그리디 알고리즘

경북대학교 전현승
dogdriip@gmail.com

Table of contents

- 0x00

그리디 알고리즘?

그리디 알고리즘

- 탐욕법, Greedy Algorithm, Greedy Method
- **Optimization problem을 푸는 방법** 중 하나
  - **최적화 문제** : 최적의 답을 구하는 문제. 특정 상황에서의 최솟값, 최댓값, 가장 빠르게 가는 경로, …

- 그리디 : “각 단계마다 **최선의 선택**을 하면 전체 최적해가 나온다!”

그리디 알고리즘

- “각 단계마다 **최선의 선택을 하면** 전체 최적해가 나온다!”
- “그냥 **좋아 보이는 것만 무작정 고르면** 최적해가 나온다!”
- “현재 선택이 다음 선택에 영향을 미치지 않는다. 각 선택은 독립적이다!”
- “모든 선택을 고려하지 않고, **그때그때 가장 좋아 보이는 선택을 하면** 답이다!”

- 굉장히 간단하고 좋아 보이는 방법이다

그리디 알고리즘의 응용

- 그래프에서 Minimum Spanning Tree를 구하는 **Kruskal 알고리즘**
- 음수가 아닌 가중치 그래프에서 최단 거리를 구하는 **다익스트라 알고리즘**
- Knapsack 문제 중 한 종류인 **Fractional Knapsack 문제**의 풀이

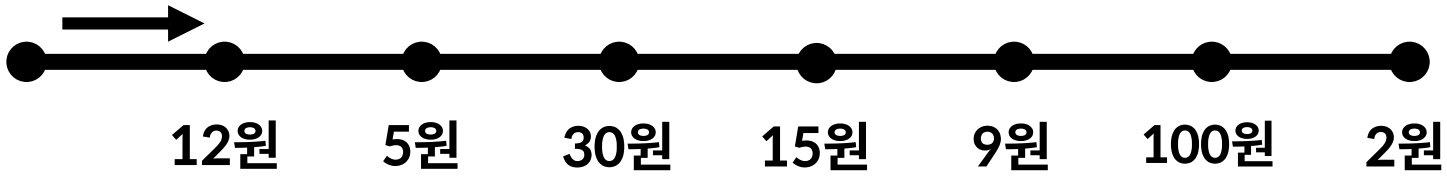
- 이 알고리즘들과 풀이의 기본 아이디어는 그리디 알고리즘이다.
- 이렇게 그리디는 문제의 풀이 그 자체가 되기도 하지만,
  다른 알고리즘의 베이스가 되기도 한다.

그리디 알고리즘은 항상 성립하는가?

- 그러나, 모든 문제에서 그리디 알고리즘을 사용하면 당연히 안 된다.
- 즉, **단계별로 최선의 선택을 했는데 전체 문제의 최적해는 아닌 경우**도 있다.

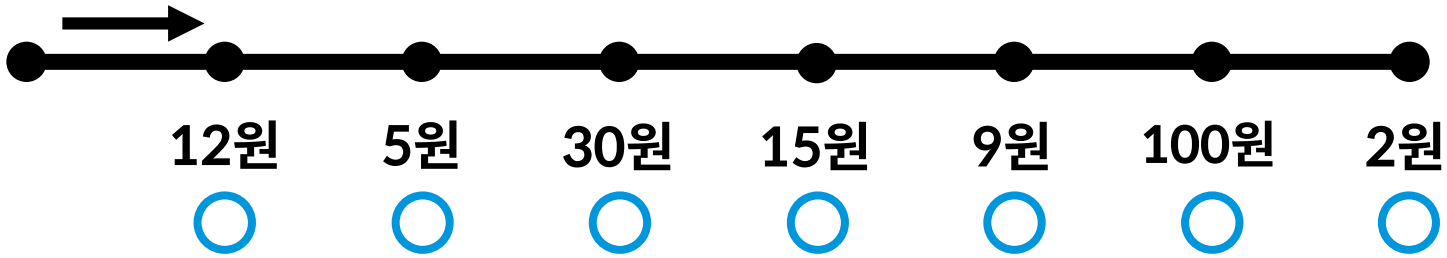
그리디 알고리즘은 항상 성립하는가? – 예시 1

- 일직선 길 위에 돈이 떨어져 있다. 길을 가면서 돈을 가장 많이 줍는 방법은?



