

实验一 递归与分治算法

一、实验目的

- 1、熟悉数据结构和基本的排序和搜索算法，熟悉编程语言的集成开发环境，掌握程序设计与实现的能力，分析算法的复杂度。
- 2、通过上机实验，要求掌握递归与分治法算法的问题描述、算法设计思想、程序设计和算法复杂性分析等。

二、实验装置

Win7, Visual C++/Java

三、实验内容

（在第一次实验题目库中选择题目，填写题目内容及输入输出要求）

1. 给定一个大小为 n 的数组 `nums`，返回其中的多数元素。多数元素是指在数组中出现次数

大于 $\lfloor n/2 \rfloor$ 的元素。假设数组是非空的，并且给定的数组总是存在多数元素。

- a) 示例 1。输入: [3, 2, 3]; 输出: 3
- b) 示例 2。输入: [2, 2, 1, 1, 1, 2, 2]; 输出: 2

2. 连续数列。

3. 至少有 K 个重复字符的最长子串。

4. 最长的美好子字符串

5. 库存管理

6. As Easy As A+B

7. Calculate e

8. Elevator

9. Max Sum

10. Number Sequence

四、程序运行结果

（说明设计思路，解释使用的数据结构，计算时间复杂度）

- 1.

（1）运行结果：

```
(base) PS C:\Users\hphp>
2
```

- (2) 思路分析：将数组一分为二进行分治，以 `left` 和 `right` 为左右边界，边界条件 `left=right`，返回该元素，为查找出子问题的多数元素，合并算法为获取左子问题的多数元素和右子问题的多数元素，计算其在子问题数组中出现的次数，选择次数的大的作为结果。
- (3) 时间复杂度： $T(n) = 2T(n/2) + O(n)$ ，为 $O(n \log n)$
- (4) 数据结构：数组

2.

- (1) 运行结果：

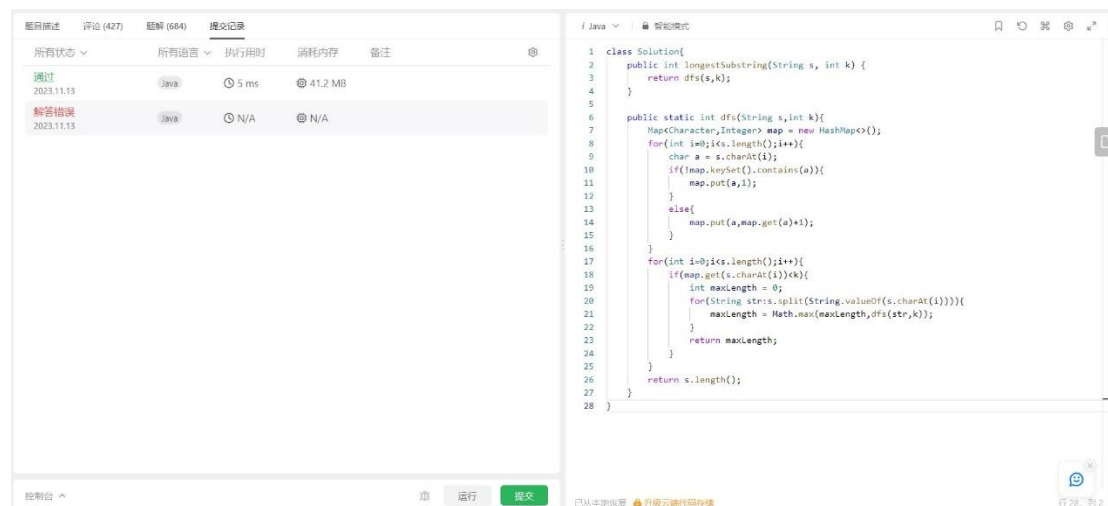
```
6
```

(2) 思路分析：可以使用动态规划算法实现，`dp[i]`数组定义为从 0 到 `i` 元素数组的最大连续子数组和，对 `dp` 初始化，`dp[0]=nums[0]`，其状态从 1 开始遍历到 `n`，对于当前 `nums[i]`，有两种选择，加入 `dp[i-1]`的子数组中，或者从自己开始重新建立子数组，即 `dp[i]=Math.max(dp[i-1]+nums[i],nums[i])`；最后对于每种状态 `i`，选择子数组之和最大的，此外，如果不加入的话，连续子数组在 `i` 处断掉，当前的连续子数组为 `dp[i-1]`，选择 `dp[i-1]`这种情况隐藏在了选择最大 `dp[i]`语句中。

- (3) 时间复杂度： $O(n)$
- (4) 数据结构：数组

3.

