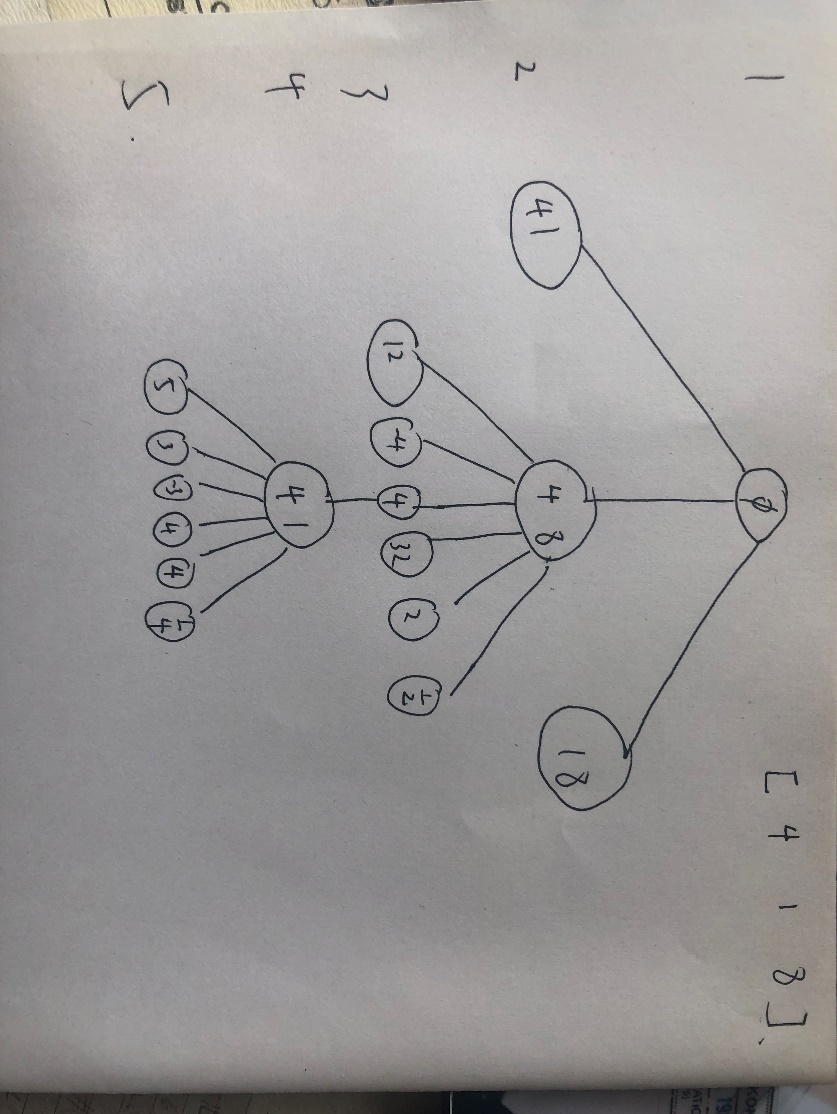
**《算法分析与设计》第5次作业**

姓名：罗煜杰 学号：71118114

算法分析题

题目1：你有4张写有1到9数字的牌。请使用回溯法判断是否能通过加减乘除及括号的运算得到24。比如给出，可以通过得到24点。注意：

1. 除法运算符'/'表示实数除法，而不是整数除法。例如。
2. '-'只能当作减法运算符，不能用作负号。
3. 数字不能合并使用，例如给出，不能写成。

答： 算法描述：从数组中随机拿出两张牌，进行加减乘除操作，然后将结果放回，此时只有三张牌，再进行上述操作，当只有一张牌时，如果数字不为24，则回溯到上一状态，继续搜索。树的根节点为空，偶数层选择牌，奇数层做加减乘除操作。下图是3张牌的简单示例图：

伪代码：

算法Count24(a,k)

