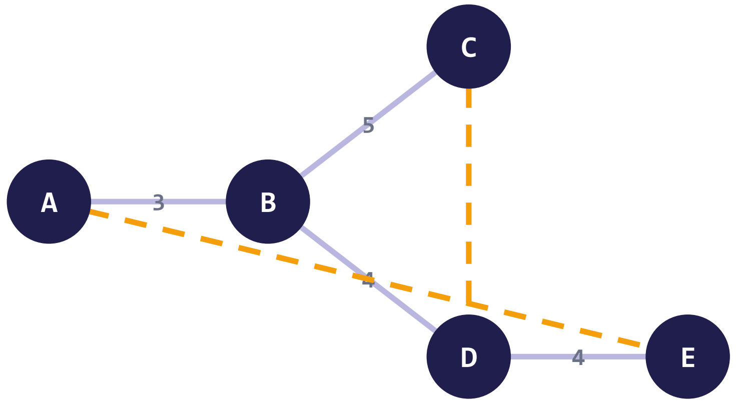
그림으로 쉽게 설명하는 파이썬 알고리즘

CHAPTER 10

근사 알고리즘

Approximation Algorithms

완벽함을 포기하되, 얼마나 손해인지는 안다



회색 = MST · 점선 = 지름길로 만든 투어

투어 ≤ 2 × OPT

CONTENTS

이번 장에서 배울 내용

9장에서 "근사해로 방향을 바꾸라"고 했다 — 이 장은 그 근사해에 수학적 보증서를 붙이는 방법이다.

10.1

왜 근사 알고리즘인가?

10.1

지수 시간 알고리즘은 현실에서
'없는 것'이나 마찬가지다.

NP-난해의 장벽 — 지수 시간은 기다릴 수 없다

NP-난해 문제의 최적해를 찾는 알려진 알고리즘들의 시간 복잡도는 다음과 같다.

$O(2^n)$

지수 시간

$O(n!)$

팩토리얼 시간

n이 작을 때 — 도시 10개

모든 경로를 계산해도 금방 끝난다.

n이 조금만 커지면 — 도시 50개

슈퍼컴퓨터로도 우주의 나이보다 긴 시간이 필요하다.

"언젠가는 끝난다"는 보장은 실무에서 아무 쓸모가 없다

9장에서 본 대로 이 문제들은 다항 시간 알고리즘이 존재하지 않을 가능성이 매우 높다. 기다리는 전략은 선택지가 아니다.

그렇다면 남은 선택지는 하나뿐이다

"최적해를 반드시 찾겠다"는 요구를 내려놓는 것이다. 다만 아무렇게나 내려놓는 것이 아니라, 얼마나 손해를 보는지 숫자로 못 박아 두고 내려놓는다.

현실 비즈니스 환경의 냉혹한 요구

택배 배송 최적화

새벽 배송 트럭은 계산이 끝날 때까지 내일 아침을 기다려 주지 않는다.

실시간 트레이딩

주식 트레이딩 시스템은 0.1초 안에 결정을 내려야 한다.

반도체 설계

수천 개의 부품 배치를 제한된 시간 안에 효율적으로 최적화해야 한다.

현실 세계에서 지수 시간 알고리즘은 "없는 것"이나 마찬가지다

아무리 정확해도 마감 시각을 넘기면 그 답은 쓸 수 없다. 정확성과 시간 중 하나를 골라야 한다면, 현장은 언제나 시간을 고른다.

그래서 필요한 것이 근사 알고리즘이다

"꽤 좋은 답을 확실히 빠르게, 그리고 얼마나 좋은지도 알려 주는" 도구다.

최적화 문제 vs 결정 문제

9장의 NP 이론은 결정 문제(Yes/No) 형식을 썼다. 10장에서는 현실에 맞게 가장 좋은 "값"을 찾는 최적화 문제를 다룬다.

외판원 순회 문제 (TSP)

결정 — 50만 원 이하인 경로가 존재하는가? (Y/N)

최적화 — 비용이 가장 적은 경로는 무엇인가? (최솟값)

배낭 문제 (knapsack)

결정 — 가치 합이 K 이상이 되게 담을 수 있는가? (Y/N)

최적화 — 담을 수 있는 최대 가치는 얼마인가? (최댓값)

결정 버전이 NP-완전 ⇒ 최적화 버전은 NP-난해

최적화 버전이 더 어렵다 — 값을 찾아야 할 뿐 아니라, 그보다 나은 값이 없음까지 보여야 하기 때문이다.

① 휴리스틱 (heuristics)

"경험상 대충 잘 맞더라" — 경험적인 법칙이나 직관에 의존하는 방법

예를 들어 TSP에서 "일단 가장 가까운 도시로 이동"을 반복하는 것이 전형적인 휴리스틱이다. 5장에서 본 가장 가까운 이웃 기법이 바로 이것이다.

장점

구현이 쉽고 실행이 빠르다.
실전에서도 꽤 괜찮은 결과를 낸다.

단점

품질 보장이 전혀 없다.
최적해와 얼마나 차이 나는지 알 수 없다.

