P, NP 문제 (1)

30 Nov 2017 | P (/blog/tags/#p) NP (/blog/tags/#np)

NP-complete (/blog/tags/#np-complete)
NP-hard (/blog/tags/#np-hard)

P/NP 문제에 대해 명철하고 해박하게 서술한 글을 소개해 보겠습니다. 서울대 컴퓨터공학부 이광근 교수님께서 쓰신 컴퓨터 과학이 여는 세계 (http://book.naver.com/bookdb/book_detail.nhn?bid=9078133)의 일부를 그대로 옮겨 왔습니다. 워낙 좋은 책이라 저 자신 또한 정리 목적으로 타이핑해 올려놓은 것이나 문제가 될 경우 바로 삭제하도록 하겠습니다. (밑줄 및 강조 표시는 제가 해놓은 것입니다)

현실적

쓸만한 알고리즘. 그 비용이 견딜 만하면 현실적인 알고리즘이다. 알고리즘의 실행비용(복잡도)이 입력 크기(n)에 대해 상수(k)승을 가지면(n^k) '현실적'이라고 본다. 예를 들어 $O(n^2)$ 혹은 $O(n^{3.5})$나 $O(n^{100})$ 등이다. 입력 크기가 커지면 다항으로 증가하는 비용(*polynomial complexity*)이다.

> 모든 게 순간이었다고 말하지 마라. 달은 윙크 한 번 하는 데 한 달이나 걸린다. - 이정록, '더딘 사랑'

왜 다항이 '현실적인' 비용인가? 그런 알고리즘은 컴퓨터 성능이 곱하기로 빨라지면($p \times \bullet$) 곱하기로 이득을 본다($p' \times \bullet$). 알고리즘 비용이 n^k라고 하자. 컴퓨터 성능이 p배 빨라지면 같은 입력에 대해서 비용이 n^k/p로 준다. 따라서 예전과 같은 시간에 처리할 수 있는 입력의 크기(즉 $n^k = m^k/p$인 m은) $p^{1/k} \times n$으로 늘릴 수 있다. $p^{1/k}$배 커진 입력을 같은 시간에 처리할 수 있게 되는 셈이다.

디지털 컴퓨터 성능은 약 2년마다 2배씩 성능이 좋아졌다. 알고리즘 비용이 $O(n)$이라 하자. 그러면 2년 후 2배 빨라진 컴퓨터로는 2배 큰 입력을 같은 시간에 처리할 수 있게 된다. 비용이 $O(n^2)$이라면? $\sqrt{2}$배 큰(약 40% 큰) 입력을 같은 시간에 처리할 수 있다.

실제 현실적인 비용은 다항 중에서도 $O(n), O(n^2), O(n^3)$까지 정도다. 그 이상의 차수는 아무리 다항이라도 실제 쓰기에는 너무 비싸다.

그러나 일단 다항에 들어오는 알고리즘을 찾으면(예를 들어 $O(n^{100})$) 그 문제는 대개 몇 년 지나면 $O(n^3)$이나 $O(n^4)$ 정도의 알고리즘을 가지게 되고, 계속 연구되면서 꾸준히 줄어든다. $O(n^{2.5}), O(n^{1.5})$ 등. 일단 다항 알고리즘을 가지게 된 문제는, 필요하다면 더 빠른(다항 차수가 적은) 알고리즘이 늘 꾸준히 찾아져 왔다.

이런 문제를 'P 클래스' 문제라고 한다. 현실적인 비용으로 풀 수 있는 문제들. 다항(*polynomial*) 알고리즘이 있는 문제들을 말한다.

비현실적

쓸 수 없는 알고리즘. 그 비용이 너무 크면 비현실적인 알고리즘이다. 알고리즘의 실행비용(복잡도)이 상수 위에 지수로 입력 크기가 올라 가면(k^n) 비현실적이라고 본다. 예를 들어 $O(2^n)$ 혹은 $O(2.5^n)$이나 $O(100^n)$이다. 입력 크기가 커지면 기하급수적으로 증가하는 비용(*exponential complexity*)이다.

이런 비용은 무시무시하다. 예를 들어 $O(2^n)$인 경우 입력 크기(n)가 100 정도만 되어도 2^{100}초다. 이 시간은 우주의 나이(10^{17}~10^{18}초)보다 길다. 우리 주변에 100개 이상 되는 입력 데이터를 다뤄야 하는 문제가 부지기수인 걸 생각하면 절망적이다.

