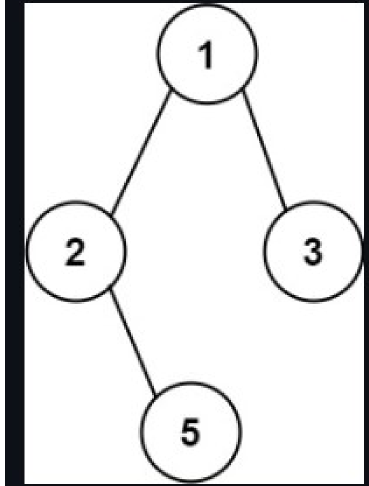
257. Binary Tree Paths

Easy

Given the `root` of a binary tree, return *all root-to-leaf paths in* ***any order***.

A **leaf** is a node with no children.

Example 1:



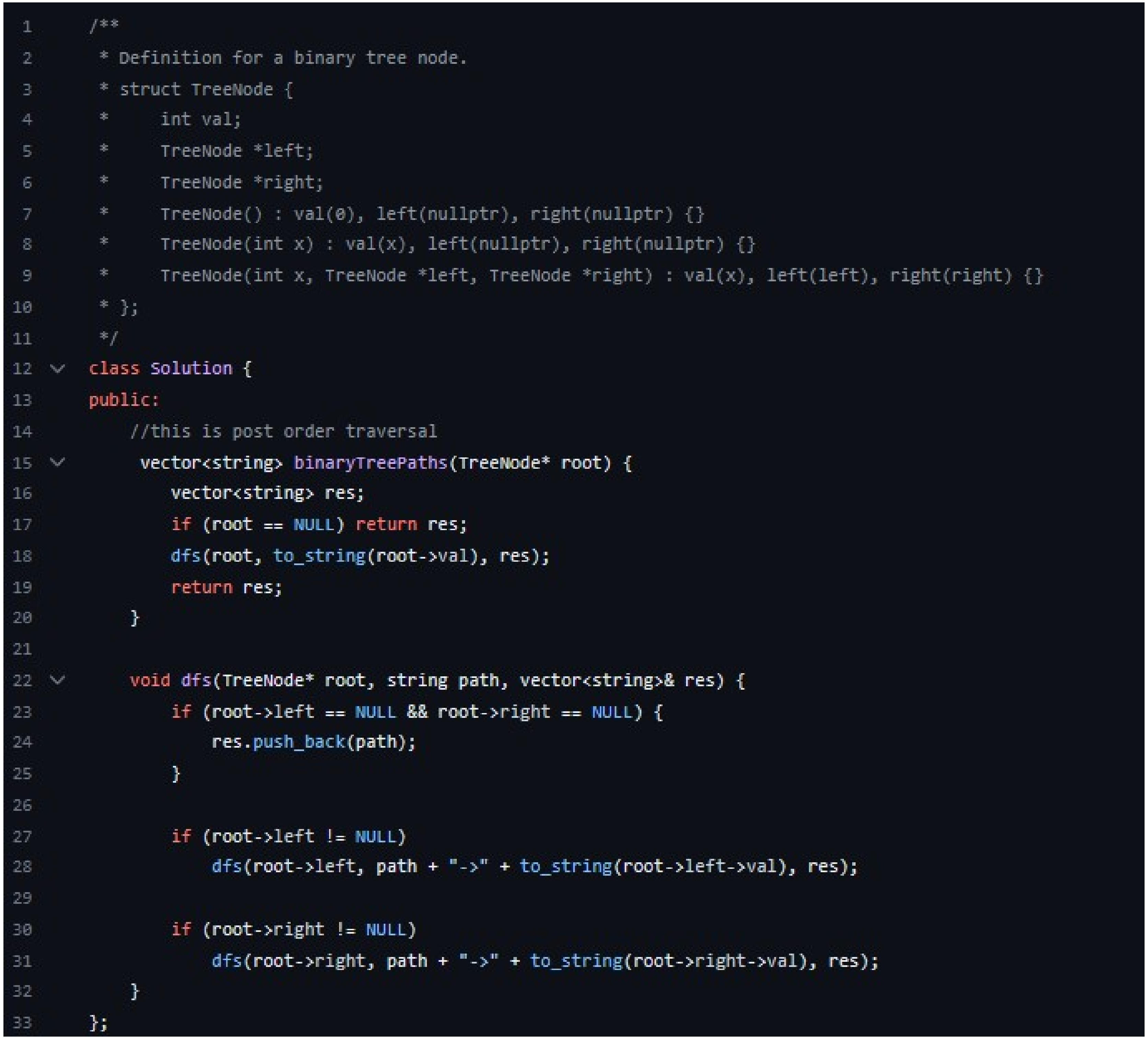
```
Input: root = [1,2,3,null,5]
Output: ["1->2->5","1->3"]
```

Example 2:

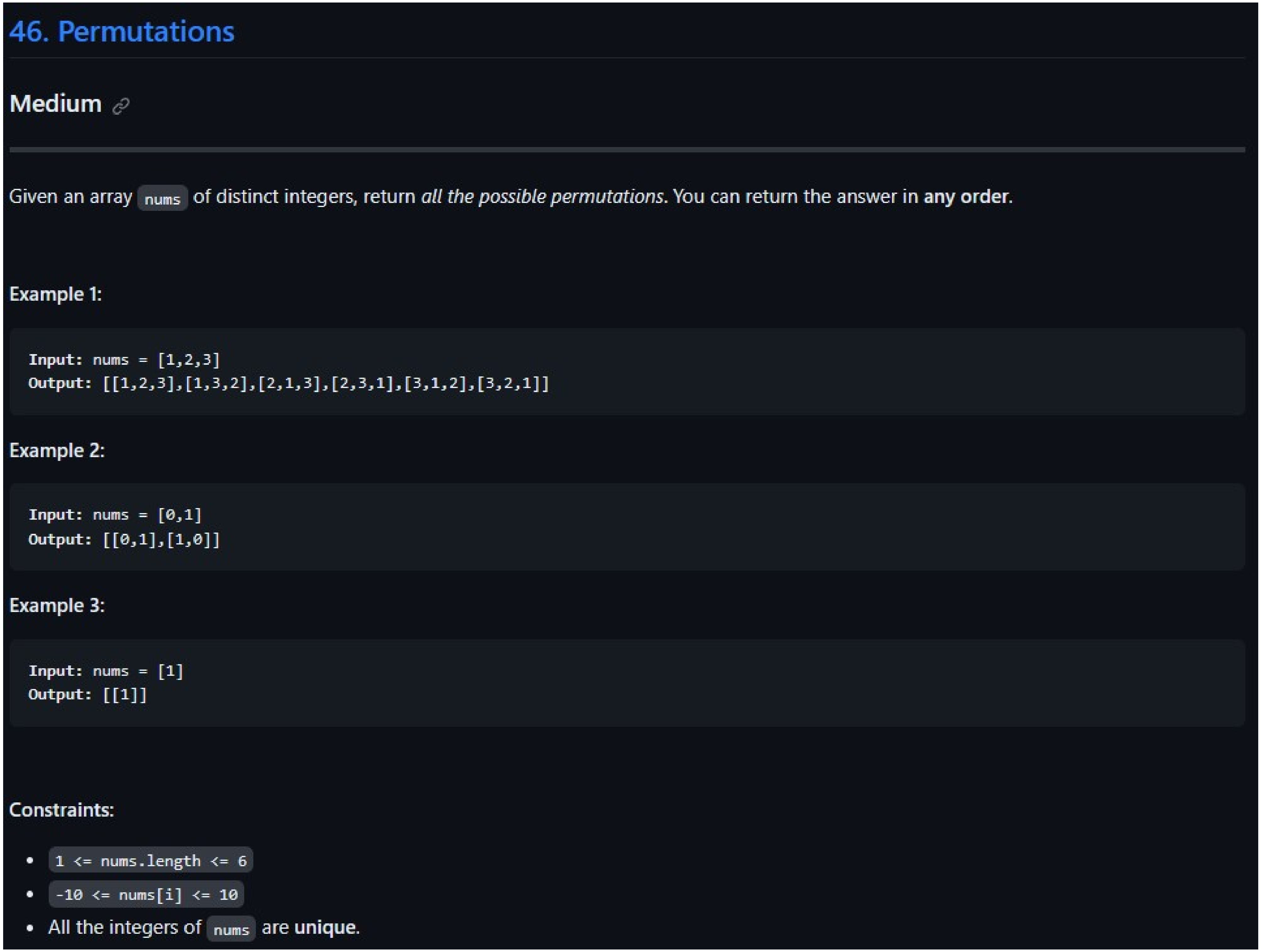
```
Input: root = [1]
Output: ["1"]
```

Constraints:

- The number of nodes in the tree is in the range `[1, 100]`.



```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    //this is post order traversal
     vector<string> binaryTreePaths(TreeNode* root) {
        vector<string> res;
        if (root == NULL) return res;
        dfs(root, to_string(root->val), res);
        return res;
    }

    void dfs(TreeNode* root, string path, vector<string>& res) {
        if (root->left == NULL && root->right == NULL) {
            res.push_back(path);
        }

        if (root->left != NULL)
            dfs(root->left, path + "->" + to_string(root->left->val), res);

        if (root->right != NULL)
            dfs(root->right, path + "->" + to_string(root->right->val), res);
    }
};
```



46. Permutations

Medium

Given an array `nums` of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

Example 2:

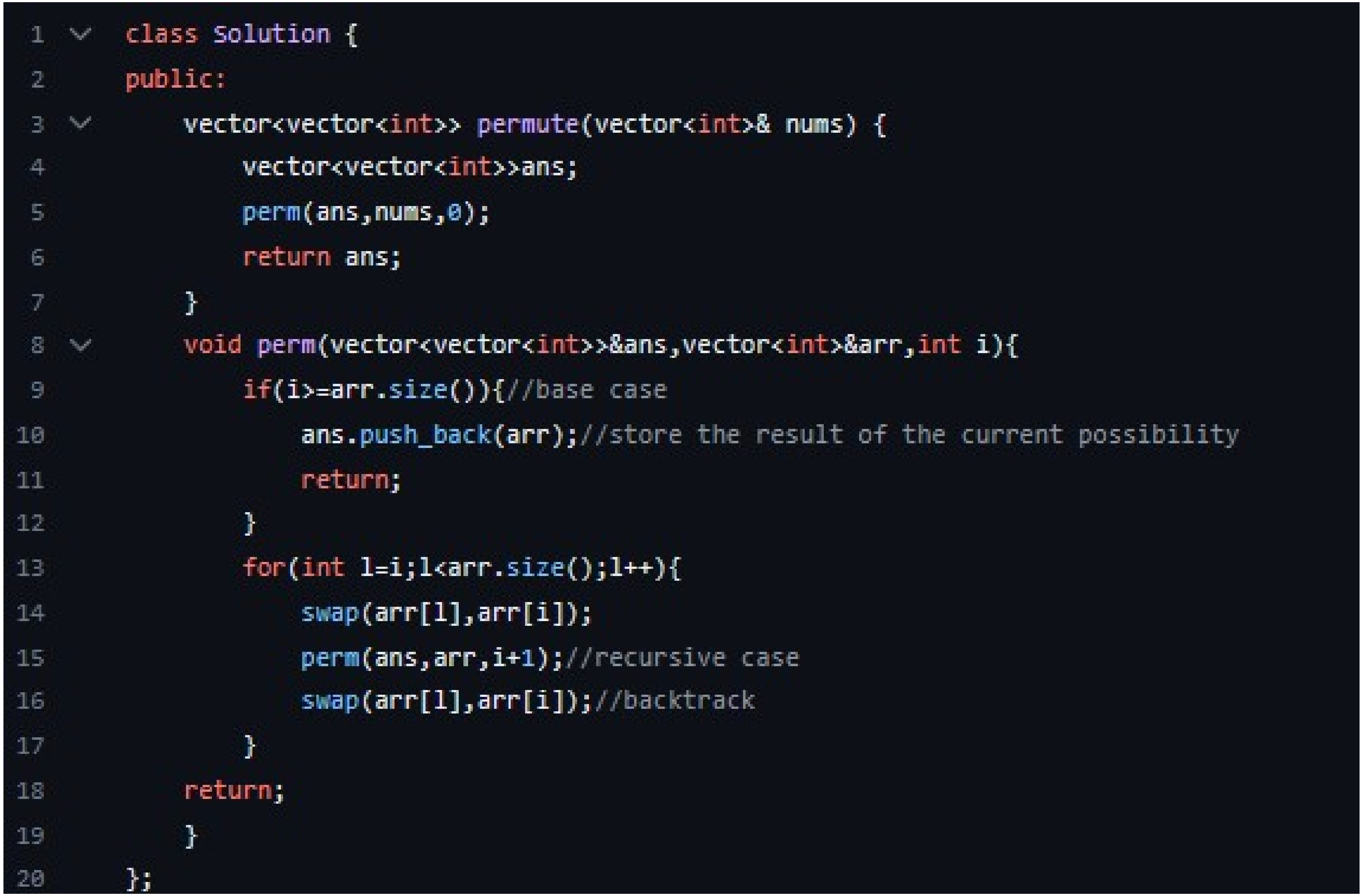
```
Input: nums = [0,1]
Output: [[0,1],[1,0]]
```

Example 3:

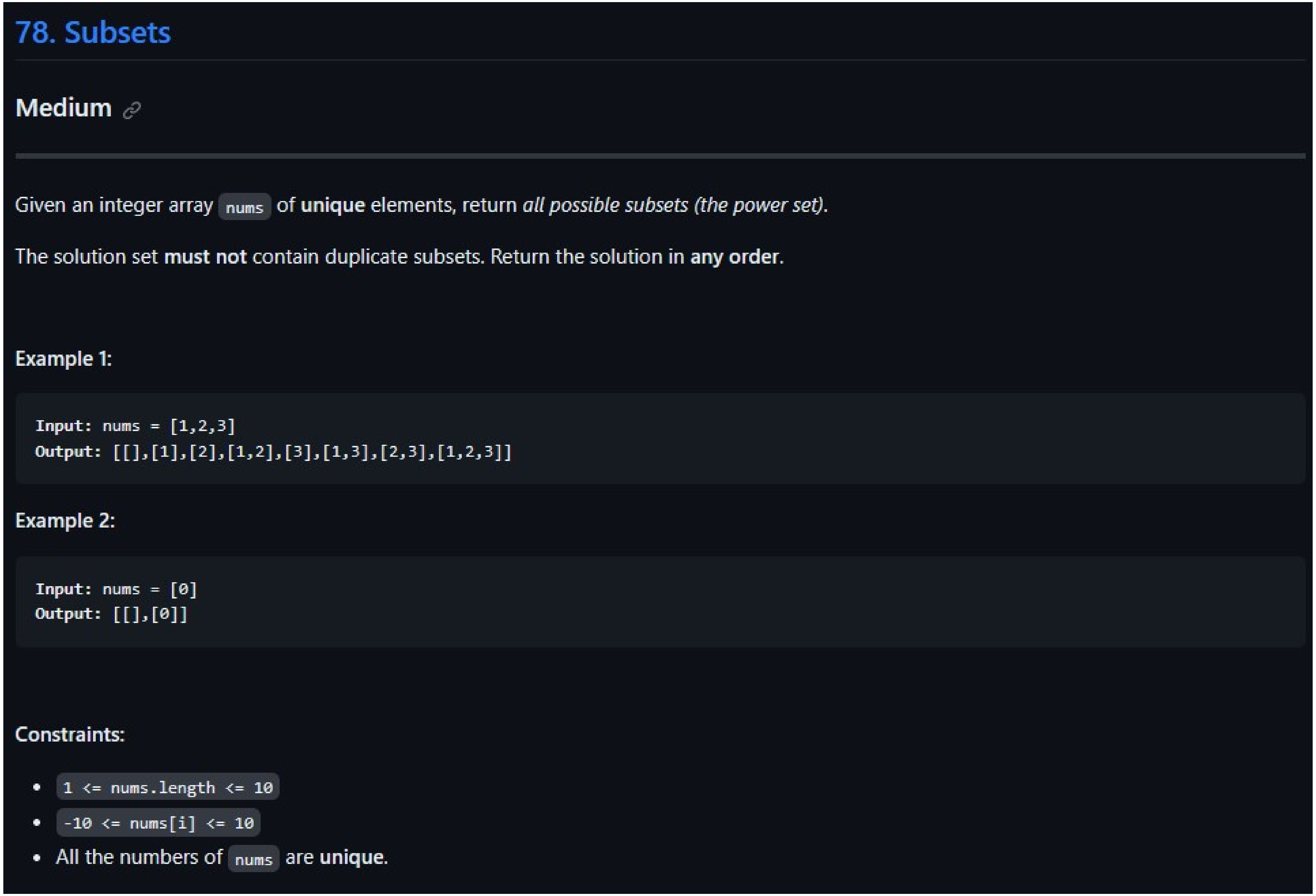
```
Input: nums = [1]
Output: [[1]]
```

Constraints:

- `1 <= nums.length <= 6`
- `-10 <= nums[i] <= 10`
- All the integers of `nums` are **unique**.



```cpp
class Solution {
public:
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>>ans;
        perm(ans,nums,0);
        return ans;
    }
    void perm(vector<vector<int>>&ans,vector<int>&arr,int i){
        if(i>=arr.size()){//base case
            ans.push_back(arr);//store the result of the current possibility
            return;
        }
        for(int l=i;l<arr.size();l++){
            swap(arr[l],arr[i]);
            perm(ans,arr,i+1);//recursive case
            swap(arr[l],arr[i]);//backtrack
        }
    return;
    }
};
```



78. Subsets

Medium

Given an integer array `nums` of **unique** elements, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

