

## 코드카피

- 코드 전체를 그대로 가져다 쓴 경우
- 특정 부분을 복사해서 변수명만 바꾼 경우
- GPT에게 해당 문제를 풀어달라고 한 뒤 해당 코드를 가져다 쓴 경우

## 코드카피의 예시 1

<https://velog.io/@jmink002/C4949>

```
int main() {
    while(true) {
        std::string input = "";
        std::stack<char> st; /* store bracket */
        //std::stack<char> small; /* small bracket */
        std::getline(std::cin, input);
        bool result = true; /* store result */


        if(input.length() == 1 && input[0] == '.') {
            break;
        }

        for(int i = 0; i < input.length(); i++) {
            if(input[i] == '[' || input[i] == '(') {
                st.push(input[i]);
            }


            if (input[i] == ']') {
                if(!st.empty() && st.top() == '[') {
                    st.pop();
                } else {
                    result = false;
                    break;
                }
            } else if(input[i] == ')') {
                if(!st.empty() && st.top() == '(') {
                    st.pop();
                } else {
                    result = false;
                    break;
                }
            }
        }
    }
}
```

```
+ int main(){
+     while(true) {
+         std::string input="";
+         std::stack<char> s;
+         std::getline(std::cin, input);
+         bool result = true;
+
+         if(input.length()==1&&input[0]=='.' ){
+             break;
+         }
+
+         for(int i=0;i<input.length();i++){
+
+             if(input[i]=='[' || input[i]=='('){
+                 s.push(input[i]);
+             }
+
+             if(input[i]==']'){
+
+                 if(!s.empty()&&s.top()==' '){
+                     s.pop();
+                 }
+
+                 else{
+                     result=false;
+                     break;
+                 }
+             }
+         }
+     }
+ }
```

## 과제 해결 성공사례

 **이민경** 수요일 오후 8:14  
안녕하세요! 문제 2840번을 풀고있는데 테스트 케이스는 전부 맞게 나오는데 백준에 제출을 하면 답이 틀렸다고 나옵니다. 어느 부분에서 틀린걸까요?

3개의 댓글

 **이민경** 수요일 오후 8:14


```
#include <iostream>
#include <vector>

using namespace std;

void wheelOfFortune(vector<char> &v, int iter) {
    int size = v.size();
    for(int p = 0; p<v.size(); p++) {
        cout << v[iter];
        iter = (iter - 1 + size)%size;
    }
}


int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);

    int n, k;
    cin >> n >> k;
    vector<char> v(n, '?');
    int iter = 0;
```

 **[튜터] 유서현** 수요일 오후 9:45  
안녕하세요 민경님! 반례입니다.

4 3  
1 A  
2 B  
1 A

참고하셔서 어떤 부분을 놓치고 계신지 파악하시면 좋을 것 같아요!

 **이민경** 목요일 오전 12:18  
감사합니다!! 해결했습니다!!

질문 채널 이용 & 튜터에게 help 요청!

## 과제 해결 방법

1. 최대한 자신의 힘으로 풀기
2. 위가 어려운 경우 튜터에게 도움을 요청하기
3. 블로그 글을 참고한 경우 출처와 참고한 부분을 표시하고 더 좋은 방법을 튜터와 함께 고민하기
  - 출처와 참고한 부분을 표시하는 방법
    1. 참고한 자료 링크를 주석으로 남긴다
    2. 참고한 부분에 대해 매 줄 주석을 달아 자료를 이해한 상태로 인용했음을 증명한다

### 면제권 관련 재공지

- 결석 한 번에 대해 면제권 사용 가능
  - 면제권은 결석의 경우에만 적용
  - 과제를 제출하지 않을 시 OUT
- OUT을 3회 받으면 제명

# 알튜비튜 브루트포스

오늘은 모든 경우의 수를 탐색하는 브루트포스에 대해 배웁니다.  
또한, 이진수로 모든 것을 표현하는 컴퓨터의 연산 방식을 활용한 비트마asking에 대해서도 배워봅시다.

“4자리 수의 암호로 된 **비밀번호**를 풀어보자.”

-> 0000 ~ 9999 까지 시도해보면 된다.

## 브루트 포스 : 무차별 대입, 완전 탐색

- 해답을 찾기 위해 가능한 모든 경우를 찾는 기법
- 시간 복잡도가 입력 크기에 비례함
- 브루트 포스 기법은 간단하지만, 매우 느리다
- 하지만, 떠올리기 가장 쉬운 방법이므로 문제를 풀 때 가장 먼저 고려해야 하는 방법
- 입력 범위와 시간 복잡도를 잘 고려하여 선택하는 것이 중요



