

#1. 재귀 함수와 완전 탐색

나정휘

<https://justicehui.github.io/>

목차

재귀 함수

백트래킹(backtracking)

분기 한정(branch and bound)

재귀 함수

재귀 함수: 자기 자신을 호출하는 함수

- 예시: $f(n) = f(n-1) + f(n-2)$
- 용도: 점화식 계산, 완전 탐색, 반복문으로 반복하기 힘든 작업, ...

다양한 알고리즘에 사용되는 개념

- 동적 계획법, 분할 정복은 재귀적/귀납적 사고를 기반으로 함
- 트리 순회, 세그먼트 트리 등 트리 구조와 관련된 알고리즘은 주로 재귀 함수를 이용

재귀 함수의 예시 - 1

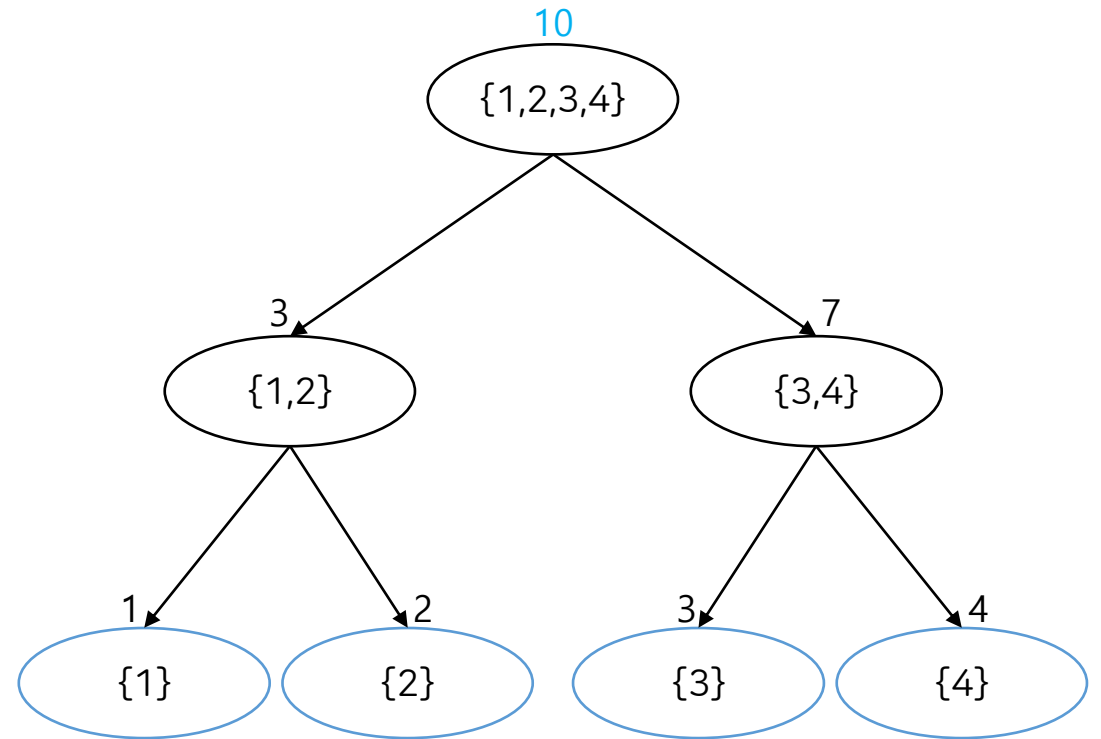
배열 A의 모든 원소의 합을 구하는 작업

- $f(l, r)$: $A[l] + A[l+1] + \dots + A[r]$ 을 구하는 함수
- $f(0, N-1)$ 을 구해야 함
- $f(l, r)$ 을 계산하는 방법
 - 일을 혼자 하는 것은 어려우니까 부하 직원 2명 고용
 - 배열을 절반으로 나눠서 직원 2명에게 분배
- 수식으로 표현
 - 배열의 중간 지점을 $m = (l + r) / 2$ 라고 하면
 - $f(l, r) = f(l, m) + f(m+1, r)$
 - $l = r$ 이면 $f(l, r) = A[l]$

재귀 함수의 예시 - 1

배열 A의 모든 원소의 합을 구하는 작업

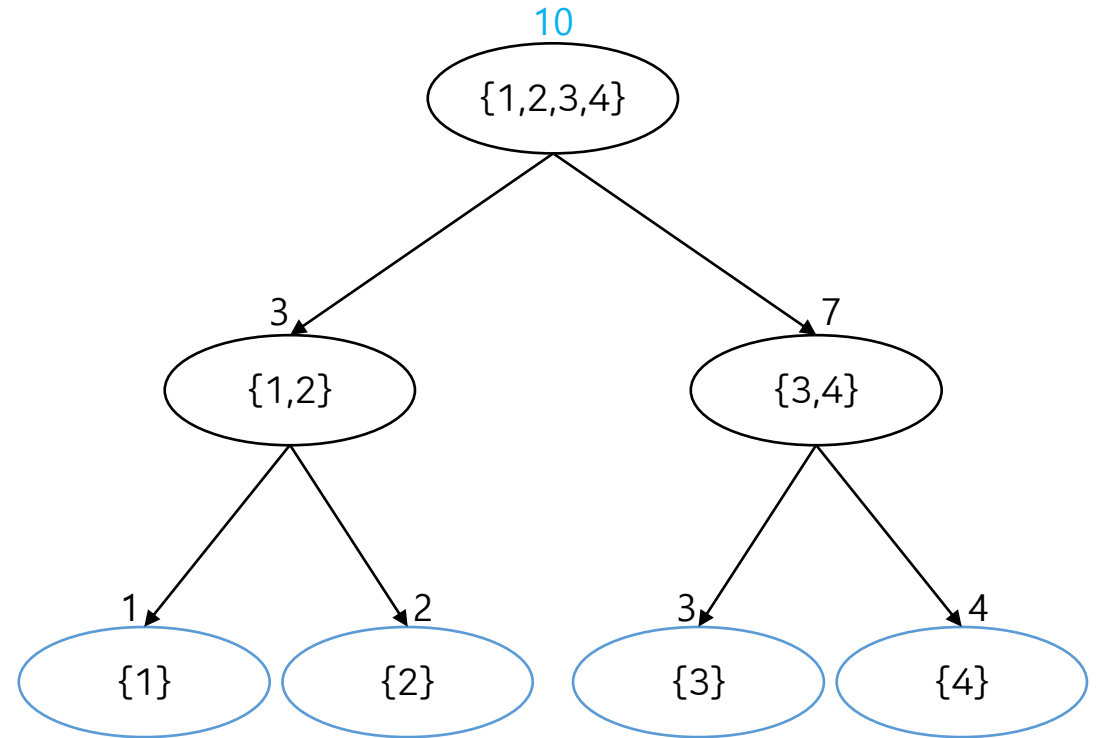
- $f(l, r)$: $A[l] + A[l+1] + \dots + A[r]$ 을 구하는 함수
- $f(l, r) = f(l, m) + f(m+1, r)$
- $l = r$ 이면 $f(l, r) = A[l]$
- 재귀 호출 과정을 살펴보면...