$O(2^n)$의 경우 컴퓨터 성능이 2배 좋아져도 같은 시간에 풀 수 있는 입력의 크기는 고작 1만큼만 늘릴 수 있을 뿐이다. 절망적이다.

> 달이 지는 것을 막아줄 / 새벽녘에 뜨는 해를
> 수평선 밑으로 / 끌어내려 줄 사람이 아무도
> 없는가 / 그러면 나는 하루 더 살 수 있을 텐데
> - 작자 미상, '심청전' 중에서

기하급수로 증가하는 비용($O(k)$, 상수 $k > 1$)의 알고리즘은, 곱하기로 빨라져도($p \times \bullet$) 더하기만큼만 더 이득을 본다($p' + \bullet$). 컴퓨터 성능이 p배 빨라지면 같은 입력에 대해서 비용이 k^n/p로 준다. 따라서 예전과 같은 시간에 처리할 수 있는 입력의 크기(즉 $k^n = k^m/p$인 m은)는 $n + \log_k p$만큼만 늘어날 뿐이다.

사실 다항과 기하급수의 분류는 실용적이지는 않다. 실제 쓸 만한 알고리즘들은 다항에만 있고 그것도 차수가 아주 낮은 것들이기 때문이다. 프로그램으로 일을 봐야 하는 실제 사용자의 입장에서 다항이냐 기하급수냐는 관심 밖이다. 실용에서 중요한 건 다항 중에서도 차수가 낮냐 높냐다.

다항과 기하급수로 나누는 용도는 다른 데 있다. 컴퓨터로 풀기 어려운 문제가 과연 뭔지를 이해하려는 데 이 분류가 동원된다. 다항과 기하급수는 급이 다른 수준이 아니고 아예 종류가 다르다. 어떤 문제가 기하급수에 남아 있는 것과 다항으로 넘어오는 것. 문제의 근본적인 어려움이 있느냐 아니냐의 기준 같은 것이다.

P의 경계

컴퓨터로 풀기 어려운 문제란 무엇인가? 문제가 쉽고 어렵고의 경계는 어디인가?

그 경계를 정확히 그을 수 있으면 좋다. 분류가 되야 문제를 효과적으로 공략할 수 있기 때문이다. 무슨 병인지가 밝혀져야 치료법이 나온다. 문제가 나타나면, "그것은 어려운 문제다"라고 확인해 주면 있지도 않은 빠르고 정확한 알고리즘을 찾아 헤매는 낭비를 피할 수 있기 때문이다.

쉬운 문제는 정의하기 쉽고 판단하기도 쉽다. 현실적인 비용, 다항비용(*polynomial complexity*)의 알고리즘을 가지고 있으면 쉬운 문제다. 문제가 나타났다. 다항 알고리즘을 찾아 나선다. 찾아졌다. 쉬운 문제다.

난감한 건 어려운 문제다. 정의는 쉽지만 판단이 어렵다. 어려운 문제가 뭔가? 쉬운 문제가 아닌 문제다. 다항 알고리즘이 없는 문제다. 정의까지는 좋다. 문제는 판단하기다. 다항 알고

리즘이 없다는 것을 어떻게 판단하는가? 문제가 나타났다. 다항 알고리즘을 찾아 나선다. 아무리 찾아도 없다. 더 찾으면 나타날까? 아니면 아예 없는 걸까? 아예 없다면 어려운 문제인 건데, 있을지 없을지를 모르겠으니 어려운 문제인지 쉬운 문제인지를 모르는 거다. 다항 풀이법을 아직 찾지 못한 문제, 아예 없는 문제일까?

어려운 문제의 경계를 판단하는 방법을 찾으려면 그 경계 가까이 가야 한다. 그 경계 가까이 있는 문제들의 집합이 있다.

> 모든 경계에는 꽃이 핀다. - 함민복, '꽃'

NP 클래스

'NP 클래스' 문제라고 한다. **운에 기대면 현실적인 비용으로 해결할 수 있는(*non-deterministic polynomial*) 문제**들.