## /<> 2309번 :일곱 난쟁이

### 문제

- 아홉 난쟁이의 키를 알고 있을 때
- 아홉 난쟁이에서 일곱 난쟁이를 골라라
- 단서 ) 일곱 난쟁이의 키의 합이 100

### 제한 사항

- 2초

**가능한 모든 경우에 대해서 키의 합이 100인지를 확인한다.**

- 아홉 난쟁이 중에 일곱 난쟁이를 뽑는 모든 경우를 고려하자
- 합이 100이 되는 경우를 찾아내자  
(\* 문제에서 적어도 하나의 답이 있음이 보장되어 있음)

## 문제를 더 쉽게 바꿔보자

- 아홉 난쟁이 중에서 난쟁이 둘을 뽑는 문제로 변형
- 아홉 난쟁이의 키의 합에서 둘의 키를 제외했을 때 100이 되는 경우
- 아홉 난쟁이 중에서 이 둘을 제외한 난쟁이가 공주가 찾는 일곱 난쟁이다!

시간 복잡도를 계산해봅시다!

## /<> 2231번 : 분해합

### 문제

- $n$ 의 분해합 =  $n$ 과  $n$ 을 이루는 각 자리수의 합
- $n$ 의 생성자 = 분해합이  $n$ 인 어떤 자연수  $m$
- (예) 245의 분해합 =  $245+2+4+5 = 256 \rightarrow 245$ 는 256의 생성자
- $n$ 의 가장 작은 생성자 구하는 문제

### 제한 사항

- $n$ 의 범위는  $1 \leq n \leq 1,000,000$

## 외판원 순회(Traveling Salesman Problem, TSP)

- 여러 개의 도시들이 주어졌을 때, 한 도시에서 출발하여 모든 도시들을 거쳐 다시 출발 도시로 최소 비용으로 돌아올 수 있는 이동 순서를 구하는 문제
- 대표적인 NP-난해 문제
- $n$ 개의 도시가 있을 때 경우의 수는?  $n! \Rightarrow O(n!)$   
 $n=16$ 이면  $n! = 20,922,789,888,000$

# 이런 문제는 어떻게 풀까요?

## 외판원 순회(Traveling Salesman Problem, TSP)

- 20,922,789,888,000개나 되는 방문 순서를 vector에 저장할 경우  
=> 상당한 양의 메모리 필요

하지만 int의 각 비트를 각 도시의 방문 여부를 저장하는 데 사용한다면?

32bit만으로 모든 도시의 방문 여부 저장 가능!

## 비트마스킹(BitMask)

- 비트 필드 각각을 하나의 원소처럼 사용하는 기법
- C++에서 int는 32비트의 공간을 사용 → int 한 개로 32개의 원소 저장 가능
- 산술 연산보다 빠른 연산 수행 가능
- 1개의 int 만으로 32개의 원소를 저장하는 효과
- 효율적인 메모리 사용
- DP, 백트래킹 등 방문 여부를 저장해야 하는 알고리즘과 함께 사용 ex) 외판원순회

## AND (&)

```
int ans = a & b;
```

1 1 0 0 1 1 0 1 0 1 1 0

1 0 0 1 0 1 1 0 1 0 1 0



1 0 0 0 0 1 0 0 0 0 1 0



## OR (|)

```
int ans = a | b;
```

1 1 0 0 1 1 0 1 0 1 1 0

1 0 0 1 0 1 1 0 1 0 1 0



1 1 0 1 1 1 1 1 1 1 1 0

## XOR (^)

```
int ans = a ^ b;
```

1 1 0 0 1 1 0 1 0 1 1 0

1 0 0 1 0 1 1 0 1 0 1 0



0 1 0 1 1 0 1 1 1 1 0 0

## NOT (~)

```
int ans = ~a;
```

1 1 0 0 1 1 0 1 0 1 1 0



0 0 1 1 0 0 1 0 1 0 0 1

## LEFT SHIFT (<<)

```
int ans = a << 3;
```

0 0 0 0 0 0 1 0 0 1 0 1



0 0 0 1 0 0 1 0 1 0 0 0

## LEFT SHIFT (<<)

```
int ans = a << 3;
```

1 1 0 0 1 1 0 1 0 1 1 0



0 1 1 0 1 0 1 1 0 0 0 0

## RIGHT SHIFT (>>)

```
int ans = a >> 3;
```

0 1 0 0 1 1 0 1 0 0 0 0



0 0 0 0 1 0 0 1 1 0 1 0

# RIGHT SHIFT (>>)

```
int ans = a >> 3;
```

1 1 0 0 1 1 0 1 0 1 1 0



1 1 1 1 1 0 0 1 1 0 1 0

## /<> 28239번 : 배고파(Easy)

### 문제

- $n$ 개의 줄에 양의 정수  $m$ 이 하나씩 주어졌을 때,  $2^x + 2^y = m$ 을 만족시키는  $(x, y)$  쌍을 찾는 문제

### 입력

- $1 \leq m \leq 10^{18}$

### 제한 사항

- 주어지는 모든  $m$ 에 대해 가능한  $x \leq y$ 인  $(x, y)$  순서쌍이 정확히 하나 존재함이 보장된다.