재귀 함수의 예시 - 1

배열 A의 모든 원소의 합을 구하는 작업

- $f(l, r)$: $A[l] + A[l+1] + \dots + A[r]$ 을 구하는 함수
- $f(l, r) = f(l, m) + f(m+1, r)$
- $l = r$ 이면 $f(l, r) = A[l]$
- 재귀 호출 과정을 살펴보면...
 - 배열의 길이가 1000000이라면?
 - 손으로 직접 호출 과정을 따라가기 어려움
 - 좋은 방법이 없을까?



재귀 함수의 예시 - 1

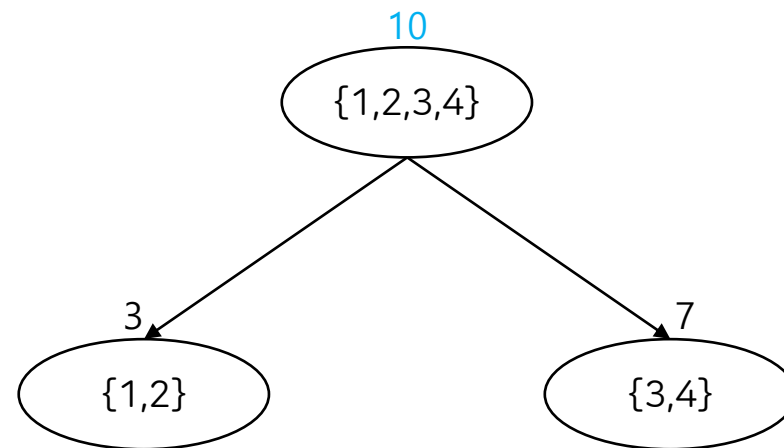
배열 A의 모든 원소의 합을 구하는 작업

- $f(l, r): A[l] + A[l+1] + \dots + A[r]$ 을 구하는 함수
- $f(l, r)$ 을 계산하는 방법
 - 일을 혼자 하는 것은 어려우니까 부하 직원 2명 고용
 - 배열을 절반으로 나눠서 직원 2명에게 분배
- 재귀 호출 과정을 파악하는 방법
 - 부하 직원이 항상 올바른 결과를 반환한다고 “믿고”
 - 내 할 일만 잘 하면 됨
 - 수학적 귀납법

재귀 함수의 예시 - 1

배열 A의 모든 원소의 합을 구하는 작업

- $f(l, r)$: $A[l] + A[l+1] + \dots + A[r]$ 을 구하는 함수
- $f(l, r) = f(l, m) + f(m+1, r)$
- $l = r$ 이면 $f(l, r) = A[l]$
- 재귀 호출 과정을 살펴보면... (애니메이션)
 - 함수 f가 올바른 결과를 반환한다고 "믿고"
 - $f(l, m) + f(m+1, r)$ 의 합만 잘 계산하면 됨



재귀 함수

재귀 호출

- 자신과 동일한 일을 하는 부하 직원에게 작업을 요청하고 그 결과물을 돌려받는 과정
- 부하 직원의 결과물이 항상 올바르다고 믿고 작업을 수행하면 됨
- = 수학적 귀납법

재귀 함수

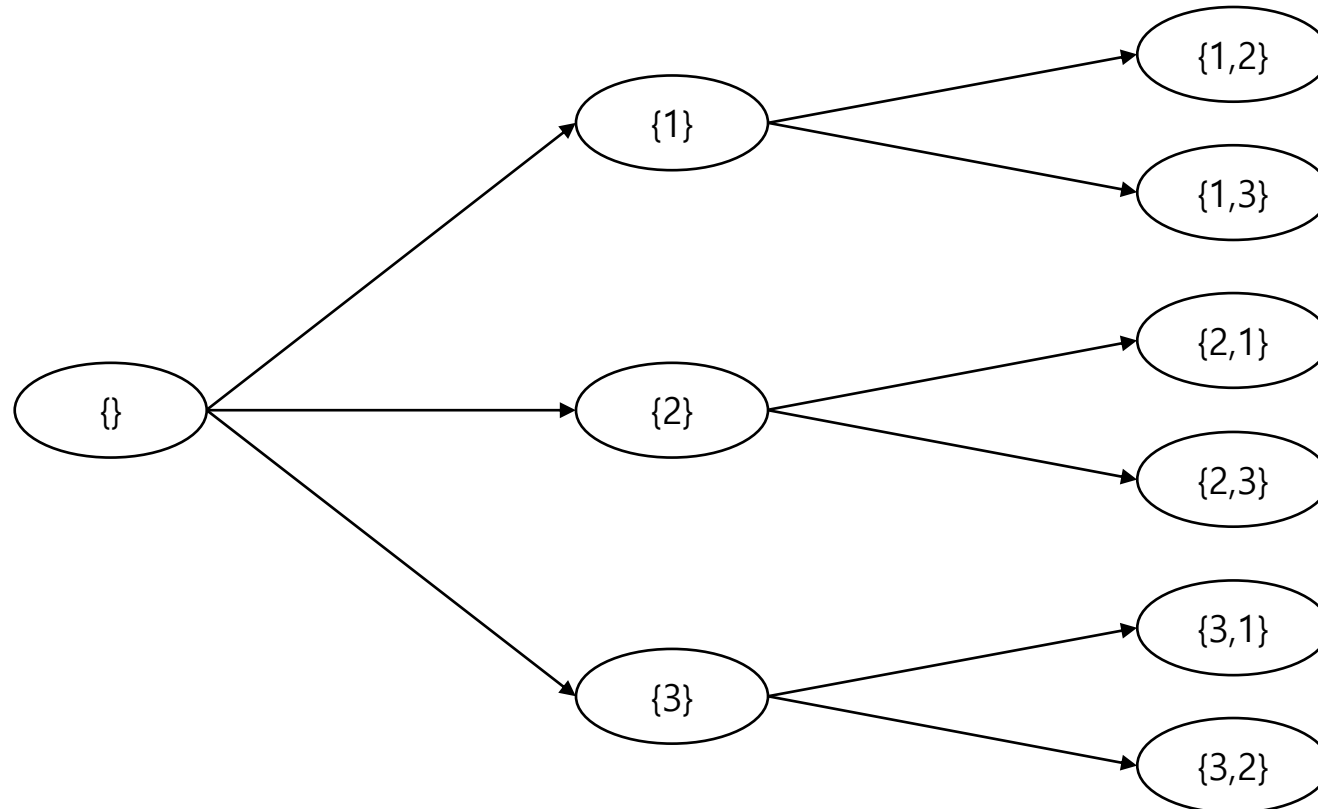
주의사항

- 재귀 호출을 하지 않는 조건(종료 조건)이 있어야 함
 - 없으면 프로그램이 종료되지 않음
- 재귀 호출 과정에 사이클이 없어야 함
 - $f(a) \rightarrow f(b) \rightarrow f(c) \rightarrow \dots \rightarrow f(a)$
 - 프로그램이 종료되지 않음
- 종료 조건에 가까워지는 방향으로 설계
 - 앞에서 본 예시는 항상 $l \leq r$ 을 만족하고 $l = r$ 이 종료 조건
 - $r - l$ 이 작아지는 방향으로 재귀 호출

