완전탐색

모든 경우의 수를 시도하여

답을 찾는 알고리즘

무식한 힘 # brute force algorithm

완전탐색 기법의 종류

01 Brute Force

02 비트 마스크

03 순열

04 BFS, DFS

01 Brute Force

for문과 if문을 이용하여 처음부터 찾을 때까지 탐색해보는 기법

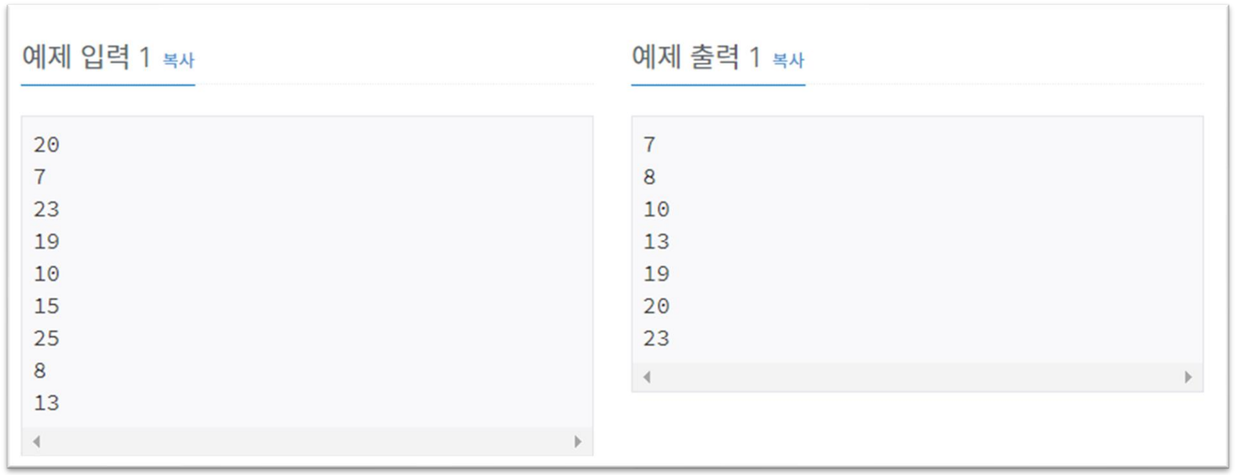
무엇이 좋은가??

1. 100% 답을 찾을 수 있다.
2. 코드 구현이 쉽다.

백준 2309번

아홉 난쟁이 중 진짜 일곱 난쟁이를 찾는 문제!

일곱 난쟁이의 키를 모두 합하면 100이다!



예제 입력 1 복사

```
20
7
23
19
10
15
25
8
13
```

예제 출력 1 복사

```
7
8
10
13
19
20
23
```

9C7 = 36 (가지) 경우의 수!

(20 ,7, 23, 19, 10, 15, 25 ,10)…

예제 입력 1 복사

```
20 → height[0]
7
23
19
10
15
25
8
13
```

```
    for(i=0; i<8; i++)
    {
        for(j=i+1; j<9; j++)
        {
            find=sum-(height[i]+height[j]);
            if(find==100)
            {
                flag=1;
                height[i]=-1;
                height[j]=-1;
                break;
            }
        }
        if(flag) break;
    }
```

02 비트 마스크

정수를 이진수로 표현하고, 비트 연산을 통해 문제를 해결해 나가는 기법

부분집합을 정수로 나타낼 수 있다!

집합 A = {1, 2, 3, 4, 5}

<A의 부분집합>

{1}

{2, 4}

{1, 2, 4}

{1, 2, 3, 4}

...

부분집합을 정수로 나타낼 수 있다!

집합 A = {1, 2, 3, 4, 5}

<A의 부분집합>

{1} → 00001 → 1

{2, 4} → 01010 → 2 + 8 = 10

{1, 2, 4} → 01011 → 1 + 2 + 8 = 11

{1, 2, 3, 4} → 01111 → 1 + 2 + 4 + 8 = 15

…

삽입

A의 부분집합 = {1, 2, 4} → 01011

3을 추가하려면? OR 연산과 시프트 연산 이용!

01011 | 1 << 3

삭제

3을 삭제하려면? AND 연산과 NOT 연산 이용!

11101 & ~1 << 3 (~1 >> 3은 10111)

...

