

im4u 봄방학 캠프

DAY 2; Discrete Optimization Problems Part I

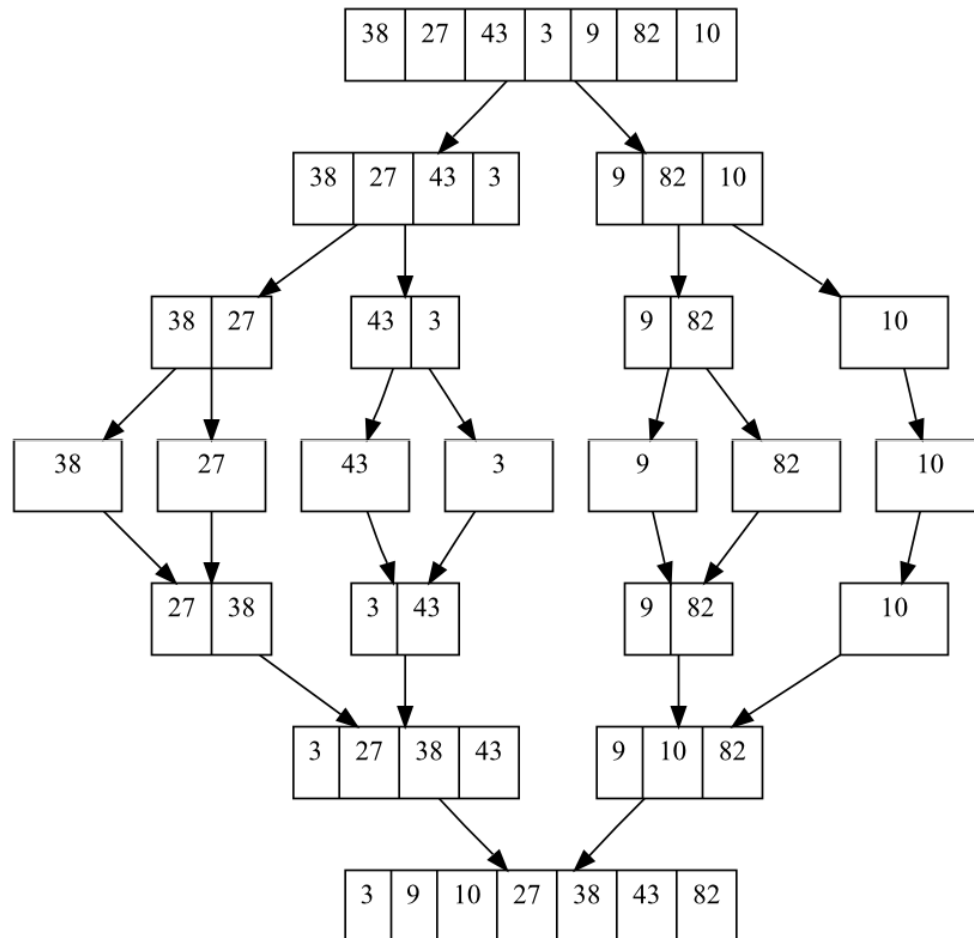
구종만

jongman@gmail.com

오늘 할 얘기

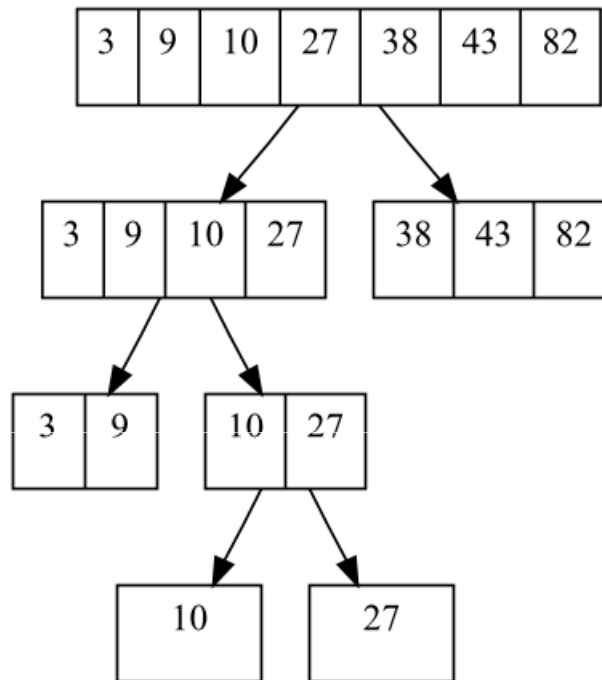
- 알고리즘 디자인 패러다임 공부하기
- 이산 최적화 문제를 대상으로 디자인 패러다임 배우기 Part I
 - 전체 탐색 (exhaustive search)
 - 분할 정복 (divide-and-conquer)
- .. With many many sample problems.

