

1、

算法思路：

使用回溯算法，在保证插入数量不超过 n 和插入（的数量大于）这两个条件下递归调用就可以了。

复杂度分析：

时间复杂度： $O(\frac{4^n}{\sqrt{n}})$

空间复杂度： $O(n)$

代码：

```
class Solution {
public:
    vector<string> generateParenthesis(int n) {
        vector<string> res;
        int lc = 0, rc = 0;
        dfs(res, "", n, lc, rc);
        return res;
    }
    void dfs(vector<string>& res, string path, int n, int lc, int rc) {
        if (rc > lc || lc > n || rc > n) return;
        if (lc == rc && lc == n) {
            res.push_back(path);
            return;
        }
        dfs(res, path + '(', n, lc + 1, rc);
        dfs(res, path + ')', n, lc, rc + 1);
    }
};
```

截图：

The screenshot shows a LeetCode submission page for the 'Generate Parentheses' problem. The submission result table indicates that the solution was accepted (通过) with a runtime of 8 ms and a memory usage of 16.1 MB. The C++ code is displayed in a text editor, showing the same code as provided in the previous block. The page also includes navigation links for problem lists, previous/next questions, and a control panel.

提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	8 ms	16.1 MB	C++

2、

算法思路：

同样采用回溯算法，每次对于一个数都有选和不选两种选择，递归调用的两个 if 条件是终止条件，当 $target=0$ 时将答案存入数组中，最后返回 ans 即可。

复杂度分析：

时间复杂度： $O(S)$ ， S =所有可行解的长度之和

空间复杂度： $O(target)$

代码：

```
class Solution {
public:
    void dfs(vector<int> &candidates,vector<vector<int> > &ans,int idx,
int target,vector<int> &temp)//传入 candidate 数组、答案、选取到哪个位置、目
标值 d 还剩余多少没选,当前已经选取的数组
    {
        if(idx==candidates.size())
        {
            return;
        }
        if(target==0)
        {
            ans.emplace_back(temp);
            return;
        }
        else{
            //不选择
            dfs(candidates,ans,idx+1,target,temp);
            //选择
            if(target-candidates[idx]>=0)
            {
                temp.emplace_back(candidates[idx]);
                dfs(candidates,ans,idx,target-candidates[idx],temp);
                temp.pop_back();
            }
        }
    }
    vector<vector<int>> combinationSum(vector<int>& candidates, int tar
get) {
        vector<vector<int> > ans;
        vector<int> temp;
        dfs(candidates,ans,0,target,temp);
        return ans;
    }
};
```

截图：

力扣 探索 题库 圈子 竞赛 面试 职位 商店

新功能 下载 App Plus 会员 中 模拟面试

题目描述 讨论 (823) 题解 (961) 提交记录

提交时间	提交结果	运行时间	内存消耗	语言
7天前	通过	12 ms	16.8 MB	C++

```

1 class Solution {
2 public:
3     void dfs(vector<int> &candidates, vector<vector<int> > &ans, int idx, int target, vector<int>
&temp) //传入candidates数组、答案、选取到哪个位置、目标值d还剩余多少没选,当前已经选取的数组
4     {
5         if(idx==candidates.size())
6         {
7             return;
8         }
9         if(target==0)
10        {
11            ans.emplace_back(temp);
12            return;
13        }
14        else{
15            //不选择
16            dfs(candidates,ans,idx+1,target,temp);
17            //选择
18            if(target-candidates[idx]>=0)
19            {
20                temp.emplace_back(candidates[idx]);
21                dfs(candidates,ans,idx,target-candidates[idx],temp);
22                temp.pop_back();
23            }
24        }
25    }
26    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {

```

题目列表 随... 39/1790 上一题 下一题 控制台 贡献

执行代码 提交

3、

算法思路：

每次以双指针为左右边界（也就是「数组」的左右边界）计算出的容量中的最大值即可。

复杂度分析：

时间复杂度：O(n)，n 为数组长度

空间复杂度：O(1)

