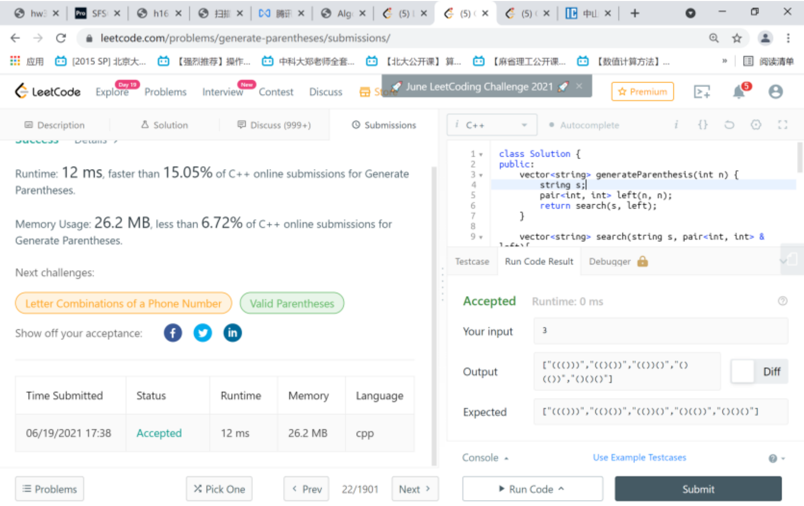
1. 算法思路：递归，回溯法，枚举

   复杂度分析：$\frac{1}{n+1}\binom{2n}{n}$，n 为括号的对数

```
class Solution {
    public:
        vector<string> generateParenthesis(int n) {
            string s;
            pair<int, int> left(n, n);
            return search(s, left);
        }

        vector<string> search(string s, pair<int, int> & left){
            vector<string> v, v1, v2;
            if ( left.first == 0 && left.second == 0 ){
                v.push_back(s);
                return v;
            }
            if ( left.first > 0 ){
                left.first--;
                v1 = search(s+'(', left);
                left.first++;
            }
            if ( left.second > left.first ){
                left.second--;
                v2 = search(s+')', left);
                left.second++;
            }
            set_union(v1.begin(), v1.end(), v2.begin(), v2.end(),
                    std::back_inserter(v));
            return v;
        }
};
```



Runtime: 12 ms, faster than 15.05% of C++ online submissions for Generate Parentheses.

Memory Usage: 26.2 MB, less than 6.72% of C++ online submissions for Generate Parentheses.

Next challenges:

Letter Combinations of a Phone Number　Valid Parentheses

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
06/19/2021 17:38	Accepted	12 ms	26.2 MB	cpp

```
class Solution {
public:
    vector<string> generateParenthesis(int n) {
        string s;
        pair<int, int> left(n, n);
        return search(s, left);
    }

    vector<string> search(string s, pair<int, int> &
left){
```

Accepted　Runtime: 0 ms

Your input: 3

Output: ["((()))","(()())","(())()","()(())","()()()"]

Diff

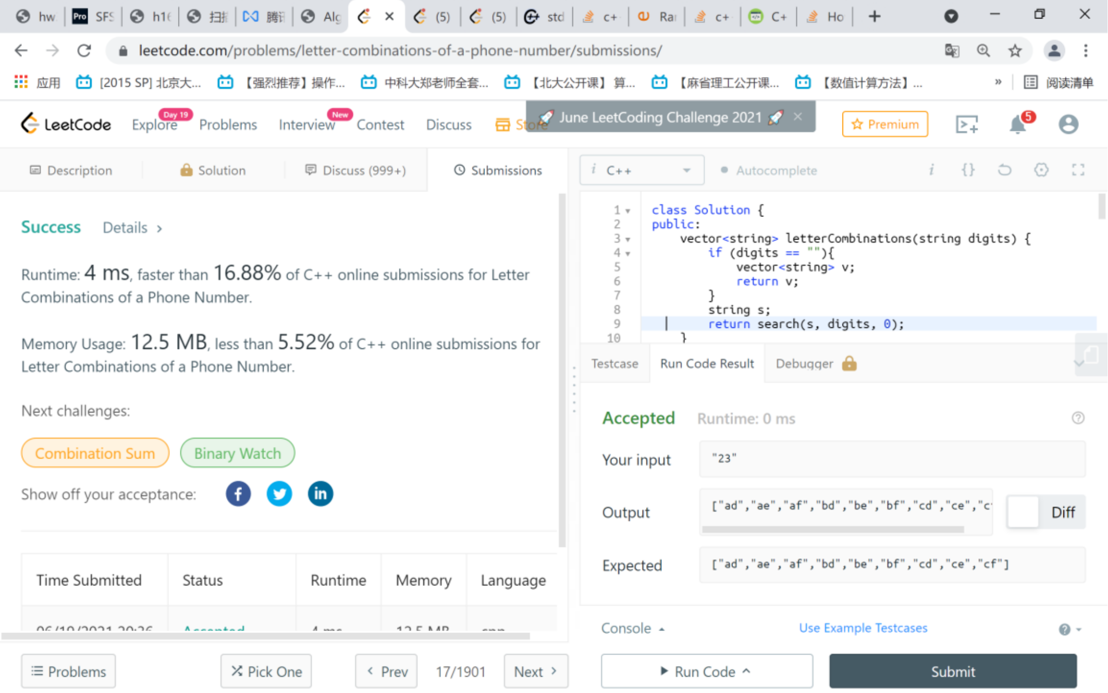
Expected: ["((()))","(()())","(())()","()(())","()()()"]