병합 정렬 (Merge Sort)



(Image from Wikipedia)

이분 검색 (Binary Search)



- 정렬된 배열에서 27 찾기

‘패턴’ 이 보이나요?

- 병합 정렬과 이분 검색
 - 문제를 해결하기 위해 입력을 두 개의 작은 입력으로 분할
- 디자인 패러다임 (paradigm): 알고리즘이 문제를 풀기 위해 취하는 전략의 형태
 - 전략을 공부하는 것은 새로운 알고리즘을 개발하는 능력을 쌓기 위한 좋은 방법이 된다

패러다임 (paradigm)

- “어떤 한 시대 사람들의 견해나 사고를 지배하고 있는 이론적 틀이나 개념의 집합체.”
 - Definition a la Thomas Kuhn
- 여기에서는, 일종의 ‘설계 방식’ 정도로 이해하면 좋습니다.
- 알고리즘의 분류 수단으로 볼 수도 있구요.

디자인 패러다임 공부하기

- 결론부터 말하자면: 연습이 킹왕짱
 - 수 많은 예제 문제를 풀다 보면, 일종의 패턴을 알아내는 직관이 발달하게 되는데..

최적화 문제들 (Optimization Problems)

- 여러 개의 답 (solution) 이 있는 문제
- 구성 요소
 - 가능한 답의 집합들 (feasible solutions)
 - 답의 적합도 기준 (measure)
- 여러 개의 답 중 가장 적합도가 높은 답을 찾아내는 문제!

Assignment Problem

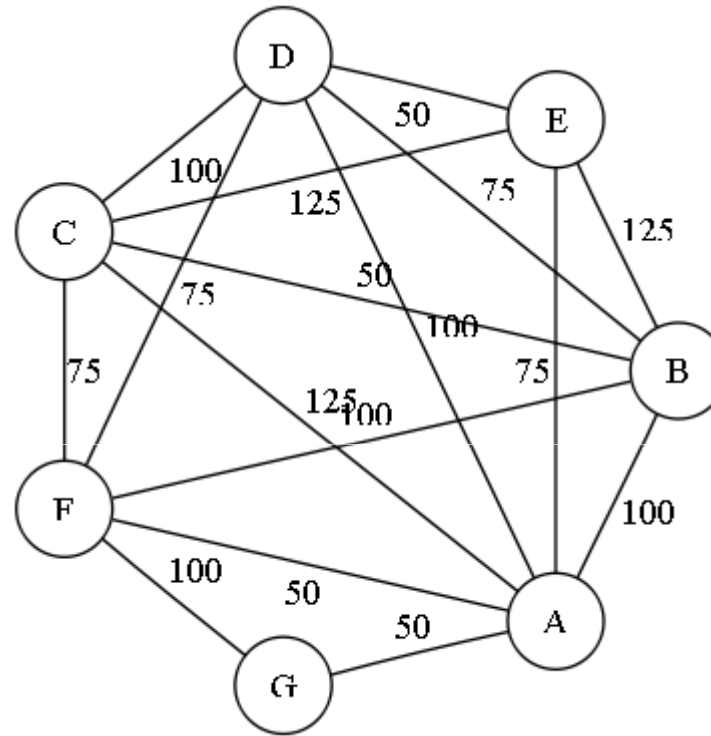
- 2008 컴바라기-SM엔터 4:4 미팅

	연희	태연	윤아	아라
영정	82	42	88	56
재현	92	64	94	84
재영	50	91	64	47
재두	80	83	89	85

- 컴바라기 4인의 호감도 합을 최대화하는 파트너 배정 방법은?

