가 조금만 커져도 알고리즘의 수행 시간은 급속도로 증가할 것입니다(이 알고리즘은 O(n!)의 성능을 갖습니다). 현재까지 이 장물뱅이의 여행 문제에 대해서 다항식 시간 내에 풀 수 있는 알고리즘은 제안되지 않았습니 다. 즉, 지금까지 알려진 이 문제의 난이도에 대한 상한은 지수 함수라는 이야기입니다.

하지만 그렇다고 이 문제가 P에 속하지 않다고 결론난 것은 아닙니다. 이 문제의 하한이 지수 함수라는 사실 역시 증명되지 않았기 때문입니다. 사실 이 문제는 NP에 속하는 문제들 중에서 가장 어려운 문제 중의 하나입니다. 이러한 결과는 다항식 시간 전형 (polynomial time transformable)을 통해서 증명되었습니다. 즉, 장물뱅이의 여행 문제가 NP에 속하는 문제 중에서 가장 어려운 문제로부터 다항식 시간 전형됨으로써 이 문제가 NP에 속하는 어떤 문제보다도 쉽지 않다는 결론을 도출할 수 있는 것입니다. 이러한 증명에서 가장 어려운 것은 NP에 속하는 문제 중에서 가장 어려운 문제를 하나 찾아내는 과정일 것입니다.

◁그림 7> P와 NP의 두 가지 가능성



쿡(Cook)은 만족 문제(satisfiability problem)가 이러한 문제라는 것을 최초로 증명하였습니다. 또한 이와는 독립적으로 레빈(Levin) 역시 타일링 문제(tiling problem)가 NP에 속하는 문제 중에서 가장 어려운 문제라는 것을 보여주었습니다. 이렇게 NP에 속하면서 가장 어려운 문제를 'NP-complete' 이라고 부릅니다.

아직 그 어느 누구도 P와 NP의 관계에 대해서 정확하게 이야기하지 못하고 있습니다. P와 NP가 같다면 다항식 시간 내에 검증할 수 있는 문제들은 항상 다항식 시간 내에 풀 수 있다는 이야기가 됩니다. 만약 P와 NP가 같지 않다면, 다항식 시간 내에 검증할 수 있는 문제들 중에 반드시 다루기 힘든 문제가 존재한다는 이야기가 됩니다.


P와 NP의 관계는 이 세기의 가장 큰 난제로 남아 있습니다. 다만 대부분의 학자들은 NP에 속하는 특정 문제들에 대해서 오랜 연구에도 불구하고 다항식 시간 내에 풀 수 있는 알고리즘을 설계하지 못했기 때문에 P와 NP가 다를 거라고 추측하고 있습니다. 이는 매우 불행한 소식입니다. 우리가 실생활에서 접하는 매우 많은 문제들이 NP-complete이기 때문입니다. 만약 P와 NP가 서로

다르다면 우리는 이러한 문제들을 결코 효율적으로 해결할 수 없을 것입니다. 이러한 불행을 어떻게 극복할 수 있을까요?

사고하는 습관을 기르자

끝으로 알고리즘의 설계에 관한 당부의 부탁을 드리며 이 지루한(?) 글을 마칠까 합니다. 실생활의 문제들에 대해서 우수한 성능을 가지는 알고리즘을 설계한다는 것은 매우 어려운 작업입니다. 이 글을 읽고 있는 그 어느 누구도 이러한 사실을 부인하지 않을 것입니다.

성공적으로 알고리즘을 설계하기 위해서는 이 글의 범위를 훨씬 넘어서는 많은 경험과 지식을 필요로 합니다. 하지만 그러한 경험과 지식 못지 않게 중요한 것이 바로 사고하는 습관이라는 것을 간과해서는 안 될 것입니다. 우리는 그 어떤 순간에도 다음과 같이 묻고 고민해야 할 것입니다. 어떻게 해야 더 좋은 알고리즘을 설계할 수 있을까요? 이것이 최상의 방법일까요? 이것은 전산학도의, 그리고 알고리즘 개발자의 숙명적인 의무입니다.

이 글의 타이틀은 알고리즘의 이해입니다. 여러분에게 이 재미있고도 중요한 학문을 너무 재미없게 소개한 것은 아닐까 걱정이 앞섭니다. 하지만 천리길도 한 걸음부터, 알고리즘 공부도 이제부터 시작이라는 마음으로 한 걸음 한 걸음 정진할 것을 부탁드립니다. 더 좋은 세상을 만들기 위해서 더 좋은 알고리즘을 부탁하며 글을 마칠까 합니다. 

정리 | 조규형 | joky@korea.ac.kr