- (1) 运行结果：



```
1 class Solution {
2     public int longestSubstring(String s, int k) {
3         return dfs(s, k);
4     }
5
6     public static int dfs(String s, int k) {
7         Map<Character, Integer> map = new HashMap<>();
8         for (int i = 0; i < s.length(); i++) {
9             char a = s.charAt(i);
10            if (!map.containsKey(a)) {
11                map.put(a, 1);
12            }
13            else {
14                map.put(a, map.get(a) + 1);
15            }
16        }
17        for (int i = 0; i < s.length(); i++) {
18            if (map.get(s.charAt(i)) < k) {
19                int maxLength = 0;
20                for (String str : s.split(String.valueOf(s.charAt(i)))) {
21                    maxLength = Math.max(maxLength, dfs(str, k));
22                }
23                return maxLength;
24            }
25        }
26        return s.length();
27    }
28 }
```

(2) 思路分析：采用递归分治方法求解，先遍历字符串中每个字符出现的次数，使用一个 `HashMap` 存储，分治策略为以出现次数小于 `k` 的字符构成的连续字符串作为分界点，分为多个子问题（字符串），直到找不到出现次数小于 `k` 的字符，返回此时的字符串长度，合并子问题，找出子问题中长度最大的即可。

- (3) 时间复杂度： $T(n) = aT(n/a) + O(1)$ ，`a` 为划分后的字符串个数

(4) 使用的数据结构: HashMap, 数组

4.

(1) 运行结果:

```
aAa
```

(2) 思路分析: 采用分治法, 划分策略为遍历每个字符, 如果一个字符在字符串中没有出现其对应大写或者小写形式, 则该字符不可能出现在结果中, 以该字符串为基准一分为二划分子字符串, 边界条件为当划分的字符串长度为 1 时, 直接返回空字符串, 合并策略为找到子问题中长度最大的字符串返回。

(3) 时间复杂度: $T(n) = 2T(n/2) + O(1)$ $O(n)$

5.

(1) 运行结果:

```
[2, 2]
```

(2) 思路分析: 将数组使用快速排序进行排序, 排好序后输出前 cnt 个数据即可

(3) 时间复杂度: $O(n \log n)$

(4) 数据结构: 数组

6.

(1) 运行结果:

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

(2) 思路分析: 进行快速排序, 快速排序采用分治思想, 首先选择一个元素作为基准元素, 在这里使用首元素作为基准元素, 将基准元素放在第一个元素的位置, 有两个指针 l, r 从第二个元素和最后一个元素开始分别向后向前遍历, l 指针遍历到第一个大于基准元素的数后停下, 右边 r 指针开始遍历, 遍历到第一个小于基准元素的数停下, 交换两元素, 如此反复, 直到 l 指针大于 r 结束, 这时候交换首元素和 r 指针位置即可, 接着以基准元素为划分点一分为二解决子问题

(3) 时间复杂度: $O(n \log n)$

(4) 数据结构: 数组

7.

(1) 运行结果:

```

2 2.5
3 2.666666667
4 2.708333333
5 2.716666667
6 2.718055556
7 2.718253968
8 2.718278770
9 2.718281526

```

(2) 思路分析：本题的核心在于求阶乘，对于 0-2，由于保留位数不同，需要额外算出来后输出，其他元素只要按照公式计算，最后保留 9 位输出即可。

(3) 时间复杂度：O(n)

(4) 数据结构：数组

8.

(1) 运行结果：

```

1 2
17
3 2 3 1
41
0

```

(2) 思路分析：按照题目规则模拟电梯运行即可，向上花费 6 秒，并且在每一层 5 秒，使用循环遍历数组模拟电梯运行即可得到答案。

(3) 时间复杂度：O(n)

(4) 数据结构：数组

9.

(1) 运行结果：

```

2
5 6 -1 5 4 -7
7 0 6 -1 1 -6 7 -5
Case 1:
14 1 4
Case 2:
7 1 6
(base) PS C:\Users\hnhnx>

```

(2) 思路分析：使用动态规划算法实现，dp[i]数组定义为从 0 到 i 元素数组的最大连续子数组和，对 dp 初始化，dp[0]=nums[0]，其状态从 1 开始遍历到 n，对于当前 nums[i]，有两种选择，加入 dp[i-1]的子数组中，或者从自己开始重新建立子数组，即 dp[i]=Math.max(dp[i-1]+nums[i],nums[i]);最后对于每种状态 i，选择子数组之和最大的，此外，如果不加入的话，连续子数组在 i 处断掉，当前的连续子数组为 dp[i-1]，选择 dp[i-1]这种情况隐藏在了选择最大 dp[i]语句中。同时定义两个标记位 start 和 end，如果遍历到的元素从自己开始重新建立子数组，则 start=i，如果遍历一个状态后该状态的子数组大小最大，则更新 end=i+1。输出 start 和 end 即可

(3) 时间复杂度：O(n)

(4) 数据结构：数组

10.