그리디 알고리즘은 항상 성립하는가? – 예시 1

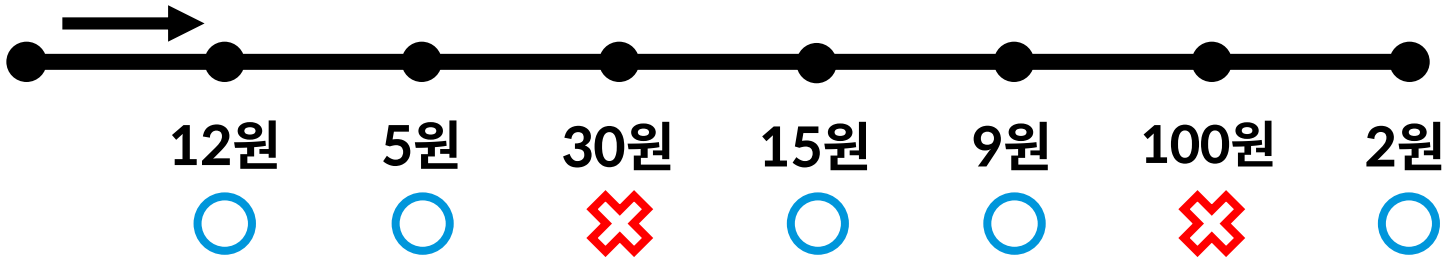
- 일직선 길 위에 돈이 떨어져 있다. 길을 가면서 돈을 가장 많이 줍는 방법은?



- 아무런 조건이 없다면?
- 그냥 가면서 있는 대로 주우면 된다. 다 주우면 된다.
- 각 단계마다 최선의 선택을 하면 전체 문제의 최적해가 된다 → **그리디 O**

그리디 알고리즘은 항상 성립하는가? – 예시 1

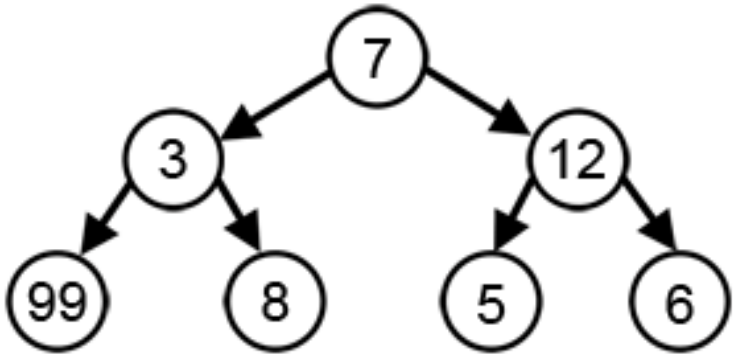
- 만약 **“연속해서 3번 이상은 주울 수 없다”**는 조건이 붙으면?



- 무턱대고 계속 주우면 안 된다. 뒤에 나오는 큰 걸 못 주울 수도 있다.
- 항상 최적해가 나오지 않는다. 현재 선택이 다음 선택에 영향을 미친다. → **그리디 X**

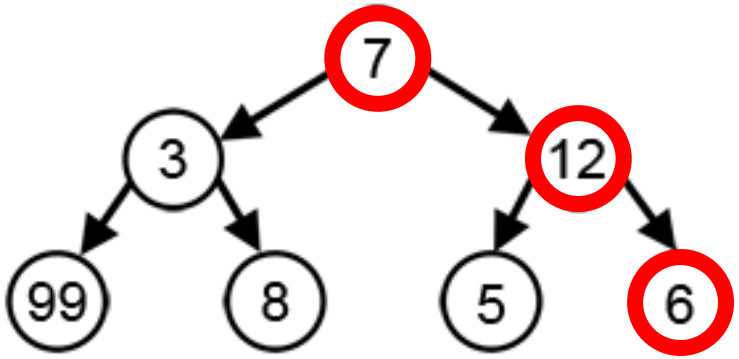
그리디 알고리즘은 항상 성립하는가? – 예시 2

- 루트에서 시작해서 내려갈 때, 합이 가장 큰 경로는?



그리디 알고리즘은 항상 성립하는가? – 예시 2

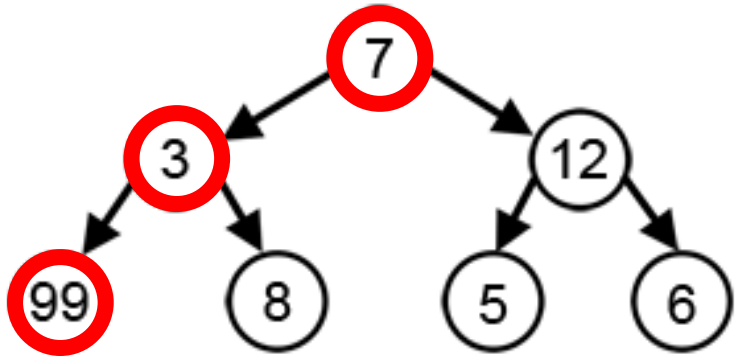
- 루트에서 시작해서 내려갈 때, 합이 가장 큰 경로는?