재귀 함수의 예시 - 2

BOJ 15649. N과 M (1)

- 1부터 N까지의 자연수 중 중복 없이 M개를 고른 수열을 모두 출력하는 문제
- 손으로 모든 순열을 구할 때는 수형도를 이용할 수 있음
- ex. $N = 3, M = 2$



재귀 함수의 예시 - 2

BOJ 15649. N과 M (1)

- f(choose, arr)
 - f(지금까지 선택한 수의 개수, 지금까지 만든 배열)
 - choose = M이면 arr 출력하고 종료
 - arr에 없는 수를 하나 골라서 맨 뒤에 추가하고 재귀 호출
 - arr[choose] = num;
 - f(choose+1, arr)
 - arr[choose] = 0;

```
void f(int n, int m, int choose, int arr[]){
    if(choose == m){ // 종료 조건
        for(int i=0; i<m; i++){
            printf("%d ", arr[i]);
        }
        printf("\n");
        return;
    }

    for(int i=1; i<=n; i++){ // n개의 수를 한 번씩 시도
        int exist = 0; // 이미 만든 배열에 i가 있으면 exist = 1
        for(int j=0; j<choose; j++){
            if(arr[j] == i) exist = 1;
        }
        if(exist == 0){
            arr[choose] = i; // i 선택
            f(n, m, choose+1, arr); // 선택한 수의 개수 1 증가
            arr[choose] = 0; // i 넣은 거 원상 복구
        }
    }
}
```

재귀 함수의 예시 - 2

BOJ 15649. N과 M (1)

- exist 변수의 값을 매번 계산하는 것을 비효율적
- use[num] = num을 사용했으면 1, 안 했으면 0
 - arr[choose] = num;
 - use[num] = 1;
 - f(choose+1, arr, use);
 - arr[choose] = 0;
 - use[num] = 0;

```
void f(int n, int m, int choose, int arr[], int use[]){
    if(choose == m){ // 종료 조건
        for(int i=0; i<m; i++){
            printf("%d ", arr[i]);
        }
        printf("\n");
        return;
    }

    for(int i=1; i<=n; i++){ // n개의 수를 한 번씩 시도
        if(use[i] == 0){ // 아직 i를 사용하지 않았다면
            arr[choose] = i; // i 선택
            use[i] = 1;
            f(n, m, choose+1, arr, use); // 선택한 수의 개수 1 증가
            arr[choose] = 0; // i 넣은 거 원상 복구
            use[i] = 0;
        }
    }
}
```

재귀 함수의 예시 - 2

BOJ 15649. N과 M (1)

- 항상 5개의 인자를 모두 넘겨줘야 할까?
 - n, m은 변하지 않는 값
 - arr, use도 포인터를 넘겨주는 것이므로 변하지 않음
 - 항상 동일한 값은 전역 변수로 빼도 됨

```
int N, M, A[8], U[9]; // 전역 변수는 0으로 초기화

void f(int choose){
    if(choose == M){
        for(int i=0; i<M; i++) printf("%d ", A[i]);
        printf("\n");
        return;
    }

    for(int i=1; i<=N; i++){
        if(!U[i]){
            A[choose] = i; U[i] = 1;
            f(choose+1);
            A[choose] = 0; U[i] = 0;
        }
    }
}
```

재귀 함수의 예시 - 3

BOJ 15650. N과 M (2)

- 1부터 N까지의 자연수 중 중복 없이 M개를 오름차순으로 고른 수열을 모두 출력하는 문제
- 현재 배열의 맨 뒤에 있는 수보다 큰 수를 선택해야 함

```
int N, M, A[8];

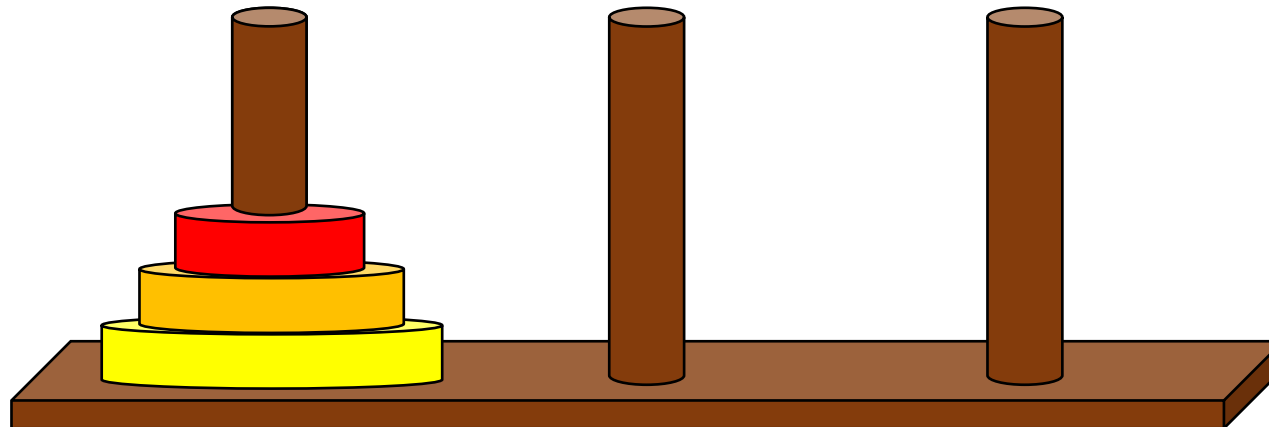
void f(int choose){
    if(choose == M){
        for(int i=0; i<M; i++) printf("%d ", A[i]);
        printf("\n");
        return;
    }

    for(int i=1; i<=N; i++){
        if(choose == 0 || A[choose-1] < i){
            A[choose] = i;
            f(choose+1);
        }
    }
}
```