(1) 运行结果:

```
1 1 3
1 2 10
0 0 0
2
5
```

(2) 该题和斐波那契数列类似，但是数据量大，不能使用递归或循环求解，否则会超时，使用矩阵快速幂解法，通过计算可以发现该递推公式满足以下公式

$$\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} A & B \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} f(n-1) \\ f(n-2) \end{bmatrix}$$

通过不断递推可以直到，要求 $f(n)$ ，只需要直到系数的矩阵的 $n-2$ 次幂即可，使用矩阵快速幂可以完成计算。

(3) 时间复杂度: $O(\log n)$

(4) 数据结构: 数组

实验一成绩: _____

教师签名: _____

实验二 动态规划和贪心算法

一、实验目的

- 1、理解动态规划法的设计思想，分析是否满足最优子结构性，刻画其结构特征，递归地定义最优值（动态规划方程），以自底向上的方式计算出最优值，构造最优解。掌握动态规划的算法框架和设计策略。
- 2、理解贪心算法的设计思想，掌握贪心算法的算法框架和设计策略，选取度量标准，逐步构造最优解。

二、实验装置

Win7, Visual C++/Java

三、实验内容

（在第一次实验题目库中选择题目，填写题目内容及输入输出要求）

1. Max Sum

2. Employment Planning

3. 龟兔赛跑

4. 母牛的故事

5. Brackets

6. 独立任务最优调度问题。

7. 石子合并问题

8. 数字三角形问题

9. 租用游艇问题

10. 圈乘运算问题

四、程序运行结果

（说明设计思路，解释使用的数据结构，计算时间复杂度）

1.

```

2
5
6 -1 5 4 -7
Case 1:
14 1 4
7
0 6 -1 1 -6 7 -5
Case 2:
7 1 6

```

- (1) dp 数组含义: $dp[i]$ 表示以索引 i 结尾的数组中的最大子序列的和
 - (2) dp 状态: $0-n$, n 为输入数组的大小
 - (3) 状态转移方程:

当遍历到数组下一个数时, 有两种选择:

 - ① 选择该数加入子序列: $dp[i]=dp[i-1]+num[i]$
 - ② 抛弃该数前面的子序列, 从该数开始重新计算子序列: $dp[i]=num[i]$

因此, 总的转移方程为 $dp[i]=\max(dp[i-1]+num[i], num[i])$
 - (4) 初始化 dp: $dp[0]=num[0]$
 - (5) 最终结果: dp 数组中最大的元素
 - (6) 输出选择范围: 设置 start 和 end 变量, 如果选择该数加入序列, 则 $end+1$, 否则 $start=i$
- 时间复杂度 $O(n)$

1.

```

3
4 5 6
10 9 11
0
199

```

- (1) dp 数组函数含义: $dp[i][j]$ 表示第 i 个阶段保留了 j 个人
- (2) 状态转移方程: 对于第 i 个月, 与每一个需要的人数进行比较, 如果小于最少需要的人数, 则需要进行雇佣, $dp[i][j]=dp[i-1][j]+hire*(num-j)*salary*j$, 之后选择最小代价 $dp[i][j]=\min(dp[i][j], dp[i-1][j]+hire*(num-j)+salary*j)$, 如果大于最少需要的人数, 则可以解雇也可以不解雇, 为 $dp[i][j]=\min(dp[i][j], dp[i-1][j]+fire*(j-num)+salary*j)$ 。
- (3) 状态的初始化: 对 $i=1$, 即第一个月进行初始化, 第一个月的人数可以从最少需要人数到 n 个月所需最大人数 $maxNum$
- (4) 状态的遍历: i 从 2 到 n , 表示月份, j 从 $num[i]$ 到 $maxNum$, 表示可选择能完成任务的保留的人数, 再从 k 遍历到 $maxNum$, 表示雇佣和解雇的所有可能。

时间复杂度为 $O(n^3)$

3.

100	100
3 20 5	3 60 5
5 8 2	5 8 2
10 40 60	10 40 60
Good job,rabbit!	What a pity rabbit!

- (1) **dp 数组含义**: 把每个充电站加上起点和终点作为间隔点, $dp[i]$ 就表示到这些间隔点的最短时间。
 - (2) **状态遍历**: 遍历 $dp[i]$, 再从 0 到 i 遍历 j , 表示从起点到 i 点的选择。
 - (3) **状态转移方程**: $dp[i]=\min(dp[i],dp[j]+t0)$, j 从 0 遍历到 i , $j=0$ 表示从起点直接到 i , $j=1$, 表示在 1 充电后行驶到 i , $j=2$, 表示在 2 充电后行驶到 i $t0$ 表示充电时间和从 j 行驶到 i 的时间之和。
 - (4) **状态的初始化**: $dp[0]=0$, 除此之外为了求最小值, 每个状态设置为 `Integer` 中的最大值
- 时间复杂度: $O(n^2)$

4.