- 그때그때 양쪽 자식 중 더 큰 자식 노드 쪽으로 내려가면 되지 않을까?
  - 이 방법대로 하면 7 → 12 → 6이 나온다

그리디 알고리즘은 항상 성립하는가? – 예시 2

- 루트에서 시작해서 내려갈 때, 합이 가장 큰 경로는?



- 하지만 이는 최적해가 아니다!
  - 7 → 3 → 99 쪽으로 가는 게 경로의 합이 더 크다.
- 각 단계마다 최선의 선택을 해도, 그게 전체 문제의 최적해가 아닐 수 있다. → **그리디 X**

그리디 알고리즘이 성립하기 위한 조건

1. Greedy choice property

   현재 선택이 미래 선택에 영향을 미치면 안 된다.

   지금 무작정 많이 선택한다고 해서, 미래의 선택에 지장이 있으면 안 된다는 것

2. Optimal substructure

   최적 부분 구조

   부분문제의 최적해가 → 전체 문제의 최적해여야 한다.

   현재 최선의 선택이 → 전체 문제를 고려해도 최선의 선택이어야 한다.

- 0x01

예시 – 동전 교환 문제

쉬운 경우부터 풀어봅시다

#5585 거스름돈

- (1000엔 이하로) 물건을 사고, 1000엔 한 장을 낸다.
- 잔돈으로 **500엔, 100엔, 50엔, 10엔, 5엔, 1엔**이 있고, 각 동전은 충분히 있다.
- 동전 개수를 가장 적게 거슬러 주는 방법은?

쉬운 경우부터 풀어봅시다

#5585 거스름돈

- 예를 들어, 물건 가격이 380엔일 경우, 잔돈은 620엔
- 500엔 1개, 100엔 1개, 10엔 2개를 거슬러주는 경우가 최적이며, 이 때의 동전 개수는 4개

쉬운 경우부터 풀어봅시다

#5585 거스름돈

- 가장 먼저 드는 생각은 : **그냥 제일 비싼 동전부터 거슬러 주면 안 될까?**
- 757엔을 거슬러 주려면
  - 500엔 1개, 100엔 2개, 50엔 1개, 5엔 1개, 1엔 2개 == 총 7개
  - 이 때가 최적이다
- 210엔을 거슬러 주려면
  - 100엔 2개, 10엔 1개 == 총 3개
  - 이 때가 최적이다

- 이렇게 풀면 된다!

쉬운 경우부터 풀어봅시다

#5585 거스름돈

- 너무 쉽게 풀렸다

```
int n; cin >> n;
n = 1000 - n;                    // 거슬러줄 돈

int coin[6] = {500, 100, 50, 10, 5, 1};
int total = 0;                   // 사용한 동전 개수
for (int i = 0; i < 6; i++) {    // 큰 액수부터 거슬러주자
    total += n / coin[i];
    n %= coin[i];
}

cout << total << '\n';
```

왜 성립하는가?

#5585 거스름돈

- 잔돈 종류를 다시 보면 : 500엔, 100엔, 50엔, 10엔, 5엔, 1엔
- **모든 동전에 대해 (액수가) 큰 동전은 작은 동전의 배수이다.**
- 이 성질 때문에 앞에서 당연하게 생각했던 방법이 가능한 것

- 큰 동전을 작은 동전 여러 개로 교체해봤자 총 동전 개수만 늘어날 뿐이다.
- 작은 동전들 여러 개를 큰 동전으로 대신할 수 있다면,
  큰 동전을 사용하는 게 무조건 이득이다.

성립하지 않는 경우도 있을까?

- 예를 들어, **60엔, 50엔, 10엔** 동전이 있을 경우에도 그리디가 성립할까?
- 일단 동전들끼리 배수 관계는 깨지게 된다

- 동전을 최소한 적게 써서 **220엔**을 만드려 한다. 앞에서 했던 방법대로 해 보면?
  - 60엔 3개를 먼저 사용하고, 남은 40엔은 10엔짜리 4개로밖에 나타낼 수 없다.
  - 그리디 알고리즘대로라면, **총 7개**
- **하지만 더 좋은 방법이 있다**
  - 50엔 4개와 10엔 2개를 사용하면, **총 6개**로 220엔을 만들 수 있다.