이 NP 클래스에는 우리가 관심 있는, 경계 가까이의 문제들이 들어온다. 현실적인 비용으로 풀 수 있는지는 아직 모르는 문제들. 알려진 알고리즘이라고는 기하급수의 비용을 가진 것 밖에는 없는 문제들. 그러나 운에 기대면 현실적인 비용으로 해결할 수 있는 문제들.

운에 기대면 현실적인 비용으로 풀 수 있다는 것이 무슨 뜻인가? 미로찾기 문제를 생각하자. 입구에서 출구로 이어지는 길을 찾아야 한다. 갈림길마다 선택을 한다. 잘못 선택하면 시간을 낭비한다. 답이 아닌 곳으로 빠져 한참을 헤매게 된다. 하지만 선택의 갈림길마다 운 좋게 늘 옳은 답을 선택했다면? 시간 낭비 없이 출구를 찾게 된다. 그런 운 좋은 선택을 현실적인 횟수만큼 해서 탈출로를 찾을 수 잇다면 '운에 기대면 현실적인 비용으로 해결할 수 있는' 문제인 거다.

정확히는 이렇다. NP 문제인지 아닌지는 예/아니오를 묻는 문제 중에서 "예"(탈출로가 있다)라고 답하기까지만 따진다. 즉 "예"라는 답을 운에 기대어 현실적인 비용으로 할 수 있으면 NP 문제라고 한다. 때문에 NP 문제는 종종 이렇게 설명하기도 한다. "예"라고 답한 근거(찾은 탈출로)를 받아서 그 근거가 맞는지를 현실적인 비용에 확인할 수 있는 문제를 NP 문제라고 한다. 답안을 받아서 그 답이 맞는지를 현실적인 비

용으로 확인할 수 있는 문제라는 설명에 수긍이 가는 게,'답을 받아서'라는 것이 '운 좋으면...'이라는 조건과 같은 말이지 않던가. 운은 답을 공짜로 알려주므로.

정의를 곱씹으면 NP 클래스는 P 문제를 모두 포함한다. 운에 기대지 않고도 쉽게 풀 수 있는 문제는 운에 기대도 쉽게 풀리는 문제다. 비행기를 타지 않아도 1시간 안에 갈 수 있는 거리는 비행기로도 1시간 안에 갈 수 있다.

그러나 위에 이야기한 대로 우리가 관심 있는 NP 클래스 문제들은 P 문제가 아니고 P와의 경계 가까이에 있는 문제다. 아직 찾지는 못했지만 현실적인 알고리즘이 있을 것만 같은 문제들.

다음과 같은 문제가 그런 NP 문제들이다. 흥미롭게도 우리 주변에 흔히 나타나는 문제들이다.

- 주어진 지도 위의 모든 도시를 한 번씩만 방문하는 경로가 있을까? 해밀턴 경로(*hamiltonan path*)라고 한다.

  지도에는 도시와 도시를 잇는 선이 있다. 도로가 있다는 뜻이다. 출발도시를 선택한다. 다음 도시는 그 도시와 도로로 연결된 도시다. 여러 이웃 도시 중에 하나를 선택해서 이동한다. 모든 도시를 한번씩 방문하는 경로가 있는 지도라면, 이런 선택이 모두 운 좋게 되면 그런 경로를 찾을 수 있다. 그런 운 좋은 선택은 도시의 개수만큼 하면 된다. NP 문제인 게 확실하다.

  운에 기대지 않는 단순한 알고리즘은 모든 경우를 살펴보는 것이다. 방문할 도시의 수가 n개라고 하자. 모두 한 번씩 방문하는 경로의 후보들은 최대 $n!$만큼이다(n개 도시들을 일렬로 순서 매기는 가짓수). 이 후보 경로 하나하나마다 주어진 지도에서 가능한지 살펴야 한다.

  다항시간에 푸는 알고리즘은 아직 없다.

- 주어진 예산으로 주어진 지도의 도시들을 다 방문하고 돌아올 수 있을까?