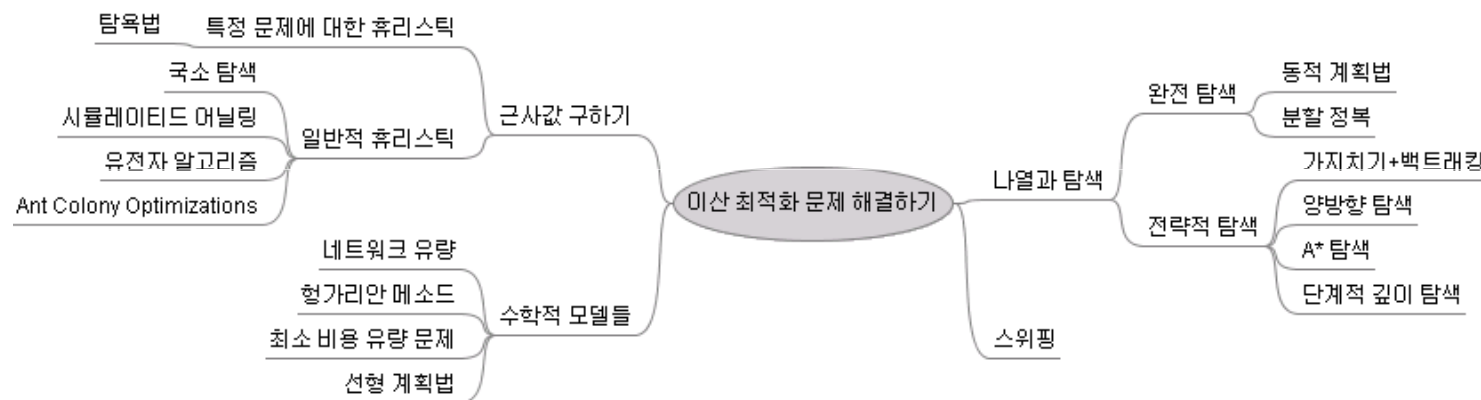
(사용된 이름은 모두 가명입니다)

