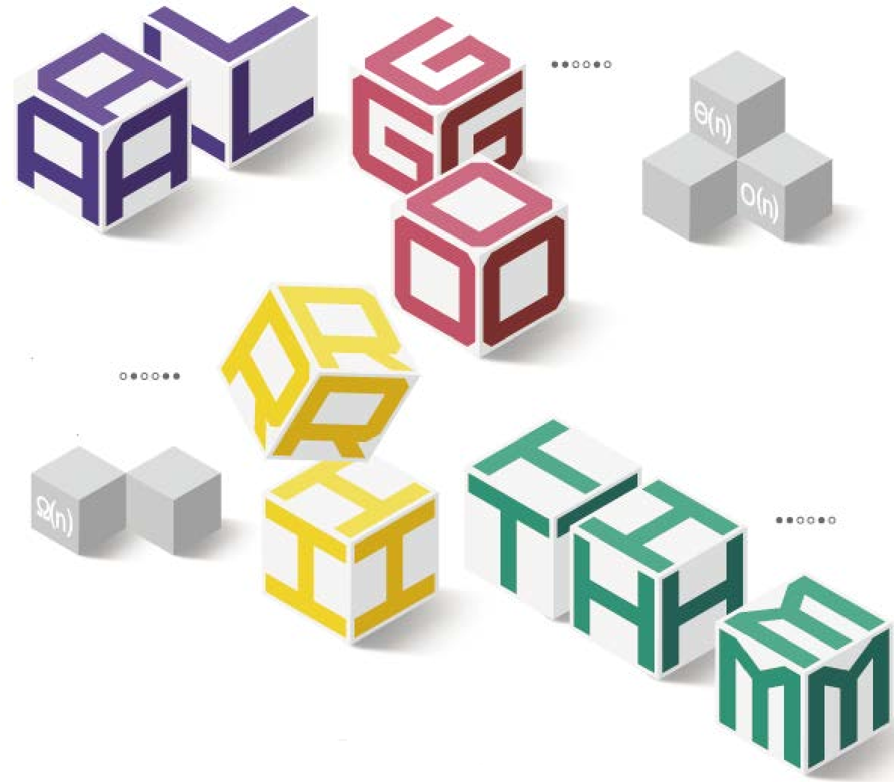


관계 중심의 사고법

쉽게 배우는 알고리즘

11장. 그리디 알고리즘

11장. 그리디Greedy 알고리즘

학습목표

- 그리디 알고리즘의 특징을 파악한다.
- 그리디 알고리즘으로 최적해가 보장되는 예와 그렇지 않은 예를 관찰한다.
- 매트로이드의 정의를 익힌다.
- 매트로이드가 만드는 문제 공간의 특성을 배운다.

그리디 알고리즘

- 눈앞의 이익만 취하고 보는 알고리즘
- 현재 시점에 가장 이득이 되어 보이는 해를 선택하는 행위를 반복한다
- 대부분 최적해와의 거리가 멀다
- 드물게 최적해가 보장되는 경우도 있다

do {

우선 가장 **좋아 보이는** 선택을 한다

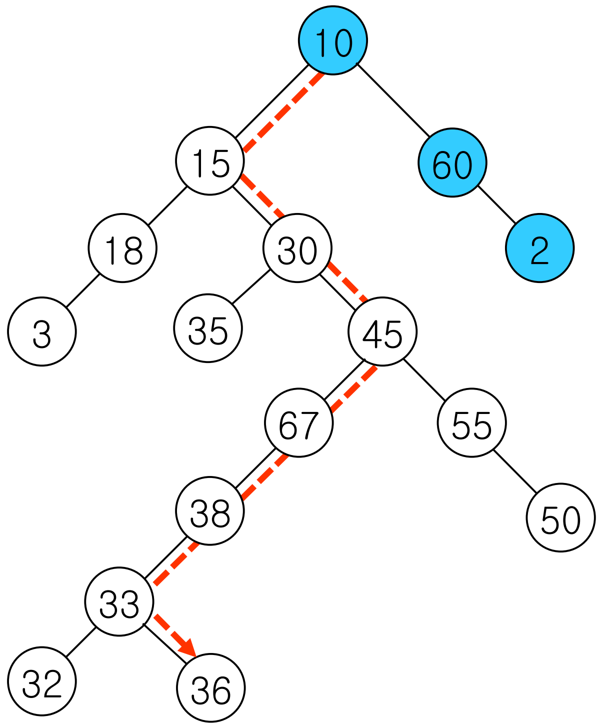
} **until** (해 구성 완료)

그리디 알고리즘의 전형적 구조

```
Greedy(C)
// C: 원소들의 총집합
{
        S ← Ø;
        while (C≠Ø and S는 아직 온전한 해가 아님) {
                x ← C에서 가장 좋아 보이는 원소;
                집합 C에서 x 제거; // C ← C-{x}
                if (S에 x를 더해도 됨) then S ← S ∪{x};
        }
        if (S가 온전한 해임) then return S;
                        else return "no solution!";
}
```