도시와 도시를 연결하는 도로가 있고, 도로의 길이만큼 비용이 든다고 하자. 주어진 입력은 도시들 사이의 도로망, 출발 도시, 그리고 주어진 예산이다. 출발한 시에서 다음 시를 선택하는 것이 필요하다. 어떻게? 섣불리 선택했다간 주어진 예산 내에 모든 도시를 방문하고 돌아오는 게 불가능할 수 있다. 돌아오는 방법이 있다면, 운 좋으면 그런 행선로를 찾을 수 있다. 몇 번 운이 좋아야 할까? 현재 도시에서 다음 도시를 선택하는 횟수만큼이다. 운에 기대어 선택해야 하는 횟수는 방문할 도시의 개수만큼이다. NP 문제인 게 확실하다.

운에 기대지 않는 단순한 알고리즘은 모든 경우를 살펴보는 것이다. 방문할 도시의 수 n개가 있고, 모든 두 도시 사이에 직통 도로가 뚫려 있다고 하면 방문하는 경로는 최대 $n!$개다. 이 모든 가능한 경로마다 비용을 보고 주어진 예산 안에 있는 것이 있으면 된다.

다항시간에 푸는 알고리즘은 아직 없다.

- 주어진 부울식(*boolean formula*)이 참이 되게 할 수 있을까?

  여기서 부울식은 변수와 그리고(*and*), 또는(*or*), 아닌(*not*)으로 조립된 식을 말한다. 예를 들어 $x((-x) + (-y))$. 이 식을 참으로 만드는 경우는 x가 1, y가 0이다. 어떤 식은 참으로 만들 방법이 없는 경우가 있다. 예를 들어 $x(-x)$가 참이 되는 경우는 없다.

  주어진 부울식이 n개의 변수로 구성돼 있으면 n개의 변수마다 0 또는 1이 가능하다. 변수마다 0과 1 중 하나를 선택해서 부울식의 참 거짓을 확인할 수 있다. 참일 수 있는 식이라면, 그 선택이 운 좋으면 참으로 만드는 경우를 찾게 될 것이다. n번 운 좋으면 참을 만드는 경우를 만들 수 있다. NP 문제인 게 확실하다.

  운에 기대지 않고 하는 단순한 알고리즘은 모든 경우를 살펴보는 것이다. 변수 n개가 0 또는 1을 가지는 경우의 수는 2^n 개다. 각 경우마다 주어진 부울식이 참인지 거짓

인지를 본다. 최대 2^n 만큼 살펴봐야 한다.

이 문제도 다항시간에 푸는 알고리즘은 아직 없다.

- 주어진 자연수를 인수분해하라.

  n자리 10진수를 받는다. 1도 아니고 자신도 아니면서 그 수를 나누는 수가 있을까? 있다면, n번만 운 좋으면 그 인자를 찾을 수 있다. 자릿수마다 0~9 중에서 운 좋게 선택하면 그 인수가 될 수 있다. NP 문제인 게 확실하다.

  운에 기대지 않고 하는 단순 알고리즘은 모든 경우를 살펴보는 것이다. 받은 자연수보다 작은 수를 2부터 시작해서 하나하나 체크해 본다. 그 수로 원래 수가 나눠지는지, n자리 10진수이므로 최대 $10^n - 1$만큼 반복해야 한다.

  디지털 컴퓨터에서 자연수를 소인수분해하는 다항 알고리즘은 아직 없다.

- 아미노산이 연결된 1차원의 아미노산 실을 접어서 3차원의 단백질 구조물을 만들라. 단, 만들어진 구조를 유지하는 데 필요한 에너지가 어느 이하가 되는 3차원 생김새를 찾아야 한다.

  단백질 접기(*protein folding*)라고 하는데, 모든 생명체가 자신의 모습을 만들 때 하는 일이다.

  기차 같이 연결된 아미노산들을 처음부터 차례로 올바른 방향으로 접어간다면 에너지 예산 안에서 지탱할 수 있는 구조물을 만들게 될 것이다. 운 좋으면 항상 올바른 방향으로 접을 수 있다. 아미노산 실의 길이만큼(아미노산의 개수 n만큼) 잘 접으면 된다. NP 문제다.

  운에 기대지 않고 하는 단순한 알고리즘은 각 아미노산마다 3차원에서 접을 수 있는 방향이 6개라면 6^n 개의 경우를 모두 시도해서 그중 에너지 조건을 만족하는 구조물인지 봐야 한다.

- 1000만 관객을 넘기는 영화를 제작하라. 제작 중 n번 디자인 선택을 해야 한다.