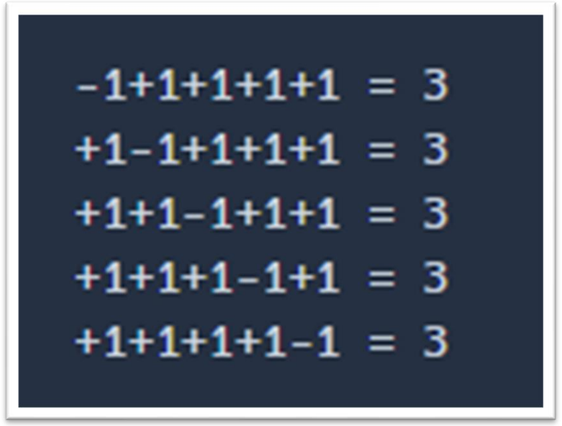
무엇이 좋은가??

1. 비트연산은 O(1)로 속도가 빠르다.
2. 코드가 간결해진다.
3. 적은 메모리를 사용한다.

프로그래머스 타겟 넘버

주어진 다섯개의 숫자들을 적절히 더하거나 빼서 타겟 넘버를 만드는 방법의 수를 구하는 문제!

[1, 1, 1, 1, 1] / 3



```cpp
#include <string>
#include <vector>

using namespace std;

int solution(vector<int> numbers, int target) {
    int answer = 0;

    for(int i=1; i<(1<<numbers.size()); i++) {
        int temp=0;
        for(int k=0; k<numbers.size(); k++) {
            if(i & (1 << k)) temp+=numbers[k];
            else temp+=(-1)*numbers[k];
        }
        if(temp==target) answer+=1;
    }
    return answer;
}

```

[1, 1, 1, 1, 1]

[+1, -1, -1, +1, +1] → 11001

[-1, -1, -1, +1, +1] → 11000

[-1, -1, -1, -1, +1] → 10000

…

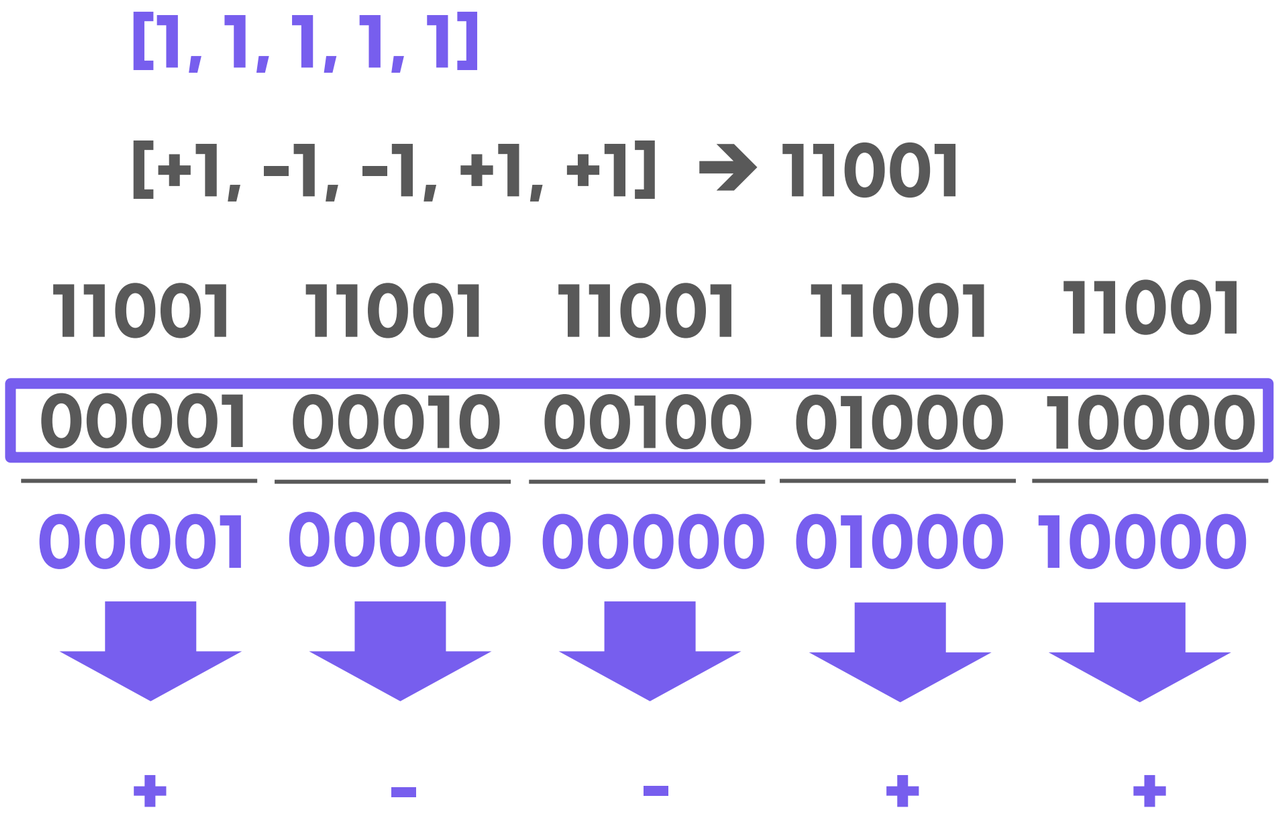
32개의 이진수로 표현 가능!

```cpp
#include <string>
#include <vector>

using namespace std;

int solution(vector<int> numbers, int target) {
    int answer = 0;

    for(int i=1; i<(1<<numbers.size()); i++) {
        int temp=0;
        for(int k=0; k<numbers.size(); k++) {
            if(i & (1 << k)) temp+=numbers[k];
            else temp+=(-1)*numbers[k];
        }
        if(temp==target) answer+=1;
    }
    return answer;
}
```