2. 算法思路：递归，回溯法，枚举

   复杂度分析：O(4^N)，N是digits字符串的长度

```
class Solution {
    public:
        vector<string> letterCombinations(string digits) {
            if (digits == ""){
                vector<string> v;
                return v;
            }
            string s;
            return search(s, digits, 0);
        }

        vector<string> search(string s, string & digits, int i){
            if ( i>=digits.length() ){
                vector<string> v;
                v.push_back(s);
                return v;
            }
            vector<string> v, w, x, y, z;
            char c = 'a' + (digits[i]-'2')*3;
            switch( digits[i] ){
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                    x = search(s+c, digits, i+1);
                    y = search(s+(char)(c+1), digits, i+1);
                    z = search(s+(char)(c+2), digits, i+1);
                    break;
                case '7':
                    x = search(s+'p', digits, i+1);
                    y = search(s+'q', digits, i+1);
                    z = search(s+'r', digits, i+1);
                    w = search(s+'s', digits, i+1);
                    break;
                case '8':
                    x = search(s+'t', digits, i+1);
                    y = search(s+'u', digits, i+1);
                    z = search(s+'v', digits, i+1);
                    break;
                default:
                    x = search(s+'w', digits, i+1);
                    y = search(s+'x', digits, i+1);
                    z = search(s+'y', digits, i+1);
                    w = search(s+'z', digits, i+1);
            }
            set_union(x.begin(), x.end(), y.begin(), y.end(),
                    std::back_inserter(v));
            set_union(z.begin(), z.end(), w.begin(), w.end(),
                    std::back_inserter(v));
            return v;
        }
};
```



Success Details ›

Runtime: 4 ms, faster than 16.88% of C++ online submissions for Letter Combinations of a Phone Number.

Memory Usage: 12.5 MB, less than 5.52% of C++ online submissions for Letter Combinations of a Phone Number.

Next challenges:

Combination Sum Binary Watch

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {
        if (digits == ""){
            vector<string> v;
            return v;
        }
        string s;
        return search(s, digits, 0);
    }
```

Accepted Runtime: 0 ms

Your input "23"

Output ["ad","ae","af","bd","be","bf","cd","ce","c

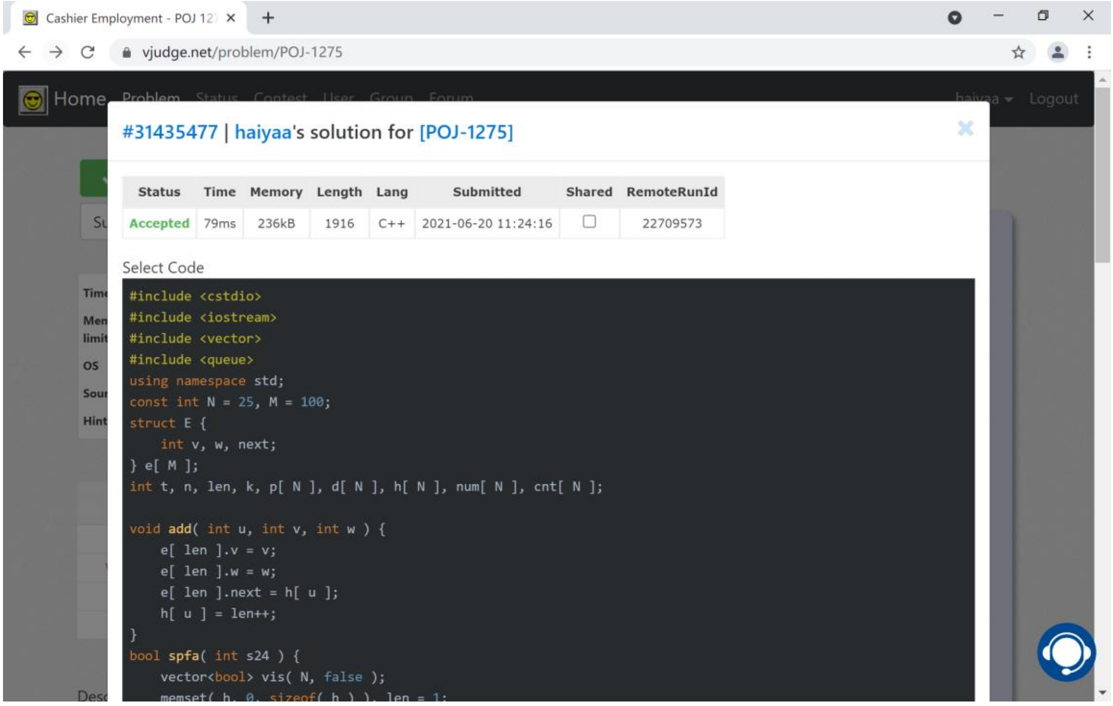
Expected ["ad","ae","af","bd","be","bf","cd","ce","cf"]

3. 算法思路：用 SPFA 解差分约束

   复杂度分析：O(|V|*|E|)，V 为构建的图中的节点数，E 为构建的图中的边数