  매번 디자인 선택(시나리오 작성, 배우 캐스팅, 촬영, 편집 등) 때마다 올바른 방향으로 해 간다면 만들 수 있다. 운 좋으면 매번(n번) 그런 방향으로 디자인 선택을 하게 된다. NP 문제다.

  운에 기대지 않고는 비현실적인 비용이 든다. 매번 마주치는 디자인 후보의 가짓수가 2개뿐이라 해도, 만들 수 있는 영화의 가짓수는 2^n 개다. 이것들을 모두 살피면서 1000만 관객을 넘길 것을 선택해야 한다.

이런 NP 문제들은 곤혹스럽다. 주변에 흔히 만나는 문제들인데 현실적인 비용으로 정확히 답을 내는 알고리즘은 아직 없다니.

> 내가 살아 있는 동안 / 다 마칠 수 있는 일이란 / 이 세상에 없다 / 진정코 이 세상이란 몇천 년 이나 걸려야 / 집 한 채 지을 수 있음이여 - 고은, '몇 천 년'

그런데 아이러니하게도 이런 NP 문제들은 그렇기 때문에 유용하기도 하다. 디지털 암호기술을 버티는 기둥이 이런 NP 문제들이기 때문이다. 메시지의 보안을 위해 이런 NP 문제들이 쓰인다. 암호화된 메시지를 도청한 사람이 암호를 풀려면 이런 NP 문제를 풀어야 하게끔 만들어 놓는 것이다. 알고리즘 복잡도(*algorithm complexity*)의 초석을 놓은 미뉴엘 블럼(Manuel Blum)이 시작한 역발상이다. 과학이 기술로 응용되는 지점에서 종종 목격되는 역발상의 지혜다.

아무튼, 어려운 문제가 뭔지를 판단하는 방법을 컴퓨터과학에서 어떻게 찾아가는지, 그 이야기를 계속하자. NP 클래스라는 문제들로 어려운 문제와 쉬운 문제의 경계를 더듬고 있다.

오리무중

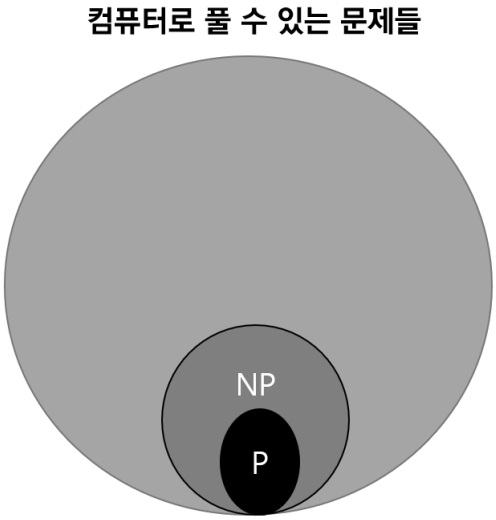
P가 NP가 같은지 아닌지는 아직 누구도 답하지 못했다.

정말 어려운 문제인지가 아리송한 경계의 문제들(NP), 이것들이 혹시 쉬운 문제(P)는 아닐까? NP 문제는 운에 기대면 현실적인 비용으로 풀 수 있는(*non-deterministic polynomial*) 문제들이다. 이런 문제들을 운에 기대지 않고도 현실적인 비용으로 풀 수 있을까? 현재의 디지털 컴퓨터(튜링기계)로는 아마도 $P \neq NP$ 일 거라고 많은 전문가들은 추측하고 있다.

$P \neq NP$라면,"어려운 문제의 경계가 어디냐"라는 질문에 조금 구체적으로 답할 수 있게 된다. 운에 기대지 않고는 비용이 너무 드는 문제들, 그런 문제들이 존재하는 게 확실해졌고, 여기부터가 어려운 문제들의 시작이라고 정의해도 되는 게 아닐까. 운에 기대기만 하면 P로 넘어가는 아슬아슬한 경계에 있으므로 P는 될 수 없으므로.

