九章算法
www.jiuzhang.com

字节跳动

2020 春招真题串讲

九章算法 令狐冲

求职课程：www.jiuzhang.com

在线刷题：www.lintcode.com

版权声明

九章算法
www.jiuzhang.com

国内算法面试题特点

题目背景老TM长

动态规划老TM考

考智商题真TM多

算法岗 > 开发岗 > 数据岗

1.万万没想到之聪明的编辑

我叫王大锤，是一家出版社的编辑。我负责校对投稿来的英文稿件，这份工作非常烦人，因为每天都要去修正无数的拼写错误。但是，优秀的人总能在平凡的工作中发现真理。我发现一个发现拼写错误的捷径：

1. 三个同样的字母连在一起，一定是拼写错误，去掉一个的就好啦：比如 helllo -> hello
2. 两对一样的字母（AABB型）连在一起，一定是拼写错误，去掉第二对的一个字母就好啦：比如 helloo -> hello
3. 上面的规则优先“从左到右”匹配，即如果是AABBCC，虽然AABB和BBCC都是错误拼写，应该优先考虑修复AABB，结果为AABCC

1.万万没想到之聪明的编辑

我特喵是个天才！我在蓝翔学过挖掘机和程序设计，按照这个原理写了一个自动校对器，工作效率从此起飞。用不了多久，我就会出任*CEO*，当上董事长，迎娶白富美，走上人生巅峰，想想都有点小激动呢！

……

万万没想到，我被开除了，临走时老板对我说：“做人做事要兢兢业业、勤勤恳恳、本本分分，人要是行，干一行行一行。一行行行行行；要是不行，干一行不行一行，一行不行行行不行。”我现在整个人红红火火恍恍惚惚的……

请听题：请实现大锤的自动校对程序

输入输出

输入描述:

第一行包括一个数字*N*，表示本次用例包括多少个待校验的字符串。

后面跟随*N*行，每行为一个待校验的字符串。

输出描述:

*N*行，每行包括一个被修复后的字符串。

输入例子1:

2

helloo

woooooooow

输出例子1:

hello

woow

模拟法

题目描述虽然很复杂度，但是题目很简单，只需要按照规则模拟即可

模拟的过程中采用双指针算法

一个指针指向整理后的字符串的末尾，一个指针扫过所有字符

```
#模拟

#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin>>n;
    string s;
    while(n--)
    {
        cin>>s;
        int j=0;
        for(int i=0;i<s.size();i++)
        {
            s[j++]=s[i];    //j还没用
            if(j>=3&&s[j-1]==s[j-2]&&s[j-2]==s[j-3])  // 你获得了一个三连
                j--;
            if(j>=4&&s[j-1]==s[j-2]&&s[j-3]==s[j-4])  // AABB
                j--;
        }
        s.erase(s.begin()+j,s.end());
        cout<<s<<endl;
    }
}
```

类似题目

https://www.lintcode.com/problem/remove-duplicates-from-sorted-array/description

万万没想到之抓捕孔连顺

我叫王大锤，是一名特工。我刚刚接到任务：在字节跳动大街进行埋伏，抓捕恐怖分子孔连顺。和我一起行动的还有另外两名特工，我提议

1. 我们在字节跳动大街的*N*个建筑中选定3个埋伏地点。
2. 为了相互照应，我们决定相距最远的两名特工间的距离不超过*D*。

我特喵是个天才！经过精密的计算，我们从*X*种可行的埋伏方案中选择了一种。这个方案万无一失，颤抖吧，孔连顺！

……

万万没想到，计划还是失败了，孔连顺化妆成小龙女，混在*cosplay*的队伍中逃出了字节跳动大街。

只怪他的伪装太成功了，就是杨过本人来了也发现不了的！

万万没想到之抓捕孔连顺

请听题：给定N（可选作为埋伏点的建筑物数）、D（相距最远的两名特工间的距离的最大值）以及可选建筑的坐标，计算在这次行动中，大锤的小队有多少种埋伏选择。

注意：

1. 两个特工不能埋伏在同一地点

2. 三个特工是等价的：即同样的位置组合(A, B, C) 只算一种埋伏方法，不能因“特工之间互换位置”而重复使用

输入输出

输入描述:

第一行包含空格分隔的两个数字 N和$D(1 \le N \le 1000000; 1 \le D \le 1000000)$

第二行包含N个建筑物的的位置，每个位置用一个整数（取值区间为$[0, 1000000]$）表示，从小到大排列（将字节跳动大街看做一条数轴）

输出描述:

一个数字，表示不同埋伏方案的数量。结果可能溢出，请对 99997867 取模

输入输出

输入例子1:

```
4 3
1 2 3 4
```

输入例子2:

```
5 19
1 10 20 30 50
```

输出例子1:

```
4
```

输出例子2:

```
1
```

例子说明1:

可选方案 (1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4)

例子说明2:

可选方案 (1, 10, 20)

九章算法
www.jiuzhang.com

又是一个典型的双指针

为什么想到双指针？

for A 的位置 然后距离A找到最远的 C 的位置，然后计算 A到C中间的方案

当 A 的位置右移时，C一定也是右移：典型的同向双指针

```cpp
#双指针，左指针最左边的特工，右指针最右边的特工
#include <iostream>
#include <vector>
using namespace std;

long long C(long long n){
    return (n-1) * n / 2;
}

int main()
{
    long long n, d, count = 0;
    cin>> n>> d;
    vector<long long> v(n);
    for (int i = 0, j = 0; i < n; i++) {
        cin>> v[i];
        while (i >= 2 && (v[i] - v[j]) > d) {
            j++;
        }
        count += C(i - j); // 右指针为i的所有情况（左边两个可以随便从(i, j-1)随便选位置
    }
    cout << count % 99997867;
    return 0;
}
```

九章算法
www.jiuzhang.com

稍微改一下题目

N 个建筑，M 个特工，相邻两个特工之间距离不超过 D

问总共有多少个方案？

N <= 10^5 M <= 10^2

领扣类似题在线评测地址

https://www.lintcode.com/problem/two-sum-difference-equals-to-target/
https://www.lintcode.com/problem/longest-substring-without-repeating-characters/description

雀魂启动

小包最近迷上了一款叫做雀魂的麻将游戏，但是这个游戏规则太复杂，小包玩了几个月了还是输多赢少。

于是生气的小包根据游戏简化了一下规则发明了一种新的麻将，只留下一种花色，并且去除了一些特殊和牌方式（例如七对子等），具体的规则如下：

总共有36张牌，每张牌是1~9。每个数字4张牌。

你手里有其中的14张牌，如果这14张牌满足如下条件，即算作和牌

14张牌中有2张相同数字的牌，称为雀头。

除去上述2张牌，剩下12张牌可以组成4个顺子或刻子。顺子的意思是递增的连续3个数字牌（例如234,567等），刻子的意思是相同数字的3个数字牌（例如111,777）

雀魂启动

例如：

1 1 1 2 2 2 6 6 6 7 7 7 9 9 可以组成1,2,6,7的4个刻子和9的雀头，可以和牌

1 1 1 1 2 2 3 3 5 6 7 7 8 9 用1做雀头，组123,123,567,789的四个顺子，可以和牌

1 1 1 2 2 2 3 3 3 5 6 7 7 9 无论用1 2 3 7哪个做雀头，都无法组成和牌的条件。

现在，小包从36张牌中抽取了13张牌，他想知道在剩下的23张牌中，再取一张牌，取到哪几种数字牌可以和牌。

输入和输出

输入描述:

输入只有一行，包含13个数字，用空格分隔，每个数字在1~9之间，数据保证同种数字最多出现4次。

输出描述:

输出同样是一行，包含1个或以上的数字。代表他再取到哪些牌可以和牌。若满足条件的有多种牌，请按从小到大的顺序输出。若没有满足条件的牌，请输出一个数字0

样例

输入例子1:

1 1 1 2 2 2 5 5 5 6 6 6 9

输出例子1:

9

例子说明1:

可以组成1,2,6,7的4个刻子和9的雀头

输入例子2:

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

输出例子2:

4 7

例子说明2:

用1做雀头，组123,123,567或456,789的四个顺子

输入例子3:

1 1 1 2 2 2 3 3 3 5 7 7 9

输出例子3:

0

例子说明3:

来任何牌都无法和牌

枚举法+DFS

for 剩余的牌的可能性:

if 胡牌(牌面 + 剩余的某张牌): 记录下结果

```
def main(nums):
    """
    遍历所有可以抓到的牌看能不能胡牌
    :return:
    """
    d = {}
    for i in nums:
        d[i] = d.get(i,0) + 1
    card_list = set(range(1,10)) - {i for i,v in d.items() if v==4}
    res = []
    for i in card_list:
        if isHu(sorted(nums + [i])):  # 如果这种抽牌方式可以和牌
            res.append(i)  # 加入和牌类型列表
    res = ' '.join(str(x) for x in sorted(res)) if res else '0'
    print(res)
```

```python
def isHu(nums):
    """
    判断是否可以胡牌
    :param nums:
    :return:
    """
    if not nums:
        return True
    n = len(nums)
    count0 = nums.count(nums[0])
    # 没出现过雀头，且第一个数字出现的次数 >= 2,去掉雀头剩下的能不能和牌
    if n % 3 != 0 and count0 >= 2 and isHu(nums[2:]) == True:
        return True
    # 如果第一个数字出现次数 >= 3，去掉这个刻子后看剩下的能和牌
    if count0 >= 3 and isHu(nums[3:]) == True:
        return True
    # 如果存在顺子，移除顺子后剩下的能和牌
    if nums[0] + 1 in nums and nums[0] + 2 in nums:
        last_nums = nums.copy()
        last_nums.remove(nums[0])
        last_nums.remove(nums[0] + 1)
        last_nums.remove(nums[0] + 2)
        if isHu(last_nums) == True:
            return True
    # 以上条件都不满足，则不能和牌
    return False
```

特征提取

小明是一名算法工程师，同时也是一名铲屎官。某天，他突发奇想，想从猫咪的视频里挖掘一些猫咪的运动信息。为了提取运动信息，他需要从视频的每一帧提取“猫咪特征”。一个猫咪特征是一个两维的vector<x, y>。如果x_1=x_2 and y_1=y_2，那么这俩是同一个特征。

因此，如果喵咪特征连续一致，可以认为喵咪在运动。也就是说，如果特征<a, b>在持续帧里出现，那么它将构成特征运动。比如，特征<a, b>在第2/3/4/7/8帧出现，那么该特征将形成两个特征运动2-3-4 和7-8。

现在，给定每一帧的特征，特征的数量可能不一样。小明期望能找到最长的特征运动。

输入输出

输入描述:

第一行包含一个正整数N，代表测试用例的个数。

每个测试用例的第一行包含一个正整数M，代表视频的帧数。

接下来的M行，每行代表一帧。其中，第一个数字是该帧的特征个数，接下来的数字是在特征的取值；比如样例输入第三行里，2代表该帧有两个猫咪特征，<1，1>和<2，2>

所有用例的输入特征总数和<100000

N满足1≤N≤100000，M满足1≤M≤10000，一帧的特征个数满足 ≤ 10000。

特征取值均为非负整数。

输出描述:

对每一个测试用例，输出特征运动的长度作为一行

样例

输入例子1:

```
1
8
2 1 1 2 2
2 1 1 1 4
2 1 1 2 2
2 2 2 1 4
0
0
1 1 1
1 1 1
```

输出例子1:

```
3
```

例子说明1:

题目简化

每一秒有好多个 <x, y>

同样的 <x, y> 可能出现在多个连续秒里

问出现时间最长的 <x, y> 持续了多久

模拟+哈希表

如何在哈希表中记录一个坐标为 Key?

```cpp
int main()
{
    int n, m;
    cin >> n;

    int len;
    pair<int, int> xy;

    while (n--)
    {
        cin >> m;

        int maxCnt = 0;
        map<pair<int, int>, int> preFeaTimes; //到上一秒特征连续出现了多少次
        map<pair<int, int>, int> feaTimes; // 到当前某个特征连续出现了多少次
        while (m--)
        {
            // 对于每一秒
            cin >> len;
            for (int i = 0; i < len; i++)
            {
                cin >> xy.first >> xy.second;

                if (preFeaTimes.count(xy))
                    feaTimes[xy] = preFeaTimes[xy] + 1;
                else
                    feaTimes[xy] = 1;

                if (feaTimes[xy] > maxCnt)
                    maxCnt = feaTimes[xy];
            }
            // 更新map

            preFeaTimes.clear();
            preFeaTimes.swap(feaTimes);
        }
        cout << maxCnt << endl;
    }

    return 0;
}
```

毕业旅行问题

小明目前在做一份毕业旅行的规划。打算从北京出发，分别去若干个城市，然后再回到北京，每个城市之间均乘坐高铁，且每个城市只去一次。由于经费有限，希望能够通过合理的路线安排尽可能的省一些路上的花销。给定一组城市和每对城市之间的火车票的价钱，找到每个城市只访问一次并返回起点的最小车费花销。

输入描述:

城市个数n（1<n≤20，包括北京）

城市间的车票价钱 n行n列的矩阵 m[n][n]

输出描述:

最小车费花销 s

样例

输入例子1:

4

0 2 6 5

2 0 4 4

6 4 0 2

5 4 2 0

输出例子1:

13

例子说明1:

共 4 个城市，城市 1 和城市 1 的车费为0，城市 1 和城市 2 之间的车费为 2，城市 1 和城市 3 之间的车费为 6，城市 1 和城市 4 之间的车费为 5，依次类推。假设任意两个城市之间均有单程票可购买，且票价在1000元以内，无需考虑极端情况。

经典问题：TSP

旅行商问题：经过所有的点一次且仅一次，返回起点，路径总长度最小

这是一个 *NP* 问题！

啥是 NP 问题?

官方说法：无法在多项式级别时间复杂度内解决的问题（只能在指数级别时间复杂度解决）

通俗说法：只能用深度优先搜索来解决的问题

啥是指数级别？组合类 O(2^n)，排列类 O(n!) ...