输入：a={a1,a2,a3,a4},k=4

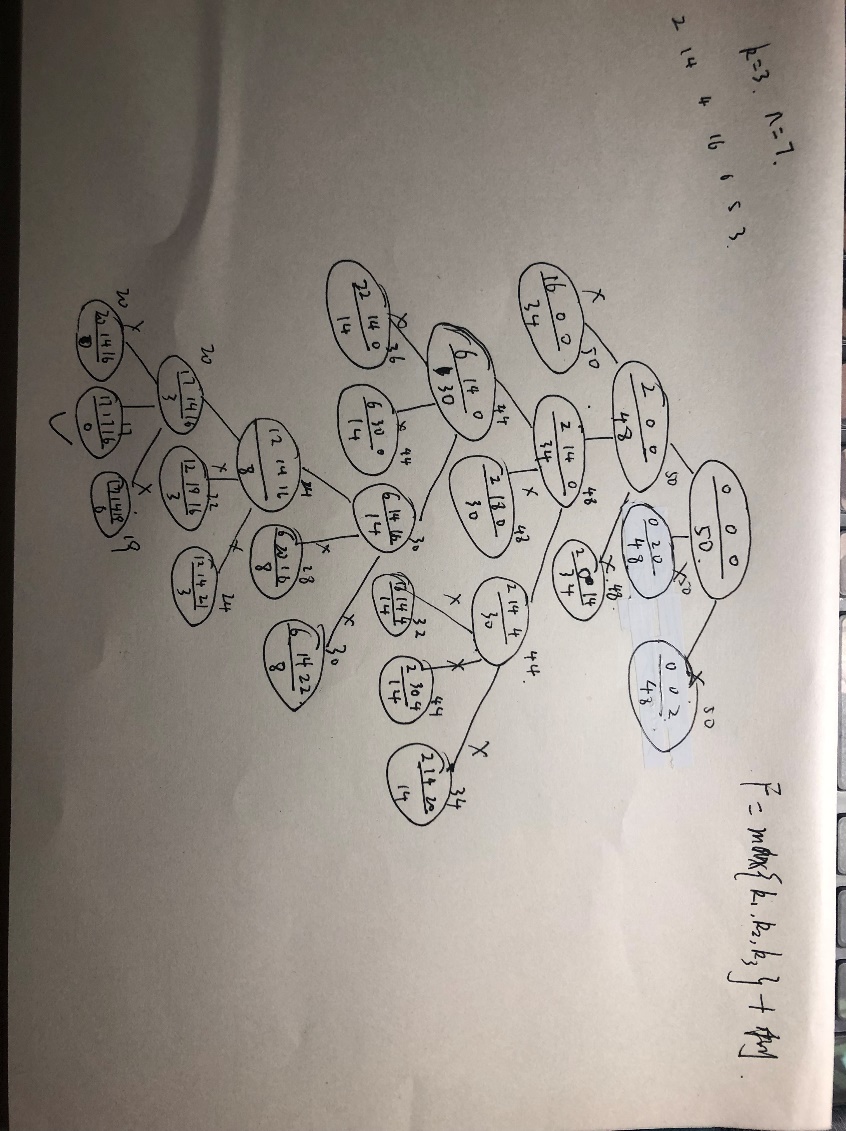
输出：所有解

1. for i🡨1 to k-1 do
2. for j🡨i+1 to k do
3. 将a[i]和a[j]移除a
4. for s🡨1 to 6//6种不同的结构
5. if s=1 then
6. 将a[i]+a[j]加入a中
7. return Count24(a,k-1)
8. else if s=2 then
9. 将a[i]-a[j]加入a中
10. return Count24(a,k-1)
11. else if s=3 then
12. 将a[j]-a[i]加入a中
13. return Count24(a,k-1)
14. else if s=4 then
15. 将a[i]\*a[j]加入a中
16. return Count24(a,k-1)
17. else if s=5 then
18. 将a[i]/a[j]加入a中
19. return Count24(a,k-1)
20. else if s=6 then
21. 将a[j]/a[i]加入a中
22. return Count24(a,k-1)