"잘 나온 것 같다"와 "얼마나 잘 나왔는지 안다"는 전혀 다른 이야기다.

② 근사 알고리즘 (approximation algorithms)

"최악의 경우에도 이 정도는 보장한다" — 수학적 엄밀함을 갖춘 방법

근사 알고리즘은 다음 두 가지를 반드시 보장한다.

보장 ① 어떤 입력이 들어오더라도 다항 시간 안에 반드시 종료한다.

보장 ② 결과값이 최악의 경우에도 최적해의 k배를 절대 넘지 않음을 수학적으로 증명한다.

k = 근사 비율 (approximation ratio)

이 장 전체를 관통하는 핵심 성능 지표

보증서가 붙어 있다는 점이 휴리스틱과의 결정적 차이다.

휴리스틱 vs 근사 알고리즘 비교 요약

구분	휴리스틱	근사 알고리즘
품질 보장	없음	수학적 증명으로 보장
실행 시간	다항 시간	다항 시간
최악의 경우	알 수 없음	α배 이내로 보장
신뢰성	운에 의존한다	수학적 안전장치가 있다

실행 시간은 같은데 왜 굳이 근사 알고리즘인가

둘 다 다항 시간에 끝난다. 차이는 결과를 신뢰할 근거가 있느냐다. 근사 알고리즘은 "최악이어도 이 선은 넘지 않는다"는 계약서를 함께 준다.

실무에서는 둘을 함께 쓰기도 한다

근사 알고리즘으로 안전선을 확보한 뒤, 휴리스틱으로 그 해를 더 다듬는 방식이 흔하다.

10.2

근사 알고리즘의 기본 개념

10.2

"얼마나 좋은가"를 숫자로 말하려면
먼저 표기법부터 정해야 한다.

해(solution)와 해의 값(value)

해 (solution)

모든 제약 조건을 만족하는 하나의 선택 또는 구성

정점 커버 → 선택된 정점 집합

TSP → 순환 경로

집합 커버 → 선택된 부분집합 모음

해의 값 (value)

그 해가 만들어 내는 목적 함수의 수치적 결과

정점 커버 → 정점의 개수

TSP → 경로의 총 길이

집합 커버 → 선택된 집합의 수

근사 분석이 보는 것은 "값"이다

"어떤 해를 선택했는가"가 아니라 "그 해의 값이 최적해의 값에 비해 얼마나 차이 나는가"를 따진다. 해의 모양이 최적해와 전혀 달라도 값만 가까우면 좋은 근사다.

OPT(I) 와 A(I) 표기법

OPT(I)

이론적으로 존재할 수 있는 가장 완벽한 정답의 값

NP-난해이므로 정확한 값을 모르는 경우가 대부분

A(I)

근사 알고리즘 A가 입력 I에 대해 실제로 계산해 낸 결과값

A(I)가 OPT(I)와 최대한 비슷하기를 바란다

I = 입력 인스턴스 — 그래프, 거리 행렬, 집합 등 문제를 정의하는 모든 정보

목표는 "모든 입력 I에 대해" 성립하는 보장이다

운 좋은 몇몇 입력에서 잘 나오는 것은 의미가 없다. 최악의 입력에서도 일정 비율 안에 든다는 것을 증명해야 한다.

근사 비율 (approximation ratio)

α(알파)로 표기하며 언제나 α ≥ 1 이다 — "최악의 경우 정답과 몇 배나 차이가 나는가?"

α가 1에 가까울수록 성능이 좋은 알고리즘

1	2	∞
완벽	좋음	나쁨

α는 "최악의 경우"에 대한 약속이다

평균적으로 얼마나 좋은지가 아니라, 가장 나쁜 입력에서도 넘지 않는 상한이다. 그래서 실제 성능은 대체로 α보다 훨씬 좋다.

α = 1.0001 이어도 상수 배 근사이고, α = log n 이면 상수가 아니다 — 이 구분이 중요하다.

최소화 문제의 근사 비율

값이 작을수록 좋은 문제다. 예를 들어 TSP의 최소 경로 비용이 여기에 속한다.

$$A(I) \le \alpha \cdot OPT(I)$$

또는 $A(I) / OPT(I) \le \alpha$

해석 "내가 구한 값이 아무리 나빠도, 최적값의 α배를 넘지 않는다"

$\alpha \ge 1$ 인 이유 최소화 문제에서 근사해는 최적해보다 작을 수 없다. 즉 언제나 $A(I) \ge OPT(I)$ 이다.

예 — 2-근사 TSP

최악의 경우에도 최적 거리의 2배를 넘지 않는 투어를 보장한다. 이 장의 §10.3에서 실제로 만들어 본다.

최대화 문제의 근사 비율

값이 클수록 좋은 문제다. 예를 들어 배낭 문제의 최대 가치가 여기에 속한다.

$$\mathrm{OPT}(I) \le \alpha \cdot A(I)$$

또는 $\mathrm{OPT}(I) / A(I) \le \alpha$

해석 "우리 알고리즘의 해는 최적값의 최소 $1/\alpha$ 이상을 보장한다"

$\alpha \ge 1$ 인 이유 최대화 문제에서 근사해는 최적해보다 클 수 없다. 즉 언제나 $A(I) \le \mathrm{OPT}(I)$ 이다.

