전수조사 방법

한국기술교육대학교 컴퓨터공학부 김상진

```
while(!morning){
    alcohol++
    dance++
} #party
```

```
while(!sleep){
    think++
    solve++
} #cse-mode
```

교육목표

- 전수조사(완전 탐색) 알고리즘 설계 방법
  - 모든 경우를 다 해보는 방법
  - 경우의 수가 제한적일 경우에만 사용 가능한 방법
    - 제한적이지만 컴퓨터가 하는 것이기 때문에 생각보다 그 범위가 큼
  - 프로그래밍 경시대회
    - 제한시간 1초: 테스트케이스 T, 입력의 크기 n이 주어짐
      - 여러 테스트 파일을 이용하여 검사함. 각 테스트 파일마다 시간 측정
        - 이 방법 외에 알고리즘의 성능을 측정하기 힘듦
    - 주먹구구 법칙. 1초당 필요한 반복문 수행 횟수가 100,000,000이 넘으면 시간초과될 수 있음
    - 예) $T = 100, n = 1,000$일 때 $O(n^2)$은 통과할까? $T \times n^2 = 10^8$
  - 전수조사 알고리즘은 보통 재귀적으로 구현함
    - 동적프로그래밍(dynamic programming), 되추적(backtracking), 분기한정법(branch-and-bound) 등을 통해 개선할 수 있음

전수조사 방법

- 가능한 모든 경우를 다 조사하는 방법
- 단순한 접근이지만 모든 경우를 다 조사하는 프로그램을 작성하는 것도 간단하지 않을 수 있음
- 모든 경우의 수가 적을 경우에만 효과적임
  - 예) 10명의 학생을 줄을 세우는 방법은? 10! = 3,628,800
    - 컴퓨터에게는 별로 큰 수가 아님
  - 최악의 경우를 고려하여 원하는 효율을 얻을 수 있는지 검토해야 함
- 정확하게 모든 경우를 조사한다면 올바른 해답을 얻을 수 있음
- 더 효율적이며 쉽게 구현할 수 있는 방법이 있으면 전수조사 방법을 사용할 필요는 없음
- 특정 경우를 확인하는 과정에서 빠르게 배제할 방법이 있다면 활용할 필요가 있음 (되추적, 분기한정법)
- 모든 경우를 조사하는 과정에서 중복 검사하거나 동일 답을 여러 번 만날 수 있음

모든 경우: 조합

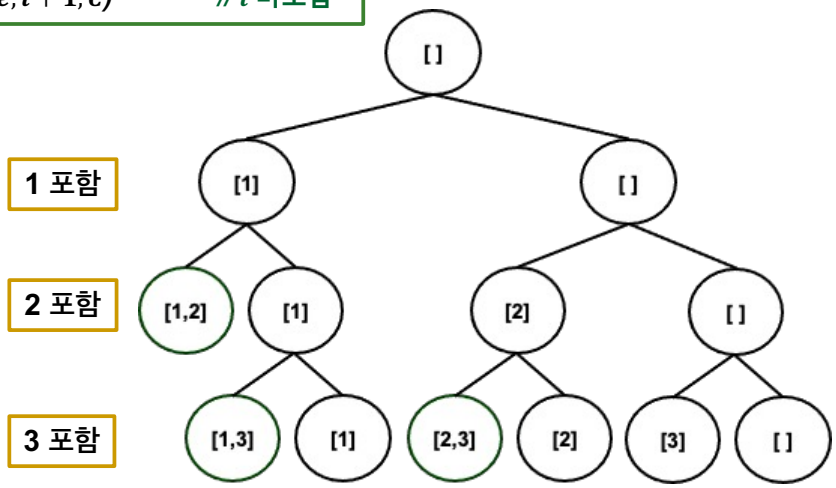
- 모든 경우를 구현할 때 가장 많이 사용하는 방법은 재귀 호출임
- 예) n개의 원소 중 m개를 고르는 모든 조합을 찾는 알고리즘
  - 총 경우의 수: $\binom{n}{m} = \frac{n!}{m!(n-m)!}$
  - n과 m에 따라 가능한 조합이 너무 많아 모든 조합을 찾는 것이 가능하지 않을 수 있음
- 한 요소는 조합에 포함될 수 있고, 포함되지 않을 수 있음
  - 이 특징을 이용하여 재귀 함수로 조합 문제를 해결하는 알고리즘을 만들 수 있음
    - 두 개의 재귀 호출(요소가 포함되는 조합을 찾는 호출과 요소가 포함되지 않는 조합을 찾는 호출)이 필요함
- 재귀 알고리즘을 작성한 후에 알고리즘 분석을 위해 재귀 트리를 그려볼 수 있음
- 거꾸로 재귀 호출 트리를 그리고 이를 바탕으로 알고리즘을 구현할 수 있음