- (1) 结果:

```

2
2
4
4
6
9
0

```

- (2) **dp 数组**: 题目中每一头母牛从第四年开始都会开始生小母牛, 可知对于第 i 年, 其牛个数等于前一年的牛个数 $dp[i-1]$ +新的母牛生的小牛数量。如果一头母牛在第 1 年出生, 那么其会在第 4 年开始生小牛, 以此新生牛的数量为 $dp[i-3]$, 可得 $dp[i]=dp[i-1]+dp[i-3]$
- (3) **dp 初始化**: $dp[1]=1,dp[2]=2,dp[3]=3$
- (4) **遍历**: 从第四年开始遍历到第 55 年
- (5) **时间复杂度**: $O(n)$

5.

- (1) 结果:


```

(O O O)
6
((O O))
6
([ ]])
4
([ ] [ ] D)
6

```

(2) 分析：这是一道区间 dp 的题， $dp[L][R]$ 表示从 L 到 R 区间内字符串括号匹配的数量，i 从位置 1 遍历到结尾，接着 k 从 i 遍历到字符串结尾，j 从 0 开始，如果 j 和 k 位置对应的括号能匹配，则 $dp[j][k]=dp[j+1][k-1]+2$ ，然后 x 从 j 到 k 区间内遍历， $dp[j][k]=\max(dp[j][k], dp[j][x]+dp[x+1][k])$ ，最后 $dp[0][str.length()-1]$ 就是最终结果。

(3) 时间复杂度： $O(n^3)$

6.

(1) 运行结果：

```

6
2 5 7 10 5 2
3 8 4 11 3 4
15

```

(2) 思路分析：

dp 数组采用二维数组形式， $dp[i][j]$ 表示第 i 个任务完成时，机器 j (0 表示 A 机器，1 表示 B 机器) 的最优用时，对于两台机器来说，完成时间为两台机器完成的最大值，对于第 i 个任务，如果在 A 机器上运行，则完成时间 $t1=\max(dp[i-1][0]+a[i], dp[i-1][1])$ ，如果在 B 机器上执行 $t2=\max(dp[i-1][0], dp[i-1][1]+a[i])$ ，如果 $t1 < t2$ ，则在 A 机器上运行更新 $dp[i][0]=dp[i-1][0]+a[i]$ ，B 保持不变 $dp[i][1]=dp[i-1][1]$ ，反之在 B 机器上运行。最后在 $dp[n][0]$ 和 $dp[n][1]$ 取最大值。

(3) 时间复杂度： $O(n)$

7.

(1) 运行结果：

```

4
4 4 5 9
43
54

```

(2) 思路分析：这是一道典型的区间 dp 问题，以最小得分为例，要求最小得分时，dp 采用二维数组形式， $dp[i][j]$ 表示从 i 到 j 区间石子合并的最小得分，遍历区间 $[i, j]$ ，即 $dp[i][j]=\min(dp[i][k]+dp[k+1][j]+\text{区间内石子和})$ ，最外层遍历区间长度，直到区间 $[1, n]$ 为止，本题中由于石子环形摆放，所以将数组复杂一遍放在后面模拟环，找 $dp[1][n]$ ， $dp[2][n+1]$ ， $dp[3][n+2]$ 中的最小值即可。

(3) 时间复杂度： $O(n^2)$

8.

(1) 运行结果:

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
3 0
```

(2) 思路分析: 从三角形底向上推导, 以输入样例中倒数第二行和最后一行为例, 对于 2 这个数组, 可以选择的路径为 4 和 5, 到 5 的路径总和最大, 则将 2 更新为 7, 以此类推, 推导至最上层即为解。

(3) 时间复杂度 $O(n^2)$

9.

(1) 运行结果:

```
3
5 15
7
12
```

(2) 思路分析: dp 数组采用一维, $dp[i]$ 表示到 i 租用站的最小租金, 以 $dp[5]$ 为例, 到第 5 个租用站的最小花费可以由以下几种情况推出:

①从 $dp[1]$ 直接到 5, 为 $dp[1]+m[1][5]$

②从 $dp[2]$ 到 5, 为 $dp[2]+m[2][5]$

③从 $dp[3]$ 到 5, 为 $dp[3]+m[3][5]$

④从 $dp[4]$ 到 5, 为 $dp[4]+m[4][5]$

取上述 4 中情况中的最小值, 即为 $dp[5]$ 的解

可得: $dp[i]=dp[1 \text{ to } i-1] + m[1 \text{ to } i-1][i]$

(3) 时间复杂度 $O(n^2)$

10.

(1) 运行结果:

```
3 12
1
0 0
```

(2) 思路分析: b 最小可以 2 为数组合, 最大为 9, 最小为 9. 所以 $(x \text{ 的各位数相加}) * 9 + 9$; x 的各位数相加最大为 $9 * n$, 建立二维 dp 数组, 与原来的圈乘生成该数的次数对比找最小。

(3) 时间复杂度: $O(n^3)$

实验二成绩: _____

教师签名: _____

实验三 回溯算法

一、实验目的

1、理解回溯法的设计思想，回溯法是一个既带有系统性又带有跳跃性的搜索算法。掌握从包含问题的所有解的解空间树中，按照深度优先的策略，从根结点出发搜索解空间树的过程。掌握回溯法的算法框架和设计策略。

二、实验装置

Win7, Visual C++/Java

三、实验内容

（在第三次实验题目库中选择题目，填写题目内容及输入输出要求）

1.组合总和

2.变形课

3.Tempter of the Bone

4. N 皇后问题

5. Train Problem I

6.子集和问题

7. 最小长度线路板排列问题

8. 运动员最佳配对问题

9. n 色方柱问题

10. 整数变换问题

四、程序运行结果

（说明设计思路，解释使用的数据结构，计算时间复杂度）

1.组合总和

（1）运行结果

```
2 3 5
8
[[2, 2, 2, 2], [2, 3, 3], [3, 5]]
```

```
2 3 6 7
7
[[2, 2, 3], [7]]
```

```
2
1
[]
```

(2) 设计思路

该题使用回溯算法求解，核心在于深度优先遍历 `candidates` 数组，直到找到一个等于目标数的组合为止，在搜索过程中使用 `sum` 遍历统计当前的加和，使用一个 `ArrayList` 来记录选择数字的操作，当 `sum` 等于目标数字时，使用一个 `List< List<Integer>>`，判断如果已经包含了重复数字的结果，则不加入，否则就加入 `List`，最后将该 `List` 输出就是结果。深搜过程中使用 `sum` 和 `target` 进行剪枝操作，如果 `sum` 已经大于 `target`，则返回。

(3) 数据结构：数组，List

(4) 时间复杂度：O(4^n)

2.变形课

(1) 运行结果

```
so
soon
river
goes
them
got
moon
begin
big
0
0
NO
YES
```

(2) 设计思路

使用 `char[][2]` 二维数组 `map` 保存每个输入字符串的首字符和结尾字符，使用深度优先搜索算法，对 `map[i]` 进行遍历，如果遍历到的首字符 `map[i][0]` 等于当前字符，将当前字符变为 `map[i][1]` 结尾字符，进入下一层搜索，直到遍历到的当前字符等于目标字符，表示转换成功，记录 `flag=1`，最后判断 `flag`，为 1 时输出 YES，否则输出 NO。

(3) 使用的数据结构：数组

(4) 时间复杂度：O(n!)

3.Tempter of the Bone

(1) 运行结果

```

4 4 5
S.X.
..X.
..X0
....
NO
3 4 5
S.X.
..X.
...0
YES

```

(2) 设计思路:

核心思路是对输入的迷宫(二维数组)进行深度优先搜索，从起点开始向上下左右 4 种情况进行遍历，遍历过程中判断条件为:

```

if(xx>=0 && yy>=0 && xx<n && yy<m){
    if(map[xx][yy]!='X' && log[xx][yy]==0)

```

表示遍历过程不越界，不会回头走且不会越过墙'X'。

遍历完一条路径，其遍历的深度就是路径的长度，记录下每条路径的最小值，最后与时间 T 进行比较，如果小于 T 则可以存活。

(3) 数据结构: 数组

(4) 时间复杂度: $O(n!)$

4. N 皇后问题

(1) 运行结果:

```

1 8 5 0
1
92
10

```

(2) 设计思路:

使用 dfs, 遍历每一次摆放皇后的位置是否可行, 关键在于如何判断可行。使用两个一维数组存储之前摆放的皇后的 x,y 位置, 在摆放新的皇后时, 遍历该数组, 同行同列的自然不符合要求, 接着要判断如果在同一对角线上, 分为两种情况, 一是与主对角线平行, 这种情况下在一条对角线上的元素 x,y 下标之和相等; 而是与副对角线平行, 此时在一条对角线上的元素 x,y 下标之差相等。如此就可以判断皇后摆放是否正确。

(3) 数据结构: 数组

(4) 时间复杂度: $O(n!)$

5. Train Problem I

(1) 运行结果:

```

1
123
321
Yes
in
in
in
out
out
out
Finish
3
123
312
No
Finish
0

```

(2) 使用栈模拟火车的进站和出站，具体模拟过程为：两个指针 i, j 初始分别指向两个序列的第一个元素，从火车进站顺序中的第一列火车开始遍历加入栈中，直到遍历到与出站顺序序列的第一列火车相同的时候进行出栈操作，接着该序列的指针+1，重复该操作直到进站序列遍历完成，最后将栈内元素出栈清空，判断是否和出站序列相同。

(3) 时间复杂度： $O(n)$

6.

(1) 运行结果

```

5 10
2 2 6 5 4
2 2 6 PS D:\algo>

```

(2) 思路分析：该问题是一个和 01 背包类似的问题，对于 n 个数，每个数都有选和不选两种选择，采用 dfs 遍历各种选择，剪枝条件为当已选择的数大于目标数则返回。

(3) 时间复杂度： $O(2^n)$

7.

(1) 运行结果：