재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

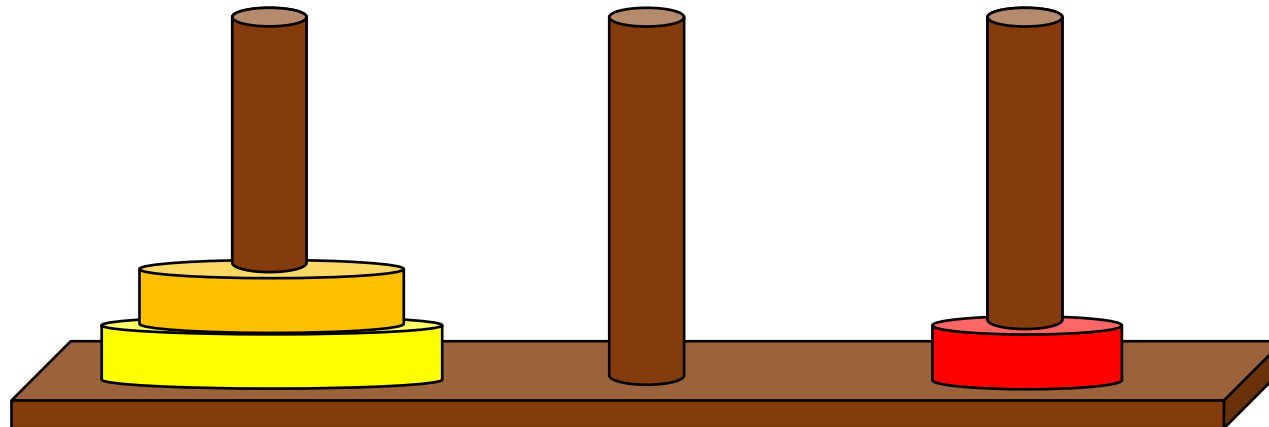
- 3개의 기둥과 크기가 서로 다른 N개의 원판이 있음
- 원판 위에는 자기 자신보다 작은 원판만 올라갈 수 있음
- 원판을 최소한으로 이동해서 첫 번째 기둥에 있는 모든 원판을 세 번째 기둥으로 옮기는 문제



재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

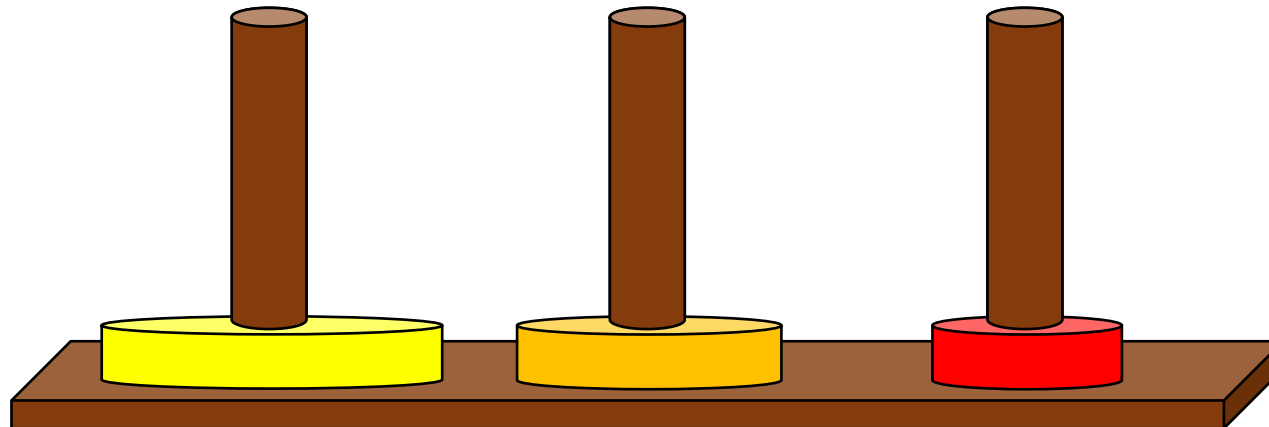
- 3개의 기둥과 크기가 서로 다른 N개의 원판이 있음
- 원판 위에는 자기 자신보다 작은 원판만 올라갈 수 있음
- 원판을 최소한으로 이동해서 첫 번째 기둥에 있는 모든 원판을 세 번째 기둥으로 옮기는 문제



재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

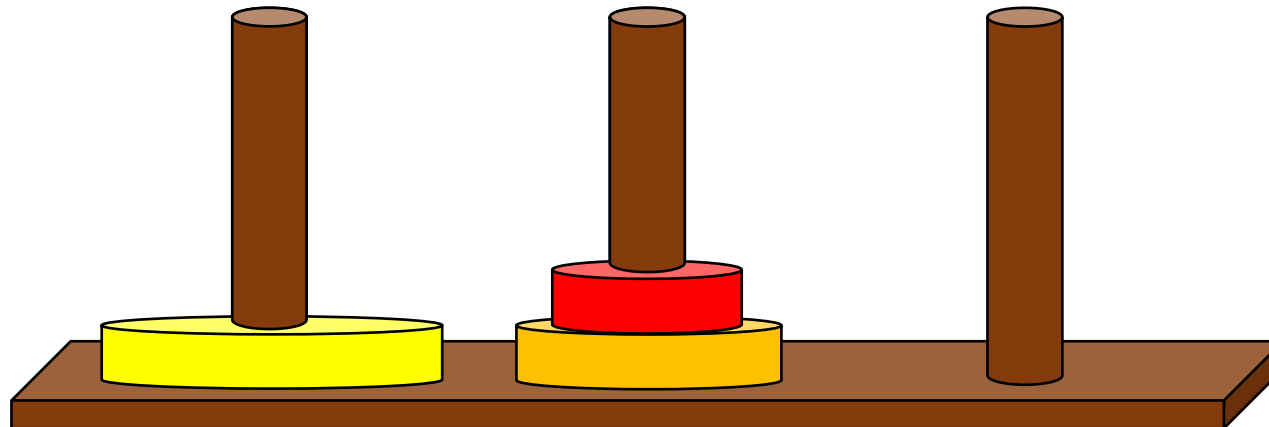
- 3개의 기둥과 크기가 서로 다른 N개의 원판이 있음
- 원판 위에는 자기 자신보다 작은 원판만 올라갈 수 있음
- 원판을 최소한으로 이동해서 첫 번째 기둥에 있는 모든 원판을 세 번째 기둥으로 옮기는 문제



재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

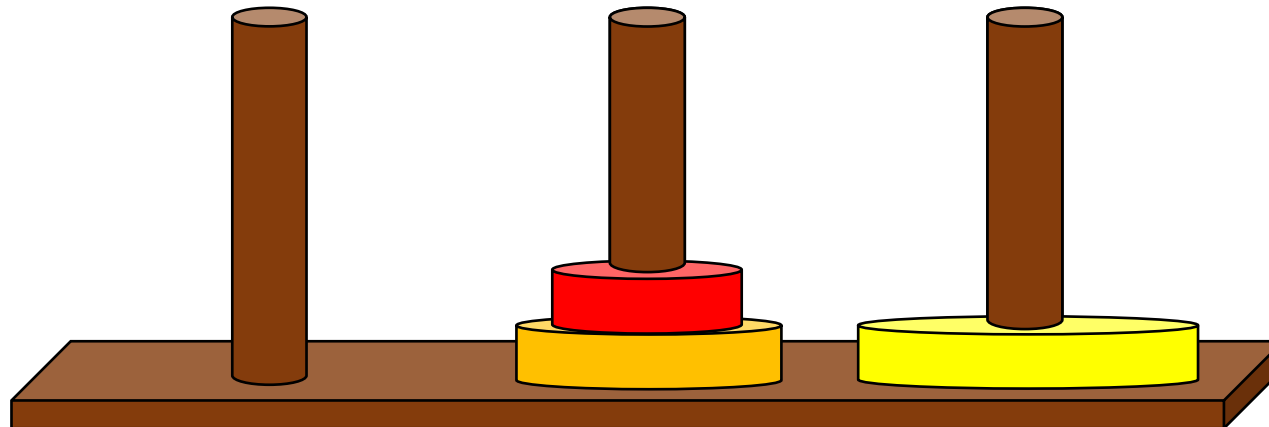
- 3개의 기둥과 크기가 서로 다른 N개의 원판이 있음
- 원판 위에는 자기 자신보다 작은 원판만 올라갈 수 있음
- 원판을 최소한으로 이동해서 첫 번째 기둥에 있는 모든 원판을 세 번째 기둥으로 옮기는 문제



