

# 算法设计与应用基础 作业4

19335174 施天予

## 1. 【Leetcode22】括号生成

### 算法思路

这道题可以用回溯法。生成n组括号，可以考虑先放左括号，再放右括号，每次在放的过程中，必须保证左括号的个数小于等于右括号的个数。运用一个backtrack函数，如果未放的右括号个数小于左括号个数，或者左括号个数或右括号个数小于0，就return；如果剩余左括号个数和右括号个数都为0，那么这是一种解。在放括号时，先放左括号，然后记左括号个数减1，递归调用backtrack，同理右括号也是如此。需要注意的是，每当一种括号的情况结束后，要把它“取出来”，这样才可以形成回溯。

### 复杂度分析

时间复杂度： $O(n \cdot 2^{2n})$ ，n为括号个数。

空间复杂度： $O(n)$ ，n为括号个数。

### 代码

```
class Solution {
public:
    vector<string> generateParenthesis(int n) {
        if (n == 0) return {};
        vector<string> res;
        string track;
        backtrack(n, n, track, res);
        return res;
    }
    void backtrack(int left, int right, string& track, vector<string>& res) {
        if (right < left) return;
        if (left < 0 || right < 0) return;
        if (left == 0 && right == 0) {
            res.push_back(track);
            return;
        }
        track.push_back('(');
        backtrack(left - 1, right, track, res);
        track.pop_back();
        track.push_back(')');
        backtrack(left, right - 1, track, res);
        track.pop_back();
    }
};
```

## 截图

执行结果：通过 [显示详情](#)

执行用时：4 ms，在所有 C++ 提交中击败了 73.35% 的用户

内存消耗：11.2 MB，在所有 C++ 提交中击败了 73.23% 的用户

炫耀一下：

[与题解，分享我的解题思路](#)

提交结果	执行用时	内存消耗	语言	提交时间	备注
通过	4 ms	11.2 MB	C++	2021/06/04 17:16	<a href="#">添加备注</a>

## 2. 【Leetcode17】电话号码的字母组合

### 算法思路

这道题可以用回溯算法。首先用哈希表存储每个数字对应的所有可能的字母。

回溯过程中维护一个字符串，表示已有的字母排列（如果未遍历完电话号码的所有数字，则已有的字母排列是不完整的）。该字符串初始为空。每次取电话号码的一位数字，从哈希表中获得该数字对应的所有可能的字母，并将其中的一个字母插入到维护的字符串后面，然后继续处理电话号码的后一位数字，直到处理完电话号码中的所有数字，即得到一个完整的字母排列。然后进行回退操作，遍历其余的字母排列。

### 复杂度分析

时间复杂度： $O(3^m \times 4^n)$ ，其中  $m$  是输入中对应 3 个字母的数字个数（包括数字 2、3、4、5、6、8）， $n$  是输入中对应 4 个字母的数字个数（包括数字 7、9）， $m+n$  是输入数字的总个数。

空间复杂度： $O(m + n)$ ， $m$  与  $n$  的含义如上。

### 代码

```
class Solution {
public:
    vector<string> ans;
    vector<string> letterCombinations(string digits) {
        if (digits == "") return ans;
        vector<string> tel(10, "");
        tel[2] = "abc";tel[3] = "def";
        tel[4] = "ghi";tel[5] = "jkl";tel[6] = "mno";
        tel[7] = "pqrs";tel[8] = "tuv";tel[9] = "wxyz";
        dfs(0, "", digits, tel);
        return ans;
    }
    void dfs(int index, string temp, string digits, vector<string> tel) {
        if (index == digits.size()) {
            ans.push_back(temp);
            return;
        }
        int c = digits[index] - 48;
        for (int i = 0; i < tel[c].size(); ++i) {
            temp.push_back(tel[c][i]);
```

```

        dfs(index+1, temp, digits, tel);
        temp.pop_back();
    }
}
};

```

## 截图

执行结果: **通过** [显示详情 >](#) [添加备注](#)

执行用时: **0 ms** , 在所有 C++ 提交中击败了 **100.00%** 的用户

内存消耗: **6.8 MB** , 在所有 C++ 提交中击败了 **8.09%** 的用户

炫耀一下:







[写题解, 分享我的解题思路](#)

提交结果	执行用时	内存消耗	语言	提交时间	备注
<b>通过</b>	0 ms	6.8 MB	C++	2021/06/18 17:22	<a href="#">添加备注</a>