만일 $P = NP$라고 판명되면, 이 세계의 많은 것이 기계적인 세상으로 위축되는 느낌이다. $P = NP$인 세상은 비유하자면 "명작을 현실적인 비용으로 컴퓨터가 자동 생산할 수 있다"가 된다. 명작 만들기는 선택의 연속이었을 것이다. 각 선택마다 수없이 많은 디자인 후보 중에서 명작으로 가는 하나를 절묘하게 선택해가면서 엮어 놓은 것 아닐까. 이 선택을 자동화해서 현실적인 비용에 늘 그런 명작이 자동으로 만들어지도록 할 수 있을까? 만든 음악을 듣고 좋다 나쁘다 판단하기는 쉽지만, 그런 음악을 만드는 것은 그보다는 차원이 다르게 어려운 문제가 아닐까. 좋은 그림, 좋은 소설, 좋은 시 등이 마찬가지 아니던가. 수많은 디자인 선택을 절묘하게 구성한 결과들일 것이다. $P = NP$면 그런 명작도 자동으로 쉽게 만들 수 있다는 이야기다. 뭔가 아니라는 느낌 아닌가.

$P \neq NP$? 튜링상(*Turing Award*)은 그 답을 기다린다. 컴퓨터 과학의 노벨상이다.



(https://imgur.com/kyu3iK5)

P의 바깥

$P \neq NP$가 사실이라면, P 말고 NP에만 있는 문제들이 있는 것이다. 그런 문제는 P가 아니므로 어려운 문제다. 그렇다면 $P \neq NP$라는 전제 아래 어떤 문제가 P 바깥에 있는지(어려운 문제인지) 판단하는 방법이 있을까?

한 방법이 겨우 있다. '겨우'라는 표현을 쓴 것은, 그 방법이 믿을 만하지만 완전하지는 않기 때문이다. 모든 P 바깥의 문제가 그 방법으로 확인되는 건 아니다. 그 방법을 통과하면 P 바깥의 문제인 것은 확실하지만(믿을 만한 방법이지만), 어떤 문제는 P 바깥의 문제인데도 그 방법을 통과하지 못할 수 있다(완전하지는 않다).

그 방법은 NP 클래스의 재미있는 성질 때문에 가능하다. 그 성질은, NP 문제 중에서도 가장 어려운 문제가 있다는 것이다. NP 클래스의 나머지 문제들은 모두 이 문제보다 쉽다. 따라서 $P \neq NP$라면 이 문제는 당연히 P 바깥이다. 내가 만난 문제가 그 문제만큼 어렵다면 그 문제는 분명 P 바깥인 거다. 이 사실을 발견하는 데는 건너풀기(*problem reduction*)라는 개념이 지렛대로 사용된다.

건너풀기

> 강 건너가 건너온다. / 누가 끌배를 끌고 있다. -
> 이문제, '독거'

NP로 분류한 문제들은 모두 단단히 연대해 있다. 달라 보이지만 사실은 한 덩어리로 움직이는 하나다. 어려운 문제 쪽에 있지만 쉬운 문제 진영에 바싹 다가서 있는 문제들. 이것들은 하나만 넘어오면 모두 한 몸 같이 넘어오는 성질을 가지고 있다.

이런 연대가 건너풀기(*reducibility, problem reduction*)라는 개념으로 확인된다. '건너풀기'란 건너에서 간접적으로 풀기다. 문제 A를 풀어야 한다. 건너편 문제 B를 풀고 그 알고리즘을 이용해서 A를 풀 수 있다. A 문제를 푸는 데 B 문제로 건너 푼 것이다. 간접적으로 푼 것이다. 단, B 문제로 푼 답을 A 문제의 답으로 옮기는 건 쉬워야(다항 비용으로 되야) 한다.

건너풀기는 일상에서 흔하다. 곱하기를 더하기로 건너 풀 수 있다. 더하기만 할 줄 알면 곱하기는 해결되기 때문이다.
5×12는 5를 12번 더하면 된다. 이게 건너풀기다.

건너풀기로 NP 문제들을 모두 지배하는 하나의 문제가 있다. NP 클래스의 문제 중에 '종결자' 역할을 하는 대표적인 문제가 있는 것이다. 이 문제만 현실적인 비용으로 풀리면, 나머지 NP 문제들은 모두 이 문제로 건너 풀 수 있다. 모든 NP 문제를 종결자 문제로 바꿔서 낼 수 있고, 종결자 문제의 알고리즘으로 풀어서 그 답을 원 문제의 답으로 바꾸면 된다. 종결자 문제의 알고리즘이 다항 알고리즘이면(그리고 문제를 바꾸고 답을 바꾸는 비용이 다항이면), 모든 NP 문제도 다항 알고리즘으로 풀리는 것이다. 이 종결자 문제로 건너 풀면서 모든 NP 문제들이 다 현실적인 비용으로 풀리는 것이다.