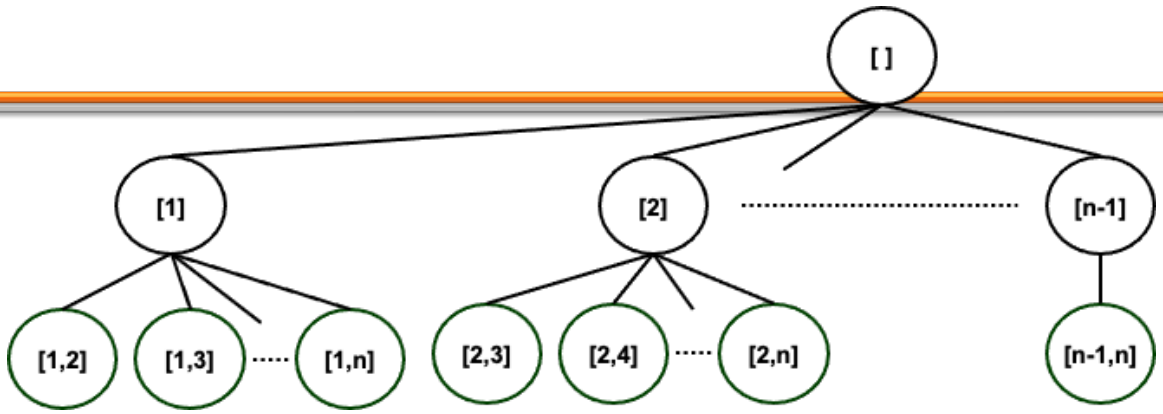
```
function combination(n, m):
    ret := []
    C := [0]×n
    combination(ret, n, m, 1, 1, C)
    return ret

procedure combination(ret, n, m, size, i, C):
    if size = m + 1 then ret.pushback(C.clone())
    if i ≥ n then return
    C[size] := i
    combination(ret, n, m, size + 1, i + 1, C)      // i 포함
    combination(ret, n, m, size, i + 1, C)          // i 미포함
```