- 그리디 알고리즘이 최적해를 보장하지 못한 사례

- 0x02

예시 – Interval Scheduling

Interval Scheduling

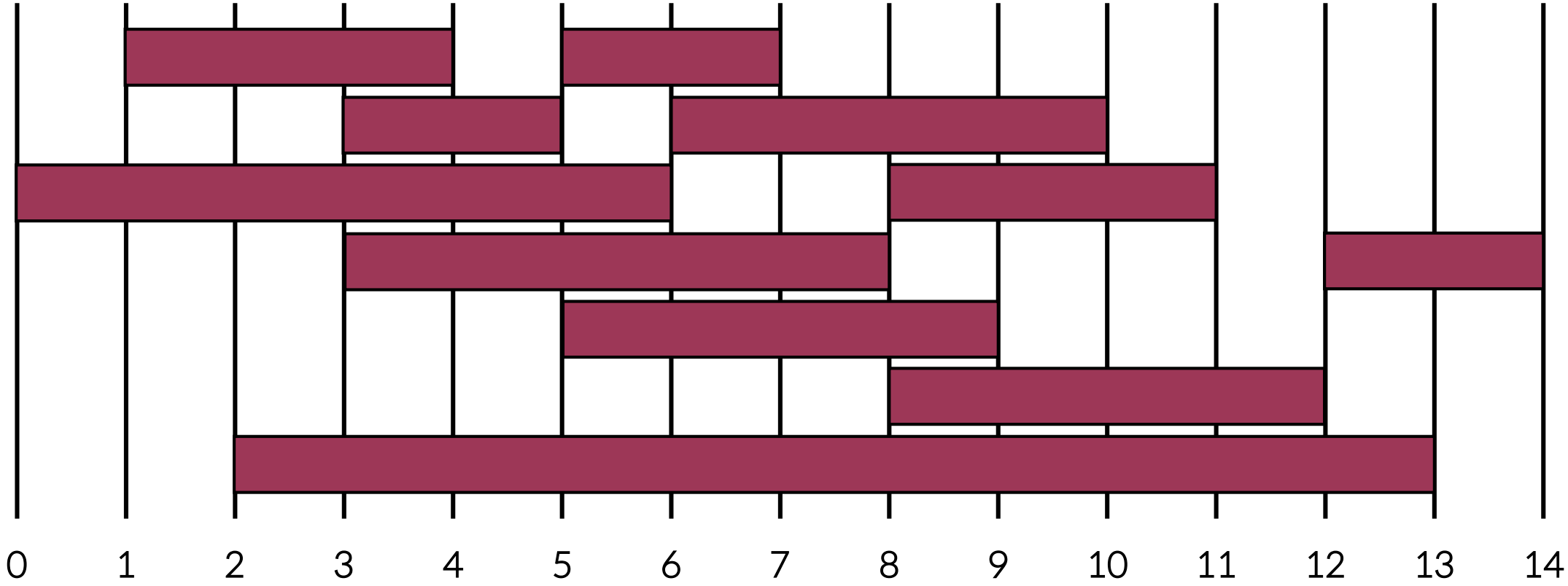
#1931 회의실배정

- 한 개의 회의실과 N개의 회의에 대하여 회의실 사용표를 만들려고 한다.
- 각 회의에 대해 시작시간과 끝나는 시간이 주어져 있다.
- **각 회의가 겹치지 않게 하면서 회의실을 사용할 수 있는 회의의 최대 개수**를 찾아보자.