NP에 있는 이런 종결자 문제를 NP-완전(*complete*) 문제라고 한다. 모든 NP 문제들이 빠짐없이 이 문제를 통해 건너 풀리기 때문이다. '완전'이라는 훈장을 달아준 이유다. NP 문제들 중에서 가장 어려운 문제라고 볼 수 있다.

- 컴퓨터과학에서 *complete*라는 단어가 종종 나온다. '빠뜨림이 없다'는 뜻이다. 그래서 '완전'이다. 참고로, 종결

자 역할은 하지만 NP 문제인지 확인되지 않은 문제는 NP-하드(*hard*) 문제라고 한다.

NP 문제들의 종결자 문제를 처음으로 발견한 사람은 스테픈 쿡(Stephen Cook)이다. 그는 건너풀기 개념을 가지고 NP에 있는 하나의 문제를 찾아서 그 문제가 NP의 모든 문제보다 어렵다는 놀라운 사실을 증명했다. 1971년이었고, 이 업적으로 1982년 튜링상을 받는다. 쿡이 찾은 최초의 NP-완전 문제는 주어진 부울식이 참일 수 있는지 결정하는 문제였다. 쿡에 이어 1972년, 스물한 개의 다양한 NP-완전 문제들이 발견된다. 특히 우리가 일상에서 마주치는 흔한 문제들이 NP-완전 문제들이었다. 예를 들어 모든 도시를 한 번씩 방문하는 경로(해밀턴 경로) 찾기 문제 등이다. 리처드 카프(Richard Karp)가 이를 증명했고 그 업적으로 튜링상을 받는다(1985년). 현재 우리가 일고 있는 NP-완전 문제는 수천가지다.

어려운 문제인지 판단하기

이제 이 NP-완전 문제가 어려운 문제를 판별하는 기준으로 쓰일 수 있다. 문제가 '어렵다'는 정의는 'P가 아니다'라는 것이다. $P \neq NP$가 사실이라면(그럴 것이라고 추측하고 있다), 주어진 문제가 적어도 NP-완전 문제만큼 어렵기만 하면 그 문제는 P 바깥의 문제다.

그래서 어떤 문제를 만났을 때, 다항 알고리즘을 찾지 못하고 있다면 NP-완전인지, 혹은 알려진 NP-완전 문제를 그 문제로 건너풀 수 있는지를 확인한다. 사실이라면 적어도 NP-완전 문제만큼 어려운 문제다. 그러면 그 문제는, $P \neq NP$가 사실이라면 P 바깥의 문제임이 확실하다.

이게 컴퓨터과학이 찾아낸 어려운 문제 판별법이다. $P \neq NP$인지를 아직 확인 못해서 아쉽지만, 지금까지 이게 최선이다.

(https://ratsgo.github.io/natural%20language%20processing/2019/09/12/embedding/)

Related Posts

한국어 관형사절의 시간 표현 09 Nov 2017
(/korean%20linguistics/2017/11/09/detertime/)
한국어의 부사절 내포 15 Sep 2017
(/korean%20linguistics/2017/09/15/advp/)
한국어의 접속문 18 Jul 2017
(/korean%20linguistics/2017/07/18/consen/)
관형사란 무엇인가 28 Jun 2017
(/korean%20linguistics/2017/06/28/nounad/)
한국어의 문장성분 01 Oct 2017
(/korean%20linguistics/2017/10/01/sentcomp/)

Comments

토론 참여하기

다음으로 로그인

또는 디스커스에 가입하세요.

이름

TOT0Ro • 3년 전

진짜 너무 감사합니다. p np를 이렇게 쉽게 설명해주시는 분 처음 만났습니다.

답글 • 공유 ›

호준 • 3년 전

늦은 나이에 외국에서 컴퓨터 공학을 공부하고 있는데 때때로 들러 좋은 글을 읽고 갑니다.
감사합니다.

답글 • 공유 ›

이동수 • 4년 전

p, np 문제 관련해서 여러번 이해를 시도하고 다양한 글을 읽어봤는데,