재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

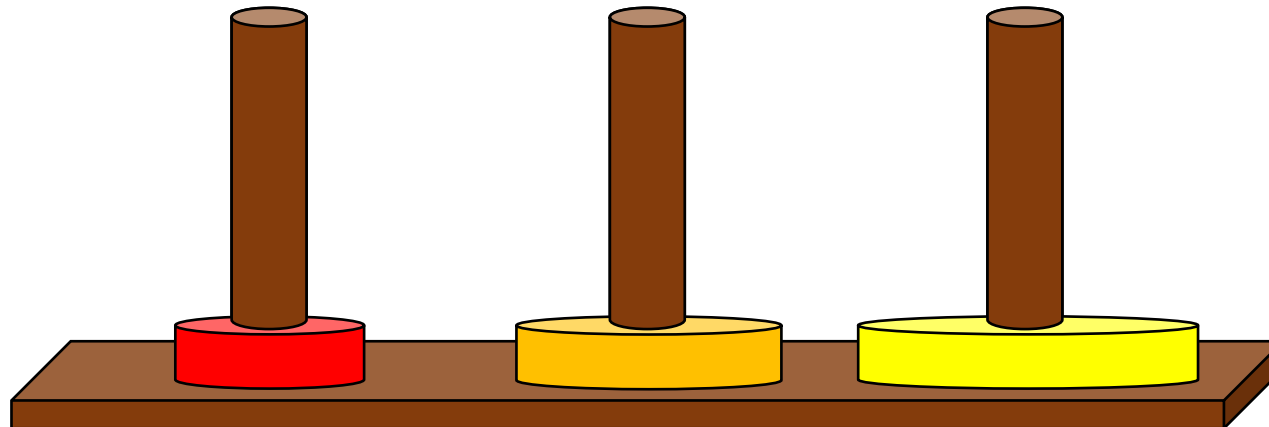
- 3개의 기둥과 크기가 서로 다른 N개의 원판이 있음
- 원판 위에는 자기 자신보다 작은 원판만 올라갈 수 있음
- 원판을 최소한으로 이동해서 첫 번째 기둥에 있는 모든 원판을 세 번째 기둥으로 옮기는 문제



재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

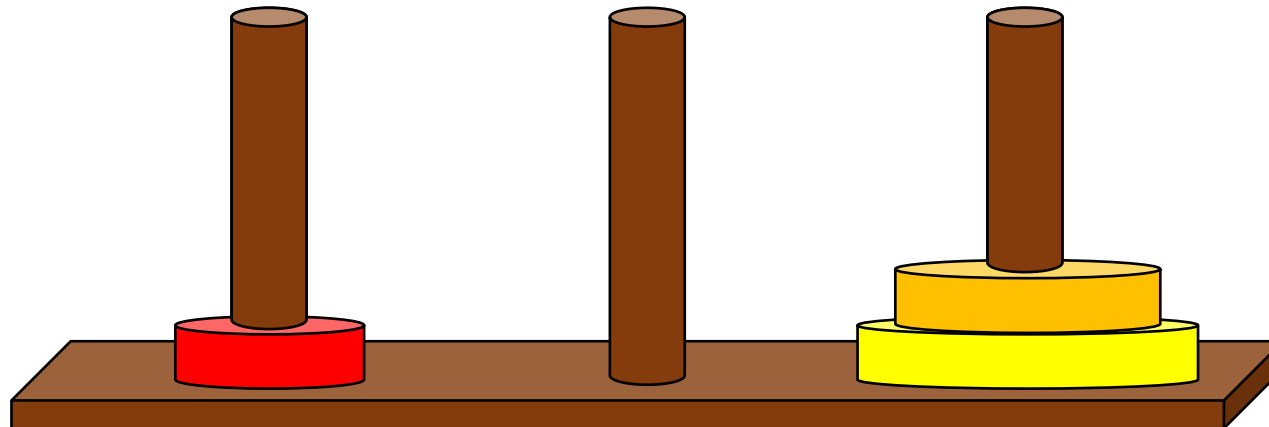
- 3개의 기둥과 크기가 서로 다른 N개의 원판이 있음
- 원판 위에는 자기 자신보다 작은 원판만 올라갈 수 있음
- 원판을 최소한으로 이동해서 첫 번째 기둥에 있는 모든 원판을 세 번째 기둥으로 옮기는 문제



재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

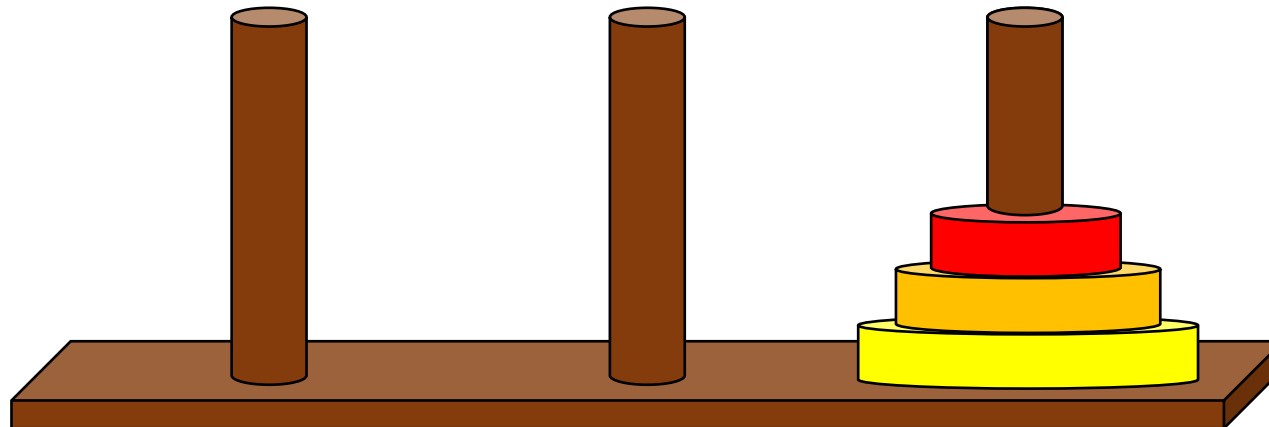
- 3개의 기둥과 크기가 서로 다른 N개의 원판이 있음
- 원판 위에는 자기 자신보다 작은 원판만 올라갈 수 있음
- 원판을 최소한으로 이동해서 첫 번째 기둥에 있는 모든 원판을 세 번째 기둥으로 옮기는 문제