代码：

```

class Solution {
public:
    int maxArea(vector<int>& height) {
        int res = 0;
        int i = 0;
        int j = height.size() - 1;
        while (i < j) {
            int area = (j - i) * min(height[i], height[j]);
            res = max(res, area);
            if (height[i] < height[j]) {
                i++;
            } else {
                j--;
            }
        }
        return res;
    }
};

```

截图：

力扣 探索 题库 圈子 竞赛 面试 职位 商店

新功能 下载 App Plus 会员 中

题目描述 评论 (1.1k) 题解 (1562) 提交记录

执行结果: **通过** 显示详情

执行用时: 16 ms, 在所有 C++ 提交中击败了 83.27% 的用户

内存消耗: 7.3 MB, 在所有 C++ 提交中击败了 68.52% 的用户

标签: 数组, 双指针, 贪心

与题解, 分享我的解题思路

提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	16 ms	7.3 MB	C++
几秒前	编译出错	N/A	N/A	C++

```

1 class Solution {
2 public:
3     int maxArea(vector<int>& height) {
4         int res = 0;
5         int i = 0;
6         int j = height.size() - 1;
7         while (i < j) {
8             int area = (j - i) * min(height[i], height[j]);
9             res = max(res, area);
10            if (height[i] < height[j]) {
11                i++;
12            } else {
13                j--;
14            }
15        }
16        return res;
17    }
18 };

```

题目列表 随机 11/1790 上一题 下一题 控制台 贡献

执行代码 提交

4、

算法思路:

差分约束的思想, 使用前缀和转化推导不等式进行解决。

复杂度分析:

时间复杂度: $O(n^2)$

空间复杂度: $O(n)$

代码:

```

#include <cstdio>
#include <cstring>
#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];
bool vis[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) {
    memset(h, 0, sizeof(h)), len = 1;
    memset(d, -0x3f, sizeof(d)); // 最小值求最长路
    memset(cnt, 0, sizeof(cnt));
    memset(vis, false, sizeof(vis));
    d[0] = 0; // 从 0 可以遍历所有的边
    // 根据枚举的 s24 进行建图

```

```

    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]);
        }
    }
}

//由于我们还需要设置 s24 为定值 所以创建 s24<=k s24 >= k 这样 s24 就是定
值了(k 是枚举的 s24 的值)
add(0, 24, s24), add(24, 0, -s24);
//spfa 看是否有解
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() {
    scanf("%d", &t);
    while (t--) {
        memset(num, 0, sizeof(num)); //num[i] 代表 i 时刻的人数
        for (int i = 1; i <= 24; i++) scanf("%d", &p[i]);
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &k);
            num[++k]++;
        }
        //枚举一下 s24 的值
        bool ok = false;
        for (int i = 1; i <= n; i++) {

```

```

        if (!spfa(i)) {
            //找到一组结果
            printf("%d\n", d[24]); //输出一下一共需要的最少人数
            ok = true;
            break;
        }
    }
    if (!ok) printf("No Solution\n");
}
return 0;
}

```

截图：

The screenshot shows a submission page for problem #27315048, titled "Condor_kid's solution for [POJ-1275]". The submission status is "Accepted". The table below shows the submission details:

Status	Time	Memory	Length	Lang	Submitted	RemoteRunId
Accepted	79ms	112kB	1626	C++	2020-09-17 03:44:44	21983549

Below the table, the code is displayed under the heading "Select Code". The code is a C++ implementation of a Shortest Path Faster Algorithm (SPFA) to find the shortest path from node 1 to node 24 in a graph with 25 nodes and 100 edges.

```

#include<iostream>
#include<cstdio>
#include<cstring>
#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];
bool vis[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) {
    memset(h, 0, sizeof(h)), len = 1;

```

5、

NP 问题：能在多项式时间内验证得出一个正确解的问题。

P 问题是 NP 问题的子集。

先证明它至少是一个 NP 问题，再证明其中一个已知的 NP-完全性问题能约化到它。