解法1：暴力搜索

对应排列式搜索，时间复杂度是 $O(N!)$

此时 $N = 20$，有点慢，会超时

解法2：状态压缩DP

说是 DP，但是依然是 NP 的解法

状态压缩 = 用一个二进制表示若干个点是否访问过的信息

f[state][i] 表示 访问了 state 中的点之后，停在点 i 的最小耗费

这里 state 是一个 N 位二进制，每一位分别表示对应的点是否访问过

```java
# 应该是状压dp，数据范围很小。
public class TSP2 {
    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);
        int cityNum = in.nextInt();// 城市数目
        int[][] dist = new int[cityNum][cityNum];// 距离矩阵，距离为欧式空间距离
        for (int i = 0; i < dist.length; i++)
            for (int j = 0; j < cityNum; j++) {
                dist[i][j] = in.nextInt();
            }
        in.close();

        int V = 1 << (cityNum - 1);// 对1进行左移n-1位，值刚好等于2^(n-1)
        // dp表，n行，2^(n-1)列
        int[][] dp = new int[cityNum][V];
        // 初始化dp表第一列
        for (int i = 0; i < cityNum; i++)  dp[i][0] = dist[i][0];

        //设想一个数组城市子集V[j]，长度为V,且V[j] = j,对于V[j]即为压缩状态的城市集合
        //从1到V-1  用二进制表示的话，刚好可以映射成除了0号城市外的剩余n-1个城市在不在子集V[j]，1代表在，0代表不在
        //若有总共有4个城市的话，除了第0号城市，对于1-3号城市
        //111 = V-1 = 2^3 - 1  = 7 ，从高位到低位表示3到1号城市都在子集中
        //而101 = 5 ，表示3,1号城市在子集中，而其他城市不在子集中
        //这里j不仅是dp表的列坐标值，如上描述，j的二进制表示城市相应城市是否在子集中
        for (int j = 1; j < V; j++)
            for (int i = 0; i < cityNum; i++) { //这个i不仅代表城市号，还代表第i次迭代
                dp[i][j] = Integer.MAX_VALUE; //为了方便求最小值,先将其设为最大值
                if (((j >> (i - 1)) & 1) == 0) {
                    // 因为j就代表城市子集V[j],((j >> (i - 1))是把第i号城市取出来
                    //并位与上1，等于0，说明是从i号城市出发，经过城市子集V[j]，回到起点0号城市
                    for (int k = 1; k < cityNum; k++) { // 这里要求经过子集V[j]里的城市回到0号城市的最小距离
                        if (((j >> (k - 1)) & 1) == 1) { //遍历城市子集V[j]
                            //设s=j ^ (1 << (k - 1))
                            //dp[k][j ^ (1 << (k - 1))，是将dp定位到，从k城市出发，经过城市子集V[s]，回到0号城市所花费的最小距离
                            //怎么定位到城市子集V[s]呢，因为如果从k城市出发的，经过城市子集V[s]的话
                            //那么V[s]中肯定不包含k了，那么在j中把第k个城市置0就可以了，而j ^ (1 << (k - 1))的功能就是这个
                            dp[i][j] = Math.min(dp[i][j], dist[i][k] + dp[k][j ^ (1 << (k - 1))]); //^异或
                            //还有怎么保证dp[k][j ^ (1 << (k - 1))]的值已经得到了呢，
                            //注意所有的计算都是以dp表为准，从左往右从上往下的计算的，每次计算都用到左边列的数据
                            //而dp表是有初试值的，所以肯定能表格都能计算出来
                        }
                    }
                }
            }
        System.out.println(dp[0][V - 1]);
    }
}
```

解法2：状态压缩DP

f[state][i] = min{f[state - 2^i][j] + distance[j][i]}

状态压缩的解法时间复杂度是 O(2^n * n^2)

当 n = 20 的时候，大概是 10^6 * 400 刚刚可以接受

解法3：随机算法

这是目前来说，解决 TSP 类问题最高效的算法

但并不是一个 100% 能够得到正确结果的算法！

解法3：随机算法

第一步：随机生成一个访问顺序

第二步：寻找访问顺序中是否存在一对点 i, j，交换 i, j 之后，使得整体的路径耗费变小

第三步：不停的寻找第二步中满足条件的点，找到了就交换

第四步：直到无法找到这样的点对时，记录下这个局部最优解，回到第一步继续做，直到限制时间耗尽

解法3：随机算法

这个算法的总体时间复杂度是不确定的，因为随机次数不确定

假设随机了K次，每一次执行 O(N^2) 次调整，每次调整花费 O(N) 的时间计算结果

时间复杂度大概为 O(K * N^3)

九章算法
www.jiuzhang.com

字节跳动面试题小结

模拟算法，枚举算法，字符串处理，深度优先搜索，都是编程的基本功

算法本身不会很难，但是要在面试的时候写好也不容易

总体来说我是比较喜欢和认可字节跳动的这种题目风格的！确实有比较好的区分度！

不像某团的题，只有会和不会两种情况。