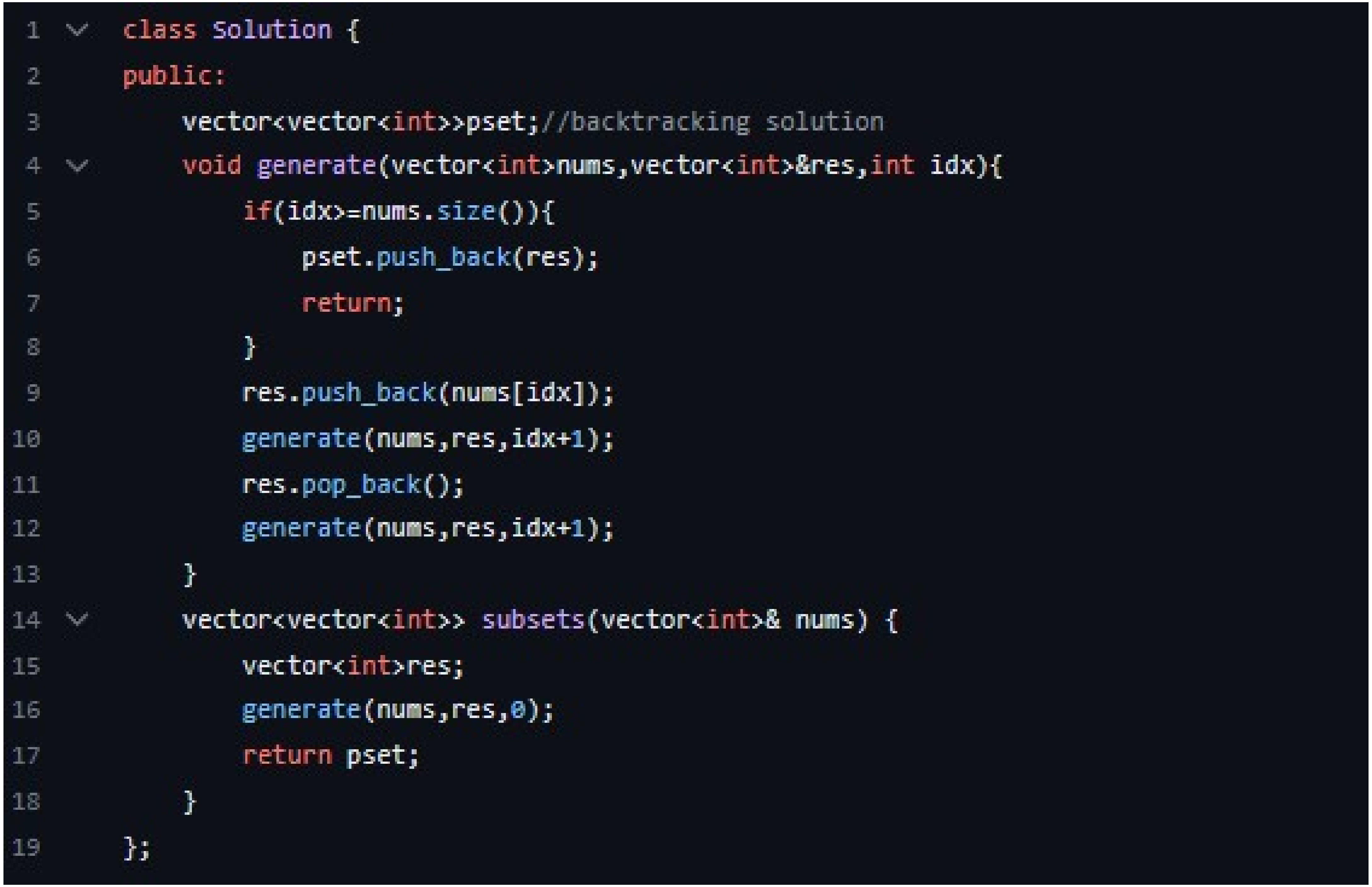
```
Input: nums = [1,2,3]
Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
```

Example 2:

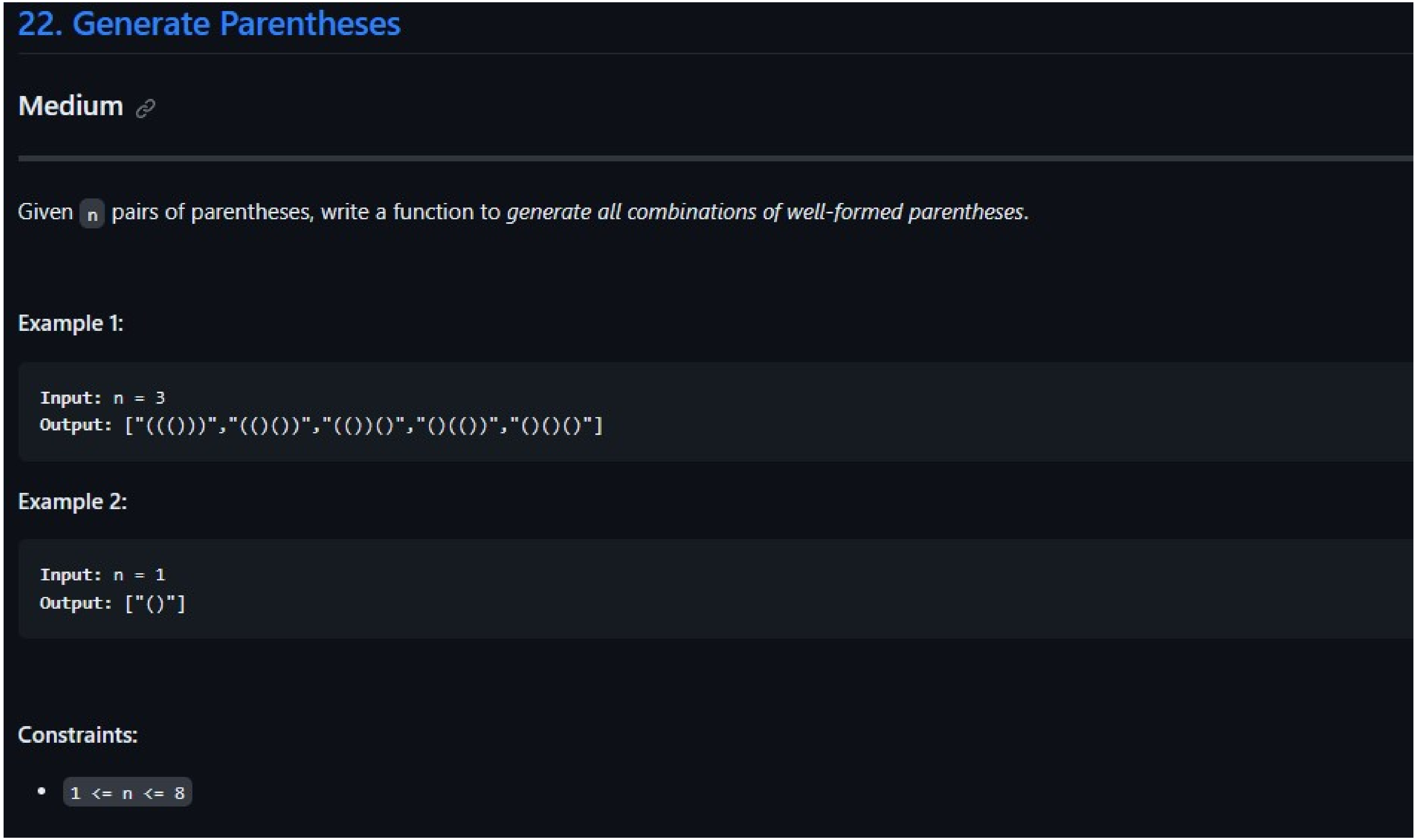
```
Input: nums = [0]
Output: [[],[0]]
```

Constraints:

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`
- All the numbers of `nums` are **unique**.



```cpp
class Solution {
public:
    vector<vector<int>>pset;//backtracking solution
    void generate(vector<int>nums,vector<int>&res,int idx){
        if(idx>=nums.size()){
            pset.push_back(res);
            return;
        }
        res.push_back(nums[idx]);
        generate(nums,res,idx+1);
        res.pop_back();
        generate(nums,res,idx+1);
    }
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<int>res;
        generate(nums,res,0);
        return pset;
    }
};
```



22. Generate Parentheses

Medium

Given `n` pairs of parentheses, write a function to *generate all combinations of well-formed parentheses*.

Example 1:

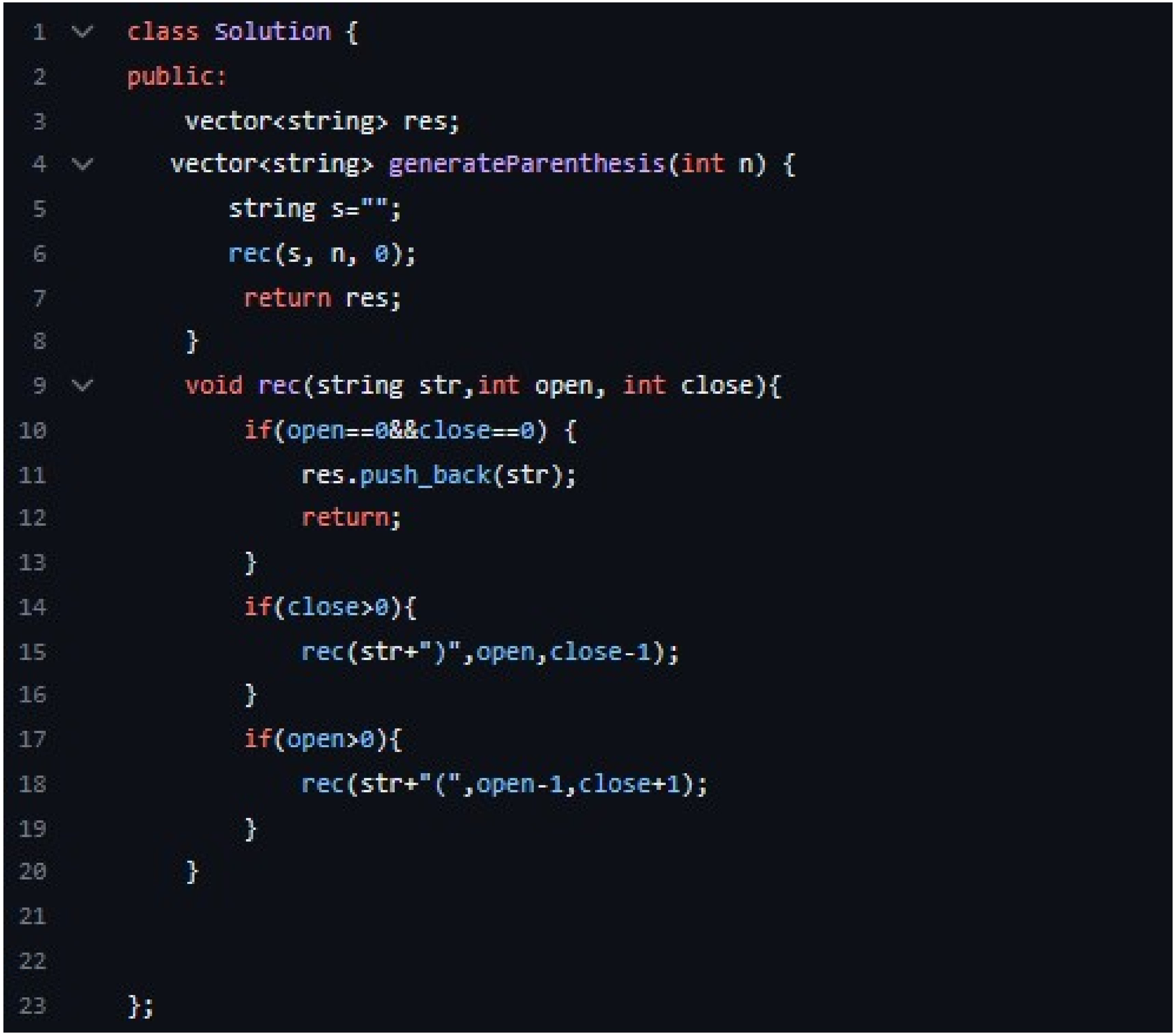
```
Input: n = 3
Output: ["((()))","(()())","(())()","()(())","()()()"]
```

Example 2:

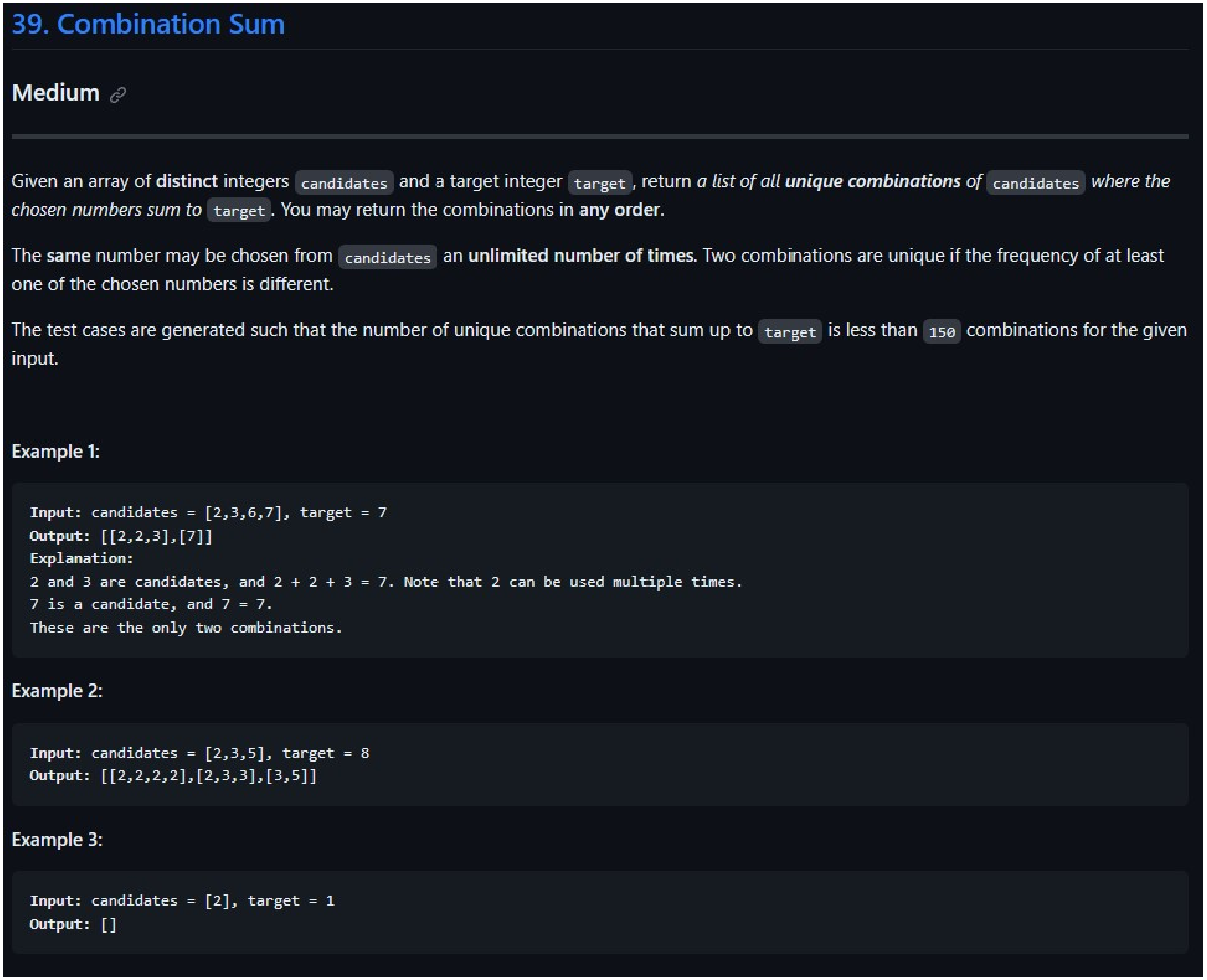
```
Input: n = 1
Output: ["()"]
```

Constraints:

- `1 <= n <= 8`



```
class Solution {
public:
    vector<string> res;
   vector<string> generateParenthesis(int n) {
       string s="";
       rec(s, n, 0);
        return res;
    }
    void rec(string str,int open, int close){
        if(open==0&&close==0) {
            res.push_back(str);
            return;
        }
        if(close>0){
            rec(str+")",open,close-1);
        }
        if(open>0){
            rec(str+"(",open-1,close+1);
        }
    }


};
```



39. Combination Sum

Medium

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all* ***unique combinations*** *of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

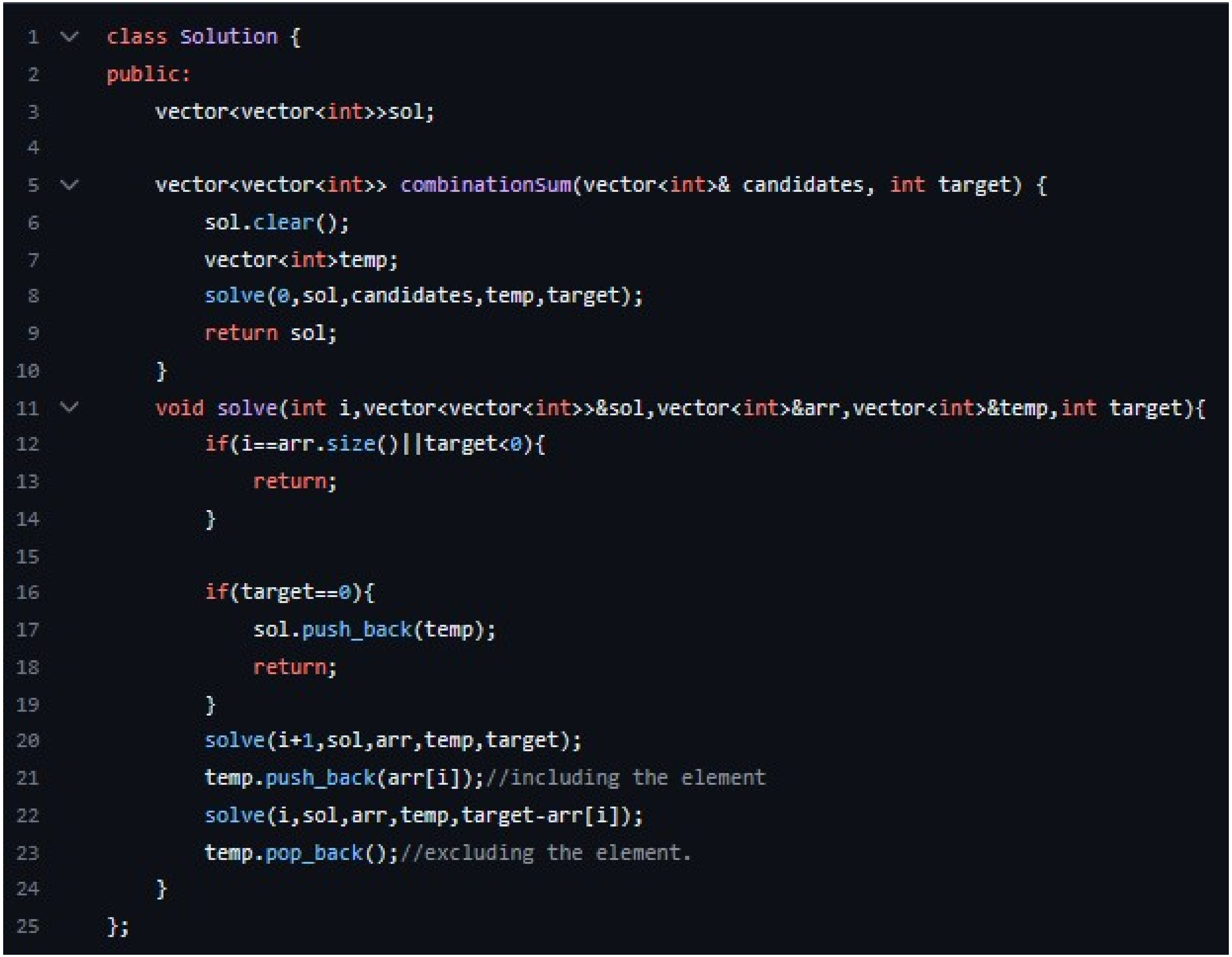
```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

Example 2:

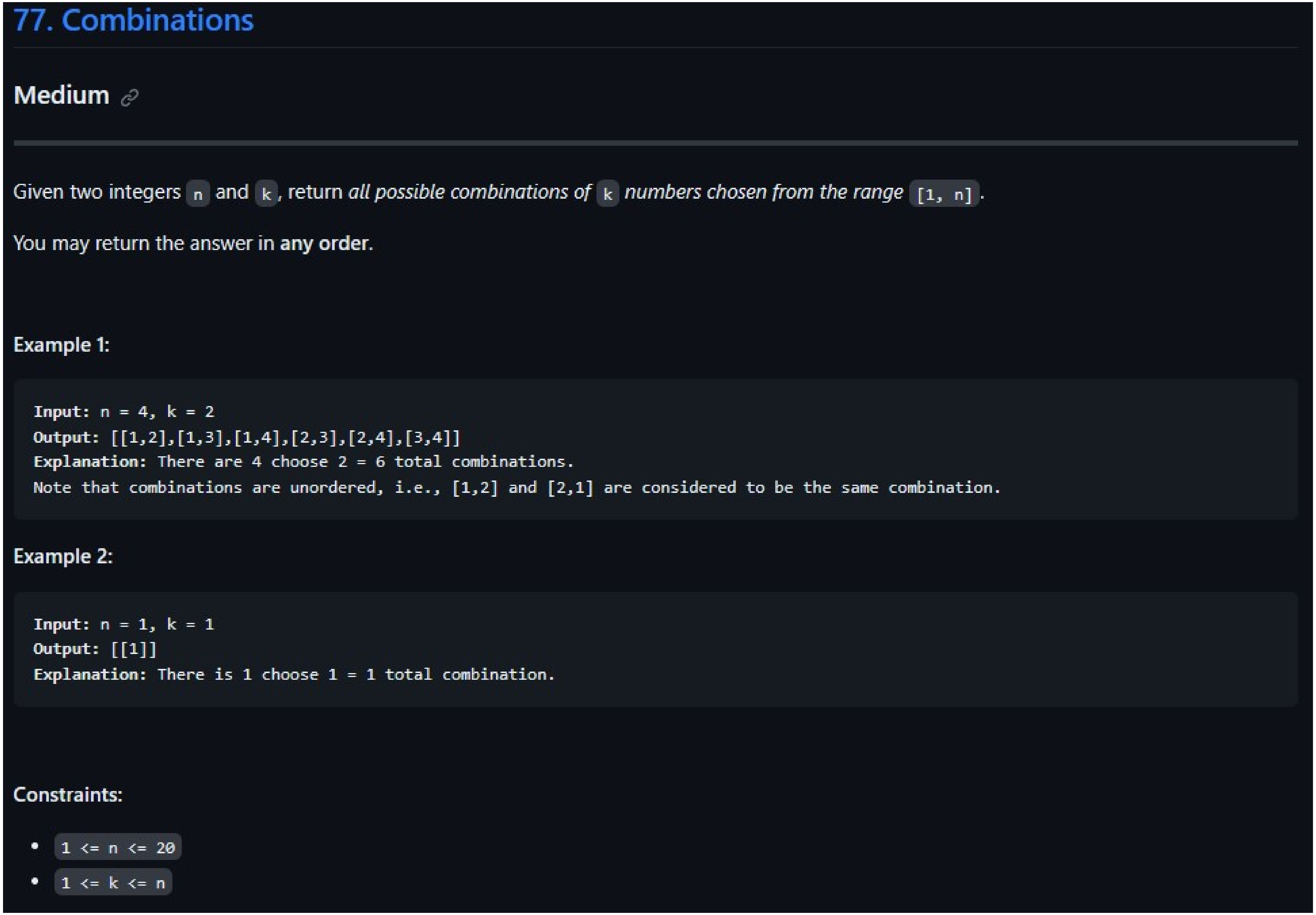
```
Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]
```

Example 3:

```
Input: candidates = [2], target = 1
Output: []
```



```cpp
class Solution {
public:
    vector<vector<int>>sol;

    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        sol.clear();
        vector<int>temp;
        solve(0,sol,candidates,temp,target);
        return sol;
    }
    void solve(int i,vector<vector<int>>&sol,vector<int>&arr,vector<int>&temp,int target){
        if(i==arr.size()||target<0){
            return;
        }

        if(target==0){
            sol.push_back(temp);
            return;
        }
        solve(i+1,sol,arr,temp,target);
        temp.push_back(arr[i]);//including the element
        solve(i,sol,arr,temp,target-arr[i]);
        temp.pop_back();//excluding the element.
    }
};
```



77. Combinations

Medium

Given two integers `n` and `k`, return *all possible combinations of* `k` *numbers chosen from the range* `[1, n]`.

You may return the answer in **any order**.

Example 1:

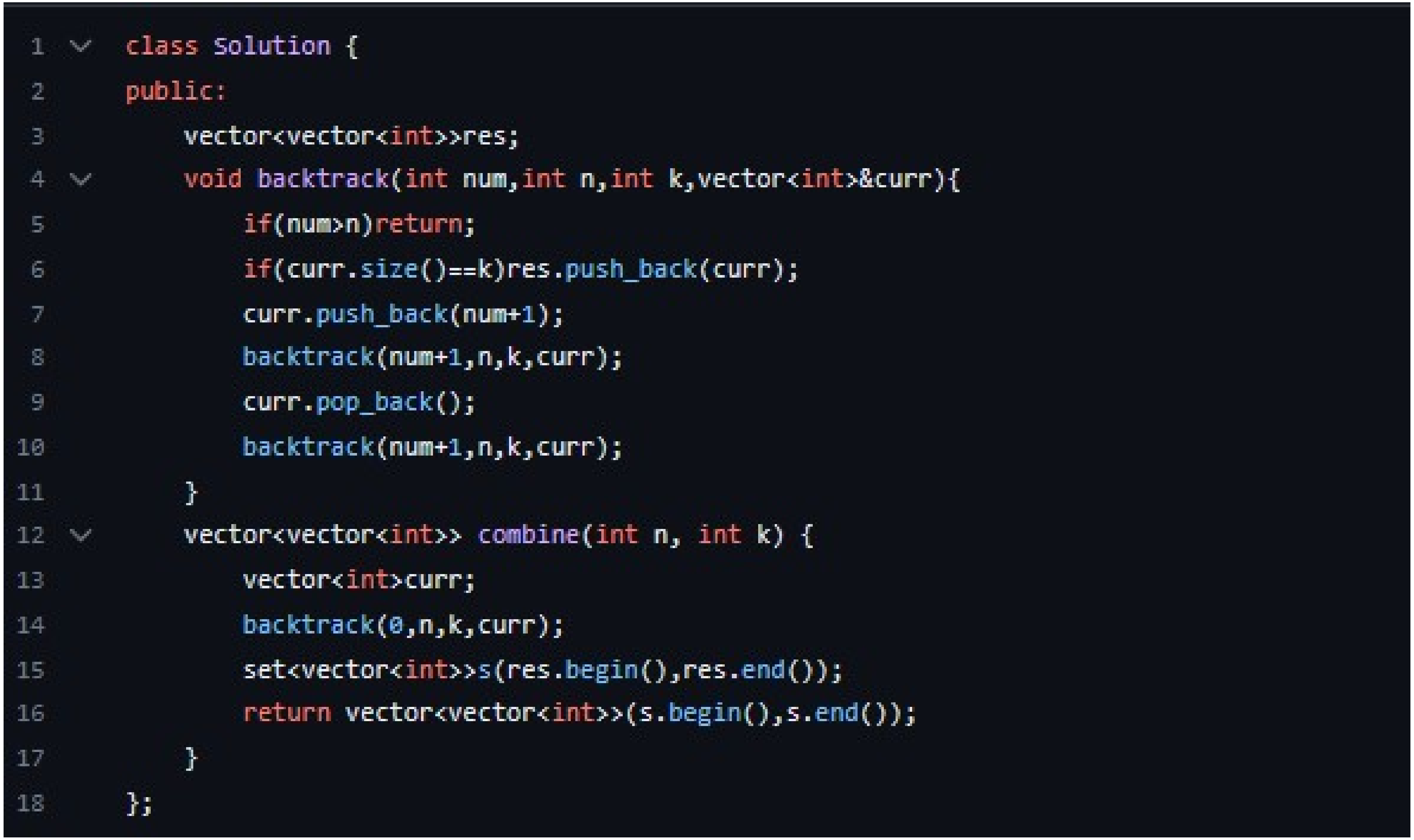
```
Input: n = 4, k = 2
Output: [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]
Explanation: There are 4 choose 2 = 6 total combinations.
Note that combinations are unordered, i.e., [1,2] and [2,1] are considered to be the same combination.
```

Example 2:

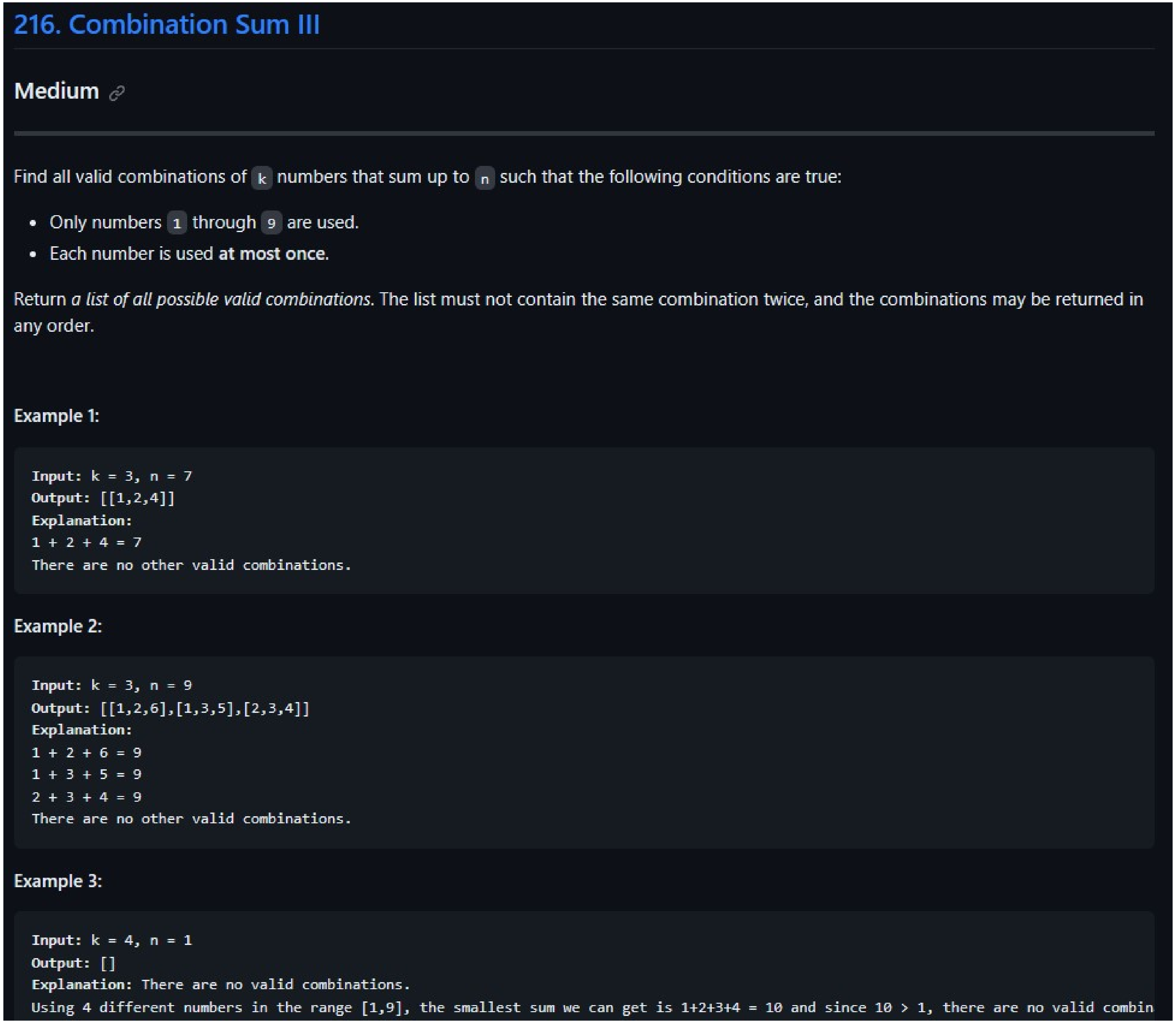
```
Input: n = 1, k = 1
Output: [[1]]
Explanation: There is 1 choose 1 = 1 total combination.
```

Constraints:

- `1 <= n <= 20`
- `1 <= k <= n`



```cpp
class Solution {
public:
    vector<vector<int>>res;
    void backtrack(int num,int n,int k,vector<int>&curr){
        if(num>n)return;
        if(curr.size()==k)res.push_back(curr);
        curr.push_back(num+1);
        backtrack(num+1,n,k,curr);
        curr.pop_back();
        backtrack(num+1,n,k,curr);
    }
    vector<vector<int>> combine(int n, int k) {
        vector<int>curr;
        backtrack(0,n,k,curr);
        set<vector<int>>s(res.begin(),res.end());
        return vector<vector<int>>(s.begin(),s.end());
    }
};
```



216. Combination Sum III

Medium

Find all valid combinations of `k` numbers that sum up to `n` such that the following conditions are true:

- Only numbers `1` through `9` are used.
- Each number is used **at most once**.

Return *a list of all possible valid combinations*. The list must not contain the same combination twice, and the combinations may be returned in any order.

Example 1:

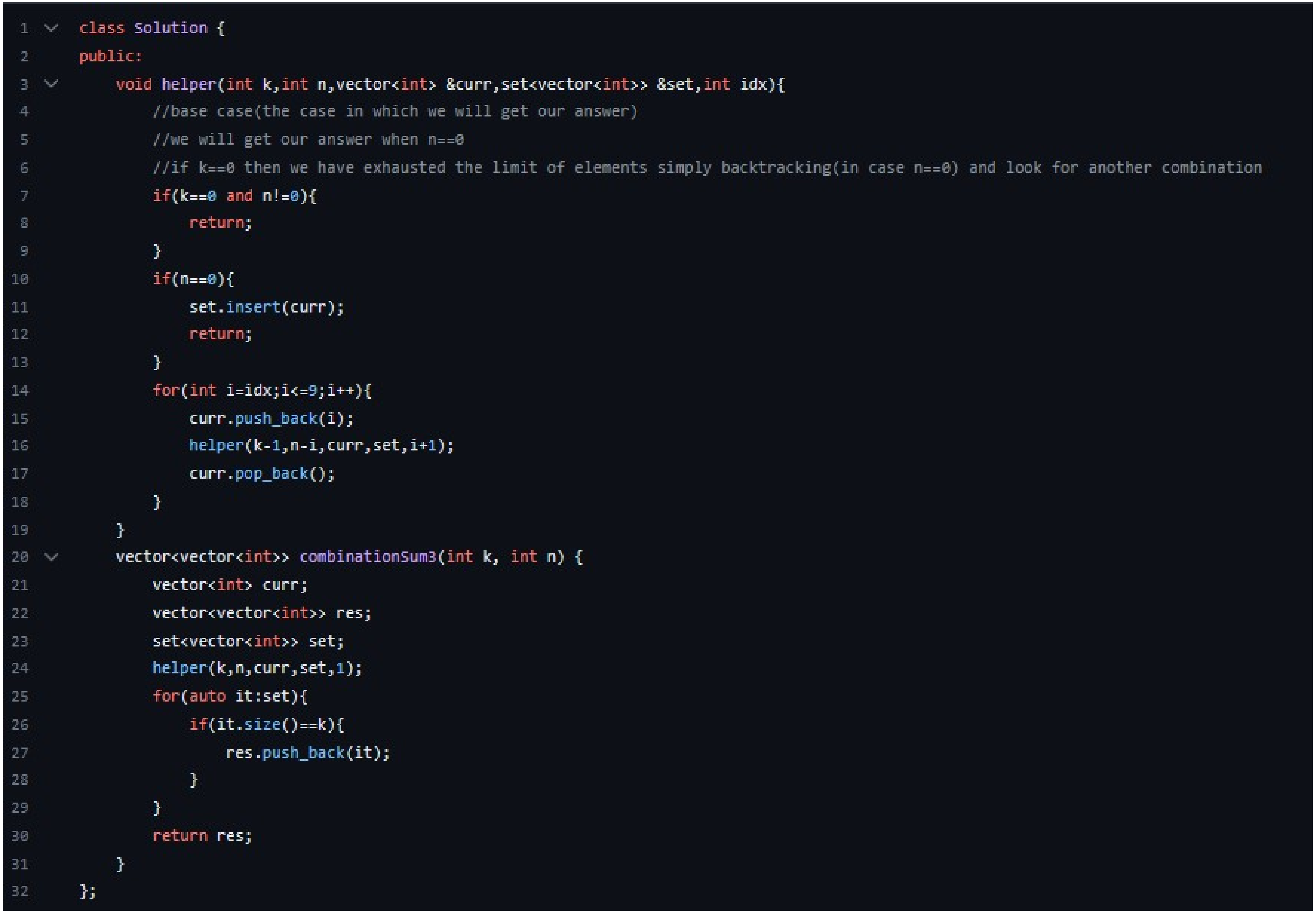
```
Input: k = 3, n = 7
Output: [[1,2,4]]
Explanation:
1 + 2 + 4 = 7
There are no other valid combinations.
```

Example 2:

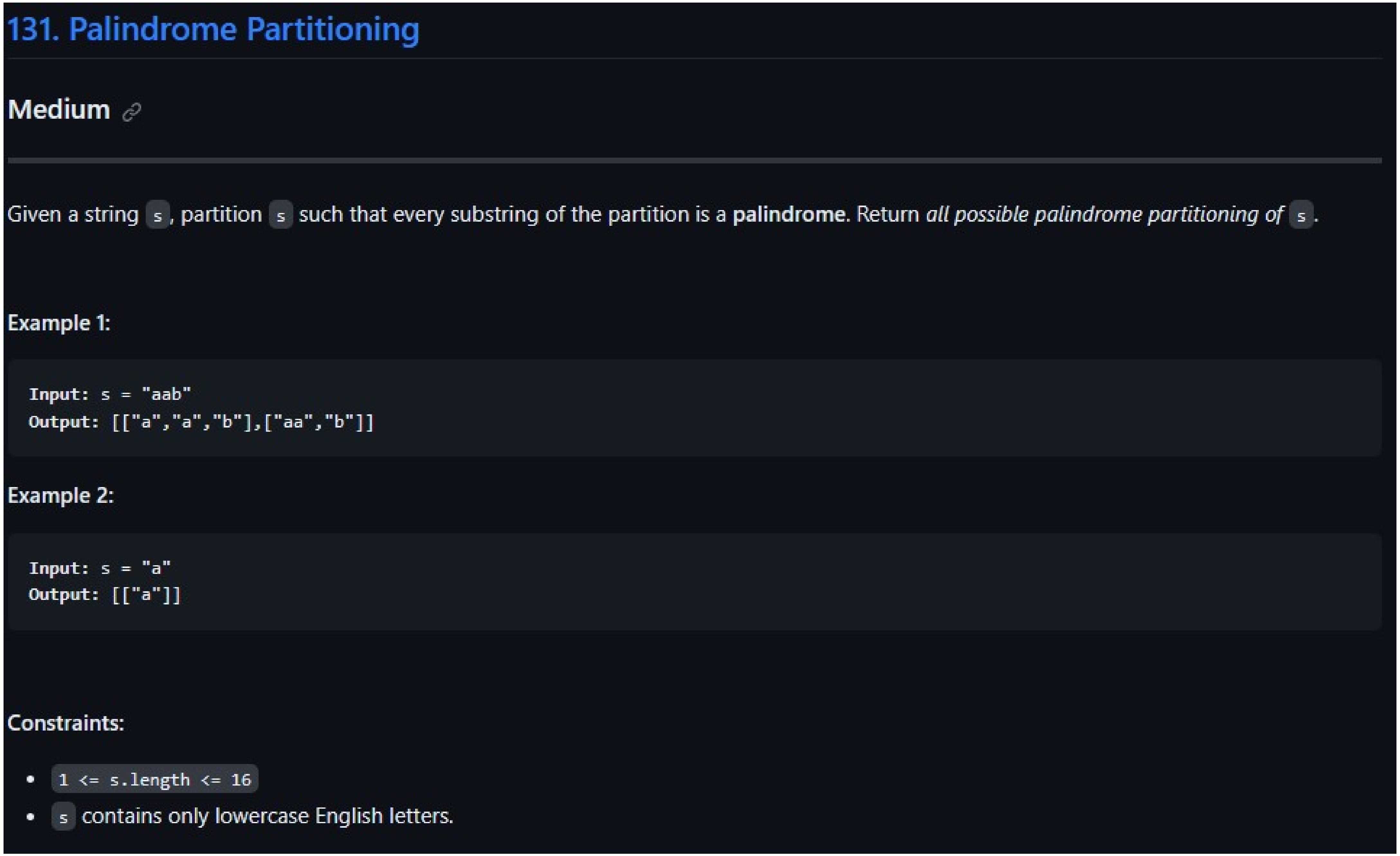
```
Input: k = 3, n = 9
Output: [[1,2,6],[1,3,5],[2,3,4]]
Explanation:
1 + 2 + 6 = 9
1 + 3 + 5 = 9
2 + 3 + 4 = 9
There are no other valid combinations.
```

Example 3:

```
Input: k = 4, n = 1
Output: []
Explanation: There are no valid combinations.
Using 4 different numbers in the range [1,9], the smallest sum we can get is 1+2+3+4 = 10 and since 10 > 1, there are no valid combin
```



```cpp
class Solution {
public:
    void helper(int k,int n,vector<int> &curr,set<vector<int>> &set,int idx){
        //base case(the case in which we will get our answer)
        //we will get our answer when n==0
        //if k==0 then we have exhausted the limit of elements simply backtracking(in case n==0) and look for another combination
        if(k==0 and n!=0){
            return;
        }
        if(n==0){
            set.insert(curr);
            return;
        }
        for(int i=idx;i<=9;i++){
            curr.push_back(i);
            helper(k-1,n-i,curr,set,i+1);
            curr.pop_back();
        }
    }
    vector<vector<int>> combinationSum3(int k, int n) {
        vector<int> curr;
        vector<vector<int>> res;
        set<vector<int>> set;
        helper(k,n,curr,set,1);
        for(auto it:set){
            if(it.size()==k){
                res.push_back(it);
            }
        }
        return res;
    }
};
```



131. Palindrome Partitioning

Medium

Given a string `s`, partition `s` such that every substring of the partition is a **palindrome**. Return *all possible palindrome partitioning of* `s`.

Example 1:

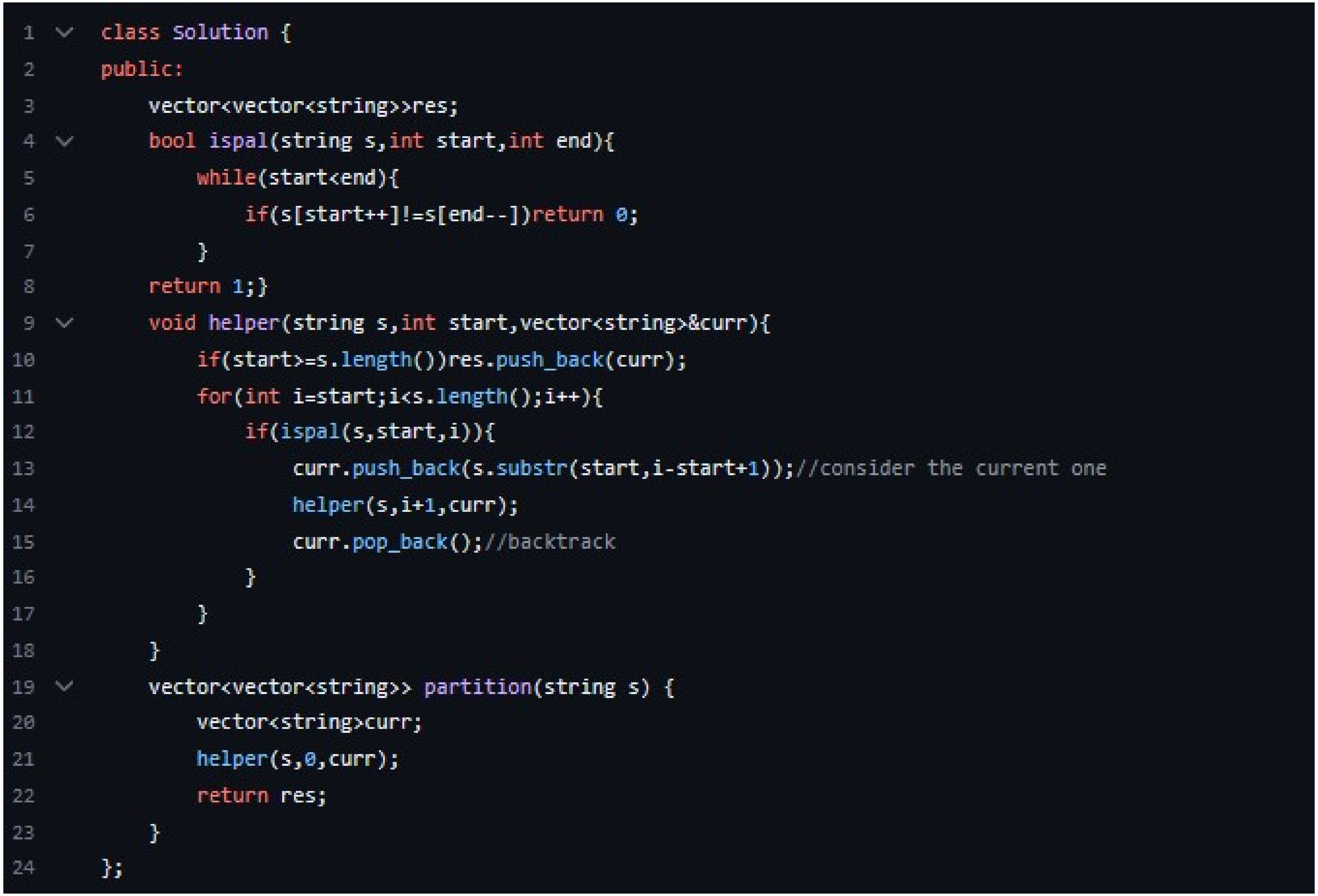
```
Input: s = "aab"
Output: [["a","a","b"],["aa","b"]]
```

Example 2:

```
Input: s = "a"
Output: [["a"]]
```

Constraints:

- `1 <= s.length <= 16`
- `s` contains only lowercase English letters.



```cpp
class Solution {
public:
    vector<vector<string>>res;
    bool ispal(string s,int start,int end){
        while(start<end){
            if(s[start++]!=s[end--])return 0;
        }
    return 1;}
    void helper(string s,int start,vector<string>&curr){
        if(start>=s.length())res.push_back(curr);
        for(int i=start;i<s.length();i++){
            if(ispal(s,start,i)){
                curr.push_back(s.substr(start,i-start+1));//consider the current one
                helper(s,i+1,curr);
                curr.pop_back();//backtrack
            }
        }
    }
    vector<vector<string>> partition(string s) {
        vector<string>curr;
        helper(s,0,curr);
        return res;
    }
};
```

51. N-Queens

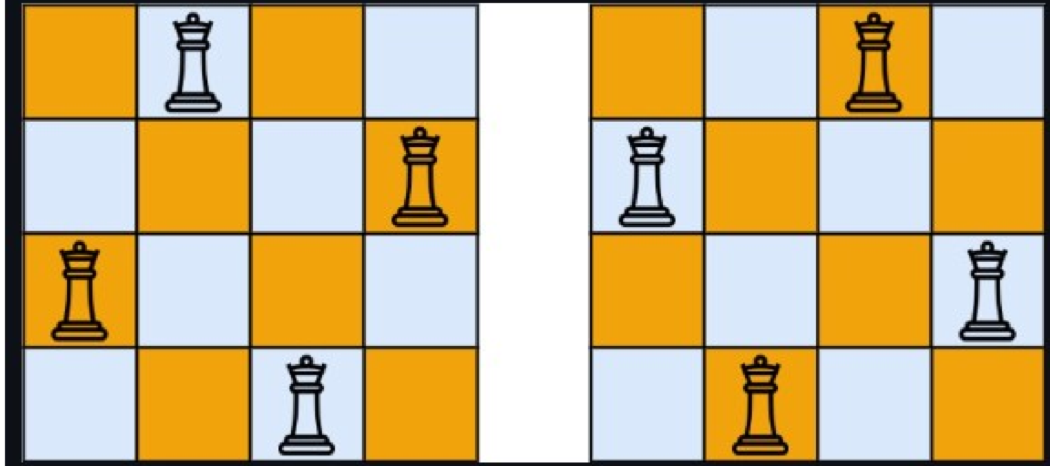
Hard

The **n-queens** puzzle is the problem of placing `n` queens on an `n x n` chessboard such that no two queens attack each other.

Given an integer `n`, return *all distinct solutions to the **n-queens puzzle***. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space, respectively.

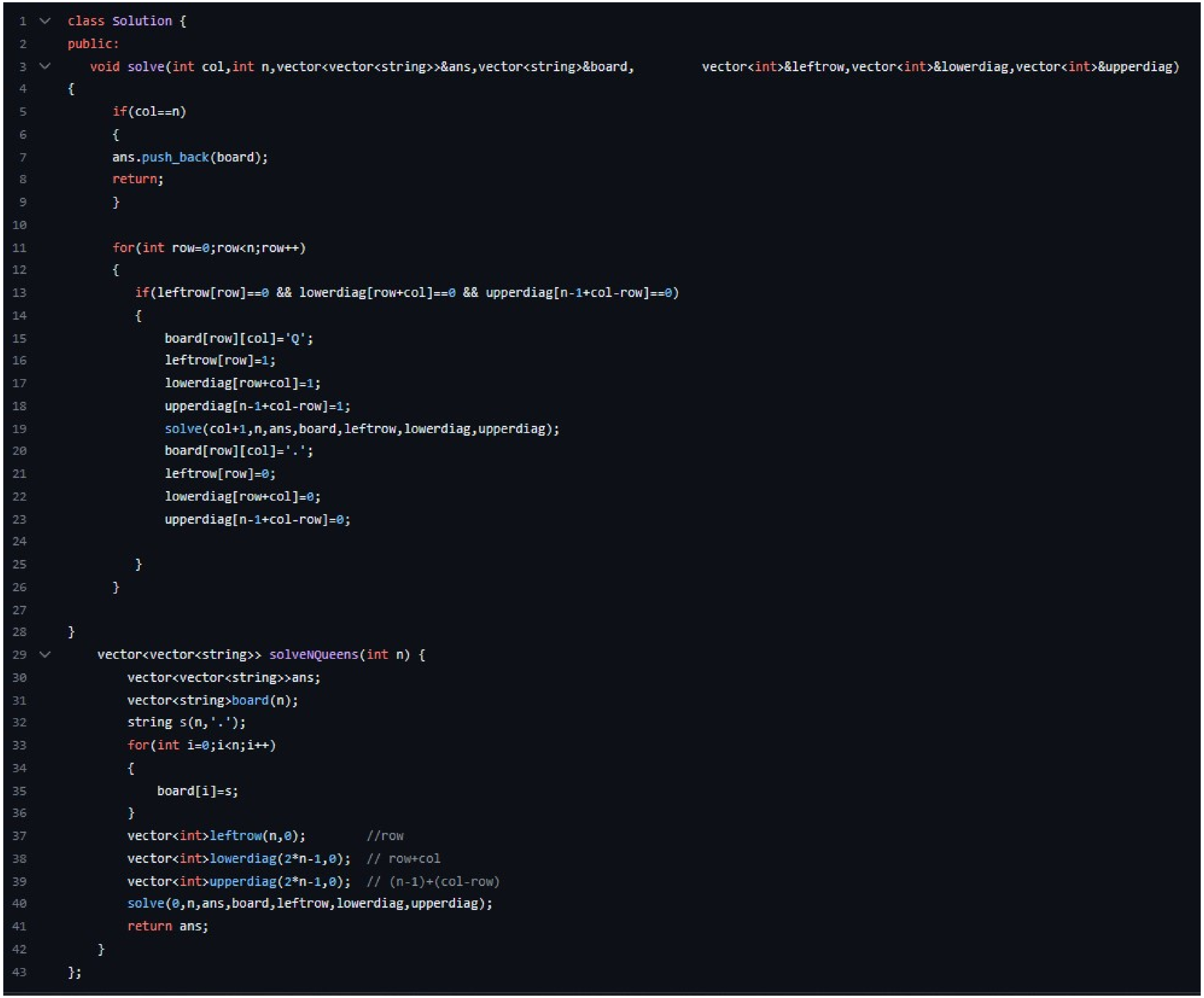
Example 1:



```
Input: n = 4
Output: [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above
```

Example 2:

```
Input: n = 1
Output: [["Q"]]
```



```cpp
class Solution {
public:
   void solve(int col,int n,vector<vector<string>>&ans,vector<string>&board,         vector<int>&leftrow,vector<int>&lowerdiag,vector<int>&upperdiag)
{
      if(col==n)
      {
      ans.push_back(board);
      return;
      }

      for(int row=0;row<n;row++)
      {
         if(leftrow[row]==0 && lowerdiag[row+col]==0 && upperdiag[n-1+col-row]==0)
         {
             board[row][col]='Q';
             leftrow[row]=1;
             lowerdiag[row+col]=1;
             upperdiag[n-1+col-row]=1;
             solve(col+1,n,ans,board,leftrow,lowerdiag,upperdiag);
             board[row][col]='.';
             leftrow[row]=0;
             lowerdiag[row+col]=0;
             upperdiag[n-1+col-row]=0;

         }
      }

}
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>>ans;
        vector<string>board(n);
        string s(n,'.');
        for(int i=0;i<n;i++)
        {
            board[i]=s;
        }
        vector<int>leftrow(n,0);        //row
        vector<int>lowerdiag(2*n-1,0);  // row+col
        vector<int>upperdiag(2*n-1,0);  // (n-1)+(col-row)
        solve(0,n,ans,board,leftrow,lowerdiag,upperdiag);
        return ans;
    }
};
```

37. Sudoku Solver

Hard

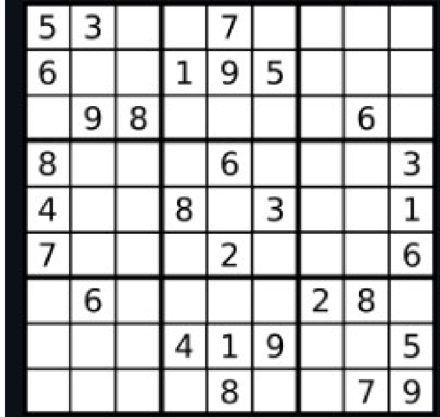
Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

1. Each of the digits `1-9` must occur exactly once in each row.
2. Each of the digits `1-9` must occur exactly once in each column.
3. Each of the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

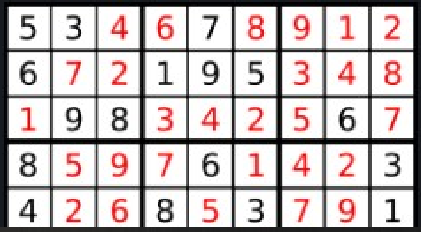
The `'.'` character indicates empty cells.

Example 1:

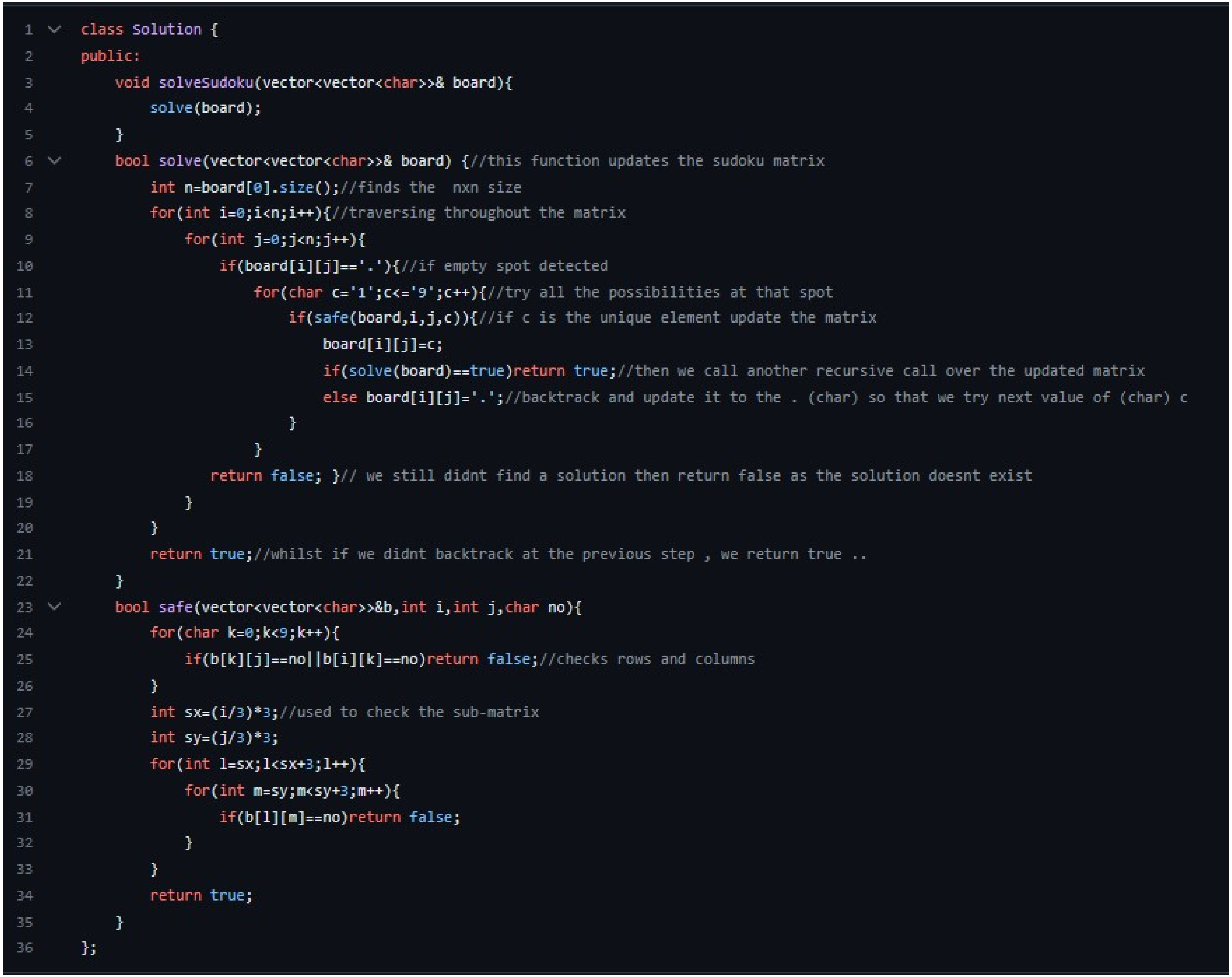


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

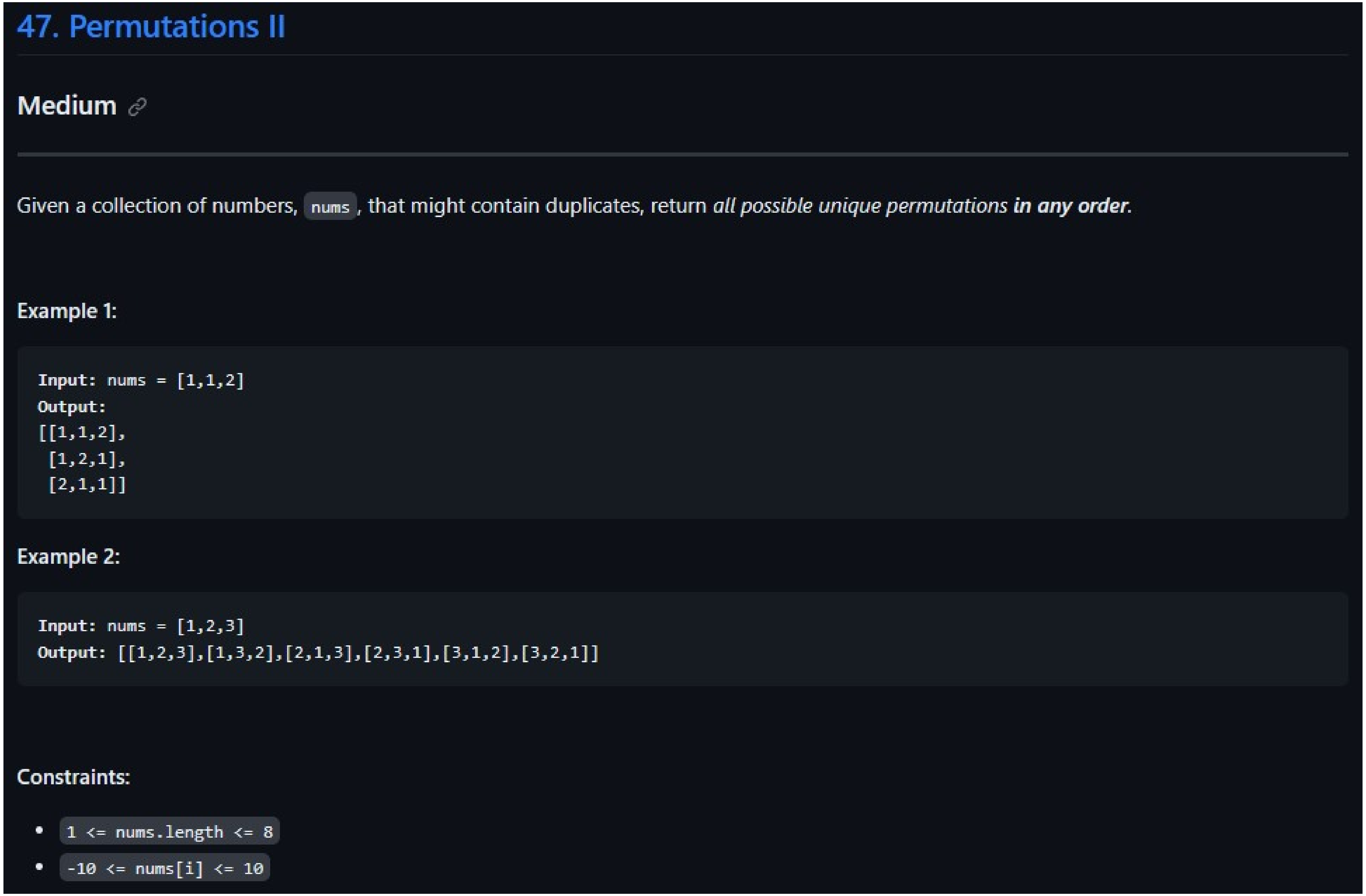
```
Input: board = [["5","3",".",".","7",".",".",".","."],["6",".",".","1","9","5",".",".","."],[".","9","8",".",".",".",".","6","."],["8
Output: [["5","3","4","6","7","8","9","1","2"],["6","7","2","1","9","5","3","4","8"],["1","9","8","3","4","2","5","6","7"],["8","5","
Explanation: The input board is shown above and the only valid solution is shown below:
```



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1