예 — 2-근사 최대화

최악의 경우에도 최적값의 절반(1/2) 이상은 확보한다는 뜻이 된다.

우리의 목표 — α = 1을 향한 끝없는 도전

"다항 시간 안에 동작하면서, 근사 비율 α가 가능한 한 1에 가까운 알고리즘을 설계하는 것"

1

α = 1 은 사실상 불가능

완벽한 최적해를 다항 시간에 낸다는 뜻이므로, 그것을 해내면 P = NP를 증명한 것이 된다.

2

현실적 목표는 상수 배

α = 2, 1.5, 1.01 처럼 1에 가까운 상수 비율을 노린다. 문제마다 도달 가능한 한계가 다르다.

3

때로는 입력에 따라 커진다

α = ln n 처럼 입력 크기와 함께 커지는 경우도 있다. 집합 커버가 대표적이다.

근사 알고리즘 연구는 "α를 얼마나 낮출 수 있는가"의 역사다

Metric TSP는 2에서 1.5로 내려왔고, 정점 커버는 아직 2에 머물러 있다. 어떤 문제는 더 내려갈 수 없다는 것까지 증명되어 있다.

10.3

거리 공간의 TSP

10.3

현실 세계의 상식 하나를 조건으로 추가하면,
TSP가 근사 가능한 영역으로 내려온다.

일반 TSP — 근사조차 불가능하다

일반적인 TSP는 다항 시간 안에 어떤 상수 비율로도 근사할 수 없다

100배, 1000배, 심지어 1,000,000배로 잡아도 소용없다. 만약 가능하다면 그것으로 P = NP가 증명되어 버리기 때문이다. (자세한 증명은 §10.6)

그렇다면 방법이 없는 것일까? 있다. 문제를 조금 좁히면 된다.

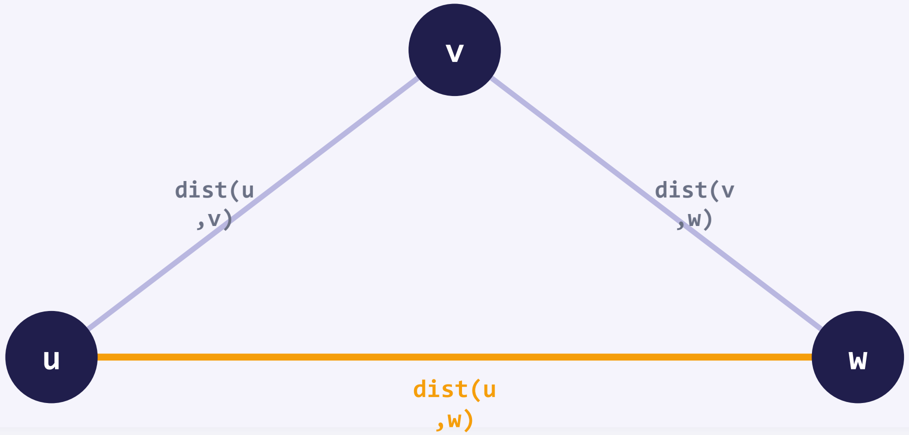
해결책 현실 세계에서는 당연히 성립하는 상식적인 조건을 하나 추가해서 문제의 난이도를 낮춘다.

거리 공간 (metric space) 의 조건 = 삼각 부등식

문제를 못 풀겠으면 문제를 바꾼다 — 근사 알고리즘 설계의 첫 번째 기술이다.

삼각 부등식 (triangle inequality)

$$\mathrm{dist}(u, w) \le \mathrm{dist}(u, v) + \mathrm{dist}(v, w)$$



직관적 의미

u에서 w로 바로 가는 것이, v를 거쳐 돌아가는 것보다 비쌀 수는 없다.

현실 세계 비유

서울에서 부산을 가는데 굳이 강릉을 찍고 가는 편이 더 빠를 수는 없다.

이 조건이 붙으면 TSP는 "근사 가능한 영역"으로 내려온다 — 지름길을 만들어도 손해가 아니기 때문이다.

MST 기반 2-근사 알고리즘 — 핵심 아이디어

"모든 도시를 연결하는 가장 싼 뼈대(MST)를 만들고, 그걸 따라 한 바퀴 돌면 TSP 경로가 된다"

1

MST 구성

모든 도시를 연결하는 최소 신장 트리를 만든다. 5장에서 배운 크루스칼이나 프림으로 충분하다.

2

왕복으로 따라가기

MST의 모든 간선을 왕복으로 훑는다. 이른바 더블 트리, 곧 오일러 투어다.

3

중복은 건너뛴다

이미 방문한 도시는 지나친다. 삼각 부등식 덕분에 건너뛰어도 손해가 없다.

핵심은 MST를 하한값(lower bound)으로 쓰는 것이다

알 수 없는 OPT를 직접 다루는 대신, 계산할 수 있는 MST와 비교한다. 근사 알고리즘 증명은 거의 언제나 이런 "계산 가능한 하한값 찾기"에서 출발한다.