종료조건.
1) 만든 조합의 크기가 m보다 커진 경우
2) 다음에 포함할 값이 n보다 커진 경우

2개의 재귀 호출로 구성됨



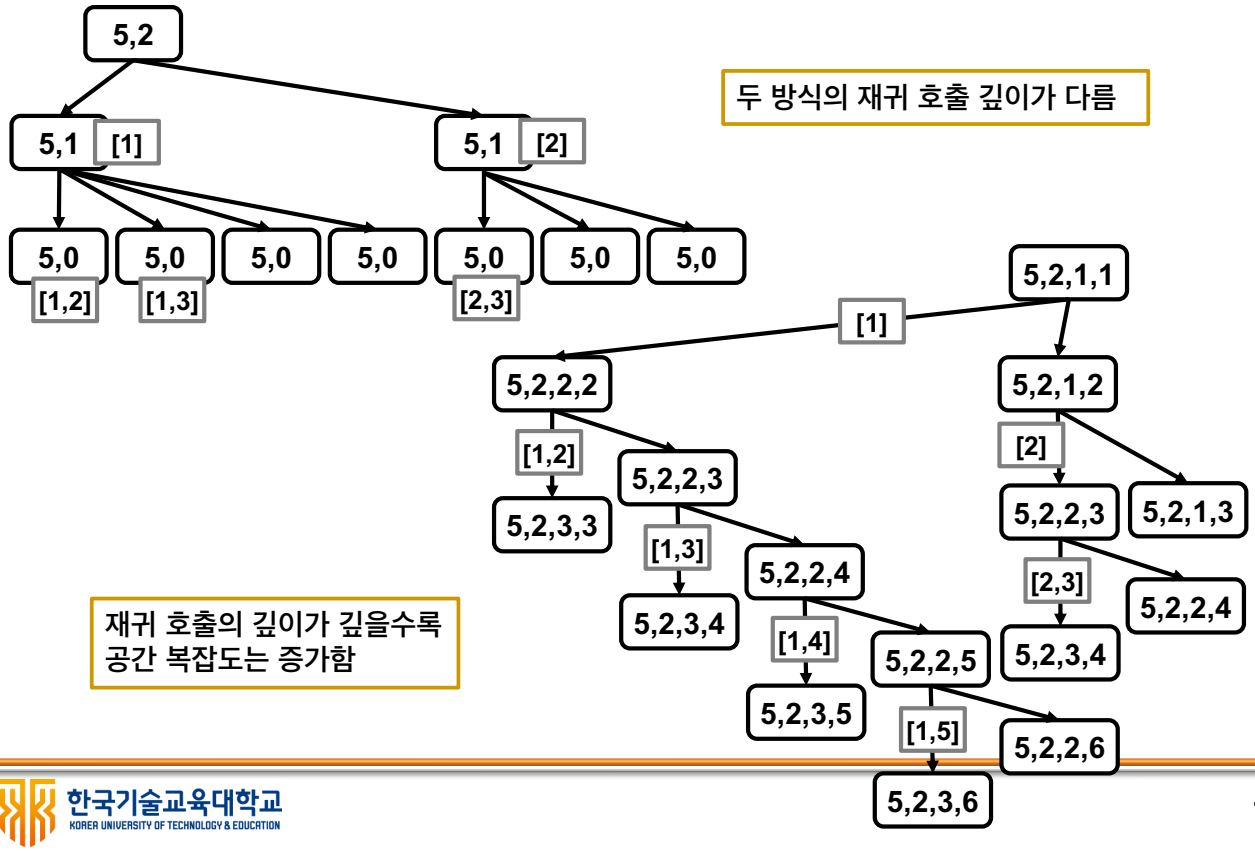



```
function combination(n, m):
    ret := []
    C := []
    combination(ret, n, m, C)
    return ret

procedure combination(ret, n, m, C):
    if m = 0 then ret.pushback(C.clone())
    next := 1 if C.isEmpty() else C.back() + 1
    for i := next to n do
        C.pushback (i)
        combination(ret, n, m − 1, C)
        C.popback(i)
```

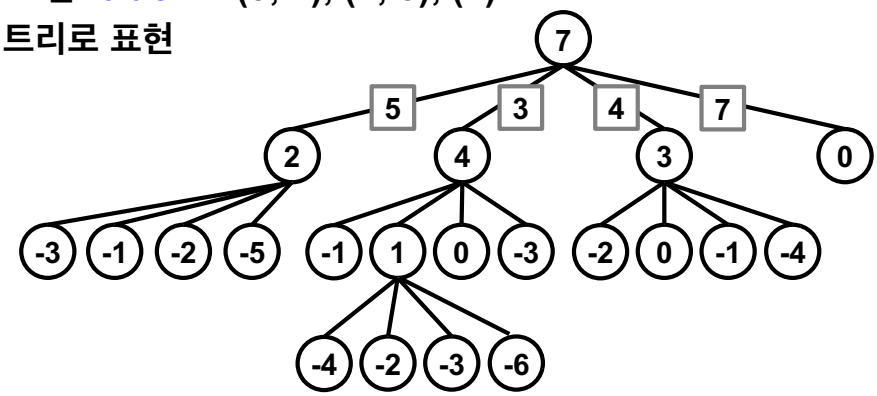
종료조건.
1) 만든 조합의 크기가 m보다 커진 경우
2) 다음에 포함할 값이 n보다 커진 경우