```cpp
class Solution {
public:
    void solveSudoku(vector<vector<char>>& board){
        solve(board);
    }
    bool solve(vector<vector<char>>& board) {//this function updates the sudoku matrix
        int n=board[0].size();//finds the  nxn size
        for(int i=0;i<n;i++){//traversing throughout the matrix
            for(int j=0;j<n;j++){
                if(board[i][j]=='.'){//if empty spot detected
                    for(char c='1';c<='9';c++){//try all the possibilities at that spot
                        if(safe(board,i,j,c)){//if c is the unique element update the matrix
                            board[i][j]=c;
                            if(solve(board)==true)return true;//then we call another recursive call over the updated matrix
                            else board[i][j]='.';//backtrack and update it to the . (char) so that we try next value of (char) c
                        }
                    }
                   return false; }// we still didnt find a solution then return false as the solution doesnt exist
            }
        }
        return true;//whilst if we didnt backtrack at the previous step , we return true ..
    }
    bool safe(vector<vector<char>>&b,int i,int j,char no){
        for(char k=0;k<9;k++){
            if(b[k][j]==no||b[i][k]==no)return false;//checks rows and columns
        }
        int sx=(i/3)*3;//used to check the sub-matrix
        int sy=(j/3)*3;
        for(int l=sx;l<sx+3;l++){
            for(int m=sy;m<sy+3;m++){
                if(b[l][m]==no)return false;
            }
        }
        return true;
    }
};
```



47. Permutations II

Medium

Given a collection of numbers, `nums`, that might contain duplicates, return *all possible unique permutations* ***in any order***.

Example 1:

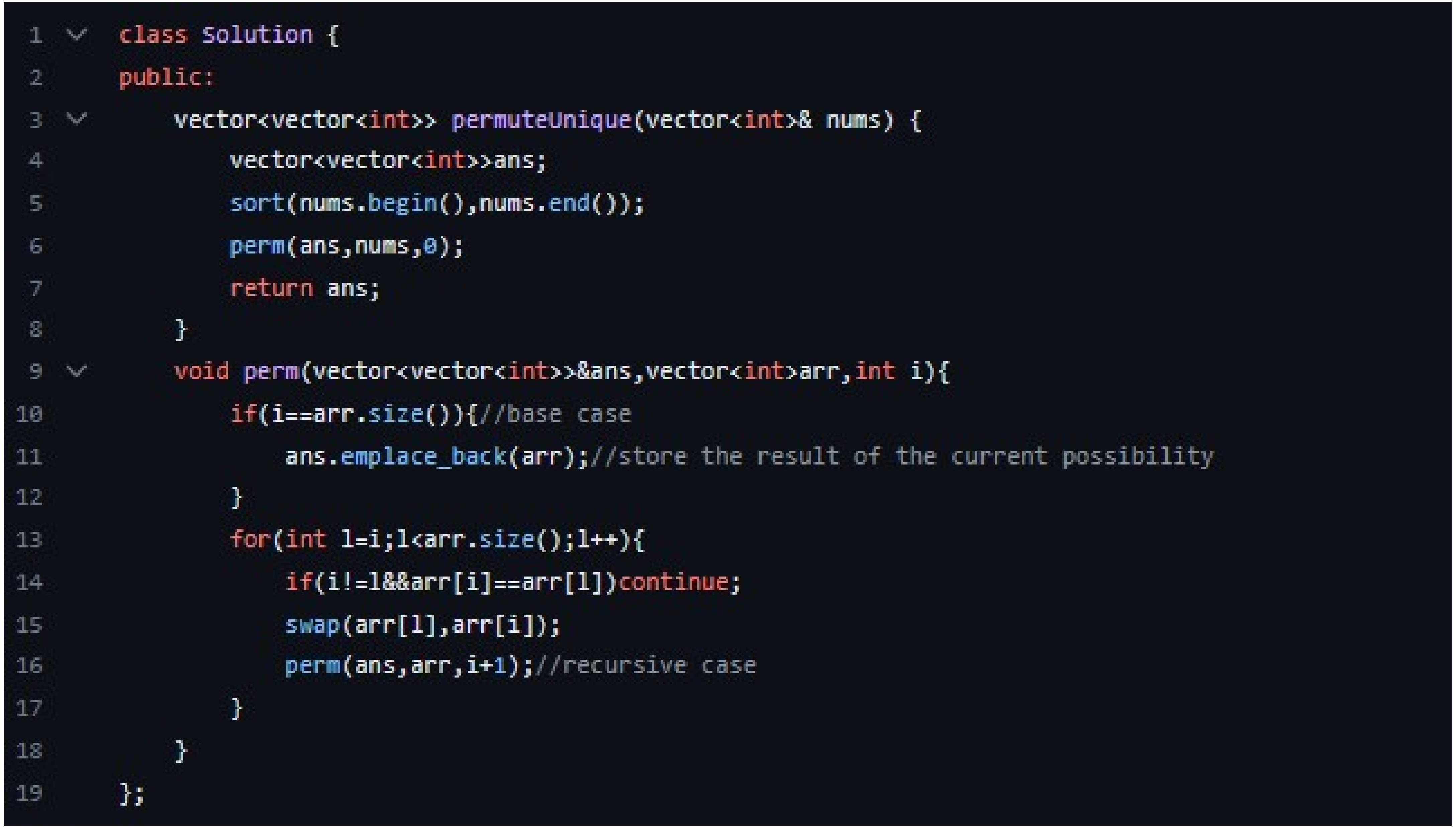
```
Input: nums = [1,1,2]
Output:
[[1,1,2],
 [1,2,1],
 [2,1,1]]
```

Example 2:

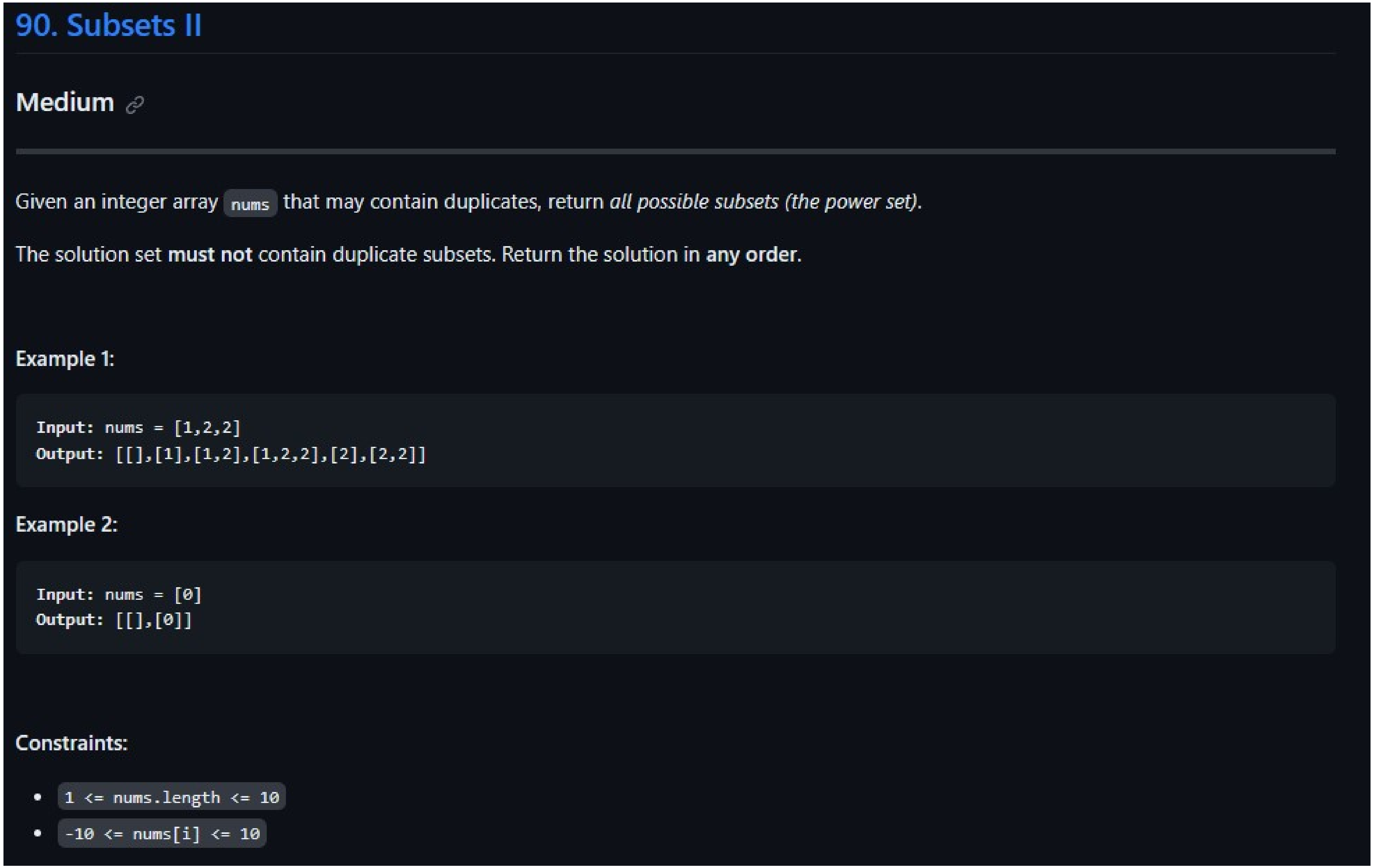
```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

Constraints:

- `1 <= nums.length <= 8`
- `-10 <= nums[i] <= 10`



```cpp
class Solution {
public:
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        vector<vector<int>>ans;
        sort(nums.begin(),nums.end());
        perm(ans,nums,0);
        return ans;
    }
    void perm(vector<vector<int>>&ans,vector<int>arr,int i){
        if(i==arr.size()){//base case
            ans.emplace_back(arr);//store the result of the current possibility
        }
        for(int l=i;l<arr.size();l++){
            if(i!=l&&arr[i]==arr[l])continue;
            swap(arr[l],arr[i]);
            perm(ans,arr,i+1);//recursive case
        }
    }
};
```



90. Subsets II

Medium

Given an integer array `nums` that may contain duplicates, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

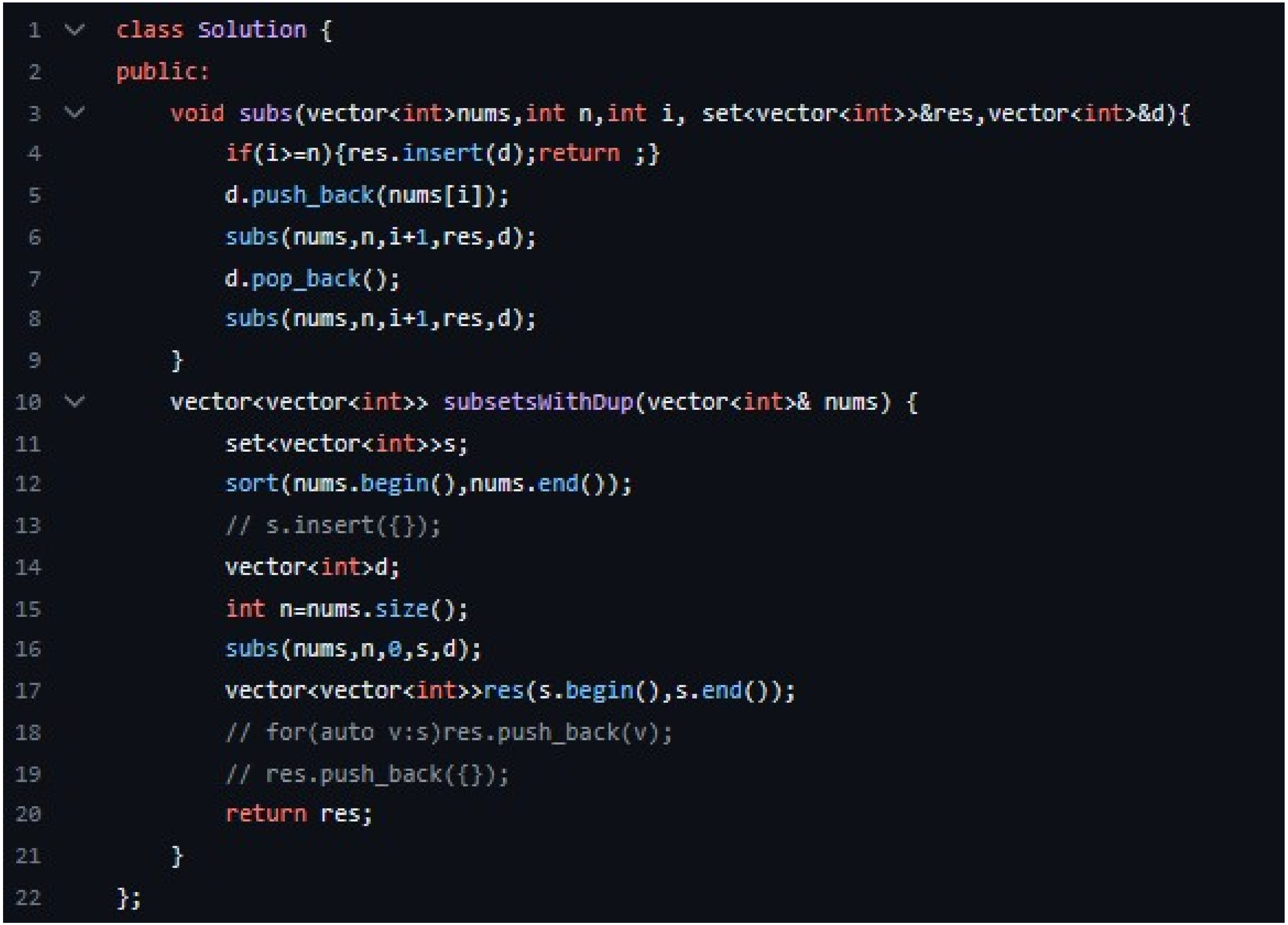
```
Input: nums = [1,2,2]
Output: [[],[1],[1,2],[1,2,2],[2],[2,2]]
```

Example 2:

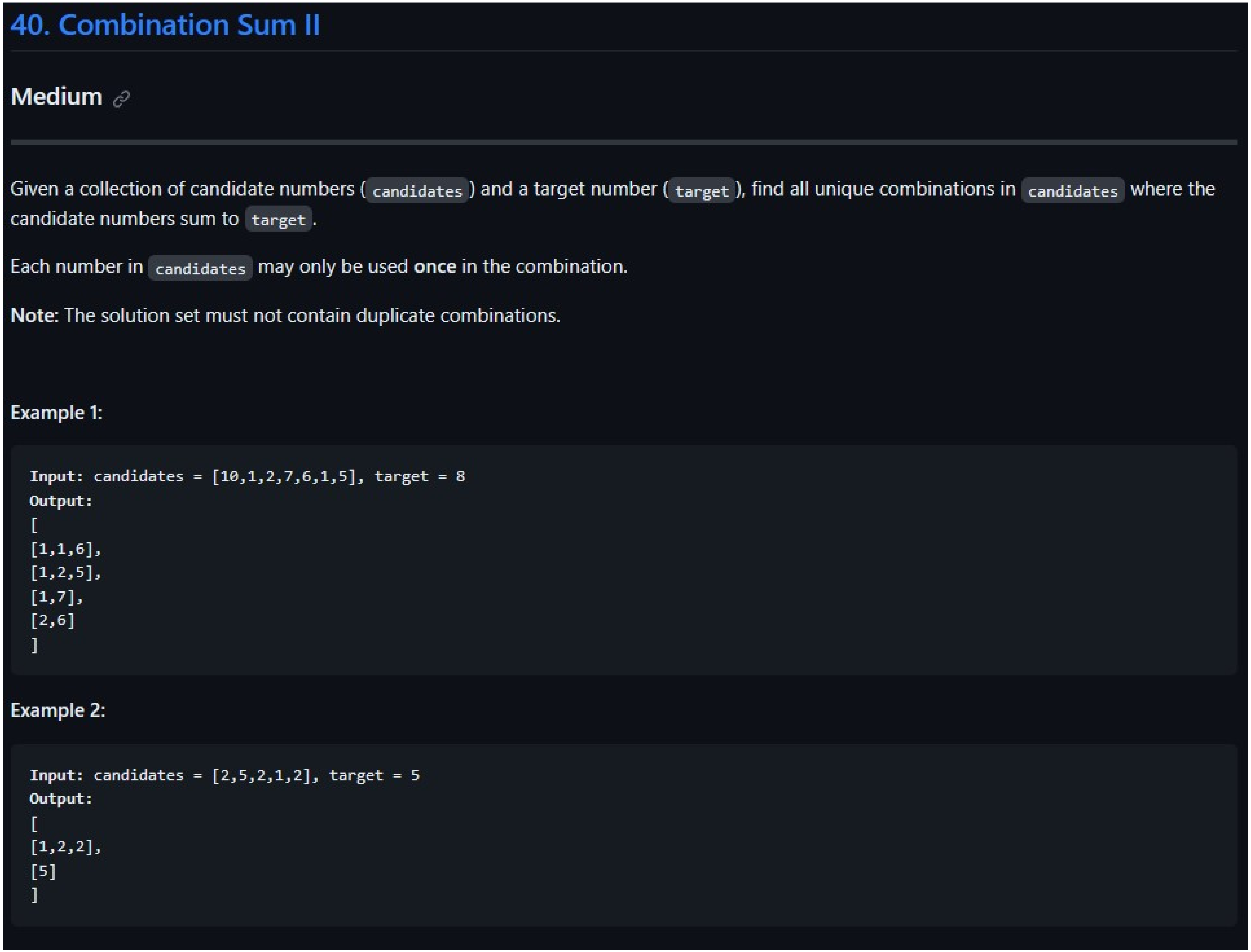
```
Input: nums = [0]
Output: [[],[0]]
```

Constraints:

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`



```cpp
class Solution {
public:
    void subs(vector<int>nums,int n,int i, set<vector<int>>&res,vector<int>&d){
        if(i>=n){res.insert(d);return ;}
        d.push_back(nums[i]);
        subs(nums,n,i+1,res,d);
        d.pop_back();
        subs(nums,n,i+1,res,d);
    }
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        set<vector<int>>s;
        sort(nums.begin(),nums.end());
        // s.insert({});
        vector<int>d;
        int n=nums.size();
        subs(nums,n,0,s,d);
        vector<vector<int>>res(s.begin(),s.end());
        // for(auto v:s)res.push_back(v);
        // res.push_back({});
        return res;
    }
};
```



40. Combination Sum II

Medium

Given a collection of candidate numbers (`candidates`) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sum to `target`.

Each number in `candidates` may only be used **once** in the combination.

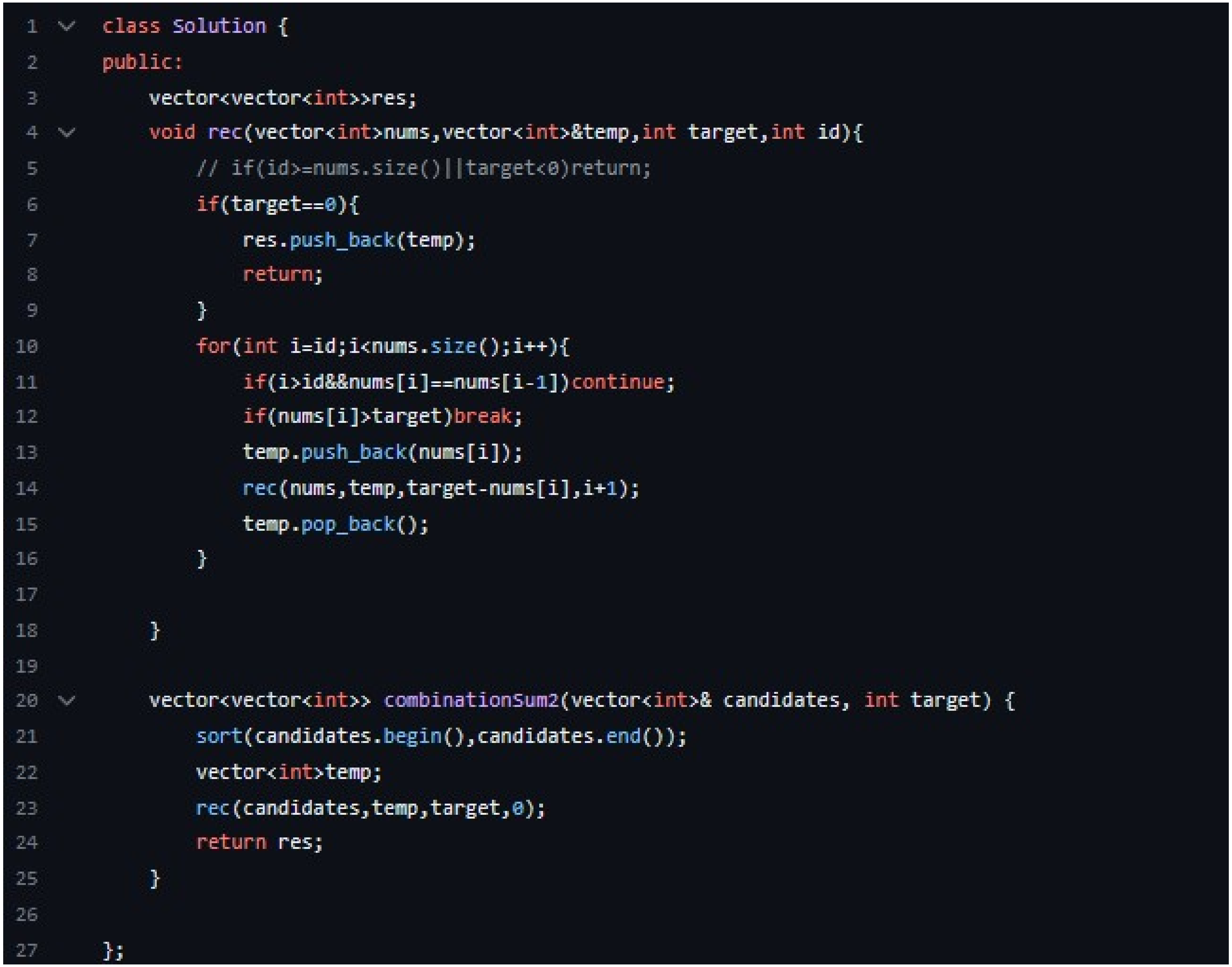
Note: The solution set must not contain duplicate combinations.

Example 1:

```
Input: candidates = [10,1,2,7,6,1,5], target = 8
Output:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]
```

Example 2:

```
Input: candidates = [2,5,2,1,2], target = 5
Output:
[
[1,2,2],
[5]
]
```



```cpp
class Solution {
public:
    vector<vector<int>>res;
    void rec(vector<int>nums,vector<int>&temp,int target,int id){
        // if(id>=nums.size()||target<0)return;
        if(target==0){
            res.push_back(temp);
            return;
        }
        for(int i=id;i<nums.size();i++){
            if(i>id&&nums[i]==nums[i-1])continue;
            if(nums[i]>target)break;
            temp.push_back(nums[i]);
            rec(nums,temp,target-nums[i],i+1);
            temp.pop_back();
        }

    }

    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
        sort(candidates.begin(),candidates.end());
        vector<int>temp;
        rec(candidates,temp,target,0);
        return res;
    }

};
```

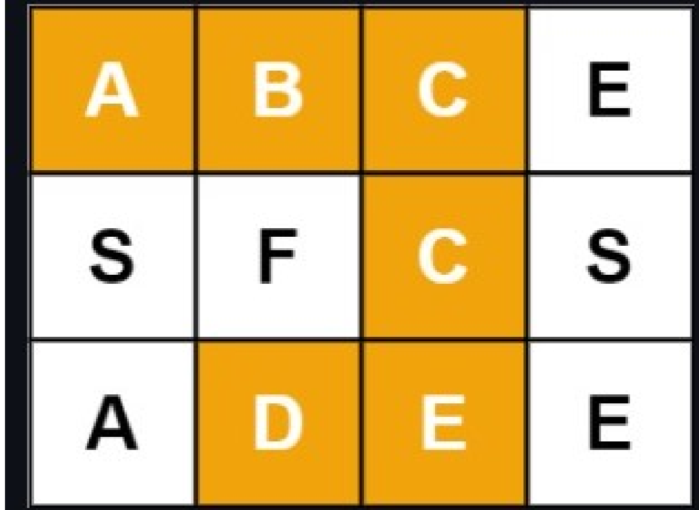
79. Word Search

Medium

Given an `m x n` grid of characters `board` and a string `word`, return `true` *if* `word` *exists in the grid.*

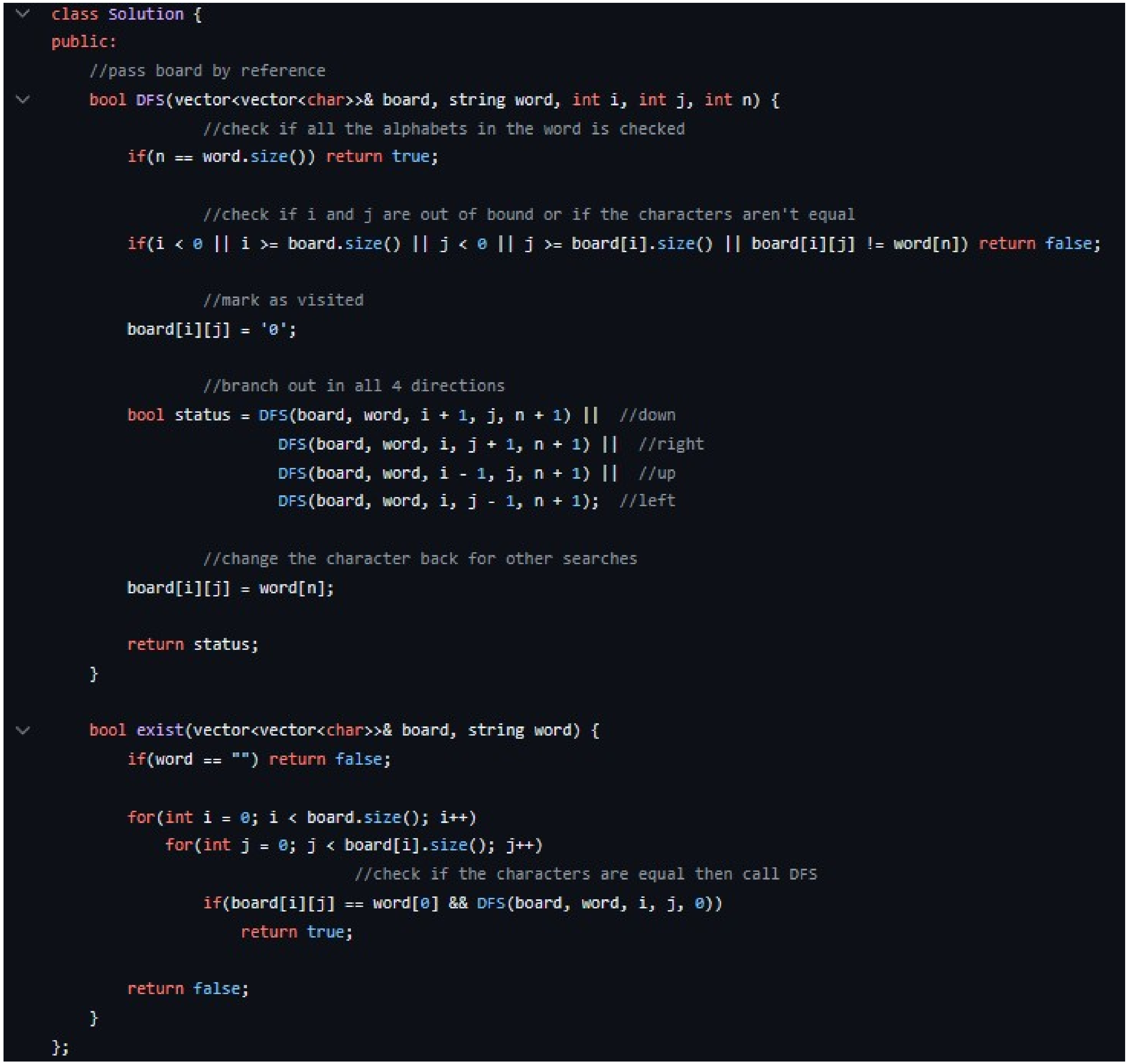
The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

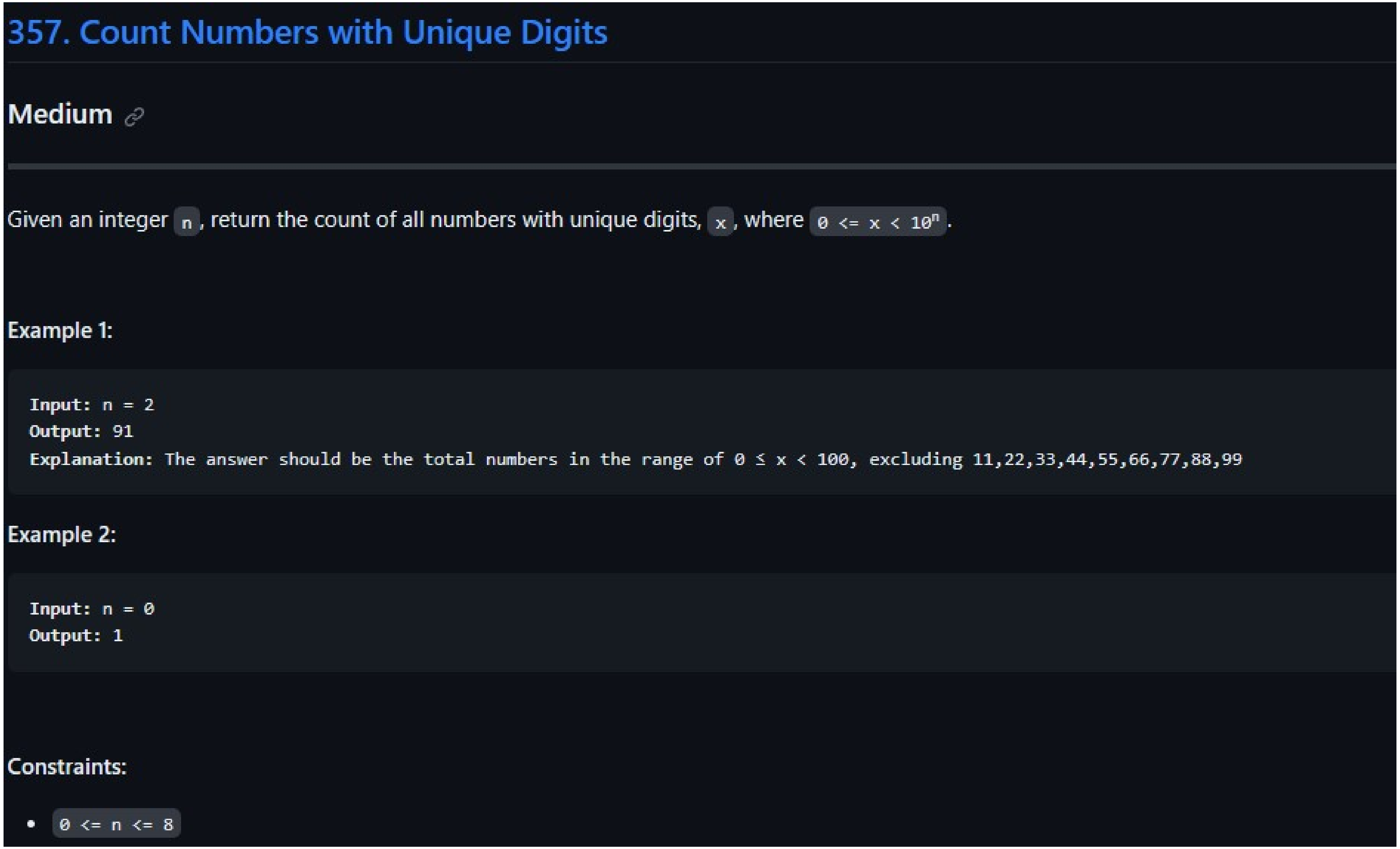
Example 1:



```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"
Output: true
```

Example 2:


```cpp
class Solution {
public:
    //pass board by reference
    bool DFS(vector<vector<char>>& board, string word, int i, int j, int n) {
                //check if all the alphabets in the word is checked
        if(n == word.size()) return true;

                //check if i and j are out of bound or if the characters aren't equal
        if(i < 0 || i >= board.size() || j < 0 || j >= board[i].size() || board[i][j] != word[n]) return false;

                //mark as visited
        board[i][j] = '0';

                //branch out in all 4 directions
        bool status = DFS(board, word, i + 1, j, n + 1) ||  //down
                        DFS(board, word, i, j + 1, n + 1) ||  //right
                        DFS(board, word, i - 1, j, n + 1) ||  //up
                        DFS(board, word, i, j - 1, n + 1);  //left

                //change the character back for other searches
        board[i][j] = word[n];

        return status;
    }

    bool exist(vector<vector<char>>& board, string word) {
        if(word == "") return false;

        for(int i = 0; i < board.size(); i++)
            for(int j = 0; j < board[i].size(); j++)
                                //check if the characters are equal then call DFS
                if(board[i][j] == word[0] && DFS(board, word, i, j, 0))
                    return true;

        return false;
    }
};
```



357. Count Numbers with Unique Digits

Medium

Given an integer `n`, return the count of all numbers with unique digits, `x`, where `0 <= x < 10`^{n}.