핵심 부등식 — Weight(MST) ≤ OPT

왜 MST의 비용이 TSP 최적해보다 항상 작을까? 네 걸음이면 증명된다.

① TSP 최적 경로(OPT)는 모든 도시를 연결하는 순환 경로다.

② 이 경로에서 간선 하나를 제거하면 신장 트리가 된다.

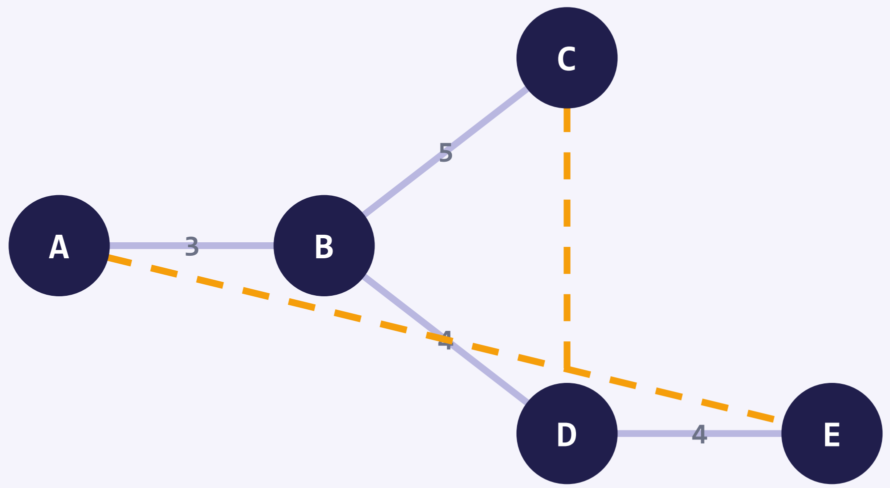
③ 간선 하나를 뺐으므로 이 신장 트리의 비용은 OPT보다 작다.

④ MST는 모든 신장 트리 중 비용이 최소이므로, MST는 그보다도 작거나 같다.

Weight(MST) ≤ OPT

강력한 하한값 확보!

MST 왕복 경로와 지름길 (shortcut)



회색 실선 = MST · 주황 점선 = 지름길로 이어 붙인 투어

MST를 DFS로 훑으면

A → B → C → B → D → E → D → B → A 라는 왕복 경로가 나온다.

중복 방문이 생긴다

B와 D를 여러 번 지난다. 이것은 TSP의 조건에 어긋난다.

지름길로 건너뛴다

C → B → D 를 C → D 로 잇는다. dist(C,B) + dist(B,D) ≥ dist(C,D) 이므로 손해가 없다.

최종 경로 : A → B → C → D → E → A

삼각 부등식이 없다면 이 "건너뛰기"가 오히려 비용을 늘릴 수도 있다.

증명 — 왜 2배를 넘지 않는가

①	MST의 간선을 두 배로 늘려 오일러 투어를 만든다 → 비용 = 2 × Weight(MST)	2·MST
②	지름길을 만들 때 삼각 부등식에 의해 비용은 줄거나 같다 → 최종 비용 ≤ 2 × Weight(MST)	≤ 2·MST
③	앞에서 얻은 하한값 Weight(MST) ≤ OPT 를 대입한다.	MST ≤ OPT

최종 비용 ≤ 2 · Weight(MST) ≤ 2 · OPT

∴ 이 알고리즘은 2-근사 알고리즘이다

알고리즘 의사코드 — MetricTSP_2Approx

metric_tsp_2approx.pseudo

```
Algorithm MetricTSP_2Approx(G, d, s)

  T     <- MST(G, d)              // 1) MST 구성
  order <- PreorderDFS(T, s)      // 2) DFS 전위순회
  tour  <- empty
  visited <- empty

  for v in order do               // 3) 지름길(shortcut)
    if v not in visited then
      append v to tour
      add v to visited

  append s to tour                // 4) 시작점으로 복귀
  return tour
```

1) MST

크루스칼 또는 프림. 여기까지는 완전히 P 클래스의 작업이다.

2) 전위순회

방문 순서를 만든다. 이 순서가 곧 투어의 뼈대가 된다.

3) 지름길

이미 방문한 도시는 건너뛴다. visited 집합 하나면 충분하다.

4) 복귀

순환 경로이므로 마지막에 출발점을 다시 붙인다.

전체 시간 복잡도는 MST 구성 단계가 지배한다

프림이나 크루스칼로 O(E log V), 이후의 순회와 지름길 처리는 O(V)에 끝난다. 완전한 다항 시간 알고리즘이다.

파이썬 구현 흐름과 실행 결과

① 데이터 입력 및 초기화
도시 이름과 인접 행렬을 설정한다.

② 플로이드-워셜
끊어진 길을 이어 완전 그래프로 만든다.

③ 프림 알고리즘으로 MST
최소 신장 트리를 구성한다.

④ DFS 전위 순회
지름길이 적용된 경로를 만든다.