Traveling Salesman Problem

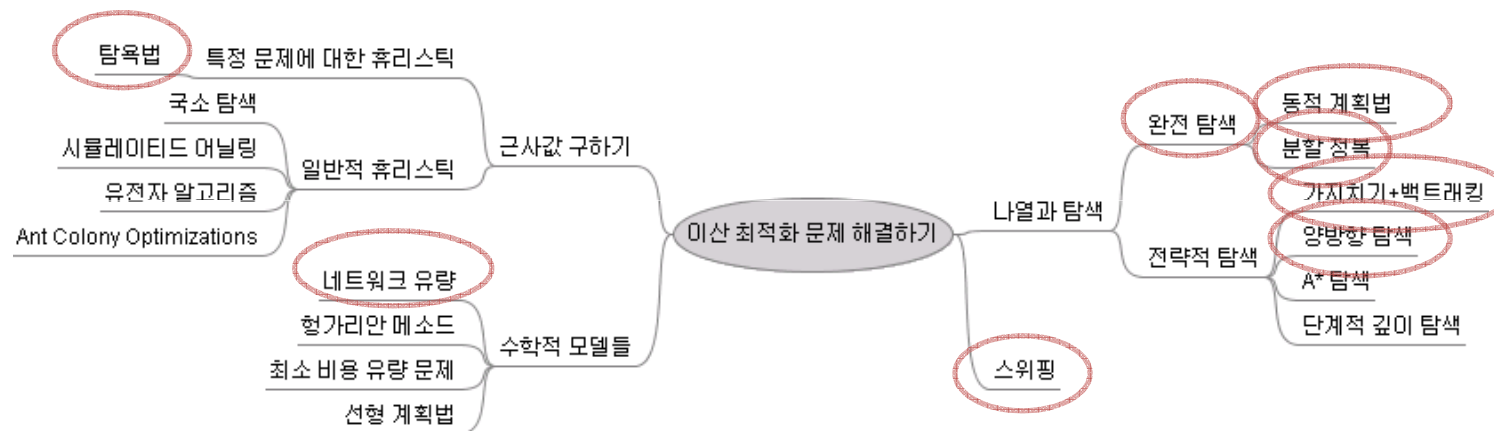


- 모든 점을 한 번씩 방문하는 가장 짧은 경로는?

최적화 문제를 해결하는 다양한 방법들



최적화 문제를 해결하는 다양한 방법들



완전 탐색 (Exhaustive Search)

- 모든 답을 다 만들어 보는 거죠!
- 입력의 크기가 작거나, 답의 개수가 작을 경우 가장 먼저 떠올라야 하는 해법
 - 답의 개수의 상한을 예측하는 것이 가장 어려운 부분
 - 답의 구조에 대한 “직관” 을 필요로 한다
 - 가끔은 작은 케이스를 풀어봄으로써 직관을 얻을 수도 있다
- 거의 항상 재귀호출로 구현된다!
- `next_permutation()` 을 쓸 때도 있음

적나라한 완전 탐색의 예

- 세계 최악(?) 의 정렬 알고리즘:
 - 입력받은 수열의 모든 배열 (permutation) 을 구하고, 그 중에 정렬된 배열을 반환한다
- $O(n!)$
- 물론 죽을 때까지 쓸 일은 없겠지만, 대충 무슨 뜻인지 이해하시라고.. ^^;

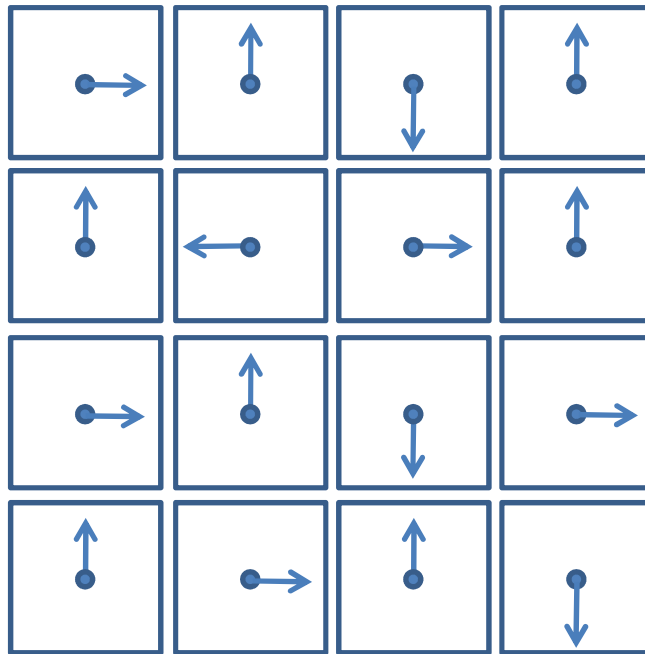
The Lucky String

- 같은 글자가 인접해 있지 않은 문자열을 행운의 문자열이라고 하자.
- “abhazpqr~~x~~” 를 재배열해서 얻을 수 있는 행운의 문자열의 수는?
- (입력 문자열의 크기 ≤ 10)

“Can We Brute-Force It?”

- n 개의 서로 다른 문자를 재배열해서 얻을 수 있는 문자열의 수: $n!$
 - 답은 언제나 $n!$ 보다 작다.
- $10! = 3,628,800$
- $n!$ 개의 순서를 모두 시도하고 이 중 Lucky String 의 수를 센다. \Rightarrow Solved!