```

8 5
1 1 1 1 1
0 1 0 1 0
0 1 1 1 0
1 0 1 1 0
1 0 1 0 0
1 1 0 1 0
0 0 0 0 1
0 1 0 0 1
4
5 4 3 1 6 2 8 7

```

(2) 思路分析：该问题是一个排序树问题，遍历电路板的各种排序顺序，每次

遍历达到层后计算该排列下 m 个连接块中的最大长度，选择各种情况下最小的进行输出。

(3) 时间复杂度 $O(n!)$

8.

(1) 运行结果:

```
3
10 2 3
2 3 4
3 4 5
2 2 2
3 5 3
4 5 1
52
```

(2) 思路分析: 定义 $p[i][j]$ 和 $q[i][j]$ 存放竞赛优势矩阵, 使用 `dfs` 遍历各种配对情况, 当遍历到底时计算该种情况下竞赛优势总和, 剪枝条件为计算剩下所有的配对情况, 如果都无法大于最大值, 则剪枝。

(3) 时间复杂度: $O(n^2 \cdot 2^n)$

9.

(1) 运行结果:

```
4
R
G
B
Y
0 2 1 3 0 0
3 0 2 1 0 1
2 1 0 2 1 3
1 3 3 0 2 2
RBGYRR
YRBGRG
BGRBGY
GYRBBB
```

(2) 思路分析: 将 n 个立方体堆叠起来, 4 个侧面每一面都有不同的颜色, 对于每一个立方体有 6 种摆放方式, `dfs` 遍历, 每一层遍历 6 种摆放方式, 如果同一面颜色不冲突就叠上去。

(3) 时间复杂度: $O(6^n)$

10.

(1) 运行结果:

```
15 4      8 7
4          13
ggfg      fffffffggggggggg
```


(2) 思路分析：采用 dfs 遍历，对于每一层来说，有两种选择，即 g 和 f 的变换，每一层遍历不同的变换方式，直到得到的结果等于目标数为止，使用一个数组记录下每一层变换的选择，结果进行输出，设定遍历深度上限，超过上限则认为无法变换至目标数。

(3) 时间复杂度： $O(2^n)$

实验三成绩：_____

教师签名：_____

实验四 分支限界算法

一、实验目的

- 1、理解分支限界法的设计思想，掌握分支限界法的算法框架和设计策略。
- 2、通过上机实验，要求掌握算法设计思想、程序设计和算法复杂性分析等。

二、实验装置

Win7, Visual C++/Java

三、实验内容

（在第四次实验题目库中选择题目，填写题目内容及输入输出要求）

1. 最小权顶点覆盖问题
2. 无向图的最大割问题
3. 最小重量机器设计问题
4. 布线问题
5. 最佳调度问题
6. I NEED A OFFER!
7. Rescue
8. 哈密顿绕行世界问题
9. 胜利大逃亡
10. A 计划

四、程序运行结果

（说明设计思路，解释使用的数据结构，计算时间复杂度）

1.
(1)

```

7 7
1 100 1 1 1 100 10
1 6
2 4
2 5
3 6
4 5
4 6
6 7
13
1 0 1 0 1 0 1

```

(2) 思路分析：最小顶点覆盖意思是将图中顶点分为两个集合 u 和 v ，对于 v 集合中的顶点，都能和 u 集合中的顶点有边相连，就称为顶点覆盖。分支界限遍历思路为每一个顶点都有成为 u 集合中一个顶点与不成为两种选择，可建立一颗子集树，遍历广度优先遍历该子集树即可，树中顶点有当前价值，是否选中，其父节点与层数属性，优先级设置为当前价值较小的优先。

(3) 时间复杂度： $O(2^n)$

(4) 数据结构：优先队列

2.

(1) 运行结果：

```

12
0 0 0 1 1 1 1
0 0 0 1 1 0 1
0 0 0 1 0 1 1
1 1 1 0 1 0 0
1 1 1 0 0 1 0
1 1 1 0 0 0 0

```

(2) 思路分析：这道题与最小权节点覆盖问题类似，都是一棵 01 子集树的问题，对于每个顶点都有选与不选两种选择，遍历到叶子节点后计算割边，记录下每种情况下最大的割边数，树中顶点有当前割边数，是否选中，其父节点与层数属性，剪枝条件为边总数减去当前割边数减去集合内部不可能成为割边的边数，如果小于 $best$ ，则说明即使再选择也无法得到最优解，舍去、

(3) 时间复杂度： $O(2^n)$

(4) 数据结构：优先队列

3.

(1) 运行结果：

```

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

```

(2) 思路分析：本题是一个子集树问题，每一个结点有 n 种不同厂商的选择，使用分支限界法遍历子集树，优先级是节点当前的重量，重量小的优先级高，剪枝条件为当前重量加上没有选择的机器中，每一个机器选择最小重量，如果相加后大于最优解，则舍去，约束条件为花费不超过限额。

(3) 时间复杂度： $O(2^n)$

(4) 数据结构：优先队列

4.

(1) 运行结果