재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

- 3개의 기둥과 크기가 서로 다른 N개의 원판이 있음
- 원판 위에는 자기 자신보다 작은 원판만 올라갈 수 있음
- 원판을 최소한으로 이동해서 첫 번째 기둥에 있는 모든 원판을 세 번째 기둥으로 옮기는 문제



재귀 함수의 예시 - 4

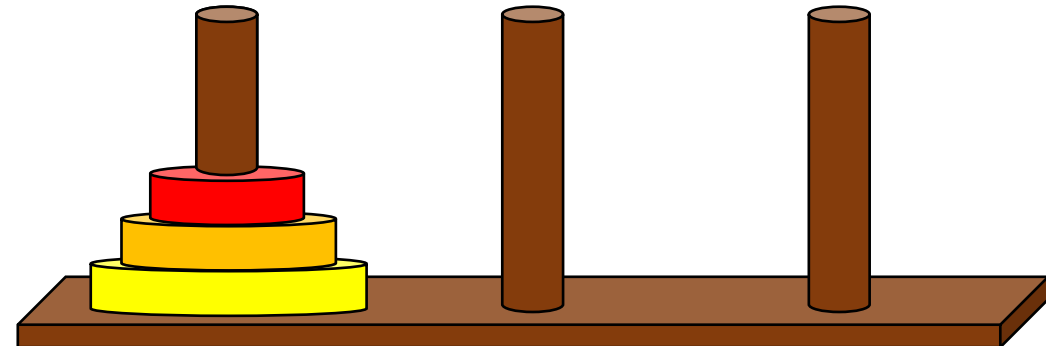
BOJ 11729. 하노이 탑 이동 순서

- 최적 전략은 어떤 형태일까?
- 첫 번째 기둥에 있는 N개의 원판을 세 번째 기둥으로 옮겨야 함
- 가장 큰 원판부터 차례대로 세 번째 기둥으로 옮김
- 가장 큰 원판을 어떻게 꺼내지?
- 위에 있는 N-1개를 잠시 두 번째 기둥으로 옮기면 됨
- N-1개를 첫 번째 기둥에서 두 번째 기둥으로 옮기고
- N번째 원판을 첫 번째 기둥에서 세 번째 기둥으로 옮기고
- N-1개를 두 번째 기둥에서 세 번째 기둥으로 옮기면 됨

재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

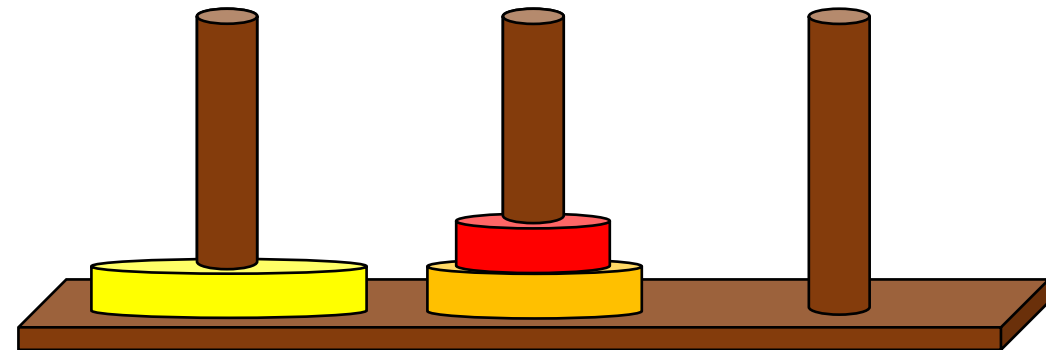
- 최적 전략은 어떤 형태일까?
- 첫 번째 기둥에 있는 N개의 원판을 세 번째 기둥으로 옮겨야 함
- 가장 큰 원판부터 차례대로 세 번째 기둥으로 옮김
- 가장 큰 원판을 어떻게 꺼내지?
- 위에 있는 N-1개를 잠시 두 번째 기둥으로 옮기면 됨
- N-1개를 첫 번째 기둥에서 두 번째 기둥으로 옮기고
- N번째 원판을 첫 번째 기둥에서 세 번째 기둥으로 옮기고
- N-1개를 두 번째 기둥에서 세 번째 기둥으로 옮기면 됨



재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

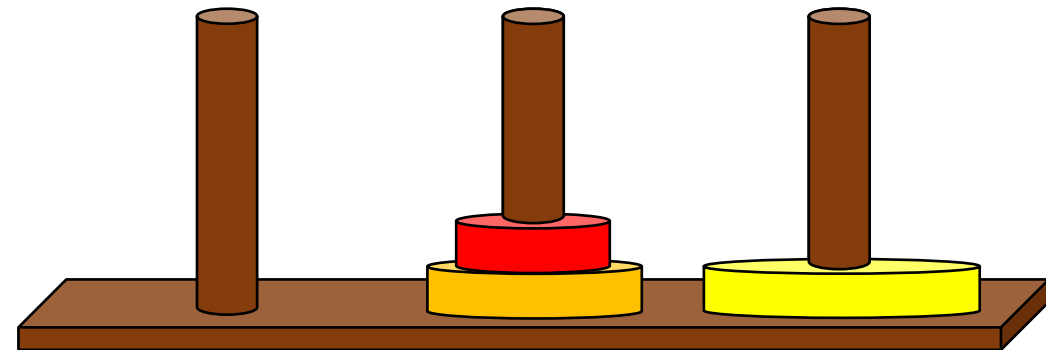
- 최적 전략은 어떤 형태일까?
- 첫 번째 기둥에 있는 N개의 원판을 세 번째 기둥으로 옮겨야 함
- 가장 큰 원판부터 차례대로 세 번째 기둥으로 옮김
- 가장 큰 원판을 어떻게 꺼내지?
- 위에 있는 N-1개를 잠시 두 번째 기둥으로 옮기면 됨
- N-1개를 첫 번째 기둥에서 두 번째 기둥으로 옮기고
- N번째 원판을 첫 번째 기둥에서 세 번째 기둥으로 옮기고
- N-1개를 두 번째 기둥에서 세 번째 기둥으로 옮기면 됨