⑤ 결과 출력
총 비용을 계산하고 경로를 표시한다.

MST

```
MST 비용: 16
  A - B (3)    B - D (4)
  D - E (4)    B - C (5)
```

TOUR

```
경로: A -> B -> C -> D -> E -> A
총 비용: 27
```

MST 비용 16의 2배는 32 — 실제 투어 27은 그 안에 확실히 들어온다.

10.4

정점 커버 문제

모든 도로를 감시하면서
CCTV 대수는 최소로 줄이는 문제.

10.4

정점 커버 문제 정의

CCTV 감시 카메라 설치 문제로 비유하면 한눈에 들어온다.

정점 (V) 교차로 — CCTV를 설치할 수 있는 위치

간선 (E) 교차로를 잇는 도로 — 감시해야 할 대상

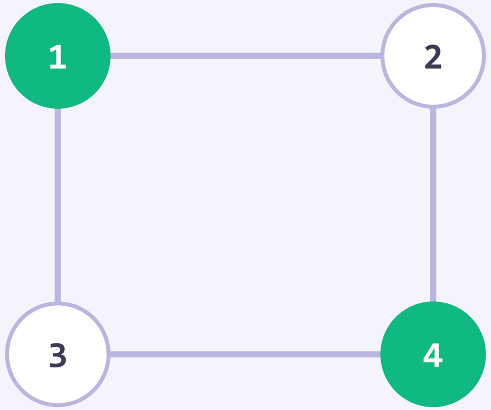
목표 모든 도로를 감시하면서 CCTV 수를 최소화한다

$G = (V, E)$에서 모든 간선 $(u, v) \in E$에 대해 $u \in S$ 또는 $v \in S$를 만족하는 가장 작은 $S \subseteq V$를 찾아라.

결정 문제 버전이 NP-완전 → 최적화 버전은 NP-난해다.

정점 커버 — 구체적인 예

정점 {1, 2, 3, 4}, 간선 (1,2), (1,3), (2,4), (3,4) 인 그래프를 보자.



초록 = 선택한 정점 {1, 4}

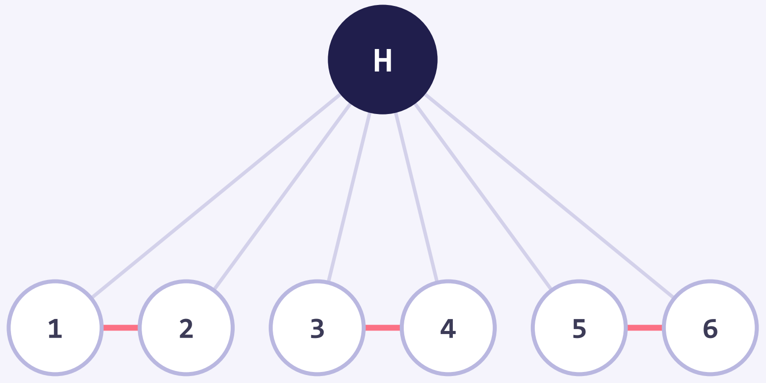
{1, 4}	모든 간선을 커버한다	크기 2
{2, 3}	역시 모든 간선을 커버한다	크기 2
{1,2,3,4}	커버하지만 지나치게 비효율적이다	크기 4
{1, 2}	간선 (3,4)를 커버하지 못한다	실패

최소 정점 커버 크기 = 2

{1, 4} 또는 {2, 3}

단순 그리디 방법의 실패

직관적인 방법 — "차수가 가장 높은 정점부터 선택한다"



허브-말단 구조 · 빨간 간선 = 말단끼리의 연결

최악의 경우 구조

가운데 허브 H는 차수가 k로 가장 높고, 말단 L1~Lk가 H에 연결된다. 말단끼리도 짝을 지어 간선으로 이어져 있다.

무슨 일이 벌어지나

단순 그리디는 차수가 가장 높은 H를 먼저 고른다. 그런데 말단끼리 이어진 간선은 여전히 커버되지 않아 정점을 더 골라야 한다.

이 단순 탐욕법은 최악의 경우 최적해보다 log n 배나 많은 정점을 고를 수 있다

즉 상수 배 근사 보장이 아니다. 직관적으로 그럴듯한 기준이 언제나 좋은 근사 비율로 이어지지는 않는다.

발상의 전환 — 매칭 기반 탐욕법

"정점" 기준으로 욕심부리지 말고
"간선" 기준으로 선택하자

간선을 고른다

아무 간선이나 하나 집는다. 어떤 기준도 필요 없다.

양 끝점을 둘 다

고른 간선의 두 정점을 모두 답에 넣는다. 하나만 넣지 않는다.

관련 간선을 지운다

그 두 정점에 붙은 간선을 전부 제거하고 다시 반복한다.

매우 무식해 보이지만, 강력한 수학적 보장을 제공한다

두 개를 한꺼번에 넣는 낭비처럼 보이는 이 행동이 오히려 2-근사를 증명해 준다.