```

3
2 3
3
10
1 3 2

```

(2) 思路分析：本题是一个排列树问题，对布线顺序进行全排列，从中选择成本最小的，采用分支限界算法，剪枝条件为当前成本大于最优成本则舍去，约束条件为当前选择不能和之前选择的节点相同，优先级为当前成本，成本越小的优先级越高。

(3) 时间复杂度： $O(n!)$

(4) 数据结构：优先队列

5.

(1) 运行结果：

```

7 3
2 14 4 16 6 5 3
17

```

(2) 思路分析：该问题是一个子集树问题，使用分支限界法解决，每一个任务都有 k 种选择，遍历每一个任务每一种可能的选择，选择运行时间最小的。优先级为当前时间，当前时间越小，优先级越高。剪枝条件为如果当前所用时间大于

最优时间，则舍去。

(3) 时间复杂度: $O(2^n)$

(4) 数据结构：优先队列

6. (1) 运行结果:

```
10 3
4 0.1
4 0.2
5 0.3
44.0%
```

(2) 思路分析：这道题是一道 01 背包问题，背包容量上限为钱，价值为概率，计算价值时使用公式 $P(A \cup B) = P(A) + P(B) - P(AB)$ ，采用一维 dp 数组，先遍历物品选择，后倒序遍历最大重量至物品的重量，状态转移方程为 $dp[j] = \max(dp[j], dp[j - w[i]] + p[i] - p[i] * dp[j - w[i]])$ 。

(3) 时间复杂度: $O(n^2)$

7. (1) 运行结果:

```
7 8
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
```

(2) 思路分析：这道题是一道 BFS 的题目，具体思路为从起点开始遍历周围上下左右四个路径，每遍历一个路径放入队列中，队列优先级采用最小路径优先，直到遍历到终点结束。

(3) 时间复杂度: $O(4^n)$

8.

(1) 运行结果:

```

1: 5 1 2 3 4 8 7 17 18 14 15 16 9 10 11 12 13 20 19 6 5
2: 5 1 2 3 4 8 9 10 11 12 13 20 19 18 14 15 16 17 7 6 5
3: 5 1 2 3 10 9 16 17 18 14 15 11 12 13 20 19 6 7 8 4 5
4: 5 1 2 3 10 11 12 13 20 19 6 7 17 18 14 15 16 9 8 4 5
5: 5 1 2 12 11 10 3 4 8 9 16 15 14 13 20 19 18 17 7 6 5
6: 5 1 2 12 11 15 14 13 20 19 18 17 16 9 10 3 4 8 7 6 5
7: 5 1 2 12 11 15 16 9 10 3 4 8 7 17 18 14 13 20 19 6 5
8: 5 1 2 12 11 15 16 17 18 14 13 20 19 6 7 8 9 10 3 4 5
9: 5 1 2 12 13 20 19 6 7 8 9 16 17 18 14 15 11 10 3 4 5
10: 5 1 2 12 13 20 19 18 14 15 11 10 3 4 8 9 16 17 7 6 5
11: 5 1 20 13 12 2 3 4 8 7 17 16 9 10 11 15 14 18 19 6 5
12: 5 1 20 13 12 2 3 10 11 15 14 18 19 6 7 17 16 9 8 4 5
13: 5 1 20 13 14 15 11 12 2 3 10 9 16 17 18 19 6 7 8 4 5
14: 5 1 20 13 14 15 16 9 10 11 12 2 3 4 8 7 17 18 19 6 5
15: 5 1 20 13 14 15 16 17 18 19 6 7 8 9 10 11 12 2 3 4 5
16: 5 1 20 13 14 18 19 6 7 17 16 15 11 12 2 3 10 9 8 4 5
17: 5 1 20 19 6 7 8 9 10 11 15 16 17 18 14 13 12 2 3 4 5
18: 5 1 20 19 6 7 17 18 14 13 12 2 3 10 11 15 16 9 8 4 5
19: 5 1 20 19 18 14 13 12 2 3 4 8 9 10 11 15 16 17 7 6 5
20: 5 1 20 19 18 17 16 9 10 11 15 14 13 12 2 3 4 8 7 6 5
21: 5 4 3 2 1 20 13 12 11 10 9 8 7 17 16 15 14 18 19 6 5
22: 5 4 3 2 1 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
23: 5 4 3 2 12 11 10 9 8 7 6 19 18 17 16 15 14 13 20 1 5
24: 5 4 3 2 12 13 14 18 17 16 15 11 10 9 8 7 6 19 20 1 5

```