2개의 재귀 호출로 구성됨



CanSum (1/2)

- 입력. 목표 정수 $m(\geq 0)$, n개의 서로 다른 정수 $x_i > 0$
- 출력. 각 x_i을 원하는 만큼 합하여 목표 정수를 만들 수 있으면 true, 없으면 false
- 예) 7, [5, 3, 4, 7]
  - 답: true ⟹ (3, 4), (4, 3), (7)
- 트리로 표현



CanSum (2/2)

- 시간 복잡도: $O(n^m)$
- 공간 복잡도: $O(m+n)$
  - m: 스택 공간

```
canSum(m, A)
    if m < 0 then return false
    if m = 0 then return true
    for all x ∈ A do
        if canSum(m − x, A) then return true
    return false
```

최악의 경우: 답이 false인 경우
트리의 높이는 주어진 정수에 1이 있는 경우: m
재귀 호출 수: $1 + n + \cdots + n^m = (n^{m+1} - 1)/(n-1) \approx n^m$

재귀함수의 성능 분석 방법

- 재귀함수의 시간 복잡도를 분석하는 방법
  - 최악의 경우 재귀 호출 트리를 그린 후에 이 트리에 있는 노드 수를 계산함
  - 이 트리의 노드 수는 트리의 깊이와 각 노드에서 만들어질 수 있는 자식 노드의 수(가지치기 수)에 의해 결정됨
    - 예) canSum의 트리 깊이: 배열에 원소 1이 있는 경우 최대 깊이가 m임
    - 예) canSum 가지치기 수: 항상 배열의 크기 n
  - 전체 비용 = (총 노드 수) × (한 노드에서 소요되는 최대 비용)
    - 예) canSum의 총 노드 수: $\leq n^m$, canSum에서 한 노드에서 소요되는 최대 비용: $O(1) \Rightarrow O(n^m)$
- 재귀함수의 공간 복잡도를 분석하는 방법
  - 최대 재귀 호출의 깊이만큼 함수 스택 공간이 필요함
  - 전체 비용 = (재귀 호출의 최대 깊이) × (함수 스택 공간의 크기) + 입력 크기
    - 예) canSum의 최대 깊이: m, 함수 스택 공간: $O(1) \Rightarrow O(m+n)$

CountSum

- 입력. 목표 정수 $m(\geq 0)$, n개의 서로 다른 정수 $x_i > 0$
- 출력. 각 x_i을 원하는 만큼 합하여 목표 정수를 만들 수 있는 조합의 수
- 예) 7, [5, 3, 4, 7]
  - 답: 3 ⟹ (3, 4), (4, 3), (7)

```
canSum(m, A)
    if m < 0 then return false
    if m = 0 then return true
    for all x ∈ A do
        if canSum(m − x, A) then return true
    return false
```

반환하는 값만 달라짐

```
countSum(m, A)
    if m < 0 then return 0
    if m = 0 then return 1
    count := 0
    for all x ∈ A do
        count += countSum(m − x, A)
    return count
```