예제로 알아보자

#1931 회의실배정

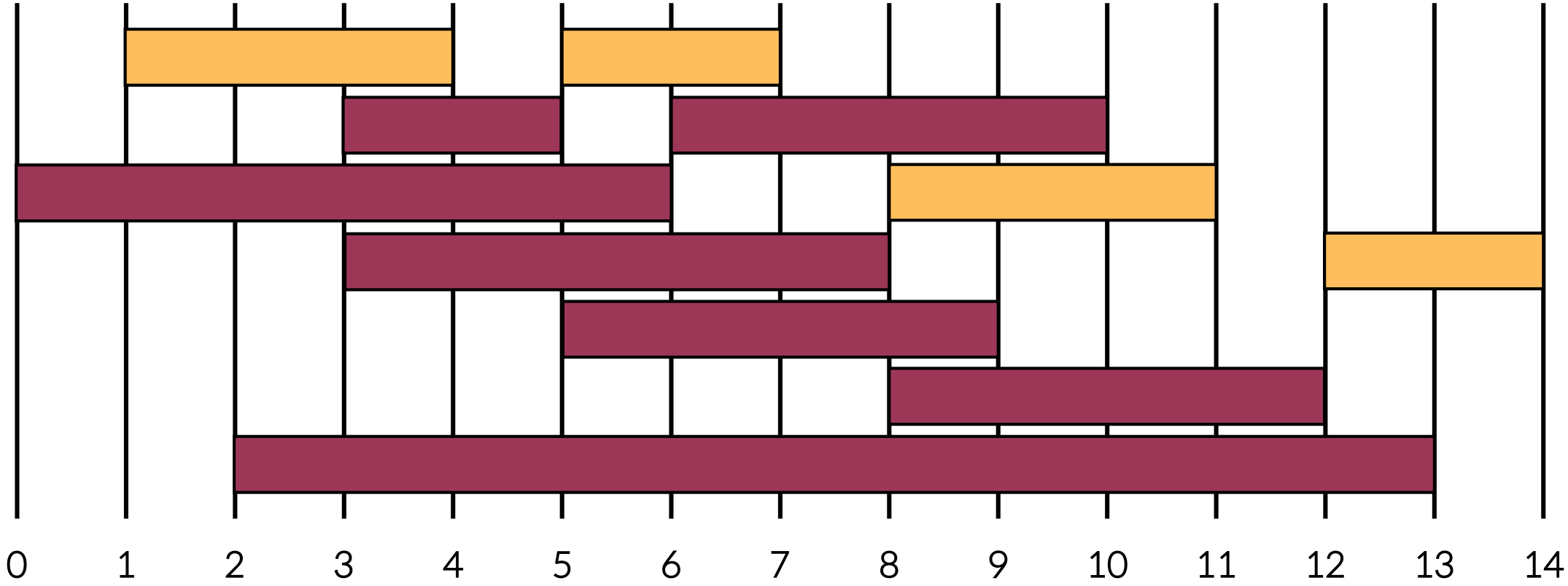
- 예제를 그림으로 나타내면 다음과 같다



예제로 알아보자

#1931 회의실배정

- 이렇게 선택하면 최대 4개의 회의를 진행할 수 있고, 이 때가 회의 개수의 최대이다.



어떻게 회의를 선택해야?

#1931 회의실배정

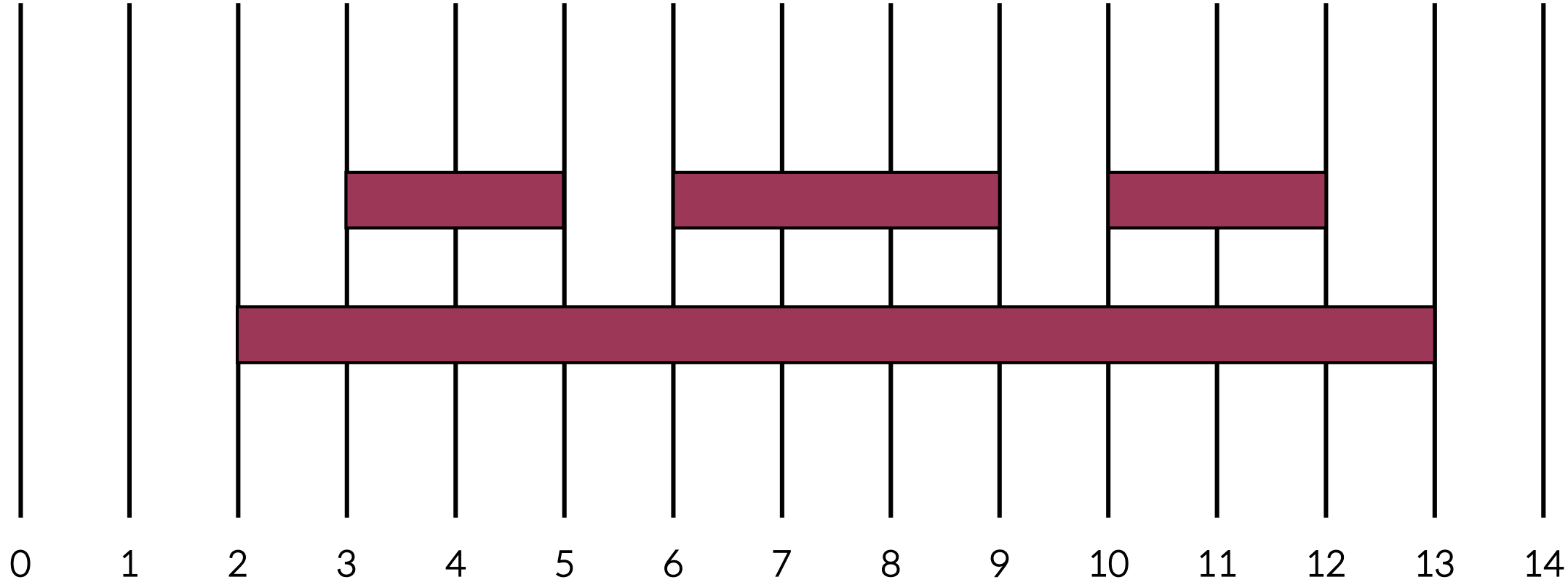
- 일단 N=100,000이므로 모든 경우의 수를 다 해보는 것은 안 된다.
  - $O(2^N)$