Synchronizing Clocks



스위치 번호	시계 번호
0	0,1,2
1	3,7,9,11
2	4,10,14,15
3	0,4,5,6,7
4	6,7,8,10,12
5	0,2,12,14,15
6	3,14,15
7	4,5,7,14,15
8	1,2,3,4,5
9	3,4,5,9,13

- 16개의 시계: 12시, 3시, 6시, 9시만 가리킨다
- 10개의 스위치: 연결된 시계를 3시간씩 전진시킨다

“Can We Brute-Force It?”

- 완전 탐색에 의한 아이디어:
 - 16개의 시계가 가질 수 있는 모든 상태를 완전탐색하면 어떨까?
- .. 어떻게 할 지는 그렇다 치더라도, 모든 상태의 수는 전부 몇 개일까?

Synchronizing Clocks

- 한 개의 시계에는 4개의 상태가 있음
- 16개의 시계가 있으므로, 전체 상태의 수는

$$4^{16} = 2^{32} = 4,294,967,296$$

- 당연히 계산 불가능한 수 ! 어떻게 해야 하나?

Synchronizing Clocks

- Key Observation 1

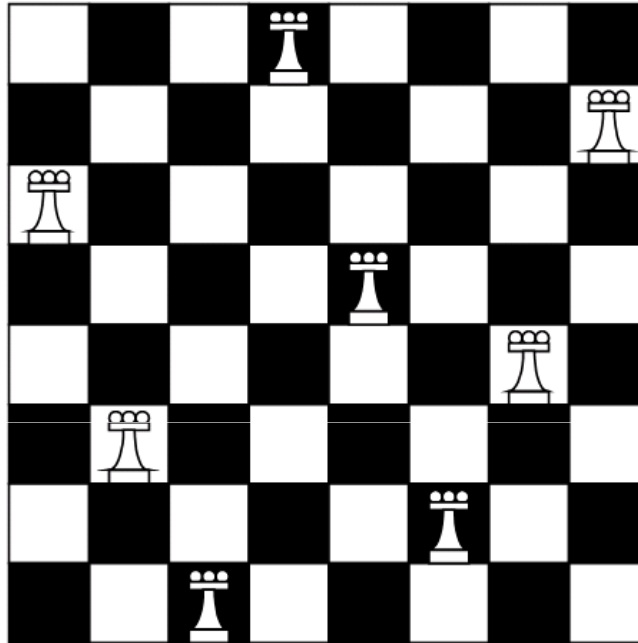
- 스위치를 누르는 순서는 중요하지 않다. 회수만이 중요하다. (*)

- Key Observation 2

- 한 스위치를 4번 누르면, 한 번도 누르지 않은 것과 다를 바가 없다.
- 따라서, 0번, 1번, 2번, 3번만이 의미가 있다.

- 결국, 스위치를 누르는 방법의 수는 $4^{10} = 1,048,576$

N-Queen



- $N \times N$ 크기의 체스판 위에 N 개의 퀸을 놓되, 서로 공격할 수 없도록 하고 싶다. 이런 경우의 수는?
- ($N \leq 12$)

“Can We Brute-Force It?”

- 답의 상한에 대한 첫 번째 시도: n^n
 - $12^{12} \approx 2^{42}$... 별로 도움이 안 된다...
- 수학적으로 답의 수를 세는 것은 꽤나 어렵다
 - 어쨌든 가우스도 한 번 틀린 문제라고 하니까요.

Smart Ideas Are Plan B

- 가장 간단한 아이디어에서 시작하자:
 - $n < 6$ 정도까지는 세 볼 만 한 듯 하니, 프로그램을 짜서 실제로 어떻게 답의 개수가 어떻게 증가하는지를 보자
- 간단한 프로그램을 짜서 아이디어를 검증하자.
- 실제로 해 보면, 1, 0, 0, 2, 10, 4, 40, 92, 352, 724, 2680, ..
- 수학적으로 잘 검증되고, ‘멋있는’ 모델들은 나중에 미뤄두자.