- 시간 복잡도: $O(n^m)$
- 공간 복잡도: $O(m+n)$

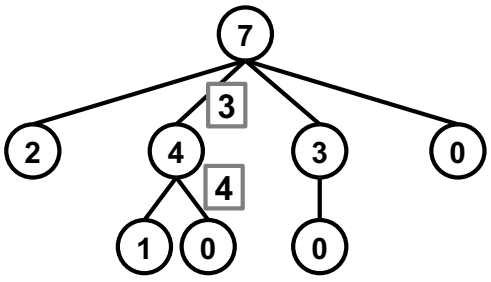
CanSum과 차이점. 중간에 중단하지 않음

HowSum

LeetCode 39

- 입력. 목표 정수 $m(\geq 0)$, n개의 서로 다른 정수 $x_i > 0$
- 출력. 각 x_i을 원하는 만큼 합하여 목표 정수를 만들 수 있는 조합
- 예) 7, [5, 3, 4, 7]
  - 답: (3, 4), (4, 3), (7) 중 하나 출력

- 예) 7, [2, 4] ⟹ null
- 예) 0, [1, 2, 3] ⟹ []




```
howSum(m, A)
    if m < 0 then return null
    if m = 0 then return []
    for all x ∈ A do
        list := howSum(m − x, A)
        if list ≠ null then
            add x to list
            return list
    return null
```

- 시간 복잡도: $O(n^m)$
- 공간 복잡도: $O(2m+n)$

- 함수 스택의 최대 깊이: m
- 답에 해당하는 list가 맨 아래에서 올라오면서 구축되며, 이 list의 최대 길이는 m

CanSum과 마찬가지로 하나를 찾으면 중단

여기서 잠깐.

- CanSum, CountSum의 반환 타입은 원시타입이지만 HowSum, BestSum은 리스트임
  - 언어에 따라 구현하는 방법에 대한 고민이 필요함
  - 자바와 파이썬은 프로그래머가 직접 동적 생성한 것을 반납할 필요가 없지만 C++는 동적 생성한 경우 반납에 대한 처리를 고민해야 함
    - 스마트 포인터를 사용하는 방법도 있지만…
  - C++의 경우 동적 생성하지 않고 값 반환을 생각할 수 있음
    - 이 경우 m < 0일 때와 m == 0일 때 반환하는 것을 구분할 수 있어야 함

```
std::vector<int>* howSum(int m, const std::vector<int>& nums)
```

```
std::vector<int> howSum(int m, const std::vector<int>& nums)
```

```
bool howSum(int m, const std::vector<int>& nums, std::vector<int>& ans)
```

전역변수의 사용은 가급적 NO

BestSum

- 입력. 목표 정수 $m(\geq 0)$, n개의 서로 다른 정수 $x_i > 0$
- 출력. 각 x_i을 원하는 만큼 합하여 목표 정수를 만들 수 있는 가장 짧은 조합 (같은 길이의 조합이 있으면 그들 중 임의로 출력)
- 예) 7, [5, 3, 4, 7]
  - 답: (7)

```
bestSum(m, A)
    if m < 0 then return null
    if m = 0 then return []
    best := null
    for all x ∈ A do
        list := bestSum(m − x, A)
        if list ≠ null then
            if best = null
                or len(best) > len(list) + 1 then
                add x to list
                best := list
    return best
```

- 시간복잡도: $O(n^m)$
- 공간복잡도: $O(3m + n)$

- 스택의 최대 깊이: m
- best와 list의 최대 길이는 m

문제: 소풍 (ALGOSPOT: PICNIC)

- 유치원에서 소풍을 갑니다. 소풍 때 학생들을 두 명씩 짝을 지어 주고자 함.
  이때 서로 친구인 학생들끼리 짝을 지어 주고자 함.
  각 학생 쌍에 대해 이들이 서로 친구인지 여부가 주어질 때,
  학생을 짝지을 수 있는 방법의 수를 계산해 주세요.
- 입력. 학생 수(짝수, <=10), 친구 정보 수, 친구 정보
  - 각 학생 0부터 $n-1$까지 번호로 나타냄
  - 예) 4 6 0 1 1 2 2 3 3 0 0 2 1 3
    - 총 학생 4명, 6개의 친구 정보
    - 친구 정보를 어떻게 유지? 무방향 그래프 (인접 리스트, 인접 행렬)
- 출력. 학생들을 짝지을 수 있는 방법의 수
  - 예) 4 6 0 1 1 2 2 3 3 0 0 2 1 3 → 3

	0	1	2	3
0	-	T	T	T
1	T	-	T	T
2	T	T	-	T
3	T	T	T	-

쌍 조합???