- 어떻게 회의를 선택해야 항상 최대 개수의 회의를 배정할 수 있을까?

어떻게 회의를 선택해야?

#1931 회의실배정

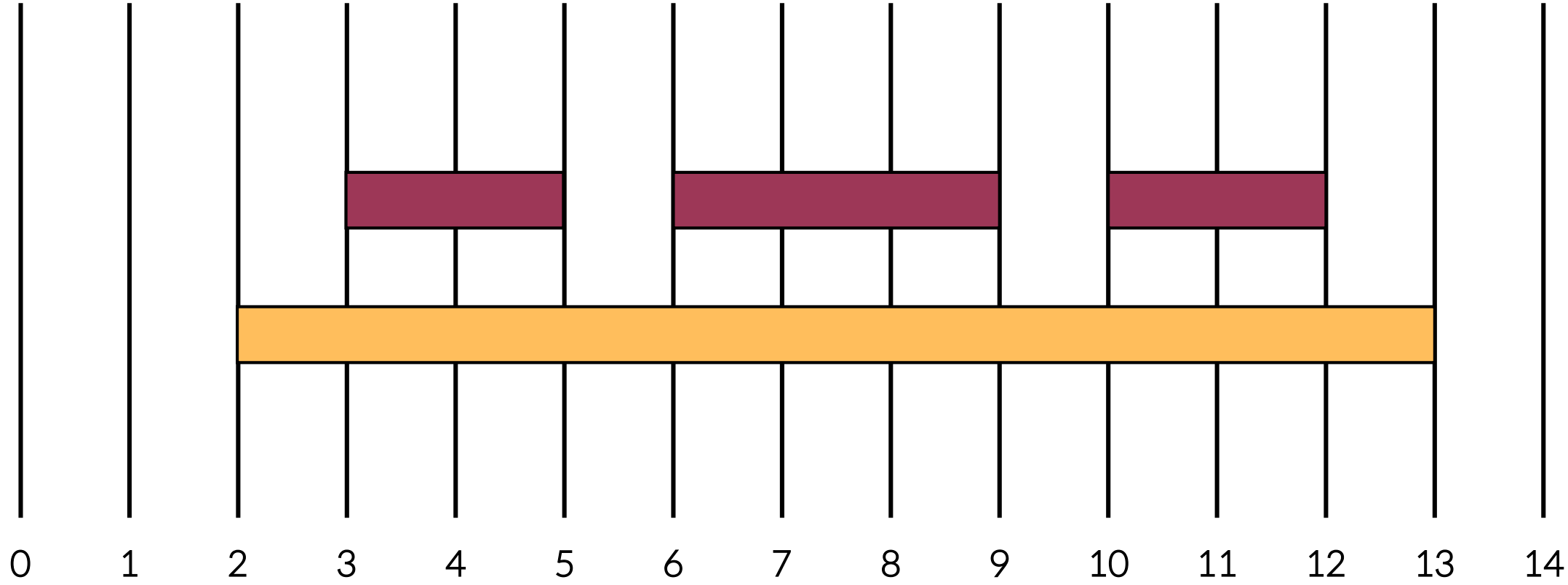
- 일찍 시작하는 회의부터 배정하면?



어떻게 회의를 선택해야?