复杂度分析：前两个for循环执行次，第三个for循环执行6次，第三个for循环体内复杂度为O(Count24(a,k-1))，所以执行=3888次，即复杂度为O(1)。

题目2：假设有个任务由台可并行工作的机器完成，完成任务需要的时间为，请使用分支限界法找出完成这个任务的最佳调度，使得完成全部任务的时间最早。比如，那么完成全部任务的最早时间是。

答： 算法描述：代价函数：F=max{, …,}+剩余的时间,从仅含空集的队列开始，每次选择一个任务分配给某一个机器，计算代价函数，取最小值，然后进行剪枝，最终得到最佳调度,建立一个k叉树，每一层表示一个任务，结点<x1,x2,…,xk>，xi表示该任务分配给第i个机器执行，每一个结点记录每个机器分配任务的所需时间, …,，剩余时间TA,剩余任务n。如图是k=3，n=7的示例图。



伪代码：

OptimalScheduling(n,k)

输入：n，k，T

输出：OST

1. F🡨∞,TA🡨T[1]+T[2]+…+T[n]，, …,初始为0
2. Q是一个FIFO的队列,将第一个任务取出分配给第一个机器，然后将该结点压入队列Q中
3. while Q不为空 do
4. 取出队列Q中的第一个结点
5. if n不为空时 then
6. 取出一个任务a
7. for i🡨1 to k do
8. xi🡨i
9. 🡨+T[a]
10. TA🡨TA-T[a]
11. FT🡨max{, …,}+TA
12. if FT<=F then
13. F🡨FT
14. 将该状态压入队列
15. else
16. return max max{, …,}

复杂度分析：while语句中的复杂度为O(k)，while语句在最坏的情况下==O(),所以该算法最坏的情况下的复杂度为O()。

算法实现题

题目3：棋盘问题：在一个给定形状的棋盘（形状可能是不规则的）上面摆放棋子，棋子没有区别。要求摆放时任意的两个棋子不能放在棋盘中的同一行或者同一列，请编程求解对于给定形状和大小的棋盘，摆放个棋子的所有可行的摆放方案。

题目内容细节见 https://vjudge.net/contest/366558

答： 算法描述：与n后问题类似，每一层的结点表示该结点可以选择的行数，然后进行搜索，当棋子还未放完而且已经没有满足的地方可放时，剪枝回溯到上一状态，k层深的结点就是一个可行解。

复杂度分析：每一个棋子选择行的时候最坏会执行n次，所以最坏的情况下的复杂度为O()

代码：

#include<iostream>

using namespace std;

static char CB[100][100];//记录棋盘的形状

static int n, k;//n是矩阵长度，k是棋子数目

static int waysOfSolution;//记录可行解的数目

static int value[1000];//保存每一次输入后的输出结果

static int times;//记录第几次输入

void solutionOfCBQ(int b,int c,int n) {//b是棋盘列数,c是棋子数目,n是棋盘的行数

bool flag[100] = { true };//记录被选择的行

//当只剩下一个棋子并且有地方放时，找到了一个可行解

if (c == 1) {

for (int i = 0;i < n;i++) {

for (int j = 0;j < b;j++) {

if (CB[i][j] == '#' && flag[i]) {

waysOfSolution++;

break;

}

}

}

}

else{//用递归的方法回溯

for (int i = 0;i < n;i++) {

if (CB[i][b - 1] == '#' && flag[i]) {

flag[i] = false;

solutionOfCBQ(b - 1, c - 1, n);

}

flag[i] = true;

}

}

}

int main() {

while (cin >> n >> k && n != -1 && k != -1) {

waysOfSolution = 0;

for (int i = 0;i < n;i++)

for (int j = 0;j < n;j++)

cin >> CB[i][j];

solutionOfCBQ(n, k, n);

value[times] = waysOfSolution;

times++;

}

for (int i = 0;i < times;i++)cout << value[i] << " ";

}