贝尔多项式实验报告

一、题目

根据贝尔多项式的定义，写出求贝尔多项式值的程序。

Exponential Bell polynomials

The *partial* or *incomplete* exponential Bell polynomials are a triangular array of polynomials given by

$$B_{n,k}(x_1,x_2,\dots,x_{n-k+1})=\sum\frac{n!}{j_1!j_2!\cdots j_{n-k+1}!}\left(\frac{x_1}{1!}\right)^{j_1}\left(\frac{x_2}{2!}\right)^{j_2}\cdots\left(\frac{x_{n-k+1}}{(n-k+1)!}\right)^{j_{n-k+1}},$$

where the sum is taken over all sequences $j_1, j_2, j_3, \dots, j_{n-k+1}$ of non-negative integers such that these two conditions are satisfied:

$$j_1+j_2+\cdots+j_{n-k+1}=k,$$

$$j_1+2j_2+3j_3+\cdots+(n-k+1)j_{n-k+1}=n.$$

The sum

$$B_n(x_1,\dots,x_n)=\sum_{k=1}^{n}B_{n,k}(x_1,x_2,\dots,x_{n-k+1})$$

is called the *n*th *complete exponential Bell polynomial*.

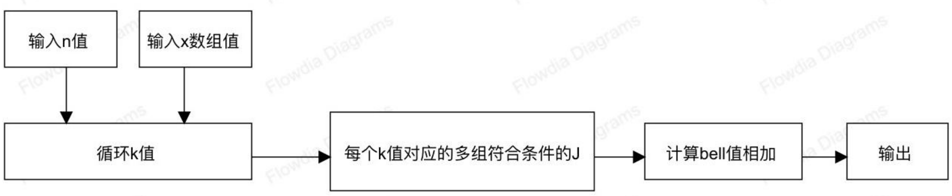
https://handwiki.org/wiki/Bell_polynomials

二、思路

1、题目理解

由题目可知，程序需要输入一个 n 值，和一个含 n 个元素的数组 X。

k 值从 1 循环到 n,每个 k 值都有若干组满足方程组的数组 J，数组 J 含（n-k+1)个元素，每组数组 J 可求一个 bell 值，最终输出的 bell 值是其和。具体流程图如下图所示：



2、突破点

得到 J 数组后求其 bell 值比较容易，重点在于如何解方程组得到符合要求的数组 J。

此处我有了三个想法：1）直接用循环，嵌套循环，将数组 J 每一个元素从 1 到 n 遍历一遍，以最后一个元素的改变作为一个最小的单位变化，每个单位变化形成一个新数组，判断新数组是否满足方程组要求。2）用线性代数行列式解方程，在相关网站寻找行列式解方程的代码，直接套数据求出解。3）将方程组和 bell 式子分别化简，从两头找到相关联的式子，不需要求出具体解。

3、具体思路

第 2）想法中，查找了相关解线性方程组的代码，但对于题目中未知数数量通常大于方程组数量的情况，都无法给出具体解值，只会显示“有无穷解”，无法解决题目问题，虽然我感觉那个代码在非负整数的前提下，改一改应该也可以求具体值。

第 3）想法中，确实没找到能化简关联的式子。

于是只好坚持第 1）想法。这里理想的嵌套循环如下：

```
int main() {
    int n = 10;//n
    int k = 1;//循环中的其中一个k值
    int J[] = { 0 };
    for (int i = 0; i < n + 1; i++) {//这里遍历次数为1到n+1的原因是，这里J的每个元素的取值范围1到n
        J[0] = i;
        for (int i = 0; i < n + 1; i++) {
            J[1] = i;
            for (int i = 0; i < n + 1; i++) {
                J[2] = i;
                //以此类推，之后下一层循环一遍后才改变上一层的值，再接着下一层循环一遍再改变上一层的值...
            }
        }
}
```