## 예제 입력 1

```
2
10
3
```

## 예제 입력 2

```
2
1052672
38654705664
```

## 예제 출력 1

```
1 3
0 1
```

## 예제 출력 2

```
12 20
32 35
```

# 비트마스킹을 이용해 풀어봅시다

## 예제 입력 1

```
2
10
3
```

- 테스트 케이스1:  $10 = 2^1 + 2^3 = 2 + 8$
- $10 = (1010)_2$
- $2 = (0010)_2$
- $8 = (1000)_2$

## 예제 출력 1

```
1 3
0 1
```

- 테스트 케이스2:  $3 = 2^0 + 2^1 = 1 + 2$
- $3 = (0011)_2$
- $1 = (0001)_2$
- $2 = (0010)_2$

# 비트마스킹은 어떻게 활용될까요?

Set{1, 2, 5, 10, 15}

1	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0
15					10					5			2	1	

집합에 속한 원소들을 각 비트필드에 할당하여 각 원소가 집합에 존재하는지 저장

N번째 비트필드가 1이면, 집합에 N이 존재한다고 판단

# 비트마스킹은 어떻게 활용될까요?

$x$   $y$   
{1, 2, 5, 10, 15}, {13}

$x$ : 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0  
15 10 5 2 1

$y$ : 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
13

# 비트마스킹은 어떻게 활용될까요?

$y$   
{13}

$$1 = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)_2$$

$$1 \ll 13 = (0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)_2$$

$$y : 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$$

# 비트마스킹은 어떻게 활용될까요?

$x$   $y$   
 $\{1, 2, 5, 10, 15\} \cup \{13\}$

$x$ : 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0

$y$ : 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0



$x|y$ : 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0

# 비트마스킹은 어떻게 활용될까요?

$\overset{x}{\{1, 2, 5, 10, 15\}} \cup \overset{y}{\{13\}}$

```
int x = 33830;    //(1000010000100110)_2  
int y = 1 << 13;  
int ans = x | y;  // x | (1 << 13)
```

# 비트마스킹은 어떻게 활용될까요?

$$\overset{x}{\{2, 4, 7, 9\}} \cap \overset{y}{\{1, 4, 5, 9\}}$$

$x$ : 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0

$y$ : 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0



$x \& y$ : 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0



# 비트마스킹은 어떻게 활용될까요?

$$\overset{x}{\{2, 4, 7, 9\}} \cup \overset{y}{\{1, 4, 5, 9\}}$$

$x$ : 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0

$y$ : 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0



$x | y$ : 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0

# 비트마스킹은 어떻게 활용될까요?

$$\overset{x}{\{2, 4, 7, 9\}} - \overset{y}{\{1, 4, 5, 9\}}$$

$x$ : 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0

$y$ : 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0



$x \& \sim y$ : 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0

# 비트마스킹은 어떻게 활용될까요?

## 1. 수학, 구현

- 입력 데이터 값 자체의 특성이나 규칙을 활용
- 추천 문제
  - /<> 1094번 : 막대기 - Silver 5
  - /<> 1740번 : 거듭제곱 - Silver 4
  - /<> 17419번 : 비트가 넘쳐흘러 - Silver 4

## 2. 브루트포스, 백트래킹

- 모든 가능한 경우의 수를 탐색하는 알고리즘과 함께 활용
- 비트마스킹을 사용하여 가능한 부분집합을 효율적으로 표현할 수 있음
- 추천 문제
  - /<> 2961번 : 도영이가 만든 맛있는 음식 - Silver 2
  - /<> 15661번 : 링크와 스타트 - Silver 1
  - /<> 1062번 : 가르침 - Gold 4

## 3. 그래프 탐색(BFS 등)

- n번째 노드의 방문 여부를 저장하기 위해 사용
- 여러 개의 조건을 따져야 하는 경우 각 조건의 만족 여부를 저장하기 위해 사용
- 추천 문제
  - /<> 1194번 : 달이 차오른다, 가자. – Gold 1
  - /<> 2234번 : 성곽 – Gold 3
  - /<> 4991번 : 로봇 청소기 – Gold 1

## 4. 다이나믹 프로그래밍(DP)

- 각 조합별 가능한 경우의 수를 DP 배열에 저장
- 각 부분 집합에 새로운 원소가 추가됐을 때의 경우의 수를 DP로 계산
- 추천 문제
  - /<> 2098번 : 외판원 순회 - Gold 1
  - /<> 1562번 : 계단 수 - Gold 1
  - /<> 1014번 : 컨닝 - Platinum 5

## 필수

/<> 1063번 : 킹 - Silver 3

/<> 1476번 : 날짜 계산 - Silver 5

/<> 11723번 : 집합 - Silver 5

## 도전

/<> 14620번 : 꽃길 - Silver 2

/<> 1052번 : 물병 - Gold 5