#1931 회의실배정

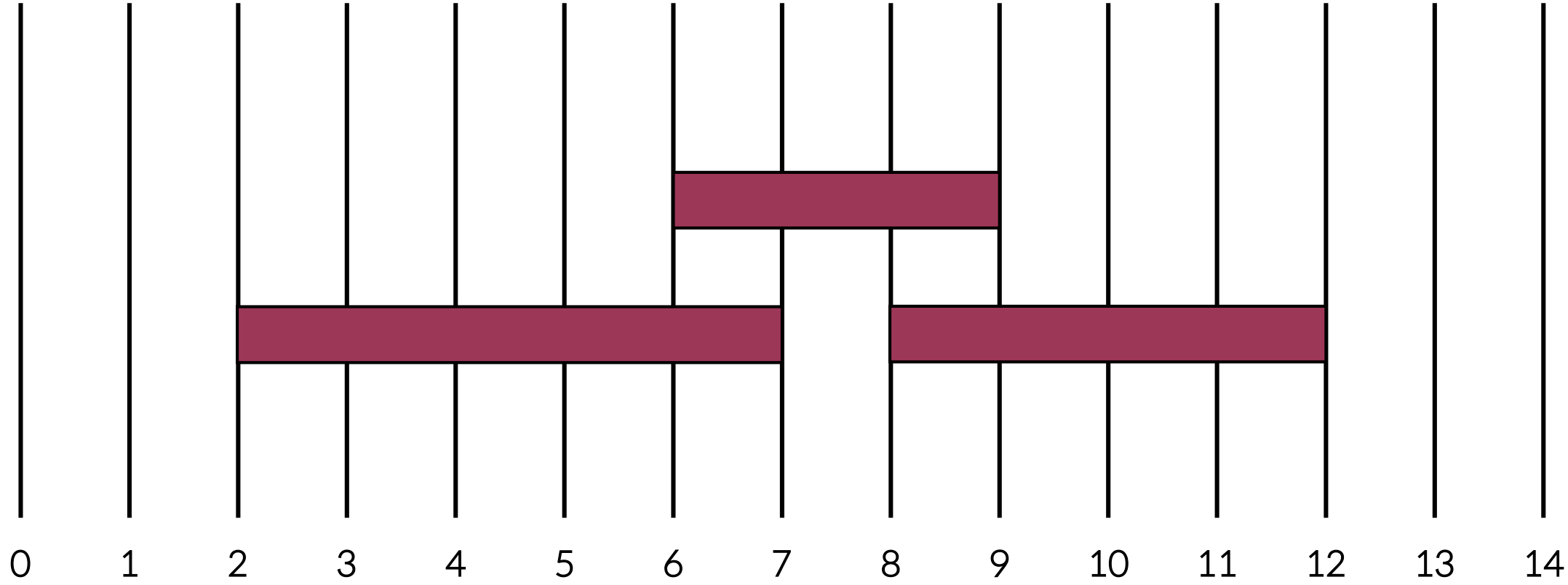
- 회의 중간에 일찍 끝나는 회의가 있는 경우 이를 선택할 수 없게 된다.



어떻게 회의를 선택해야?

#1931 회의실배정

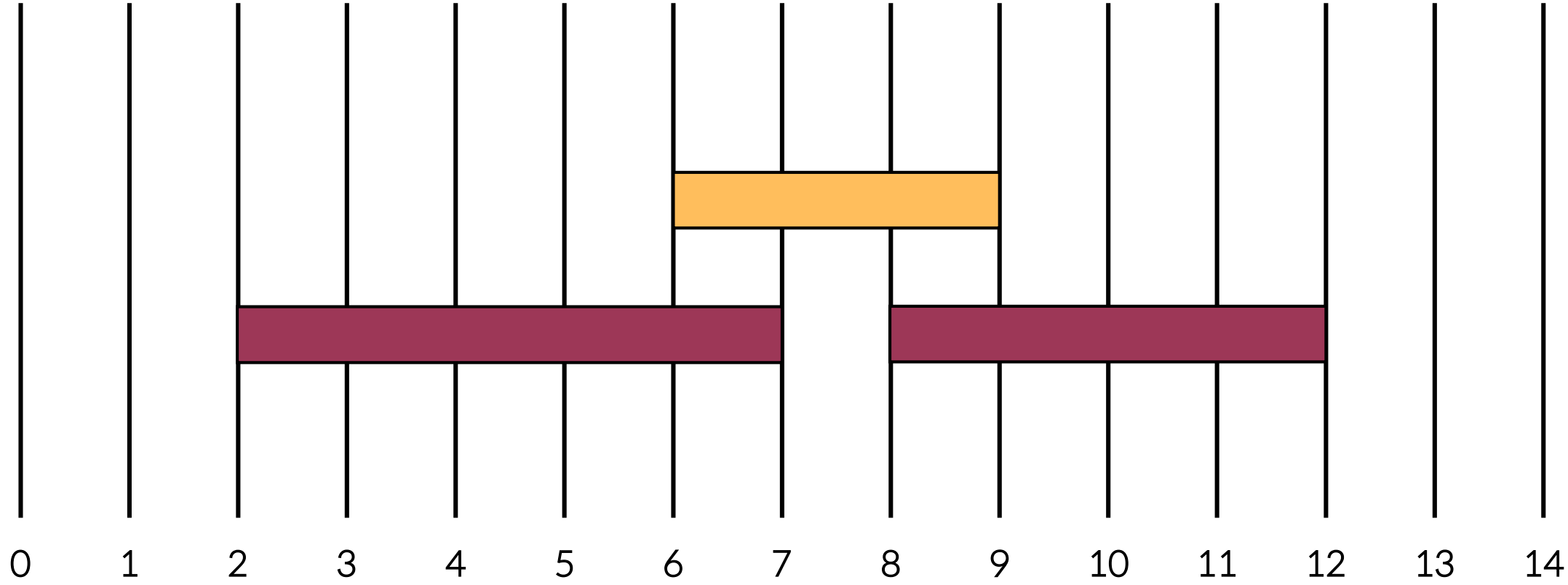
- 짧은 회의부터 배정하면?