- 모든 경우 찾기
  - 한 쌍을 결정한 후 재귀적으로 나머지 쌍 결정
  - 문제. 중복
    - 예) A B C D
      - (AB, CD), (AC, BD), (AD, BC)
      - (BA, CD), (BC, AD), (BD, AC)
      - …
  - 가능한 짝의 수: $(n-1)\times(n-3)\times\cdots < n!$
    - $n=10: 945$
    - 친구 관계에 따라 더 작음

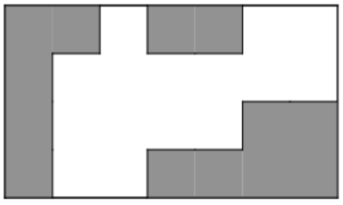
중복 제거

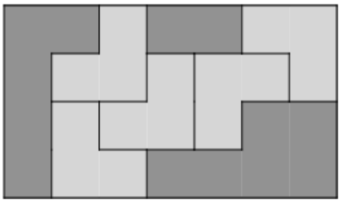
- 중복을 제거하는 방법
  - 방법 1. 집합과 같은 자료구조를 이용하여 중복 요소를 만날 때마다 배제
  - 방법 2. 특정 형태를 갖는 답만 고려
    - 사전 순으로 가장 먼저 오는 답
      - 예) (0, 1), (2, 3): [(2, 3), (0, 1)], [(1, 0), (2, 3)], …

문제: 게임판 덮기 (ALGOSPOT: BOARDCOVER)

- H x W 크기의 게임판이 있고, 게임판은 검은 칸과 흰 칸으로 구성된 격자 모양을 하고 있음. 이 중 모든 흰 칸을 세 칸짜리 L자 모양의 블록으로 덮고 싶음. 블록은 자유롭게 회전 가능하지만 겹치지 않아야 하며, 검은 칸이나 게임판 밖으로 나갈 수 없음
- 가정. 게임판에 있는 흰 칸의 수는 50을 넘지 않음
- 입력. H, W, W개의 #와 .로 구성된 H줄
  - 예) 3 7
    #.....#
    #.....#
    ##...##
  - 내부적으로 어떻게 표현?
- 출력. 덮을 수 있는 경우의 수
  - 예) 0

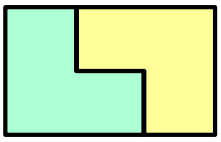
- 50을 초과하지 않는다는 것이 중요함. 경우의 수가 전수조사로 가능하다는 것임
- 전체 경우의 수?





문제: 게임판 덮기

- 예외적 상황?
- 이 문제에서 중복은?
  - 일정한 고정 방향으로 조사
- 회전은?
  - 총 4가지 모양
    - 한 위치에서 매번 4개의 재귀 호출?



3자리 짝수 찾기

- 입력. n개의 정수 d_i, $3 \leq n \leq 100$, $0 \leq d_i \leq 9$
- 출력. 주어진 정수로 만들 수 있는 독특한 3자리 짝수를 오름차순으로 출력
- 전수조사?
  - (100, 3) = 161,700
  - 전수조사 후 트리 기반 집합에 삽입 후 출력?
    - 정렬된 상태로 출력해야 한다고 정렬 알고리즘의 사용이 반드시 필요한 것은 아님
  - 중복이 많음
    - 같은 수가 4개 이상 등장하면 불필요함
    - 빈도수 배열
    - 작은 수부터 만들면 나중에 정렬할 필요가 없음
  - 거꾸로 100부터 2씩 증가하면서 해당 수를 주어진 d_i로 만들 수 있는지 조사함