그리디 알고리즘으로 최적해가 보장되지 않는 예

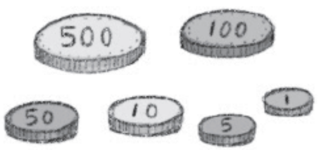
이진 트리의 최적합 경로 찾기



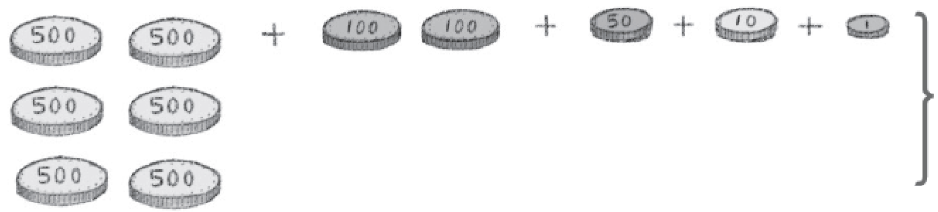
그리디 알고리즘으로 최적해가 보장되지 않는 예 2

동전 바꾸기

동전의 액면



3,256원 만들기

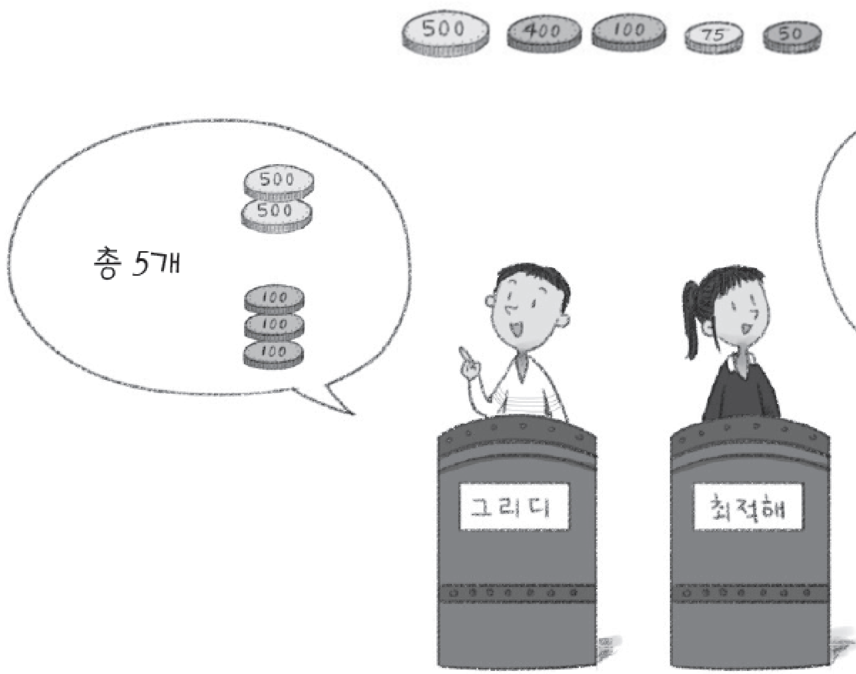


이렇게 동전의 액면이 모두 바로 아래 액면의 배수가 되면
그리디 알고리즘으로 최적해가 보장된다

액면이 바로 아래 액면의 배수가 되지 않으면 그리디 알고리즘으로 최적해가 보장되지 않는다.

예: 다음 페이지

액면이 바로 아래 액면의 배수가 되지 않으면 그리디
알고리즘으로 최적해가 보장되지 않는다



그리디 알고리즘이 최적해를 보장하는 예

최소 신장 트리 찾기를 위한
프림 알고리즘과 크루스칼 알고리즘

```
Prim (G, r)
{
    S ← Φ ;
    정점 r을 방문되었다고 표시하고, 집합 S에 포함시킨다;
    while (S≠V) {
        S에서 V-S를 연결하는 간선들 중 최소길이의 간선 (x, y) 를 찾는다; ▷ (x∈S, y∈V-S)
        정점 y를 방문되었다고 표시하고, 집합 S에 포함시킨다;
    }
}
```

greedy한 부분

그리디 알고리즘이 최적해를 보장하는 예 2

회의실 배정 문제

- 회의실 1개
- 여러 부서에서 회의실 사용 요청
  - n개의 회의 $\{a_1, a_2, \ldots, a_n\}$
  - 회의 시작 시간 s_i와 종료 시간 f_i를 명시해서 신청
- Greedy한 아이디어들
  - 소요 시간이 가장 짧은 회의순 배정
  - 시작 시간이 가장 이른 회의순 배정
  - **종료 시간이 가장 이른 회의순 배정**

이것만이 최적해를 보장한다

그리디 알고리즘이 최적해를 보장하는 예 2

회의실 배정 문제

- a_k를 종료시간이 가장 이른 회의라고 하자. a_k는 최적해에 포함된다.
- 증명: A를 최적해라고 하고, a_j를 A에서 종료시간이 가장 이른 회의라고 하자. $a_j = a_k$이면, 증명 끝. $a_j \neq a_k$이면, $A - \{a_j\} \cup \{a_k\}$가 최적해가 된다.

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- 해: $\{a_3, a_9, a_{11}\}$
- 최적해: $\{a_1, a_4, a_8, a_{11}\}, \{a_2, a_4, a_9, a_{11}\}$

그리디 알고리즘이 최적해를 보장하는 예 2

회의실 배정 문제

```
Greedy(s,f)
{
    회의는 종료시간에 의해 정렬되어 있다고 가정
    A ← {a_1}
    j ← 1
    for i ← 2 to n
        if s_i ≥ f_j  then {
            A ← A ∪ {a_i};
            j ← i;
        }
    return A
}
```

Thank you