Example 1:

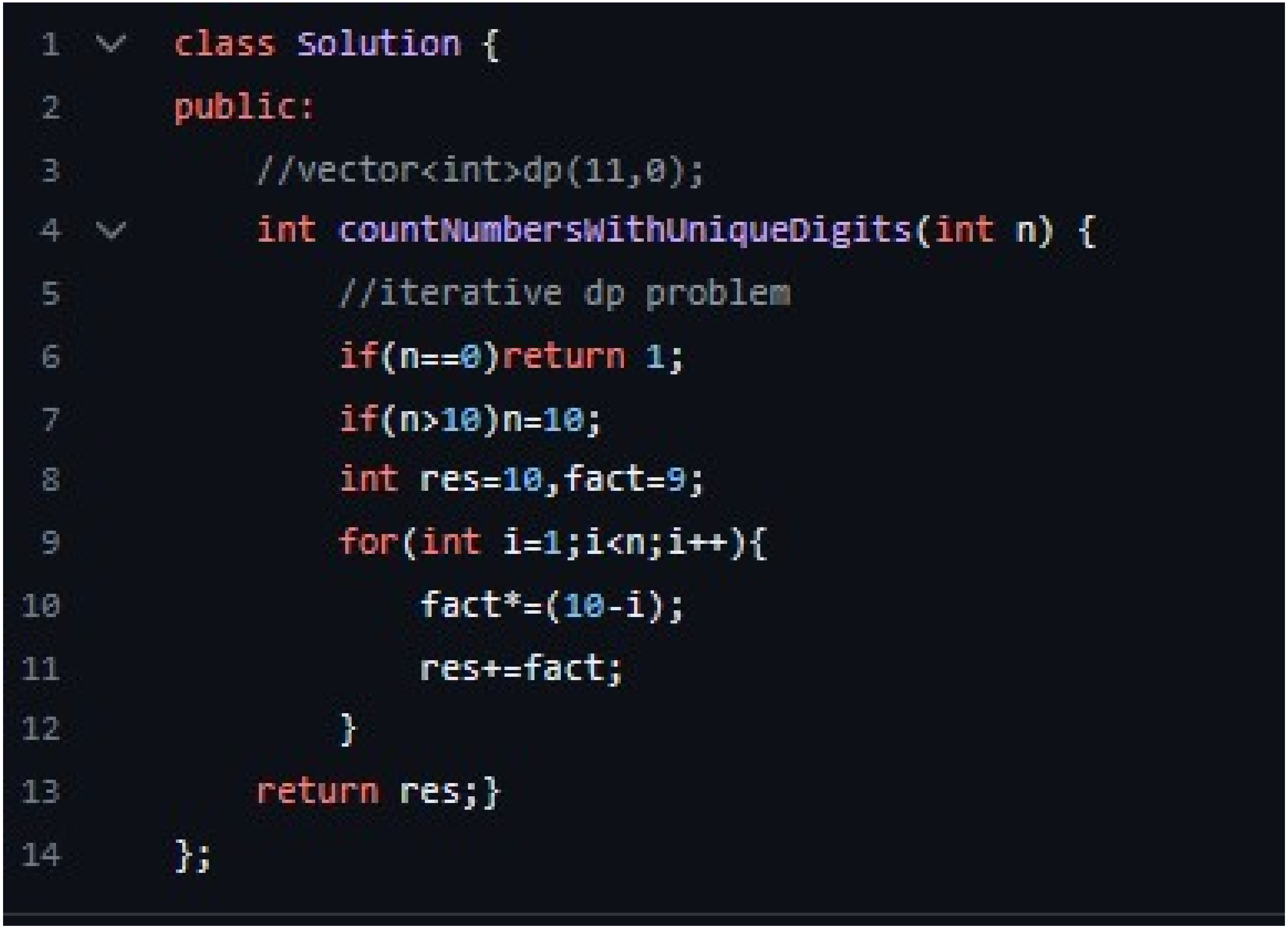
```
Input: n = 2
Output: 91
Explanation: The answer should be the total numbers in the range of 0 ≤ x < 100, excluding 11,22,33,44,55,66,77,88,99
```

Example 2:

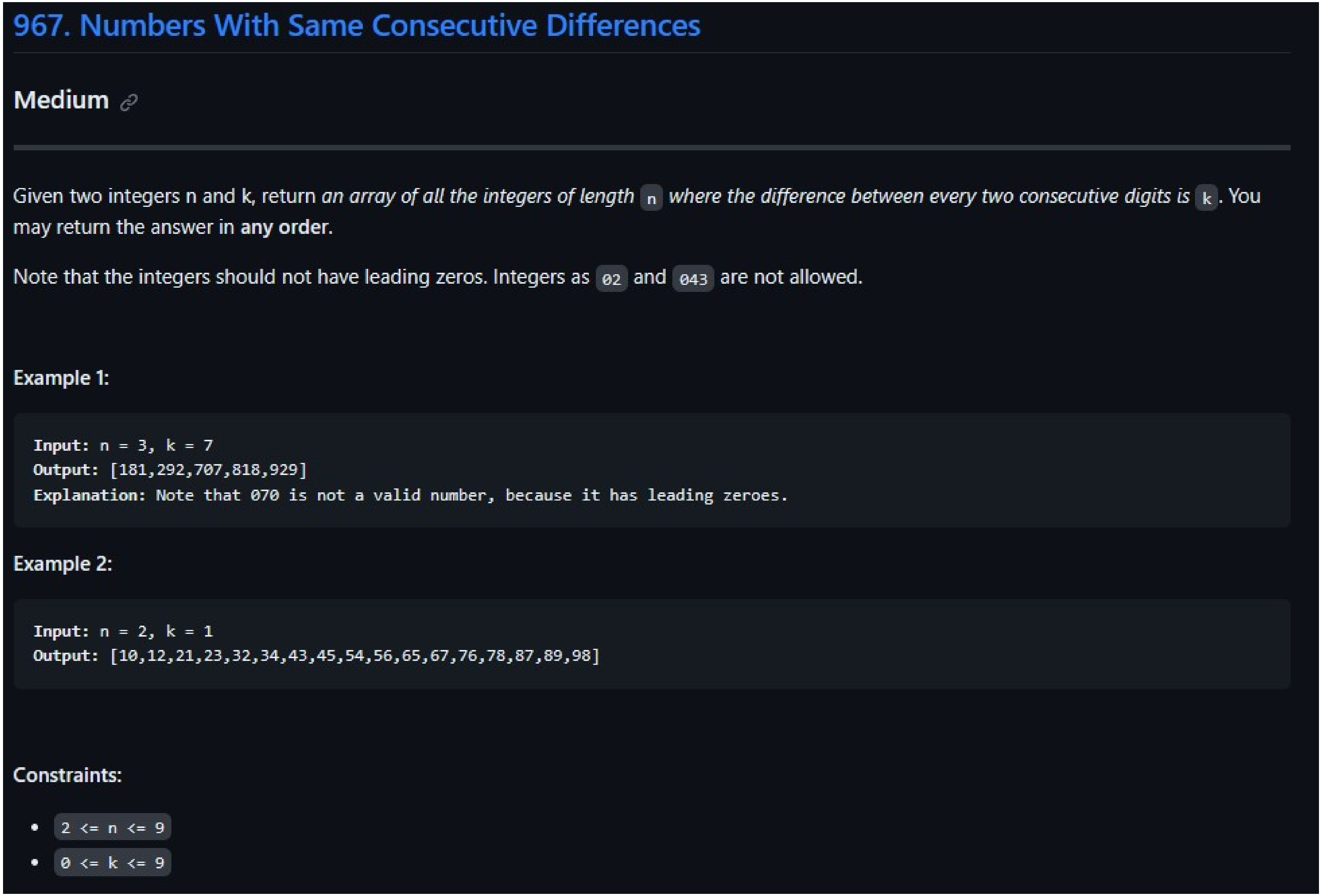
```
Input: n = 0
Output: 1
```

Constraints:

- `0 <= n <= 8`



```cpp
class Solution {
public:
    //vector<int>dp(11,0);
    int countNumbersWithUniqueDigits(int n) {
        //iterative dp problem
        if(n==0)return 1;
        if(n>10)n=10;
        int res=10,fact=9;
        for(int i=1;i<n;i++){
            fact*=(10-i);
            res+=fact;
        }
    return res;}
};
```



967. Numbers With Same Consecutive Differences

Medium

Given two integers n and k, return *an array of all the integers of length* `n` *where the difference between every two consecutive digits is* `k`. You may return the answer in **any order**.

Note that the integers should not have leading zeros. Integers as `02` and `043` are not allowed.

Example 1:

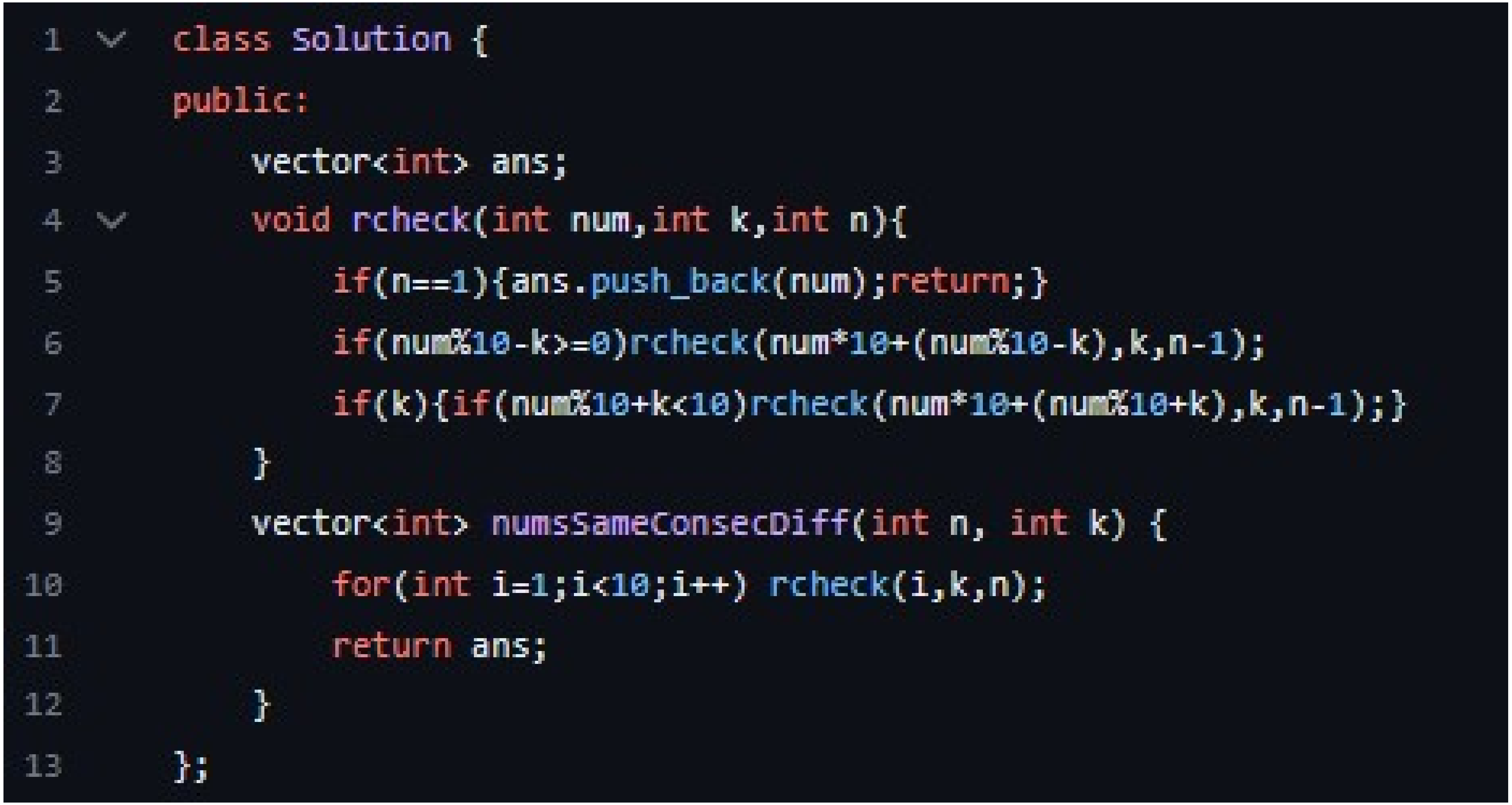
```
Input: n = 3, k = 7
Output: [181,292,707,818,929]
Explanation: Note that 070 is not a valid number, because it has leading zeroes.
```

Example 2:

```
Input: n = 2, k = 1
Output: [10,12,21,23,32,34,43,45,54,56,65,67,76,78,87,89,98]
```

Constraints:

- `2 <= n <= 9`
- `0 <= k <= 9`



```cpp
class Solution {
public:
    vector<int> ans;
    void rcheck(int num,int k,int n){
        if(n==1){ans.push_back(num);return;}
        if(num%10-k>=0)rcheck(num*10+(num%10-k),k,n-1);
        if(k){if(num%10+k<10)rcheck(num*10+(num%10+k),k,n-1);}
    }
    vector<int> numsSameConsecDiff(int n, int k) {
        for(int i=1;i<10;i++) rcheck(i,k,n);
        return ans;
    }
};
```

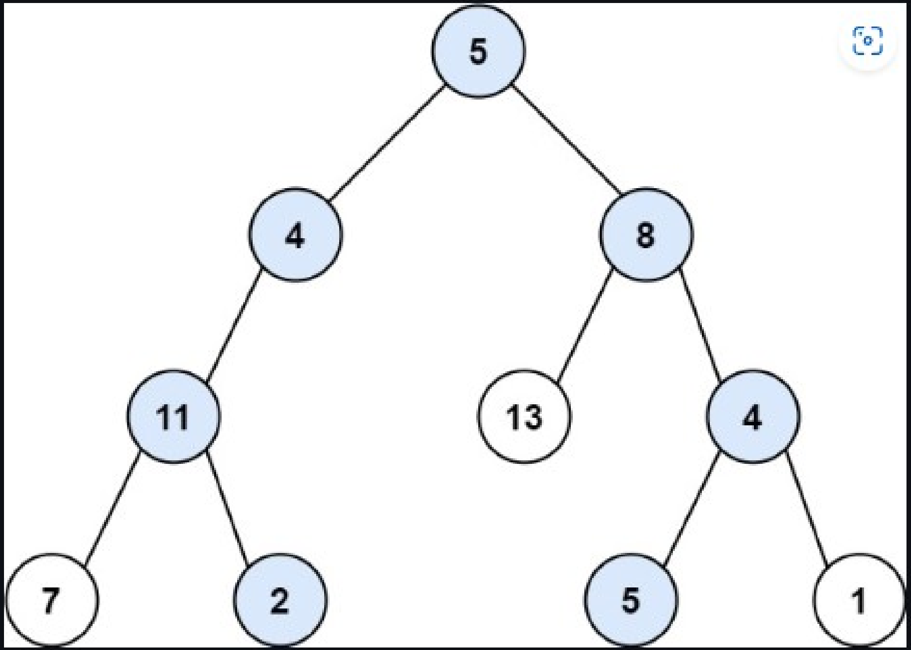
113. Path Sum II

Medium

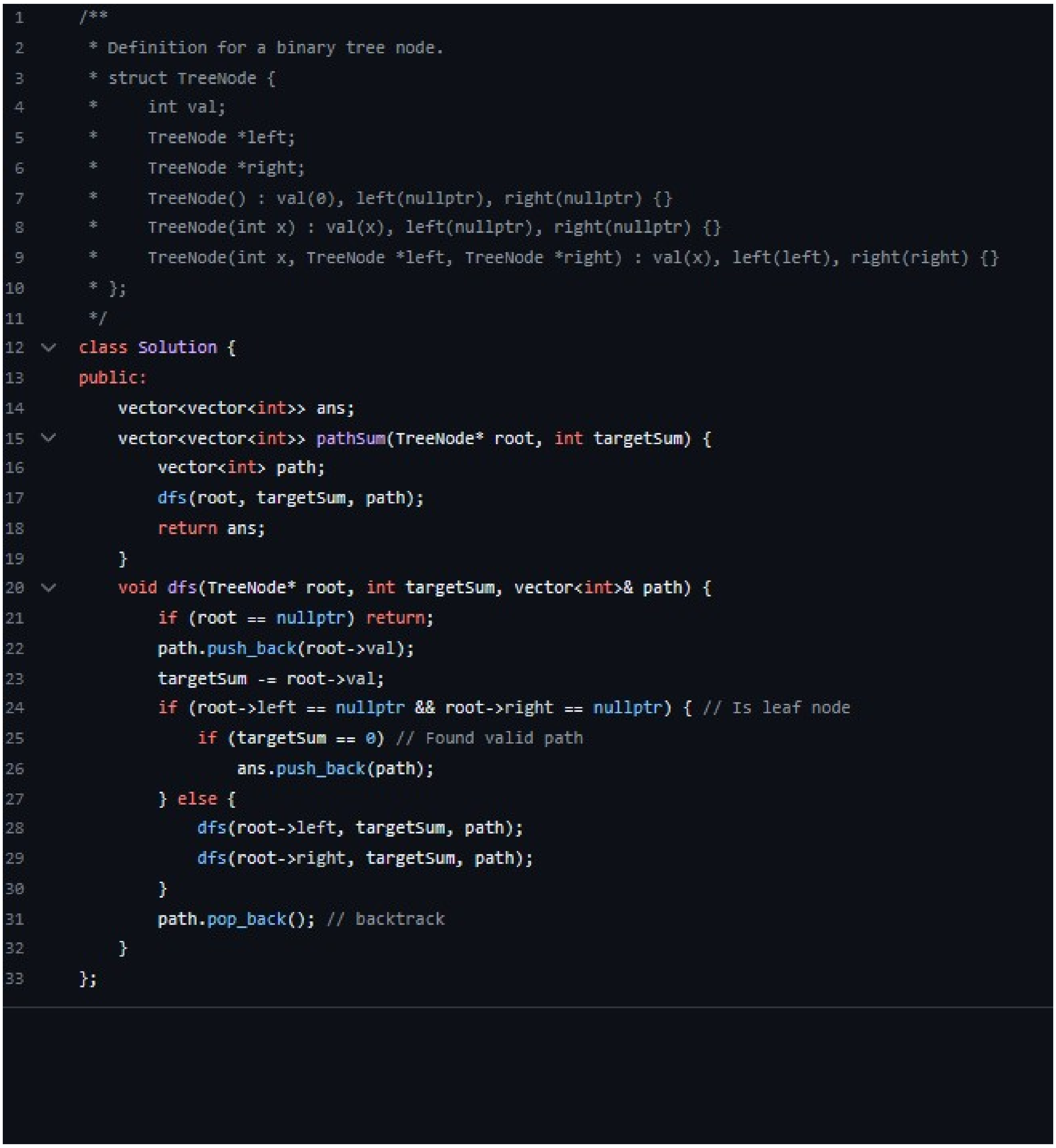
Given the `root` of a binary tree and an integer `targetSum`, return *all **root-to-leaf** paths where the sum of the node values in the path equals* `targetSum`*. Each path should be returned as a list of the node **values**, not node references.*

A **root-to-leaf** path is a path starting from the root and ending at any leaf node. A **leaf** is a node with no children.