## 3. 【POJ1275】差分约束

### 算法思路

这题比较困难，是一道差分约束题，要寻找不等关系，写出不等式。

$num[i]$  表示第  $i$  个小时开始的有多少个人。

$r[i]$  表示第  $i$  个小时最少雇佣多少人。

$s[i]$  表示第  $i$  个小时开始工作的有多少人。

$$1、s[i] - s[i-1] \geq 0$$

$$2、s[i-1] - s[i] \geq -num[i] \quad (1 \leq i \leq 24)$$

$$3、s[i] - s[i-8] \geq r[i] \quad (8 \leq i \leq 24)$$

$$4、s[i] - s[i+16] \geq r[i] - ans \quad (1 \leq i \leq 8)$$

$$5、s[24] - s[0] \geq ans$$

不断二分查找后， $ans$  就是最终答案即  $s[24]$ 。

## 复杂度分析

时间复杂度：  $O(n^3 * T)$ ， $T$ 为总的样例个数， $n$ 为人数。

空间复杂度：  $O(N)$ ， $N$ 为开辟的数组长度。

## 代码

```
#include<iostream>
#include<cstring>
#include<vector>
#include<queue>
using namespace std;

const int N = 1005;
struct Edge {
    int v, cost;
    Edge(int a = 0, int b = 0) : v(a), cost(b) {}
};
vector<Edge> e[N];
bool vis[N];
int cnt[N];
int dist[N];
int num[25], r[25];
int ans, n, T;

void addedge(int u, int v, int w) {
    e[u].push_back(Edge(v, w));
}

bool fun(int start, int n) {
    memset(vis, false, sizeof(vis));
    memset(cnt, 0, sizeof(cnt));
    memset(dist, INT_MAX, sizeof(dist));
    vis[start] = true;
    cnt[start] = 1;
    dist[start] = 0;
    queue<int> Q;
    Q.push(start);
    while(!Q.empty()) {
        int u = Q.front();
        Q.pop();
        vis[u] = false;
        for(int i = 0; i < e[u].size(); ++i) {
            int v = e[u][i].v;
            if (dist[v] < dist[u] + e[u][i].cost) {
                dist[v] = dist[u] + e[u][i].cost;
                if (!vis[v]) {
                    vis[v] = true;
                    Q.push(v);
                    if(++cnt[v] > n) return false;
                }
            }
        }
    }
    return true;
}
```

```

int main() {
    cin >> T;
    while (T--) {
        for (int i = 1; i <= 24; ++i)
            cin >> r[i];
        for (int i = 0; i <= 24; ++i)
            num[i] = 0;
        cin >> n;
        for(int i = 0; i < n; ++i) {
            int x;
            cin >> x;
            num[x+1]++;
        }
        ans = 1;
        bool flag = true;
        int left = 0, right = n;
        while(left < right) {
            for (int i = 0; i <= 24; ++i)
                e[i].clear();
            ans = (left + right) / 2;
            for (int i = 1; i <= 24; ++i) {
                addedge(i-1, i, 0);
                addedge(i, i-1, -num[i]);
            }
            for (int i = 8; i <= 24; ++i)
                addedge(i-8, i, r[i]);
            for (int i = 1; i <= 8; ++i)
                addedge(i+16, i, r[i]-ans);
            addedge(0, 24, ans);
            if (fun(0,25)) {
                right = ans;
                flag = false;
            }
            else left = ans + 1;
        }
        if (!flag) cout << right << endl;
        else cout << "No Solution" << endl;
    }
    return 0;
}

```

## 截图

#31421506 | stysty's solution for [POJ-1275]

Status	Time	Memory	Length	Lang	Submitted	Shared	RemoteRunId
Accepted	16ms	256kB	2182	C++	2021-06-18 17:41:07	<input type="checkbox"/>	22706804

## 4.简要描述什么是 P 问题， NP 问题和 NPC 问题？ 以及如何证明 NP-hard 问题 Q？

## P问题

P问题是具有多项式算法的判定问题。P问题就是可以有一个确定型图灵机在多项式时间内解决的问题。即那些存在 $O(n)$ ， $O(nk)$ ， $O(n\log n)$ 等多项式时间复杂度解法的问题。比如排序问题、最小生成树、单源最短路径。直观的讲，我们将P问题视为可以较快解决的问题。

## NP问题

NP问题是指一个复杂问题不能确定是否在多项式时间内找到答案，但是可以在多项式时间内验证答案是否正确。NP类问题数量很大，如完全子图问题、图着色问题、旅行商问题等。

## NPC问题

NPC问题的定义有两个条件。首先，它得是一个NP问题；然后，所有的NP问题都可以约化到它。

## 如何证明NP-hard问题？

要判断NP-hard，可以使用一个叫Reduction（归约）的方法。直观来说，如果能用一个问题的求解器来求解另一个已知是NP-hard问题，那么这个问题也是NP-hard的。