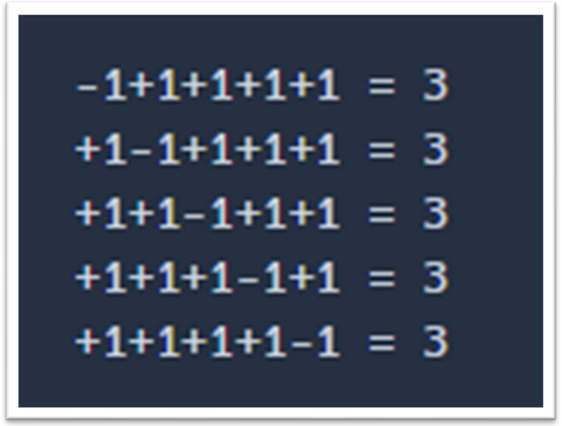
03 순열

**서로 다른 n개의 원소에서 r개를 뽑아 한 줄로 세우는 경우의 수로,
이를 통해 문제를 해결해 나가는 기법**

프로그래머스 타겟 넘버

주어진 다섯개의 숫자들을 적절히 더하거나 빼서 타겟 넘버를 만드는 방법의 수를 구하는 문제!

[1, 1, 1, 1, 1] / 3



[1, 1, 1, 1, 1]

-1, +1, +1, +1, +1	➔	01111
+1, -1, +1, +1, +1	➔	10111
+1, +1, -1, +1, +1	➔	11011
+1, +1, +1, -1, +1	➔	11101
+1, +1, +1, +1, -1	➔	11110

0을 음수로, 1을 양수라 할 때,

2개의 서로 다른 원소(0,1) 중
중복을 허용하여 5개를 뽑는
“중복순열”이라고 볼 수 있다!

```
10  void permutation(vector<int> &numbers, int n, int &target){
11      if(n==numbers.size()){
12          int sum=0;
13
14          for(int i=0;i<numbers.size();i++){
15              if(temp[i]==0){
16                  sum-=numbers[i];
17              }
18              else{
19                  sum+=numbers[i];
20              }
21          }
22
23          if(sum==target)
24              total++;
25          return;
26      }
27
28      for(int i=0;i<=1;i++){
29          temp[n]=i;
30          permutation(numbers, n+1,target);
31      }
32  }
```

0은 음수, 1은 양수를 의미!

permutation 함수를 통해
5개의 문자를 0 또는 1로 채워 넣는다!

0 0 0 0 0

↑

temp[0]

```
10 void permutation(vector<int> &numbers, int n, int &target){
11     if(n==numbers.size()){
12         int sum=0;
13 
14         for(int i=0;i<numbers.size();i++){
15             if(temp[i]==0){
16                 sum-=numbers[i];
17             }
18             else{
19                 sum+=numbers[i];
20             }
21         }
22 
23         if(sum==target)
24             total++;
25         return;
26     }
27 
28     for(int i=0;i<=1;i++){
29         temp[n]=i;
30         permutation(numbers, n+1,target);
31     }
32 }
```

0 0 0 0 0

↑

temp[0]