재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

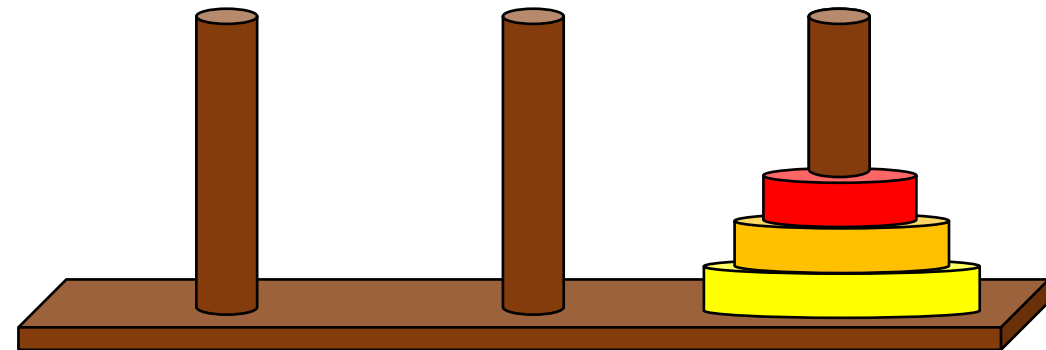
- 최적 전략은 어떤 형태일까?
- 첫 번째 기둥에 있는 N개의 원판을 세 번째 기둥으로 옮겨야 함
- 가장 큰 원판부터 차례대로 세 번째 기둥으로 옮김
- 가장 큰 원판을 어떻게 꺼내지?
- 위에 있는 N-1개를 잠시 두 번째 기둥으로 옮기면 됨
- N-1개를 첫 번째 기둥에서 두 번째 기둥으로 옮기고
- N번째 원판을 첫 번째 기둥에서 세 번째 기둥으로 옮기고
- N-1개를 두 번째 기둥에서 세 번째 기둥으로 옮기면 됨



재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

- 최적 전략은 어떤 형태일까?
- 첫 번째 기둥에 있는 N개의 원판을 세 번째 기둥으로 옮겨야 함
- 가장 큰 원판부터 차례대로 세 번째 기둥으로 옮김
- 가장 큰 원판을 어떻게 꺼내지?
- 위에 있는 N-1개를 잠시 두 번째 기둥으로 옮기면 됨
- N-1개를 첫 번째 기둥에서 두 번째 기둥으로 옮기고
- N번째 원판을 첫 번째 기둥에서 세 번째 기둥으로 옮기고
- N-1개를 두 번째 기둥에서 세 번째 기둥으로 옮기면 됨



재귀 함수의 예시 - 4

BOJ 11729. 하노이 탑 이동 순서

- $f(n, a, b, c)$: n 개의 원판을 a 에서 c 로 옮기는 과정을 구하는 함수
 - $n-1$ 개의 원판을 a 에서 b 로 이동 $f(n-1, a, c, b)$
 - n 번째 원판을 a 에서 c 로 이동: $f(1, a, b, c)$
 - $n-1$ 개의 원판을 b 에서 c 로 이동 $f(n-1, b, a, c)$
 - 종료 조건: $n = 1$ 일 때 원판 1개를 a 에서 c 로 이동
- 이동 횟수
 - $T(1) = 1$
 - $T(n) = 2T(n-1) + 1$
 - $T(n) = 2^n - 1$



```
void f(int n, int a, int b, int c){  
    if(n == 1){  
        printf("%d %d\n", a, c);  
        return;  
    }  
    f(n-1, a, c, b);  
    f(1, a, b, c);  
    f(n-1, b, a, c);  
}
```

백트래킹과 분기 한정

백트래킹(backtracking, 퇴각 검색)

- 특정 제약 조건을 만족하는 문제의 해를 찾기 위한 전략
 - 문제의 해는 주로 n -tuple $(x_1, x_2, \dots, x_n)$ 으로 표현
 - $x_1, x_2, x_3, \dots, x_n$ 의 값을 차례대로 확정 짓는 방식
 - 만약 현재까지 구성한 해 $(x_1, x_2, \dots, x_i)$ 로 시작하는 정답이 존재하지 않으면 이전 상태로 돌아감 (퇴각)
- 예시 문제
 - $n \times n$ 크기의 체스판에 n 개의 퀸이 서로 공격하지 못하도록 배치
 - n 개의 물건이 있고, i 번째 물건의 무게가 w_i , 가격이 v_i 일 때, 무게 합이 k 이하면서 가격 합이 최대인 집합 찾기
 - 길이가 $L_1, L_2, \dots, L_n$ 인 n 개의 막대가 있을 때, 막대를 적당히 연결해서 막대의 길이가 모두 같도록 만들기

분기 한정(branch and bound)

- 최적해를 찾을 가능성이 없다면 분기를 하지 않고 탐색을 중단하는 것
- 우리가 방금 전까지 한 그것!

백트래킹과 분기 한정

N-Queen 문제

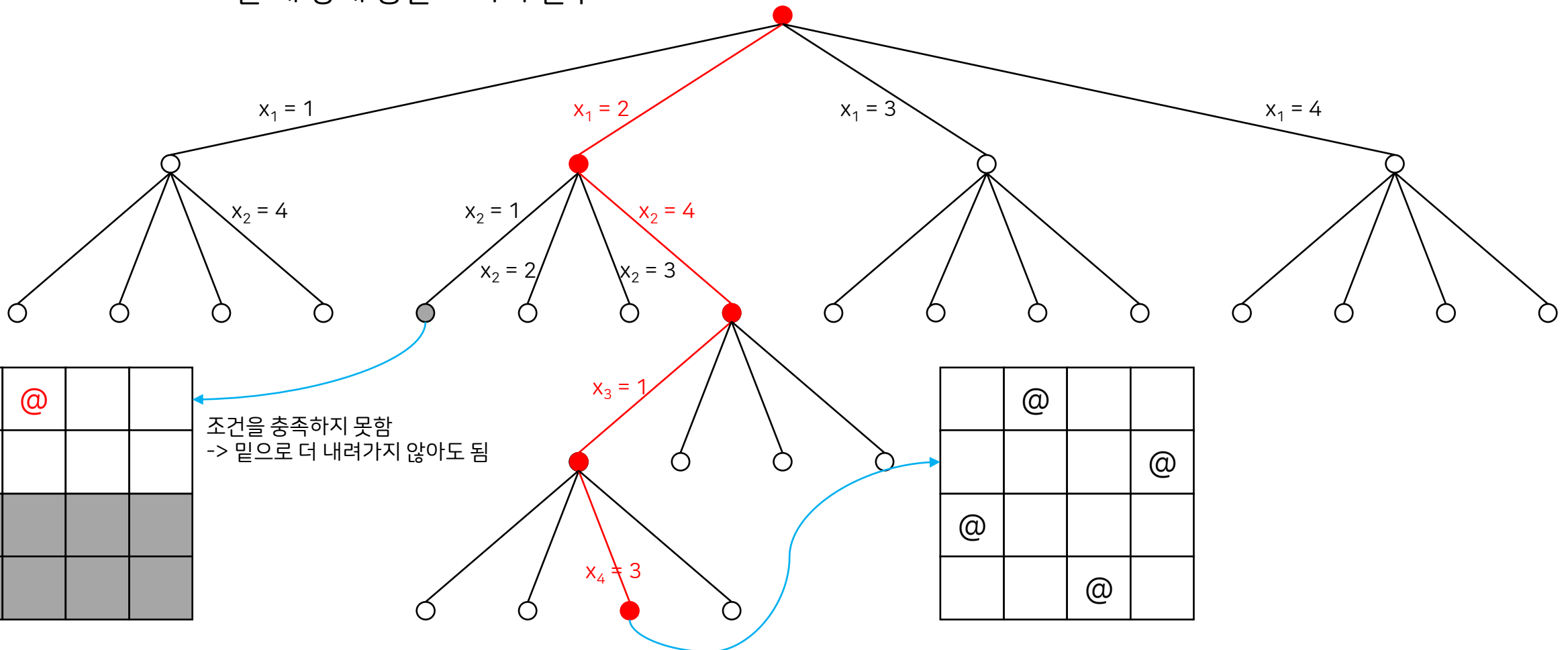
- $n \times n$ 크기의 체스판에 n 개의 퀸이 서로 공격하지 못하도록 배치
- $x_i = i$ 번째 행에 배치되는 퀸의 열 번호
- 기본적인 제약 조건: $\{(x_1, x_2, \dots, x_n) \mid 1 \leq x_i \leq n\}$
- 탐색해야 하는 상태 공간의 크기: $O(n^n)$
- $(x_1, x_2, x_3, x_4) = (2, 4, 1, 3)$