매칭 기반 탐욕법 — 알고리즘

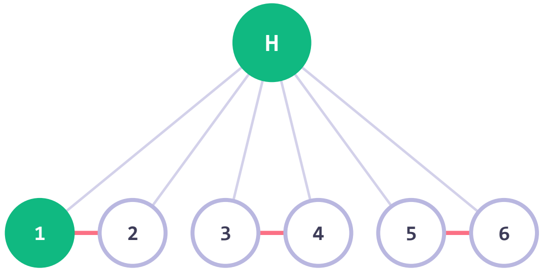
①	결과 정점 집합 C를 공집합으로 초기화한다.
② -a	그래프에 간선이 남아 있는 동안, 아무 간선 (u, v)를 하나 고른다. 정말 아무거나 좋다.
② -b	양 끝점 u와 v를 모두 C에 추가한다. 두 개 다 넣는다.
② -c	u 또는 v와 연결된 모든 간선을 그래프에서 제거한다.
③	더 이상 간선이 없으면 집합 C를 반환한다.

핵심은 간선의 양 끝점을 욕심쟁이처럼 둘 다 가져가는 전략이다.

매칭 기반 탐욕법 — 단계별 실행

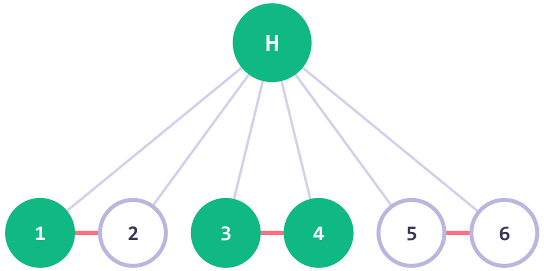
앞의 허브-말단 그래프에서 실제로 돌려 보자.

Step 1



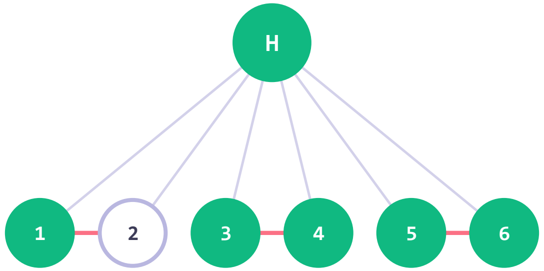
(H, L1) 선택
C = {H, L1}
H·L1 관련 간선 제거

Step 2



(L3, L4) 선택
C = {H, L1, L3, L4}
관련 간선 제거

Step 3



(L5, L6) 선택
C = {H, L1, L3, L4, L5, L6}
모든 간선 커버 완료!

$$|C| = 6 = 2 \times 3 = 2 \times \mathrm{OPT}$$

최적해는 L1·L3·L5 중 각 짝에서 하나씩 → OPT = 3

정확히 2배 — 2-근사 보장이 실제로 확인된다.

증명 — $|C| \le 2 \cdot OPT$

① 매칭 발견 알고리즘이 고른 간선들의 집합 M은 서로 꼭짓점을 공유하지 않는다. 곧 매칭(matching)이다.

② 하한값 M의 각 간선마다 최소 한 개의 정점이 OPT에 반드시 들어가야 한다. 따라서 $|M| \le OPT$ 이다.

③ 결론 우리 알고리즘은 간선마다 정점을 두 개씩 골랐다. 따라서 $|C| = 2 \cdot |M|$ 이다.

$$|C| = 2 \cdot |M| \le 2 \cdot OPT$$

∴ 매칭 기반 탐욕법은 2-근사 알고리즘이다

알고리즘 의사코드 — 매칭 기반 정점 커버

vertex_cover.pseudo

```
Matching-Greedy-Vertex-Cover(G = (V, E)):

  C <- 공집합                    // 선택된 정점 집합

  while E 가 비어있지 않은 동안 do
    choose any edge (u, v) in E     // 아무 간선이나 선택
    C <- C + {u, v}                 // 양 끝점을 모두 추가
    remove all edges incident to u or v

  return C
```

"아무 간선이나"가 맞다

어떤 간선을 골라도 2-근사가 성립한다. 고르는 규칙이 필요 없다는 점이 이 알고리즘의 매력이다.

시간 복잡도 O(V + E)

간선을 한 번씩만 훑고 지운다. 매우 효율적이다.

정점 커버의 현재 최선

더 나은 상수 비율은 아직 발견되지 않았다. 2가 사실상 한계라는 것이 유력한 추측이다.

"무식한 전략이 오히려 수학적 안전장치를 마련해 준다"

정점 하나만 골랐다면 매칭이라는 하한값과 연결할 방법이 없었을 것이다. 두 개를 다 가져가는 낭비가 곧 증명의 근거가 되었다.

좋은 근사 알고리즘은 대체로 "증명하기 좋은 구조"를 일부러 만들어 낸다.

10.5

집합 커버 문제

10.5

상수 배가 아니라 로그 배 —
그리고 그것이 사실상 최선이다.

집합 커버 문제 정의

유니버스 U 커버해야 할 전체 요소의 집합. 크기는 n개다.

부분집합 모임 S $S_1, S_2, \ldots, S_m$ — 각각 유니버스의 일부 요소로 이루어진다.