但是在这里总嵌套次数为（n-k+1）次，含有参数，不能实际全部写出来。

我想到排序算法里的桶排序，将前后循环写进一个函数，函数里面又包含函数本身，可以无限嵌套，通过控制停止条件，就可以控制嵌套次数。这里停止嵌套的条件是 p=num,即超出 J 元素数。其中最底层即最后一个元素的值改变是数组改变的单位变化，最底层的元素变化一次相当于产生一个新数组，然后对新数组进行判断是否符合方程组条件，符合方程组的为一组 J，输出并将其 bell 值加入 sum 值。具体函数代码如下：

```
void circle(int p, int num, int n, int* J, int k, int* X, int& sum) {
    if (p < num) {
        for (int q = 0; q <= n; q++) {
            J[p] = q;
            circle(p + 1, num, n, J, k, X, sum);
        }
    }
    else {
        if (judge(J, n, num, k) == 1) {
            cout << "J:";
            for (int i = 0; i < num; i++) { cout << J[i] << " "; }
            cout << endl;
            cout << "Bell:" << Bell_only(n, k, X,J) << endl;
            cout << endl;
            sum += Bell_only(n, k, X, J);
```

判断是否满足方程组条件的判断代码如下图所示：

```
int judge(int J[], int n, int num, int k) {
    int F=0;
    for (int i = 0; i < num; i++) {
        F += J[i];
    }
    int G=0;
    for (int i = 0; i < num; i++) {
        G += ((i + 1) * J[i]);
    }
    if (F == k && G == n) {
        return 1;
    }
    else return 0;
}
```

具体计算 bell 值的代码如下图所示：

```
int factorial(int p) {
    int factorial = 1;
    if (p == 0) {
        return 1;
    }
    else{
        for (int i = 1; i < p + 1; i++) {
            factorial *= i;
        }
        return factorial;
    }
}//写一个纯数字的阶乘


int Bell_only(int n, int k, int* X, int* J) {
    double part0 = factorial(n);// 分子里的n的阶乘
    double part1 = 1.0;//这里一定要用浮点数！！不能用整数，否则小于1的项会直接省成1，对结果造成较大影响！！！
    for (int i = 0; i < n - k + 1; i++) { part1 *= factorial(J[i]); }//分母里的阶乘的乘积
    double part2 = 1.0;
    for (int i = 0; i < n - k + 1; i++) {
        double t = factorial(i + 1);//i+1的阶乘
        double temp = pow(X[i] / t, J[i]);
        part2 *= temp;
    }//外面的阶乘的乘积
    double bell_only = (part0 / part1) * part2;
    return bell_only;
}//求一个固定n,k，和一组J的bell多项式
```

三、测试

$$
\begin{aligned}
&Y_1(y_1) = y_1, Y_2(y_1, y_2) = y_2 + y_1^2, \\
&Y_3(y_1, y_2, y_3) = y_3 + 3y_2y_1 + y_1^3, \\
&Y_4(y_1, y_2, y_3, y_4) = y_4 + 4y_3y_1 + 3y_2^2 \\
&\qquad\qquad + 6y_2y_1^2 + y_1^4.
\end{aligned}
$$

通过以上四个例子对程序进行测试结果如下：

在输入 n 值和 X 数组元素值后，会打印每组成立的 J 数组与其对应的 bell 值，最后输出一个总 bell 值。与图上四种情况的结果相同。程序功能正常进行。

```
请输入n: 1
请输入n个X:
1
J:1
Bell:1

Bell_all: 1
```

```
请输入n: 2
请输入n个X:
1
1
J:0 1
Bell:1

J:2
Bell:1

Bell_all: 2
```

```
请输入n: 3
请输入n个X:
1
1
1
J:0 0 1
Bell:1

J:1 1
Bell:3

J:3
Bell:1

Bell_all: 5
```

```
请输入n: 4
请输入n个X:
1
1
1
1
J:0 0 0 1
Bell:1

J:0 2 0
Bell:3

J:1 0 1
Bell:4

J:2 1
Bell:6

J:4
Bell:1

Bell_all: 15
```

在输入 n 值和 X 数组元素值后，会打印每组成立的 J 数组与其对应的 bell 值，最后输出一个总 bell 值。与图上四种情况的结果相同。程序功能正常进行。