MatchCounting

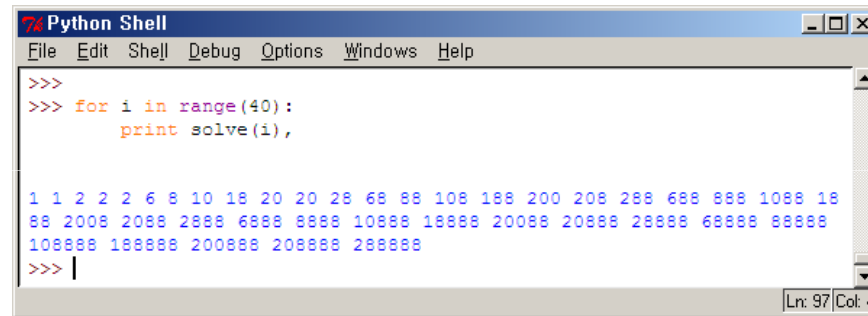
- $N (\leq 128)$ 개의 성냥개비가 있다.



- 이 성냥개비들로 만들 수 없는 가장 작은 양의 정수는?
 - 예) $N=5$ 이면 6 은 만들 수 없다
 - 예) $N=9$ 이면 20 은 만들 수 없다

“Can We Brute-Force It?”

- 수학적으로 해결하기란 까다롭다
- $n \leq 40$ 정도까지만 무식하게 돌려볼까?



```
Python Shell
File Edit Shell Debug Options Windows Help
>>>
>>> for i in range(40):
>>>     print solve(i),
1 1 2 2 2 6 8 10 18 20 20 28 68 88 108 188 200 208 288 688 888 1088 18
88 2008 2088 2888 6888 8888 10888 18888 20088 20888 28888 68888 88888
108888 188888 200888 208888 288888
>>> |
```

- 거의 대부분이 8 로 구성되어 있네?
- Key Observation: 888 은 만들 수 있고, 8888 은 만들 수 없으면, 답은 4자리.

Divide & Conquer

- 알고리즘 디자인 패러다임의 대표격
- 삼국지에서도 나오니까 말 다했죠?
- 병합 정렬과 이분 검색이 분할 정복의 대표적 예
- 두 가지 목적
 - 얼핏 보기에 오래 걸리는 문제를 빨리 해결
 - 재귀적으로 정의되는 문제들을 해결

Fast Matrix Exponentiation

- Given a 100x100 matrix A , compute A^n .
- Simple, but with $n \leq 100,000$?
- 행렬 곱셈을 $O(n^3)$ 으로 한다고 하면, 곱셈 1000 번도 간신히 하겠는 걸..
 - Strassen 알고리즘을 써도 달라지는 건 별로 없다

Fast Matrix Exponentiation

- Key Observation:

- $A^{2n} = A^n \cdot A^n$
- $A^{2n+1} = A \cdot A^{2n}$

```
Matrix pow(const Matrix& A, int n) {  
    if(n == 0) return identity();  
    if(n % 2) return A * pow(A, n-1);  
    Matrix B = pow(A, n/2);  
    return B*B;  
}
```

- 최대 $O(\lg n)$ 번의 곱셈! 가분하다..

Number of Passwords

- 어떤 시스템의 패스워드는 A 글자 이상, B 글자 이하로 구성되어야 하며, n 종류의 문자를 사용할 수 있다.
 - 예) $A = 4, B = 4, n = 10$ 이면 0000 ~ 9999
 - 예) $A = 4, B = 8, n = 36$ 이면 알파벳 소문자 + 숫자
- Given $A, B \leq 2^{32}$ and $n \leq 128$, return the number of possible passwords (mod 1000000007)

Number of Passwords

- 실제 답의 개수를 구하기란 매우 쉽다

$$f(n, m) = n + n^2 + n^3 + \cdots + n^m$$

$$S = f(n, B) - f(n, A - 1)$$

- $f(n, m)$ 을 계산하는 데 $O(m)$ 시간이 걸리면 말짱 황...
- 어떻게 풀까요?