목표 선택한 집합들의 합집합이 U가 되도록, 최소 개수의 부분집합을 고른다.

예시

$U = \{1, 2, 3, 4, 5, 6\}$ · $S_1 = \{1,2,3\}$, $S_2 = \{2,4\}$, $S_3 = \{3,4,5\}$, $S_4 = \{5,6\}$

$S_1 \cup S_3 \cup S_4 = \{1,2,3,4,5,6\} = U$ → 세 개만 골라도 전체를 커버한다

정점 커버 문제를 일반화한 형태이기도 하다.

집합 커버 문제의 현실 응용

방송국 송신탑 설치

유니버스 = 전국의 도시

부분집합 = 각 송신탑이 커버하는 범위

목표 = 최소 송신탑으로 전국을 커버

학교 동아리 대표 선출

유니버스 = 전교생 100명

부분집합 = 각 동아리의 부원 명단

목표 = 최소 동아리 대표로 전교생을 커버

"최소한의 자원으로 전체를 덮는다"는 형태의 문제는 어디에나 있다

재고 배치, 백신 접종 거점 선정, 소프트웨어 테스트 케이스 선택까지 — 이름만 다를 뿐 같은 구조다.

정점 커버는 "각 간선을 덮는" 특수한 집합 커버로 볼 수 있다.

그리디 접근 알고리즘

매 순간 가장 "가성비"가 좋은 선택을 하는 탐욕 알고리즘이다.

① 아직 선택하지 않은 부분집합들을 살펴본다.

② 각 집합이 새로 커버할 수 있는 원소의 수를 계산한다.

③ 가장 많이 커버하는 집합을 선택한다.

④ 선택한 집합의 원소들을 커버 완료로 표시한다.

⑤ 전체가 커버될 때까지 ①~④를 반복한다.

가성비 = "아직 커버되지 않은 원소를 얼마나 많이 커버하는가"

그리디 알고리즘 실행 예시

$U = \{1,2,3,4,5\}$ · $S_1=\{1,2,3\}$, $S_2=\{1,4\}$, $S_3=\{2,5\}$, $S_4=\{3,4,5\}$

라운드 1

S_1(3개), S_4(3개), S_2(2개), S_3(2개)

→ S_1 선택 (동률이면 인덱스가 작은 것)

남은 원소: {4, 5}

라운드 2

S_4가 {4, 5}를 모두 커버한다 (2개)

→ S_4 선택

남은 원소: 없음 (완료!)

결과

선택: $\{S_1, S_4\}$

총 두 개의 부분집합으로 전체를 커버했다.

매 라운드마다 "지금 가장 많이 덮는 집합"을 다시 계산한다

이미 커버된 원소는 이득으로 세지 않는다는 점이 핵심이다. 그래서 라운드가 갈수록 각 집합의 가성비가 떨어진다.

그리디 집합 커버 — 의사코드

greedy_set_cover.pseudo

```
Greedy-Set-Cover(U, S = {S1, ..., Sm}):

  Covered  <- 공집합
  Selected <- 공집합

  while Covered != U do
    bestSet <- NULL
    bestGain <- -1

    for each X in S do
      gain <- |X - Covered|          // 새로 커버하는 원소 수
      if gain > bestGain then
        bestGain <- gain; bestSet <- X

    Selected <- Selected + {bestSet}
    Covered  <- Covered + bestSet

  return Selected
```

gain 계산이 핵심

|X − Covered| — 이미 덮인 원소는 세지 않는다. 이 한 줄이 "가성비"의 정의다.

이중 반복문

라운드마다 모든 집합을 훑는다. 시간 복잡도는 대략 $O(n \cdot m)$이다.

동률 처리

gain이 같으면 인덱스가 작은 것을 고른다. 어느 쪽을 골라도 보장은 그대로다.

정점 커버와 달리, 이 알고리즘의 근사 비율은 상수가 아니다.

집합 커버 — 근사 비율 분석

$(\ln n + 1)$ - 근사 알고리즘

근사 비율이 상수가 아니라 입력 크기에 따라 커진다

n	ln n + 1	의미
10	≈ 3.3	최적해의 약 3.3배
100	≈ 5.6	최적해의 약 5.6배
1,000	≈ 7.9	최적해의 약 7.9배
1,000,000	≈ 14.8	최적해의 약 14.8배

나쁜 소식처럼 보이지만 실은 그렇지 않다

원소가 백만 개여도 최적해의 15배 안쪽이다. 로그는 아주 천천히 자란다. 지수 시간을 기다리는 것과는 비교할 수 없다.

왜 ln n 인가 — 비용 분담 아이디어

① 비용 할당 집합을 하나 선택할 때 비용 1원이 든다고 하자. 이 1원을 새로 커버된 N개의 원소가 1/N씩 나눠 낸다.

② 효율성 감소 초반에는 큰 집합을 고르므로 원소당 비용이 1/100, 1/50처럼 작다. 후반에는 작은 집합만 남아 1/1까지 커진다.

③ 조화 급수 원소별 비용을 모두 더하면 $1 + 1/2 + 1/3 + \dots + 1/n = H_n$ 이 되고, 이 값은 대략 ln n 이다.