```

38: 5 4 8 9 16 15 14 13 12 11 10 3 2 1 20 19 18 17 7 6 5
39: 5 4 8 9 16 15 14 18 17 7 6 19 20 13 12 11 10 3 2 1 5
40: 5 4 8 9 16 17 7 6 19 18 14 15 11 10 3 2 12 13 20 1 5
41: 5 6 7 8 4 3 2 12 13 14 15 11 10 9 16 17 18 19 20 1 5
42: 5 6 7 8 4 3 10 9 16 17 18 19 20 13 14 15 11 12 2 1 5
43: 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1 2 3 4 5
44: 5 6 7 8 9 16 17 18 19 20 1 2 12 13 14 15 11 10 3 4 5
45: 5 6 7 17 16 9 8 4 3 10 11 15 14 18 19 20 13 12 2 1 5
46: 5 6 7 17 16 15 11 10 9 8 4 3 2 12 13 14 18 19 20 1 5
47: 5 6 7 17 16 15 11 12 13 14 18 19 20 1 2 3 10 9 8 4 5
48: 5 6 7 17 16 15 14 18 19 20 13 12 11 10 9 8 4 3 2 1 5
49: 5 6 7 17 18 19 20 1 2 3 10 11 12 13 14 15 16 9 8 4 5
50: 5 6 7 17 18 19 20 13 14 15 16 9 8 4 3 10 11 12 2 1 5
51: 5 6 19 18 14 13 20 1 2 12 11 15 16 17 7 8 9 10 3 4 5
52: 5 6 19 18 14 15 11 10 9 16 17 7 8 4 3 2 12 13 20 1 5
53: 5 6 19 18 14 15 11 12 13 20 1 2 3 10 9 16 17 7 8 4 5
54: 5 6 19 18 14 15 16 17 7 8 9 10 11 12 13 20 1 2 3 4 5
55: 5 6 19 18 17 7 8 4 3 2 12 11 10 9 16 15 14 13 20 1 5
56: 5 6 19 18 17 7 8 9 16 15 14 13 20 1 2 12 11 10 3 4 5
57: 5 6 19 20 1 2 3 10 9 16 15 11 12 13 14 18 17 7 8 4 5
58: 5 6 19 20 1 2 12 13 14 18 17 7 8 9 16 15 11 10 3 4 5
59: 5 6 19 20 13 12 11 10 9 16 15 14 18 17 7 8 4 3 2 1 5
60: 5 6 19 20 13 14 18 17 7 8 4 3 10 9 16 15 11 12 2 1 5

```

(2) 思路分析：该题是一道分支限界法的题目，从起点开始对每一个点遍历其所能到达的所有地方，放入队列中，在遍历到最后一个节点时，判断能否到达起点，到达起点则输出结果。

(3) 时间复杂度： $O(n!)$

9.

(1) 运行结果：

```

1
3 3 4 20
0 1 1 1
0 0 1 1
0 1 1 1
1 1 1 1
1 0 0 1
0 1 1 1
0 0 0 0
0 1 1 0
0 1 1 0
11

```

(2) 思路分析：这是一道 BFS 的问题，和迷宫问题类似，这道题在二维的迷宫问题上变成了三维，有前后左右上下六个维度，遍历有个维度的选择，放入队列中，计算出到终点的时间，如果小于等于给出的时间，则可行。否则输出-1.

(3) 时间复杂度： $O(6^n)$

10

(1) 运行结果：

```

1
5 5 14
S*#*
.#...
.....
****
...#
...P
#*...
***..
...*.
*#...
Node{time=1, i=0, x=1, y=0}
Node{time=2, i=0, x=2, y=0}
Node{time=2, i=0, x=0, y=0}
Node{time=1, i=1, x=1, y=1}
Node{time=2, i=1, x=0, y=1}
Node{time=1, i=0, x=1, y=0}
Node{time=3, i=1, x=0, y=0}
Node{time=3, i=0, x=2, y=1}

```

```
Node{time=7, i=0, x=3, y=4}
Node{time=7, i=0, x=1, y=4}
Node{time=8, i=0, x=0, y=4}
Node{time=8, i=0, x=4, y=4}
Node{time=8, i=1, x=4, y=3}
Node{time=9, i=1, x=4, y=4}
Node{time=8, i=0, x=4, y=2}
Node{time=9, i=0, x=4, y=1}
Node{time=10, i=0, x=4, y=0}
Node{time=10, i=1, x=3, y=4}
Node{time=11, i=1, x=2, y=4}
Node{time=12, i=1, x=1, y=4}
Node{time=12, i=1, x=2, y=3}
Node{time=13, i=1, x=1, y=3}
Node{time=14, i=1, x=0, y=3}
Node{time=13, i=1, x=0, y=4}
YES
```

(2) 思路分析：这是一道 BFS 的问题，和迷宫问题类似，这道题在二维的迷宫问题上加上了传送门机制，到达传送门后传送至另一个迷宫，在遍历过程中判断是否到达传送门，做出相应处理即可。

(3) 时间复杂度： $O(4^n)$

实验四成绩：_____

教师签名：_____