Number of Passwords

- 또 다시, 절반으로 나눕니다

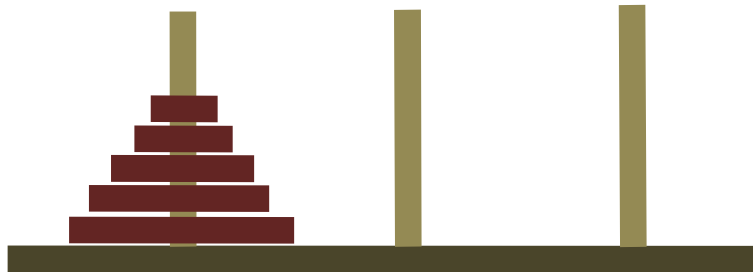
$$\begin{aligned}f(n,8) &= n + n^2 + n^3 + n^4 + n^5 + n^6 + n^7 + n^8 \\&= (n + n^2 + n^3 + n^4) + (n^5 + n^6 + n^7 + n^8) \\&= (n + n^2 + n^3 + n^4) + n^4(n^1 + n^2 + n^3 + n^4) \\&= (n + n^2 + n^3 + n^4)(1 + n^4)\end{aligned}$$

- 따라서,

$$f(n,2a+1) = f(n,2a) + n^{2a+1}$$

$$f(n,2a) = f(n,a) \cdot (1 + n^a)$$

하노이의 탑

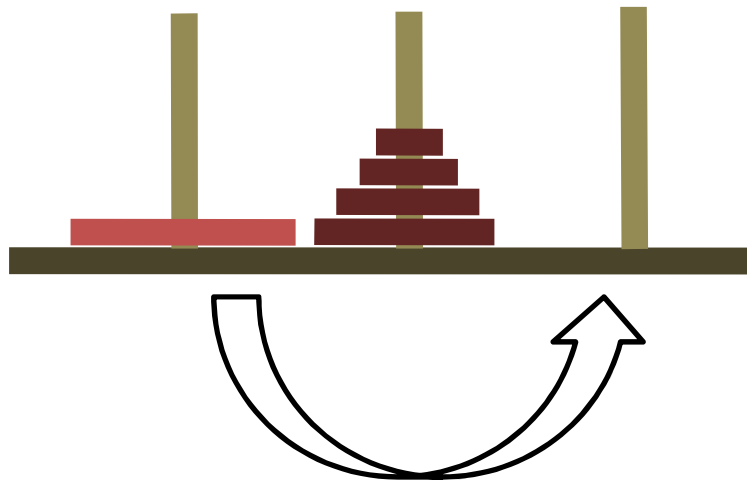


```
class HanoiTowers {  
public:  
    int getDiscCount() const;  
    int moveDisc(int fromTower, int toTower);  
};  
  
void solveHanoi(HanoiTowers& tower) {  
    // solve here  
}
```

- 어떻게 옮길까?

하노이의 탑

- 가장 큰 원판과 나머지 $n-1$ 개의 원판으로 나눠서 생각해 보자
 - 가장 큰 원판이 움직일 때는, 다른 $n-1$ 개의 원판은 다른 하나에 순서대로 모여 있어야 한다



하노이의 탑

- 재귀호출을 이용해 $n-1$ 개와 마지막 한 개로 나눈다

```
// move top n discs from A to B
void solveHanoi(HanoiTowers& tower, int A = 0, int B = 2, int n = -1) {
    if(n == 0) return;
    if(n == -1) n = tower.getDiscCount();
    int C = (0+1+2) - A - B;
    solveHanoi(tower, A, C, n-1); // move n-1 discs to a tem tower
    tower.moveDisc(A, B); // move the largest disc
    solveHanoi(tower, C, B, n-1); // move n-1 discs to the target
tower
}
```

Lessons Learned

- 이산 최적화 문제를 푸는 방법들
- 완전 탐색
 - 완전 탐색으로 풀 수 있는지를 판단하기가 어렵다
 - 가장 간단한 방법으로 시작해서 문제에 대한 직관을 얻기
- 분할 정복
 - 익숙해지면 어떻게 나눠야 할 지 패턴이 보이기 시작