어떻게 회의를 선택해야?

#1931 회의실배정

- 짧은 회의부터 선택한다 해도 항상 최대 회의를 배정할 수 있는 것도 아니다.



정답은…

#1931 회의실배정

- 정답은 **‘일찍 끝나는 회의부터 배정하기’**
- 왜?

- 모든 회의들 중 가장 일찍 끝나는 회의를 X라 하자.
- 최적해 (회의를 가장 많이 배정했을 때의 회의의 목록) 중
  X를 포함하지 않는 최적해가 있다 가정하자.
- **그 최적해의 맨 앞 회의 대신 X를 사용**해도
  뒤 회의와 겹치지 않으며 조건을 만족하고, 총 회의의 수는 동일하다.
- 따라서, 맨 앞 회의를
  현재 사용할 수 있는 회의들 중 가장 일찍 끝나는 회의로 바꿔치기할 수 있고,
  이렇게 해서 최적해를 만들 수 있으므로 이 방법은 성립한다.

Interval Scheduling

#1931 회의실배정

- 회의들을 종료시간 기준으로 오름차순 정렬,
  종료시간이 같은 경우에는 시작시간이 빠른 순으로 정렬
- 앞에서부터 보면서 회의가 겹치지 않게 최대한 많이 배정

Interval Scheduling

#1931 회의실배정

- 입력받고 정렬까지

```
using pii = pair<int, int>;
...

pii conf[100001];
int n; cin >> n;
for (int i = 0; i < n; i++) {
    cin >> conf[i].first >> conf[i].second;  // first: 시작시간, second: 종료시간
}

sort(conf, conf + n, [](const pii& a, const pii& b) {
    if (a.second == b.second) {  // 종료시간이 같다면, 시작시간이 빠른 게 먼저
        return a.first < b.first;
    }
    return a.second < b.second;  // 그렇지 않다면, 종료시간이 빠른 게 먼저
});
```

Interval Scheduling

#1931 회의실배정

- 맨 앞 회의부터 보면서 넣을 수 있는 회의는 배정

```
int cnt = 0;  // 최대 회의 배정 개수
int last = 0;  // 마지막 회의가 끝난 시간
for (int i = 0; i < n; i++) {
    if (last <= conf[i].first) {
        // 마지막 회의가 끝난 시간이 다음 회의 시작시간보다 작거나 같다면
        // 회의를 배정할 수 있는 것
        cnt++;
        last = conf[i].second;
    }
}

cout << cnt << '\n';
```

Practice

#4796 캠핑

#1789 수들의 합

#1049 기타줄

#13305 주유소

#1449 수리공 항승

#1541 잃어버린 괄호

#1080 행렬

#2812 크게 만들기

#11000 강의실 배정

#1744 수 묶기

#2437 저울

#1700 멀티탭 스케쥴링

Sources

- https://en.wikipedia.org/wiki/Greedy_algorithm

See also

- “Proof Techniques: Greedy Stays Ahead” – *Tom Wexler, Alexa Sharp*
  http://www.cs.cornell.edu/courses/cs482/2003su/handouts/greedy_ahead.pdf