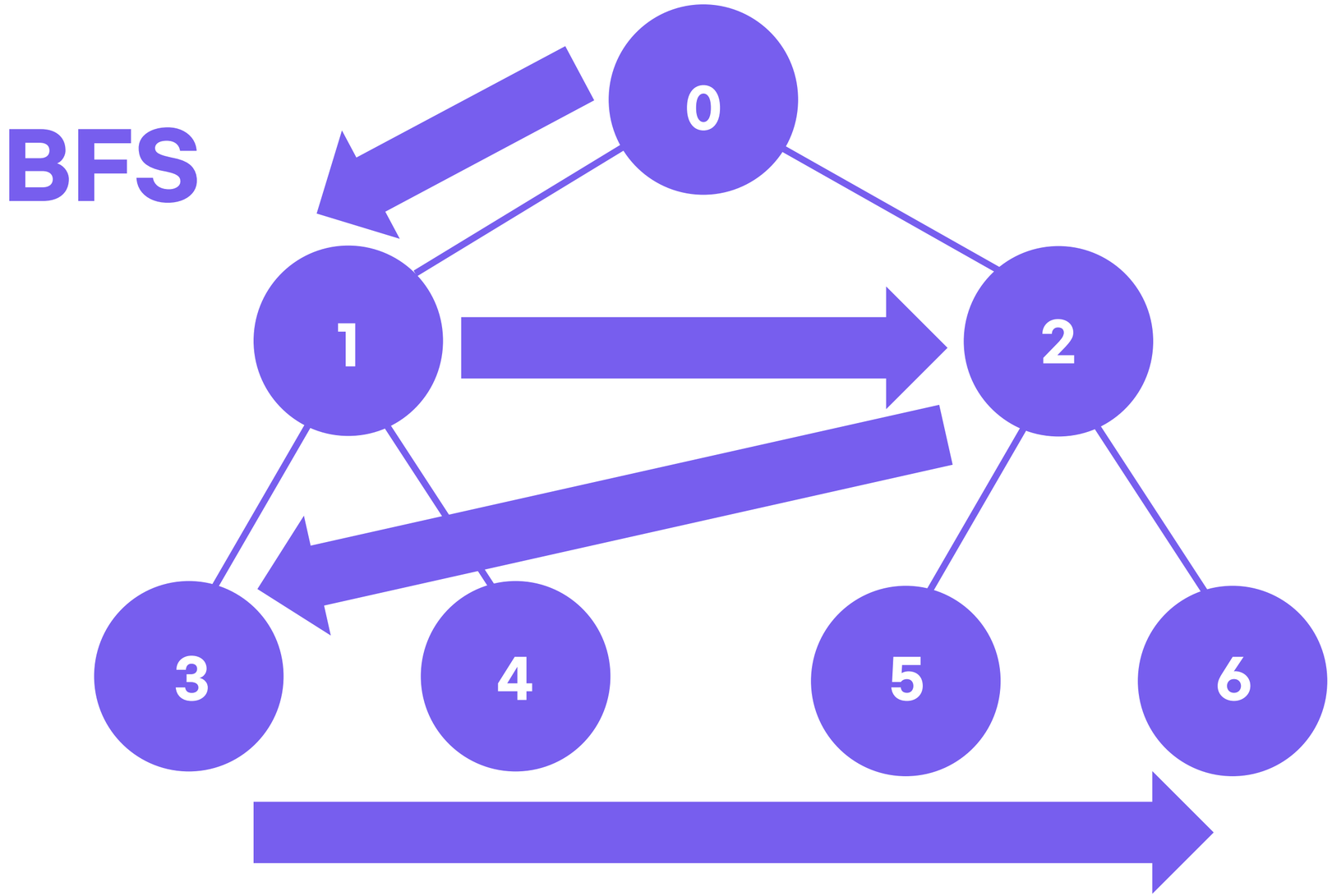
00이면 음수,
1이면 양수로 취급하여 더해준다.