	@		
			@
@			
		@	

백트래킹과 분기 한정

N-Queen 문제

- 상태 공간 트리: 모든 가능한 해의 집합을 트리로 나타낸 것
- $n = 4$ 일 때 상태 공간 트리의 일부



백트래킹과 분기 한정

N-Queen 문제



```
bool feasible(vector<int> queen, int new_col){
    int new_row = queen.size();
    for(int i=0; i<queen.size(); i++){
        // 같은 열에 이미 퀸이 있는 경우
        if(queen[i] == new_col) return false;
        // \ 방향 대각선에 이미 퀸이 있는 경우
        if(i - queen[i] == new_row - new_col) return false;
        // / 방향 대각선에 이미 퀸이 있는 경우
        if(i + queen[i] == new_row + new_col) return false;
    }
    return true;
}

void n_queen(int n, vector<int> queen){
    if(queen.size() == n){ print(n, queen); return; }
    for(int p=0; p<n; p++){
        if(feasible(queen, p)){ // 다음 행에서 p번 열에 퀸을 놓을 수 있다면
            queen.push_back(p); // x[i] = p인 해를 모두 탐색
            n_queen(n, queen);
            queen.pop_back(); // x[i] = p인 해 탐색 종료
        }
    }
}
```

▼ Input

5

► Arguments

▼ Output

@....
..@..
....@
.@...
...@.

@....
...@.
.@...
....@
..@..

.@...
...@.
@....
..@..
....@

.@...
....@
..@..
@....
...@.

..@..
@....
...@.
.@...
....@

백트래킹과 분기 한정

0/1 배낭 문제

- n 개의 물건이 있음
- i 번째 물건의 무게는 w_i , 가격은 v_i
- 무게의 합이 k 를 넘지 않도록 물건을 몇 개 선택할 때, 만들 수 있는 가격의 합의 최댓값을 구하는 문제
- 기본적인 제약 조건: $\{(x_1, x_2, \dots, x_n) \mid x_i \in \{0, 1\}\}$
- 탐색해야 하는 상태 공간의 크기: $O(2^n)$
- 모든 해를 찾은 뒤, 가격의 합의 최댓값을 구하면 됨



```
int N, K, W[111], V[111], Max;

void knapsack(int idx, int weight, int value){
    if(idx == N){
        Max = max(Max, value); return;
    }
    // idx번째 물건을 선택하지 않는 경우
    knapsack(idx+1, weight, value);

    // idx번째 물건을 사용하는 경우
    if(weight + W[idx] <= K)
        knapsack(idx+1, weight+W[idx], value+V[idx]);
}
```

백트래킹과 분기 한정

0/1 배낭 문제

- 모든 해를 탐색할 필요는 없지 않을까?
 - ex) 남은 모든 물건을 선택하더라도 지금까지 구한 것보다 가격이 낮은 경우
 - ex) 남은 물건 중 가성비가 제일 좋은 물건으로 가득 채워도 지금까지 구한 것보다 가격이 낮은 경우

```
bool feasible(int idx, int weight, int value){
    // 남은 물건을 전부 선택하더라도 Max를 이길 수 없는 경우
    int remain = 0;
    for(int i=idx; i<N; i++) remain += V[i];
    if(value + remain <= Max) return false;

    // 가성비(Vi / Wi)가 가장 좋은 물건으로
    // 남은 공간을 모두 채워도 Max를 이길 수 없는 경우
    int mx = 0;
    for(int i=idx; i<N; i++){
        // now = ceil(V[i] / W[i])
        int now = (V[i] + W[i] - 1) / W[i];
        mx = max(mx, now);
    }
    if(value + mx * (K - weight) <= Max) return false;

    return true;
}

void knapsack(int idx, int weight, int value){
    if(idx == N){ Max = max(Max, value); return; }
    if(!feasible(idx, weight, value)) return;
    knapsack(idx+1, weight, value);
    if(weight + W[idx] <= K) knapsack(idx+1, weight+W[idx], value+V[idx]);
}
```

백트래킹과 분기 한정

0/1 배낭 문제

- 모든 해를 탐색할 필요는 없지 않을까?
 - ex) 남은 모든 물건을 선택하더라도 지금까지 구한 것보다 가격이 낮은 경우
 - ex) 남은 물건 중 가성비가 제일 좋은 물건으로 가득 채워도 지금까지 구한 것보다 가격이 낮은 경우
- 조금 더 생각해 보면...
 - 탐색 초반에 충분히 좋은 답을 찾으면, 이후에 탐색 범위를 많이 줄일 수 있음
 - 탐색 순서를 적당히 조정하면(ex. 가성비가 좋은 것부터 본다면) 좋지 않을까?
 - 탐색 범위를 줄일 수 있는 방법을 고민해 보자!

요약

- 재귀 함수
 - 자신을 되부르는 함수
 - 기본 경우(base case)가 잘 정의되어 있어야 함
 - 수학적 귀납법(증명법)과 밀접한 관련이 있음
- 백트래킹
 - 문제의 답을 점진적으로 찾아나가는 방법
 - 깊이 우선 탐색 방법으로 만족하는 해를 찾거나, 모든 해를 찾는 방법
 - 탐색 중 길을 잘못든 것을 알았을 때는 과감히 포기하고 대안을 시도하는 방법
- 분기 한정
 - 가장 좋은 해를 찾는 최적화 문제에 효과적인 탐색 알고리즘
 - 끝까지 가 보지 않고 부분해 중 최적해 가능성이 없는 것을 미리 가지치기하는 방법
 - 가지치기에 한정 함수(bounding function)라는 경험적 방법을 사용함