Ssafy_9W-3_Subset-Comb-Greedy

Ssafy_9W-3_Subset-Comb-Greedy

1. 부분집합

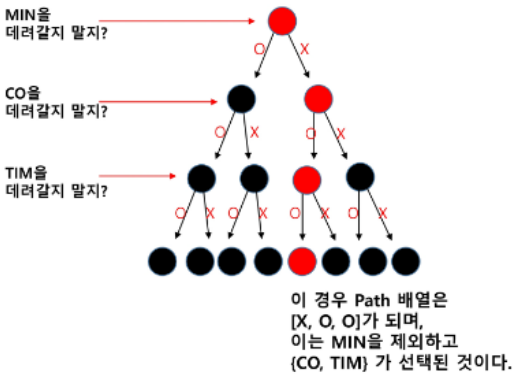
{A, B, C}
{}
{A}
{B}
{A, B}
{C}
{A, C}
{B, C}
{A, B, C}

- **집합에 포함된 원소들을 선택**하는 것
- 표는 {A, B, C}로 만들 수 있는 부분집합의 예
- 부분집합은 공집합도 포함.
- 구현 방법은 완탐과 Binary Counting 방법이 있다.
  - 완탐의 경우는 학습용으로 추천
  - Binary Counting은 부분 집합이 필요할 때 사용하는 것을 추천

1.1. 구현

1.1.1. 완전 탐색

- 완탐으로 부분집합 구하기
- {MIN, CO, TIM}에서 부분집합 구하는 문제



- Branch: 2, Level: 3

```
arr = ['O', 'X']
path = []
name = ['MIN', 'CO', 'TIM']

def print_name():
    print('{', end='')
    for i in range(3):
        if path[i] == '0':
            print(name[i], end=' ')
    print('}')

def run(lev):
```

```
    if lev == 3:
        #. print(path)
        print_name()
        return

    for i in range(2):
        path.append(arr[i])
        run(lev+1)
        path.pop()

run(0)
```

1.1.2. Binary Counting

10진수	이진수	{A, B, C}
0	000	{}
1	001	{A}
2	010	{B}
3	011	{A, B}
4	100	{C}
5	101	{A, C}
6	110	{B, C}
7	111	{A, B, C}

- 원소 수에 해당하는 N개의 비트열을 이용
- 0 0 1이면 {A}, 1 1 0이면 {B, C}

```
arr = ['A', 'B', 'C']
n = len(arr)

def get_sub(tar):
    for i in range(n):
        if tar & 0x1: #. 마지막 자리 확인해서 요소의 포함 여부를 결정
            print(arr[i], end='') #. 포함하는 경우 요소를 포함
        tar >> 1 #. 여부 확인한 경우를 제거

for tar in range(0, 1 << n): #. 만들 수 있는 총 개수는 2^n
    print('{', end='')
    get_sub(tar)
    print('}')
```

2. 조합

- Combination
- 서로 다른 N 개의 원소 중 r개를 순서 없이 골라낸 것
- 순열의 경우 {A, B, C, D, E} 중 1, 2, 3등을 뽑을 때 A, B, C와 A, C, B는 다르지만 조합의 경우 5명 중 3명을 뽑기만 하기에 A, B, C와 A, C, B는 동일한 경우다.

```
arr = ['A', 'B', 'C', 'D', 'E']
path = []

n = 3
def run(lev, start):
    if lev == n:
        print(path)
        return

    for i in range(start, 5):
        path.append(arr[i])
        run(lev+1, i+1) #. 선택한 요소의 다음 요소부터 포함 여부를 결정
        path.pop()

run(0, 0)
```

- 위 코드는 level = n, branch = 5인 경우

2. 탐욕 알고리즘

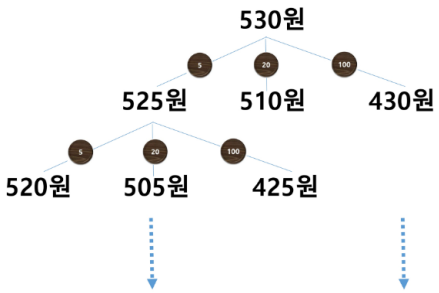
- Greedy
- 결정 시 현재 기준으로 가장 좋아보이는 선택지로 결정하여 답을 도출

2.1. 예시 문제

2.1.1. 동전 교환

- 10, 50, 100, 500 동전이 있고, 최소한의 동전 수로 거스름 돈을 주려 한다.
- 만약 1730원을 거슬러주기 위해 사용할 수 있는 최소 동전의 개수는?
- 큰 동전부터 거슬러 주면 된다. 즉, 500원 3개, 100원 2개, 10원 3개.

```
coin_list = [500, 100, 50, 10]
tar = 1730

cnt = 0
for coin in coin_list:
    possible_cnt = tar // coin

    cnt += possible_cnt
    tar -= coin * possible_cnt

print(cnt)
```



- 만약 동전이 5, 20, 100 단위로 존재한다면, 530원을 거슬러 주기 위해 사용해야 하는 최소 동전의 수는?
- 완탐 시 0원이 될 때까지 모든 경우의 수를 확인해본다. 즉, 최소 level이 되는 경우를 찾으면 된다.