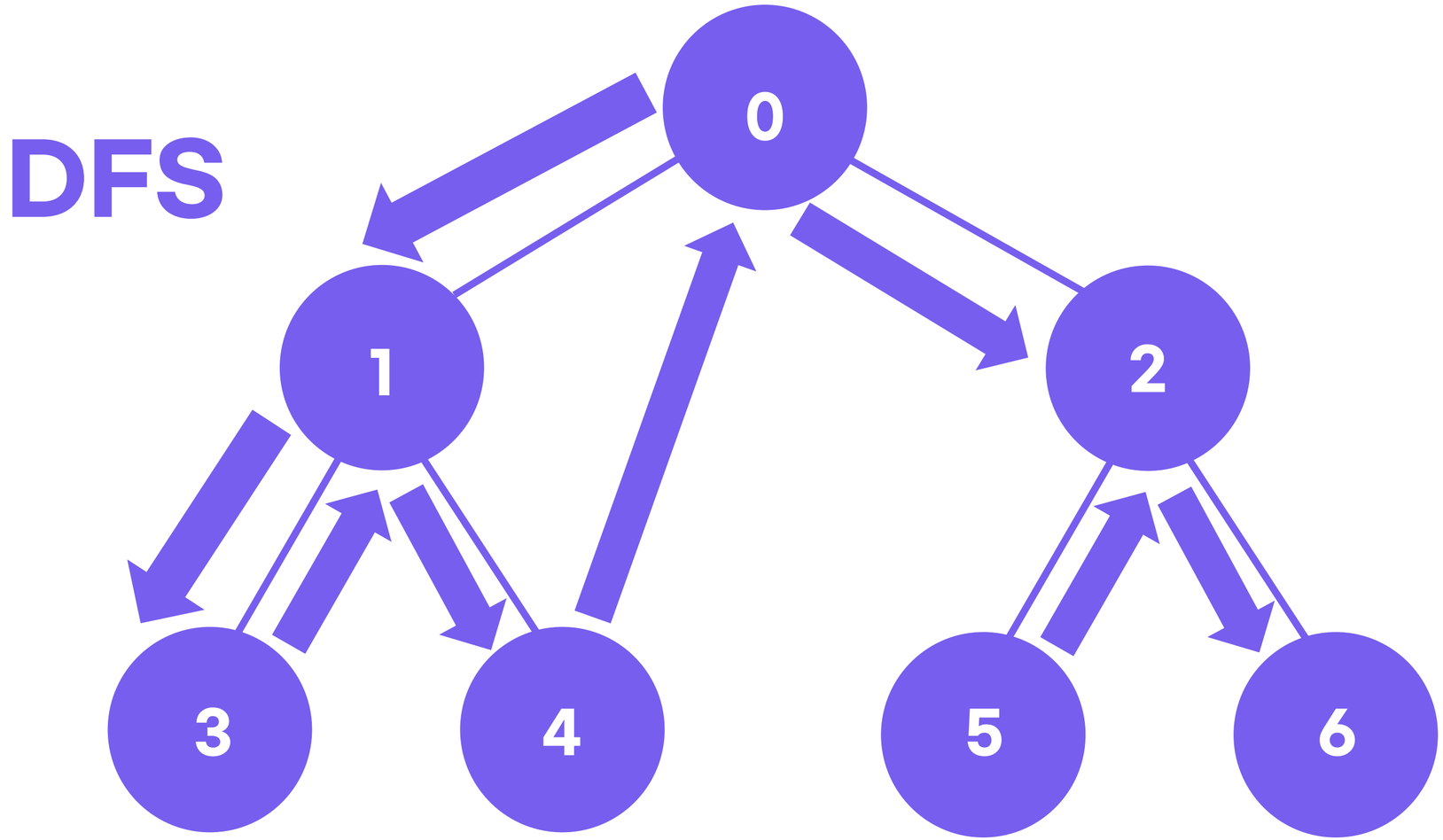
다섯개의 합이 target과 같으면,
방법의 수인 total을 증가해준다.

return 하면 함수를 호출한 곳으로
돌아가고, 이 과정이 반복된다.

04 BFS, DFS

BFS, DFS를 이용하여 문제를 해결해 나가는 기법


BFS
0
1
2
3
4
5
6


DFS
0
1
2
3
4
5
6

BFS

최단 경로를 보장한다.

DFS

찾는 노드가 깊은 단계에 있을 경우 유리하다.

프로그래머스 타겟 넘버

**주어진 다섯개의 숫자들을
적절히 더하거나 빼서
타겟 넘버를 만드는 방법의 수
를 구하는 문제!**

[1, 1, 1, 1, 1] / 3

```
-1+1+1+1+1 = 3
+1-1+1+1+1 = 3
+1+1-1+1+1 = 3
+1+1+1-1+1 = 3
+1+1+1+1-1 = 3
```

```
void DFS(vector<int> &numbers, int &target, int sum, int n) {
    if(n >= numbers.size()) {
        if(sum==target) total++;
        return;
    }

    DFS(numbers, target, sum+numbers[n], n+1);
    DFS(numbers, target, sum-numbers[n], n+1);
}

int solution(vector<int> numbers. int target) {
    int answer=0;

    DFS(numbers, target, numbers[0], 1);
    DFS(numbers, target, -numbers[0], 1);

    answer=total;

    return answer;
}
```

sum = sum+numbers[n]

sum = sum-numbers[n];

+1 +1 +1 +1 +1

+1 -1 +1 -1 -1

...