④ 최적해 비교 따라서 그리디가 고른 집합의 수는 $H_n \times \mathrm{OPT}$ 이하가 된다. 곧 약 ln n 배다.

$$H_n = 1 + 1/2 + 1/3 + \dots + 1/n \approx \ln n$$

놀라운 사실 — 이 탐욕 알고리즘이 현실적 최선이다

P ≠ NP를 가정하면, 집합 커버 문제에 대해 이보다 본질적으로 더 나은 (상수 배) 근사 알고리즘을 만드는 것이 불가능하다는 사실이 증명되어 있다.

1

처음엔 효율적이다

큰 집합부터 고르므로 초반 몇 번의 선택으로 대부분이 덮인다.

2

갈수록 비효율적이다

남은 원소가 흩어져 있어, 작은 집합을 여러 개 골라야 한다.

3

오차가 누적된다

그 누적의 한계가 수학적으로 정확히 ln n이다. 우연이 아니라 필연이다.

"더 좋은 알고리즘이 없다"는 것도 하나의 결과다

이 사실을 알면 개선 시도에 시간을 쓰는 대신, 문제 자체의 조건을 바꿀 수 있는지를 먼저 검토하게 된다.

10.6

근사조차 어려운 문제들

10.6

모든 NP-난해 문제가
효율적으로 근사 가능한 것은 아니다.

일반 TSP의 절망 — 근사 불가능

P ≠ NP를 가정하면, 일반 TSP에 대해서는 어떤 상수 $k \geq 1$ 에 대해서도 다항 시간 k-근사 알고리즘이 존재하지 않는다.

증명의 핵심 아이디어는 해밀턴 순환 문제(NP-완전)를 TSP로 변환하는 것이다.

변환 간선이 있으면 비용 1, 간선이 없으면 비용 M(매우 큰 값)으로 설정한다.

순환 있음 OPT = n 이므로, k-근사 알고리즘의 결과는 $n \cdot k$ 이하가 된다.

순환 없음 반드시 비용 M짜리 간선을 써야 하므로 OPT ≥ M, 결과도 M 이상이 된다.

$M \gg n \cdot k$ 로 잡으면 두 구간이 완전히 분리된다 → 근사 결과만 보고 Yes/No를 판별할 수 있다.

Gap을 이용한 근사 불가능성 증명

[n, n·k] ← GAP → [M, k·M]

해밀턴 순환 있음 | 누구의 영역도 아님 | 해밀턴 순환 없음

결론 ① 근사 알고리즘의 결과만 보고도 해밀턴 순환의 존재 여부(NP-완전 문제)를 판별할 수 있게 된다.

결론 ② P ≠ NP 라면 이것은 모순이다. 따라서 그런 근사 알고리즘은 존재할 수 없다.

일반 TSP는 "적당히 타협해서 풀기조차 불가능한 문제"다.

근사의 한계 (inapproximability results)

모든 NP-난해 문제가 효율적으로 근사 가능한 것은 아니다. 문제마다 도달 가능한 한계가 다르다.

문제	근사 비율	상태
Metric TSP	α = 2 (또는 1.5)	상수 배 근사 가능
정점 커버	α = 2	상수 배 근사 가능
집합 커버	α = ln n	로그 배 근사 가능 — 이것이 최선
일반 TSP	불가능	어떤 상수로도 근사 불가
최대 클리크	n^(1-ε) 불가	다항 시간 근사가 극도로 제한된다

문제 자체의 본질적 특성이 근사의 한계를 결정한다

알고리즘 설계자의 실력 문제가 아니다. 어떤 문제는 태생적으로 타협의 여지가 있고, 어떤 문제는 없다.

Metric TSP처럼 조건을 조금 좁히면 불가능이 가능으로 바뀌기도 한다 — 이것이 실무의 돌파구다.

SUMMARY

10장 정리

NP-난해 앞의 현실적 선택

지수 시간은 기다릴 수 없다. 최적해를 포기하되, 얼마나 손해인지는 숫자로 못 박는다.

휴리스틱 vs 근사 알고리즘

둘 다 다항 시간이지만, 근사 알고리즘만 "최악에도 α배 이내"라는 증명서를 준다.

Metric TSP — 2-근사

MST를 하한값으로 삼고, 삼각 부등식으로 지름길을 만든다. 투어 ≤ 2 × OPT.

정점 커버 — 2-근사

아무 간선이나 골라 양 끝점을 다 넣는다. 매칭이 하한값이 되어 $|C| = 2|M| \le 2\cdot\mathrm{OPT}$.

집합 커버 — ln n 근사

가성비가 가장 좋은 집합부터 고른다. 조화 급수 때문에 ln n 배가 되고, 이것이 최선이다.

근사조차 불가능한 문제

일반 TSP와 최대 클리크는 상수 배 근사도 불가능하다. 한계는 문제의 본질이 정한다.

CHAPTER 10

Q & A

완벽한 답이 아니어도 괜찮습니다.
얼마나 어긋났는지 알고 있다면 그것으로 충분합니다.