```
#include <cstdio>
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
const int N = 25, M = 100;
struct E {
    int v, w, next;
} e[ M ];
int t, n, len, k, p[ N ], d[ N ], h[ N ], num[ N ], cnt[ N ];

void add( int u, int v, int w ) {
    e[ len ].v = v;
    e[ len ].w = w;
    e[ len ].next = h[ u ];
    h[ u ] = len++;
}
bool spfa( int s24 ) {
    vector<bool> vis( N, false );
    memset( h, 0, sizeof( h ) ), len = 1;
    memset( d, -0x3f, sizeof( d ) );
    memset( cnt, 0, sizeof( cnt ) );
    d[ 0 ] = 0;
    for ( int i = 1; i <= 24; i++ ) {
        add( i - 1, i, 0 ), add( i, i - 1, -num[ i ] );
        if ( i >= 8 ) {
            add( i - 8, i, p[ i ] );
        }
        else {
            add( i + 16, i, -s24 + p[ i ] );
        }
    }
    add( 0, 24, s24 ), add( 24, 0, -s24 );
    queue<int> q;
    q.push( 0 );
    while ( !q.empty() ) {
        int u = q.front();
        q.pop();
        vis[ u ] = false;
        for ( int j = h[ u ]; j; j = e[ j ].next ) {
            int v = e[ j ].v;
            int w = d[ u ] + e[ j ].w;
            if ( w > d[ v ] ) {
                d[ v ] = w;
                cnt[ v ] = cnt[ u ] + 1;
                if ( cnt[ v ] >= 25 ) return true;
                if ( !vis[ v ] ) q.push( v ), vis[ v ] = true;
            }
        }
    }
    return false;
}
int main() {
    cin >> t;
    while ( t-- ) {
        memset( num, 0, sizeof( num ) );
        for ( int i = 1; i <= 24; i++ ){
            cin >> p[ i ];
        }
        cin >> n;
        for ( int i = 1; i <= n; i++ ) {
            cin >> k;
            num[ ++k ]++;
        }
        bool ok = false;
        for ( int i = 1; i <= n; i++ ) {
            if ( !spfa( i ) ) {
                cout << d[ 24 ] << endl;
                ok = true;
                break;
            }
        }
        if ( !ok ){
            cout << "No Solution\n";
        }
    }
}
```



#31435477 | haiyaa's solution for [POJ-1275]

Status	Time	Memory	Length	Lang	Submitted	Shared	RemoteRunId
Accepted	79ms	236kB	1916	C++	2021-06-20 11:24:16	□	22709573

Select Code

```
#include <cstdio>
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
const int N = 25, M = 100;
struct E {
    int v, w, next;
} e[ M ];
int t, n, len, k, p[ N ], d[ N ], h[ N ], num[ N ], cnt[ N ];

void add( int u, int v, int w ) {
    e[ len ].v = v;
    e[ len ].w = w;
    e[ len ].next = h[ u ];
    h[ u ] = len++;
}
bool spfa( int s24 ) {
    vector<bool> vis( N, false );
    memset( h, 0, sizeof( h ) ), len = 1;
```

4. P 问题：所有能在多项式时间内解决的搜索问题叫做 P 问题。

NP 问题：所有能在多项式时间内验证一个解是否正确的搜索问题叫做 NP 问题。

NPC 问题：一个搜索问题是搜索问题，当且仅当所有的搜索问题都能归约到该问题。

证明一个问题 Q 是 NP 难问题的方法：将另一个 NP 难问题归约到 Q