const에 대한 고찰

programmers ***n으로 표현*** 문제를 풀다가 예상치 못한 구간에서 error가 발생했는데, 그게 뭐냐면

일단 전체 코드는 다음과 같음

```cpp
#include <iostream>
#include <unordered_set> // 중복을 허용하지 않는 set
#include <vector>
#include <string>
using namespace std;
// 프로그래머스 - LV3 - N으로 표현
int solution(int n, int goal_num) {
    // dp[i]는 n을 정확히 i번 사용하여 만들 수 있는 모든 숫자의 집합입니다.
    vector<unordered_set<int>> dp(9);
    dp[1].insert(n);// 맨처음 경우 -> n을 1번 사용해서 만들 수 있는 수는 n밖에 없음
    if(n==goal_num){ // 예를 들어 n=5이고 goal_num =5이면 그냥 그 숫자를 바로 반환하면 되므로 1 리턴
        return 1;
    }
    // 최대 8번까지 시도
    for (int i = 2; i <= 8; i++) {// possible return value is 1~8 else return -1
        for (int j = 1; j < i; j++) {
            for (int x : dp[j]) {
                for (int y : dp[i - j]) {
                    dp[i].insert(x + y);
                    dp[i].insert(x - y);
                    dp[i].insert(x * y);
                    if (y != 0) {
                        dp[i].insert(x / y);
                    }
                }
            }
        }
        // 숫자의 결합
        dp[i].insert(stoi(string(i, '0' + n)));

        // 목표 숫자가 dp[i]에 포함되면 반환
        if (dp[i].find(goal_num) != dp[i].end()) {
            return i;
        }
    }

    // 목표 숫자를 만들 수 없을 경우 -1 반환
    return -1;
}

int main(){
    int n, goal_num;
    cin>>n>>goal_num;
    cout<<solution(n, goal_num)<<endl;
}
```

뭐 딱 보면 별 문제가 없어 보이는 코드임

근데 저기서 내가 원래 하던대로, range_based for loop를

```
for(auto &x : dp[j]){
  for(auto &y : dp[i-j]){

  }
}
```

로 바꾸면 별 이상이 없는데 반면에, 저걸

```
for(int &x : dp[j]){
  for(int &y : dp[i-j]){

  }
}
```

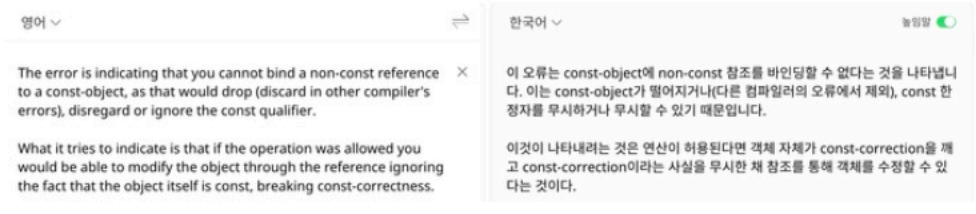
로 바꾸면 error가 뜨는것을 확인할 수 있었다.

error message를 확인해보니

```
/solution0.cpp:16:23: error: binding reference of type 'int' to value of type 'const int'
drops 'const' qualifier
            for (int &x : dp[j]) {
                      ^ ~
```

이렇게 뜨는거임;;

아니 내가 const를 쓴것도 없는데 왜 const 오류가 나는거지? 하고 stackoverflow에서 뒤져보니까



읽어보니까 **const - correction**문제라고 나와있는데, **const-object에 non-const 참조를 바인딩할 수 없대..** 그래서 내가 쓴 코드에 대체 뭐가 const냐고 chatgpt에게 물어봤는데

내가 쓴 dp가 자료형이 vector<unordered_set<int>>dp(9)자나?

근데 찾아보니까 unordered_set 자체가 const 한 성질을 가지고 있대..

그래서 int&y를 하면 안되고, 할거면 const int&y를 하거나 아예 auto를 써서 auto &y를 하래

처음엔 vector에 문제가 있는건가 해서 vector<unordered_set<int>> dp(9); 얘가 size가 fix되어 있어서 const 성질이 생긴건가 했는데 그건 또 아니더라구..

하나하나 다 체크해봤는데

```cpp
#include<iostream>
#include<unordered_set>
#include<vector>
using namespace std;
vector<unordered_set<int>>dp;
// dp를 사이즈free로 설정해두고하면
// 맨첨 dp의 사이즈가 0이라서
int main(){
    dp.push_back({0});
    dp.push_back({1});
    dp.push_back({3});
    // 이렇게 dp 자체의 push_back을 몇개 해주고 (dp자체는 vector니까 push_back)
    dp[0].insert(3);
    dp[1].insert(3);
    dp[1].insert(4);
    dp[2].insert(4);
    dp[2].insert(1);
    // dp내부는 unordered_set이니까 저렇게 insert해주고
    for(int x : dp[2]) cout<<x<<endl;
    // 어쨋든 저 dp[2] 안의 값은 unordered_set이니까 int&x 는 안되는거지
    // unordered_set은 const한 값이니까
    // 그래서 const int &x를 하거나 const int x 를 하거나 int x를 하거나 auto &x를 하거나 그래야댐

    vector<int>v;
    v.push_back(3);v.push_back(2);v.push_back(4);
    for(int & c : v){cout<<c<<endl;}
    // vector는 size free 인 상태에서도 int&c해도 잘 나오고

    unordered_set<int>k;
    k.insert(3);
    k.insert(2);
    for(int  c : k){cout<<c<<endl;}
    // unordered_set은 안되었음

    vector<int>vv(2);
    vv.push_back(1);
    vv.push_back(3);
    for(int &x : vv)cout<<x<<endl;
    // 혹시 vector를 size 있게 하고 출력해봤는데 잘 되더라
}
```

결론

unordered_set은 const한 성질이 있어서 int &x같은건 안되고 const를 붙혀주거나 auto를 써야 reference로 참조가 가능하다

vector 자료형은 상관없다