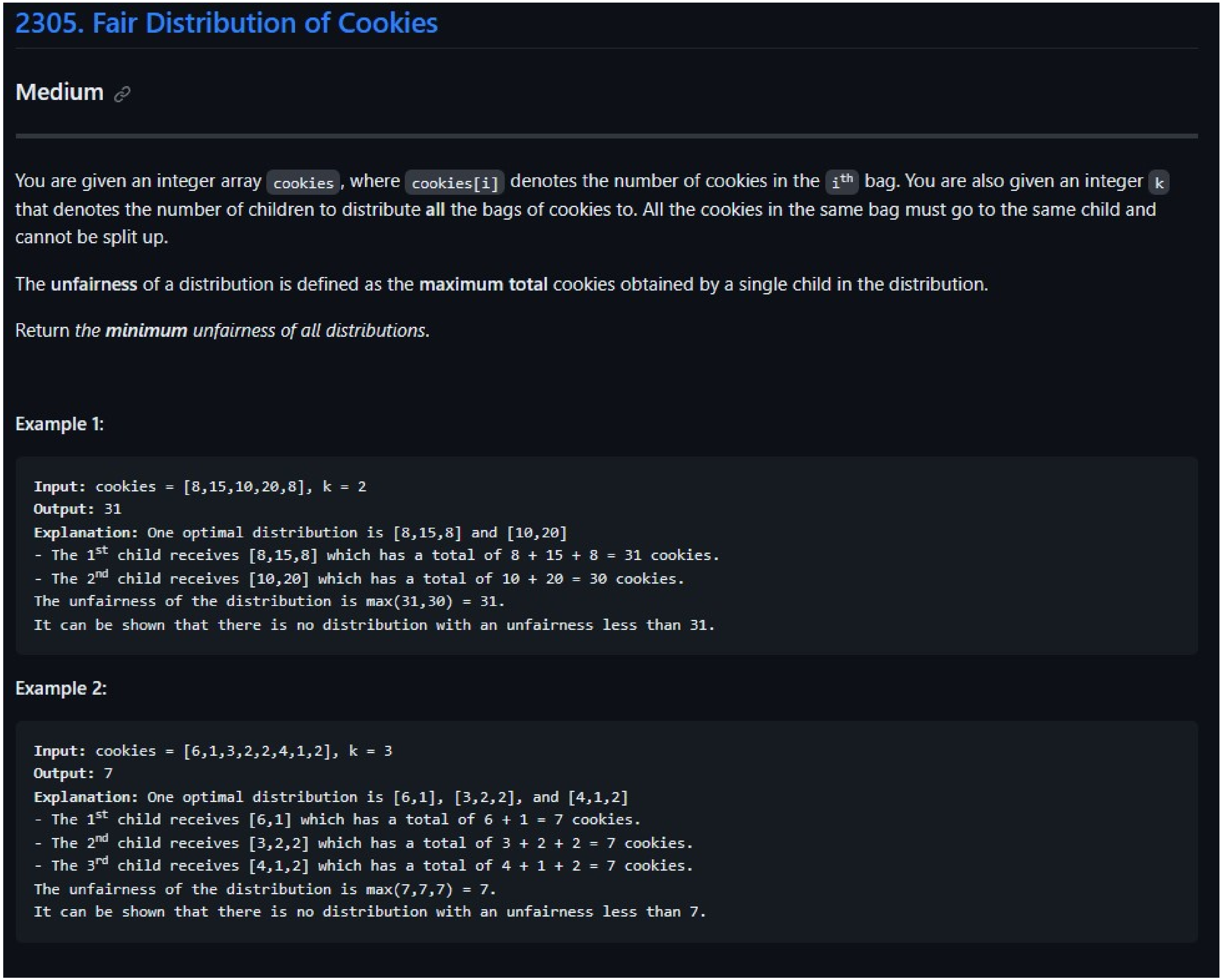
Example 1:



```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
Output: [[5,4,11,2],[5,8,4,5]]
Explanation: There are two paths whose sum equals targetSum:
5 + 4 + 11 + 2 = 22
5 + 8 + 4 + 5 = 22
```



```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<vector<int>> ans;
    vector<vector<int>> pathSum(TreeNode* root, int targetSum) {
        vector<int> path;
        dfs(root, targetSum, path);
        return ans;
    }
    void dfs(TreeNode* root, int targetSum, vector<int>& path) {
        if (root == nullptr) return;
        path.push_back(root->val);
        targetSum -= root->val;
        if (root->left == nullptr && root->right == nullptr) { // Is leaf node
            if (targetSum == 0) // Found valid path
                ans.push_back(path);
        } else {
            dfs(root->left, targetSum, path);
            dfs(root->right, targetSum, path);
        }
        path.pop_back(); // backtrack
    }
};
```



2305. Fair Distribution of Cookies

Medium

You are given an integer array `cookies`, where `cookies[i]` denotes the number of cookies in the i^{th} bag. You are also given an integer `k` that denotes the number of children to distribute **all** the bags of cookies to. All the cookies in the same bag must go to the same child and cannot be split up.

The **unfairness** of a distribution is defined as the **maximum total** cookies obtained by a single child in the distribution.

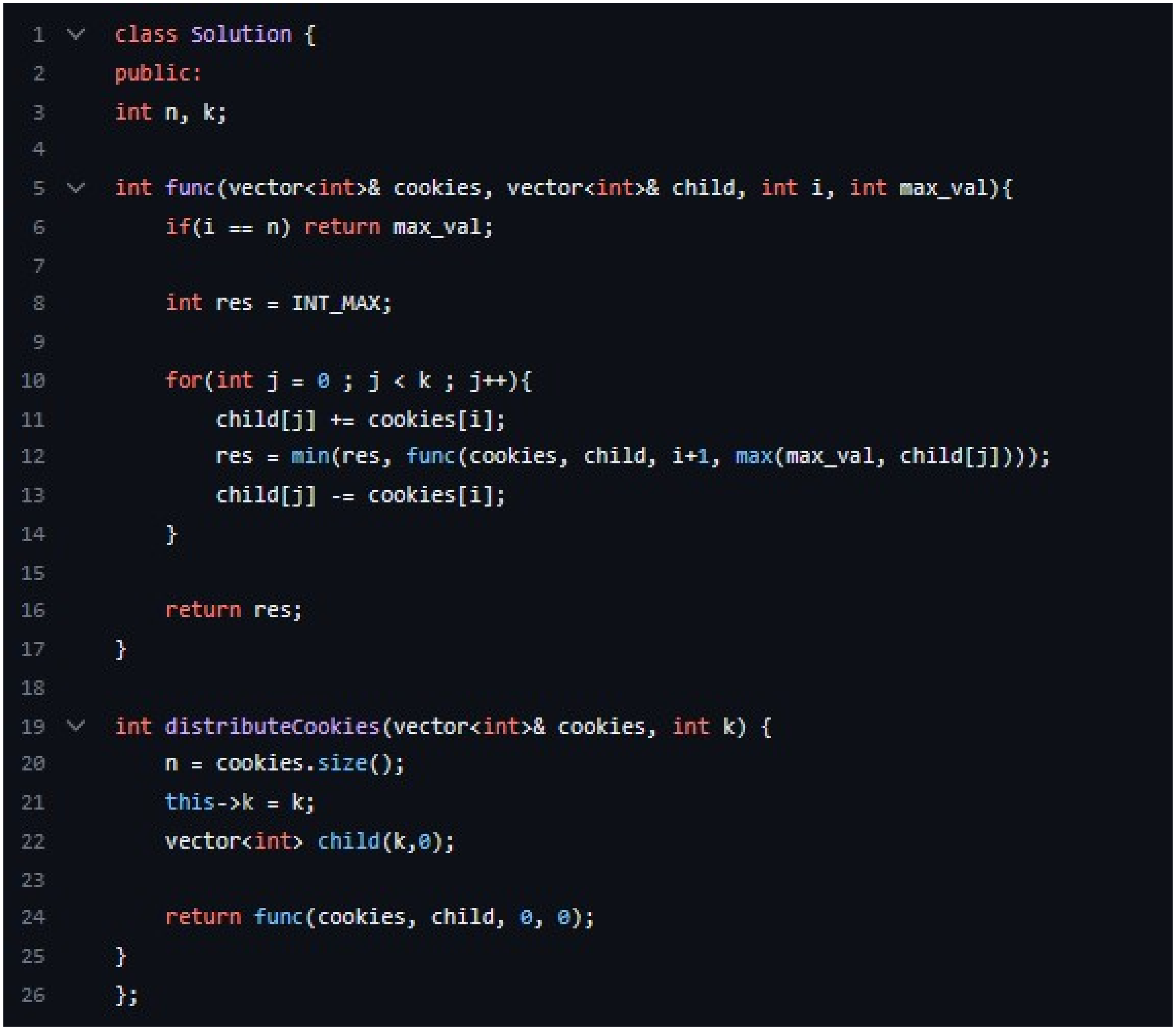
Return *the **minimum** unfairness of all distributions.*

Example 1:

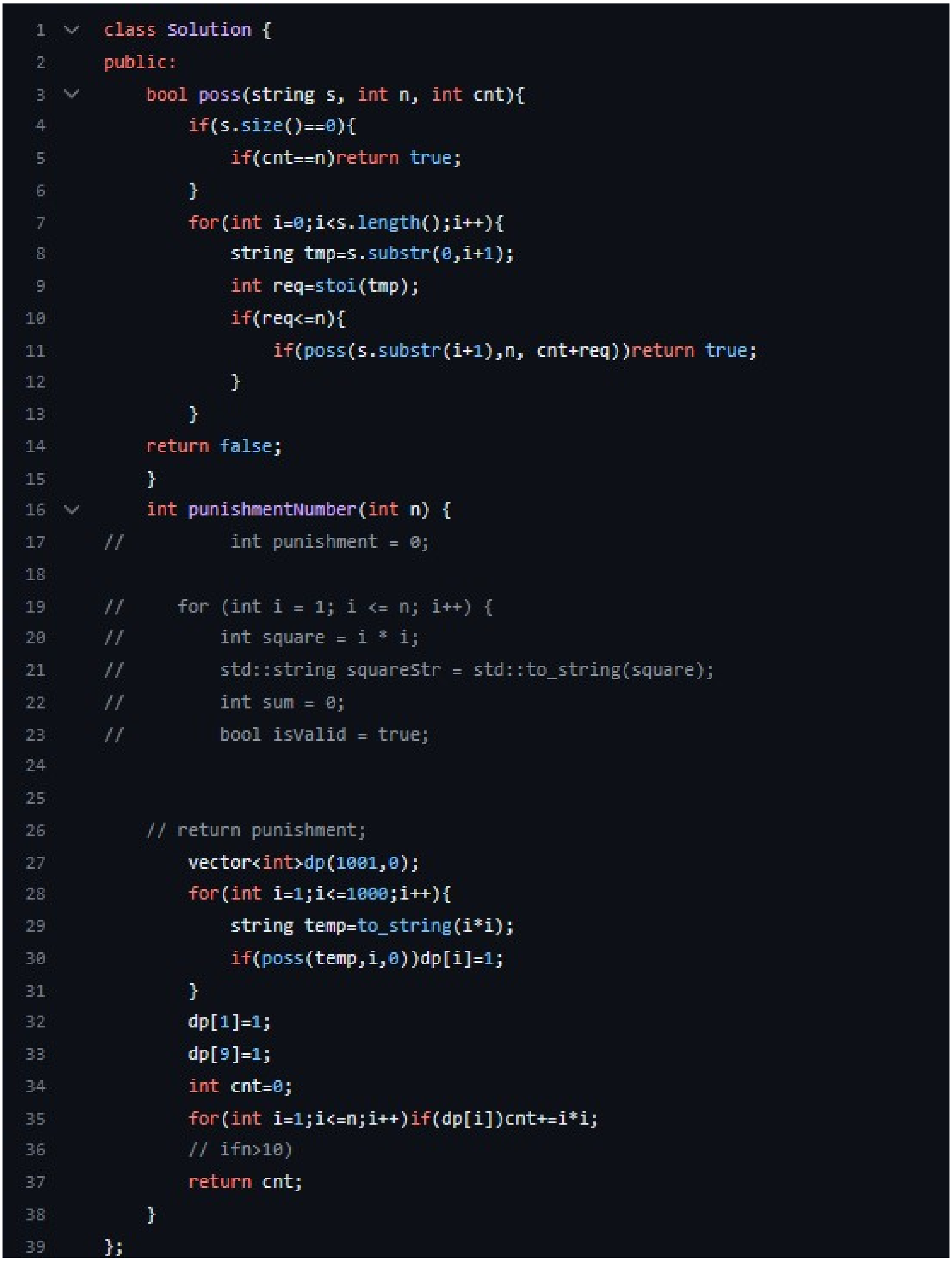
```
Input: cookies = [8,15,10,20,8], k = 2
Output: 31
Explanation: One optimal distribution is [8,15,8] and [10,20]
- The 1st child receives [8,15,8] which has a total of 8 + 15 + 8 = 31 cookies.
- The 2nd child receives [10,20] which has a total of 10 + 20 = 30 cookies.
The unfairness of the distribution is max(31,30) = 31.
It can be shown that there is no distribution with an unfairness less than 31.
```

Example 2:

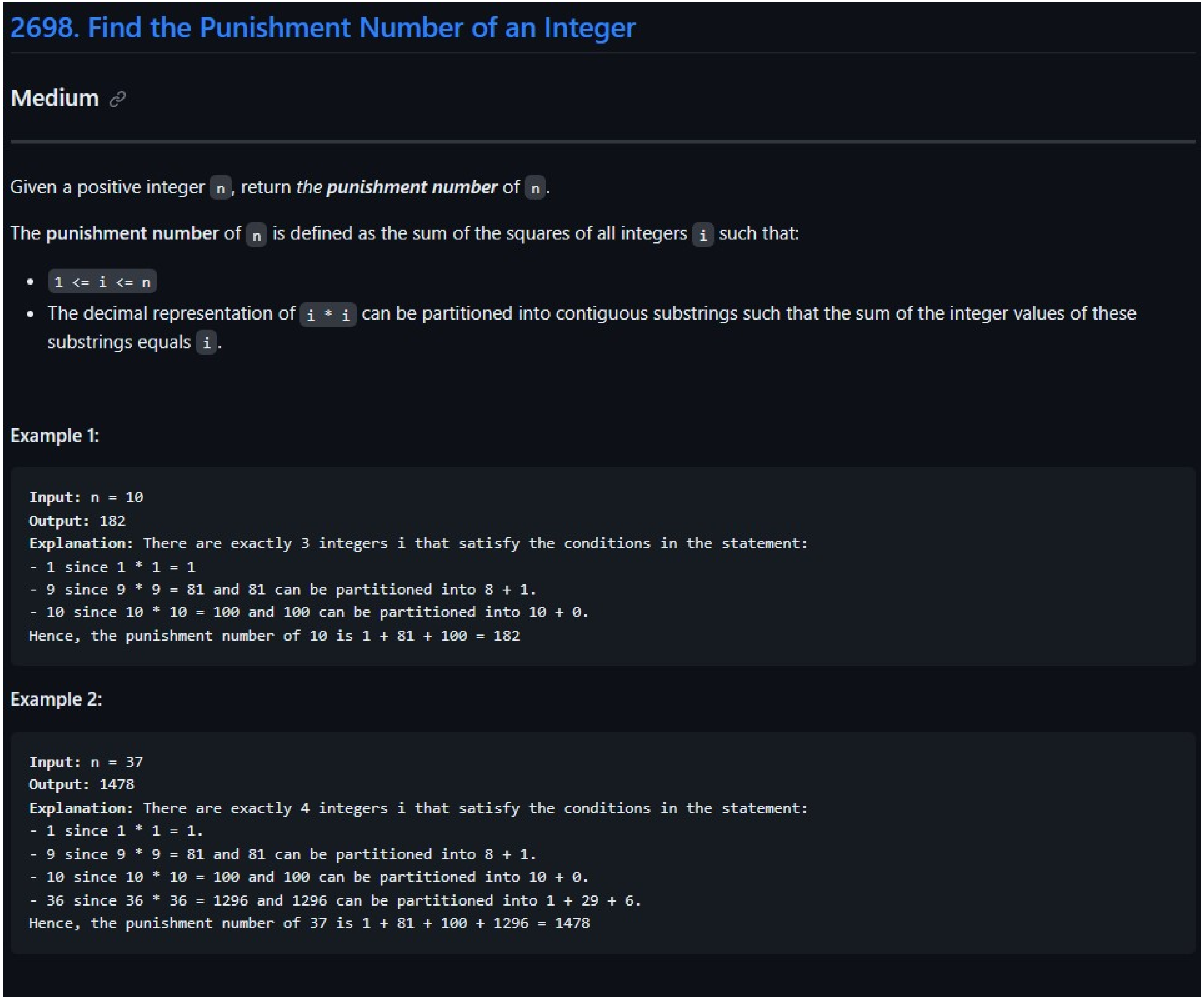
```
Input: cookies = [6,1,3,2,2,4,1,2], k = 3
Output: 7
Explanation: One optimal distribution is [6,1], [3,2,2], and [4,1,2]
- The 1st child receives [6,1] which has a total of 6 + 1 = 7 cookies.
- The 2nd child receives [3,2,2] which has a total of 3 + 2 + 2 = 7 cookies.
- The 3rd child receives [4,1,2] which has a total of 4 + 1 + 2 = 7 cookies.
The unfairness of the distribution is max(7,7,7) = 7.
It can be shown that there is no distribution with an unfairness less than 7.
```



```cpp
class Solution {
public:
int n, k;

int func(vector<int>& cookies, vector<int>& child, int i, int max_val){
    if(i == n) return max_val;

    int res = INT_MAX;

    for(int j = 0 ; j < k ; j++){
        child[j] += cookies[i];
        res = min(res, func(cookies, child, i+1, max(max_val, child[j])));
        child[j] -= cookies[i];
    }

    return res;
}

int distributeCookies(vector<int>& cookies, int k) {
    n = cookies.size();
    this->k = k;
    vector<int> child(k,0);

    return func(cookies, child, 0, 0);
}
};
```



```cpp
class Solution {
public:
    bool poss(string s, int n, int cnt){
        if(s.size()==0){
            if(cnt==n)return true;
        }
        for(int i=0;i<s.length();i++){
            string tmp=s.substr(0,i+1);
            int req=stoi(tmp);
            if(req<=n){
                if(poss(s.substr(i+1),n, cnt+req))return true;
            }
        }
    return false;
    }
    int punishmentNumber(int n) {
//          int punishment = 0;

//     for (int i = 1; i <= n; i++) {
//         int square = i * i;
//         std::string squareStr = std::to_string(square);
//         int sum = 0;
//         bool isValid = true;


    // return punishment;
        vector<int>dp(1001,0);
        for(int i=1;i<=1000;i++){
            string temp=to_string(i*i);
            if(poss(temp,i,0))dp[i]=1;
        }
        dp[1]=1;
        dp[9]=1;
        int cnt=0;
        for(int i=1;i<=n;i++)if(dp[i])cnt+=i*i;
        // ifn>10)
        return cnt;
    }
};
```



2698. Find the Punishment Number of an Integer

Medium

Given a positive integer `n`, return *the* ***punishment number*** of `n`.

The **punishment number** of `n` is defined as the sum of the squares of all integers `i` such that:

- `1 <= i <= n`
- The decimal representation of `i * i` can be partitioned into contiguous substrings such that the sum of the integer values of these substrings equals `i`.

Example 1:

```
Input: n = 10
Output: 182
Explanation: There are exactly 3 integers i that satisfy the conditions in the statement:
- 1 since 1 * 1 = 1
- 9 since 9 * 9 = 81 and 81 can be partitioned into 8 + 1.
- 10 since 10 * 10 = 100 and 100 can be partitioned into 10 + 0.
Hence, the punishment number of 10 is 1 + 81 + 100 = 182
```

Example 2:

```
Input: n = 37
Output: 1478
Explanation: There are exactly 4 integers i that satisfy the conditions in the statement:
- 1 since 1 * 1 = 1.
- 9 since 9 * 9 = 81 and 81 can be partitioned into 8 + 1.
- 10 since 10 * 10 = 100 and 100 can be partitioned into 10 + 0.
- 36 since 36 * 36 = 1296 and 1296 can be partitioned into 1 + 29 + 6.
Hence, the punishment number of 37 is 1 + 81 + 100 + 1296 = 1478
```