- 단, 위의 경우 그리디는 예외케이스가 발생할 수 있다.
- 만약 100원을 거슬러야 하는 경우 그리디로 접근했을 때는 4개(70, 10, 10, 10)지만 최소 개수는 2개(50, 50)다.
- 즉, 10, 50, 100, 500 처럼, 모든 동전이 배수 관계인 경우는 그리디가 성립한다.
- 하지만 10, 50, 70 처럼 배수 관계가 아닌 동전이 있다면 그리드는 성립하지 않는다.
- 즉, 그리디 알고리즘은 간단해 보이나, 예외 없이 맞는 규칙인지 증명이 어렵다.

2.1.2. 0-1 Knapsack

- 물건의 개수 N, 물건 별 무기 W, 가격 P이 주어질 때, 최대 이득을 구하는 문제.
- 물건이 하나씩만 존재하고, 최대 30kg 까지 짐을 담을 때
- kg 당 가치가 가장 높은 것을 먼저 담으면 답을 구할 수 없다.
- 즉, 완탐 or DP로 접근해야 한다.
- 위 문제의 경우 물건 1을 2개, 3을 1개 넣는 것 보다 물건 2, 3을 하나씩 담는 게 더 값이 높다.

2.1.3. Fractional Knapsack

- 0-1 Knapsack과 달리, 물건을 원하는 만큼 자를 수 있는 Knapsack 문제.
- 즉, 물건 1에서 5kg, 물건 2에서 10kg, 물건 3에서 15kg을 꺼내는 경우가 가능하다.
- 이 경우 그리디가 성립하며, kg당 가격이 가장 높은 물건을 최대한 담으면 된다.
- 즉, 물건 1 전체, 2를 전체, 3을 나머지 모두로 사용하면 50 + 140 + 30 = 220으로 최대가 된다.

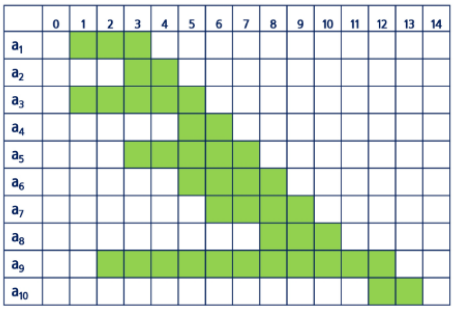
```
n = 3
target = 30
things = [(5, 50), (10, 60), (20, 140)] #. (kg, price)

#. (price / kg) 기준으로 내림차순
things.sort(key = lambda x: (x[1] / x[0]), reverse=True)
#. sort 결과 = [(5, 50), (20, 140), (10, 60)]

sum_v = 0
for kg, price in thgings:
    per_price = price / kg

    #. 가방에 남은 용량이 얼마 없다면,
    #. 물건을 잘라 가방에 넣고 끝낸다.
    if target < kg:
        sum_v += traget * per_price
        break

    sum_v += price
    target -= kg

print(int(sum_v))
```

3. 활동 선택 문제

3.1. Meeting room

- Activity-selection problem
- 회의실이 하나인 회사에서 여러 팀들이 원하는 회의식 예약 시간이 주어질 때, 가능한 많은 회의가 열리기 위해 팀을 배정하는 문제
- 희망 회의 개수, 시작 시각, 종료 시각을 입력 받는다.
- 그리디로 풀 경우 회의 종료 시각이 가장 빠른 회의를 먼저 선택하면 된다.

(1,4), (3,5), (1,6), (5,7), (3,8), (5,9), (6,10), (8,11), (2,13), (12,14)



- 위의 경우 회의들을 종료 시각을 기준으로 오름차순 정렬했다.
- 이후 종료시각이 가장 빠른 회의를 확정한다.
- 이전 종료시각 이후에 시작하는 회의중 가장 빨리 마치는 회의를 확정한다.
- 위 과정을 반복하면 가장 회의가 많이 열리는 경우를 확인할 수 있다.

4. Baby-jin