算法分析2-7

> 用分治法将这个d次多项式转换成d/2次多项式的乘积，直到d=1时停止递归
>
> $$T(d) = O(1), \qquad d = 1$$
>
> $$T(d) = 2T(d/2) + O(dlogd), \quad d > 1$$
>
> 根据主定理解此递归式可得$T(d) = O(dlog^2 d)$

算法分析2-15

> 循环赛日程表
> 当n为奇数时，增设一个虚拟选手变成偶数问题求解
> 当总数为偶数时，又会根据n/2分为偶数和奇数的情况，偶数的情况就和书上讲的$n = 2^k$的情况一致，奇数的情况在赋值的时候会和偶数的情况不一样，详细情况见代码

```
class Solution {
public:
        //左下角
        void UpperleftToBottomleft(int size, vector<vector<int>>& ans)
        {
                int m = size / 2;
                for (int i = m; i < size; i++) {
                        for (int j = 0; j < m; j++) {
                                ans[i][j] = ans[i - m][j] + m;
                        }
                }
        }
        //左上角到右下角
        void UpperleftToLowerright(int size, vector<vector<int>>& ans)
        {
                int m = size / 2;
                for (int i = m; i < size; i++) {
                        for (int j = m; j < size; j++) {
                                ans[i][j] = ans[i - m][j - m];
                        }
                }
        }
        //左下角到右上角
        void BottomleftToUpperright(int size, vector<vector<int>>& ans)
        {
                int m = size / 2;
                for (int i = 0; i < m; i++) {
                        for (int j = m; j < size; j++) {
                                ans[i][j] = ans[i + m][j - m];
                        }
                }
        }
        //n/2为偶数，调用上述函数
        void copy(int n, vector<vector<int>>& ans)
        {
                UpperleftToBottomleft(n, ans);   //左下角
                UpperleftToLowerright(n, ans);   //右下角
                BottomleftToUpperright(n, ans);   //左上角
        }
        //n/2为奇数，赋值，注释以n=6为例
        void copyodd(int n, vector<vector<int>>& ans)
        {
                int m = n / 2;
                vector<int>temp(n);
                //数组里暂存4 5 6 4 5 6，方便之后进行赋值
                for (int i = 0; i < m; i++) {
                        temp[i] = m + i + 1;
                        temp[m + i] = temp[i];
                }
                for (int i = 0; i < m; i++) {
                        for (int j = 0; j < m + 1; j++) {
                                //例如n=6时，根据前三行3*4的矩阵为后三行3*4的矩阵赋值
                                //如果当前位置>m，则取temp中尚未比赛的人进行比赛(j从0开始)
                                //后三行中的对应位置直接取对手（当前位置的行数）
                                if (ans[i][j] > m) {
                                        ans[i][j] = temp[i];
                                        ans[temp[i] - 1][j] = i + 1;
                                }
                                //当前位置<m，则对应位置=当前位置+m
                                else {
                                        ans[m + i][j] = ans[i][j] + m;
                                }
                        }
                        //给5、6列赋值，当前位置取temp中尚未比赛的人进行比赛(j从1开始)
                        //后三行中的对应位置直接取对手（当前位置的行数）
                        for (int j = 1; j < m; j++) {
                                ans[i][m + j] = temp[i + j];
                                ans[temp[i + j] - 1][m + j] = i + 1;
                        }
                }
```

```
        }


        void Arrange(int n, vector<vector<int>>& ans)
        {
                if (n == 1) {
                        ans[0][0] = 1;
                        return;
                }
                //n是偶数
                if (n % 2 == 0) {
                        Arrange(n / 2, ans);
                }
                //n是奇数
                else {
                        Arrange(n + 1, ans);
                        return;
                }

                if (n == 2 || (n / 2) % 2 == 0) {
                        copy(n, ans);
                }
                else {
                        copyodd(n, ans);
                }

        }

};
```

算法实现2-6排列的字典序问题

> 目前要解决的问题有如下三个

> 1、有排列计算字典序值

```
        int NumberToArrange(vector<int>nums)
        {
                int nums_size = nums.size();
                int coe = 0;
                int sum = 0;   //序列总和
                //求每一个位置的字符对应的之前的序列数
                for (int i = 0; i < nums_size; i++) {
                        coe = 0;   //阶乘前的系数
                        for (int j = i + 1; j < nums_size; j++) {
                                if (nums[i] > nums[j]) {
                                        coe++;
                                }
                        }
                        sum += coe * factorial(nums_size - 1 - i);
                }
                return sum;
        }
```

> 2、由字典序值得到排列

```
vector<int> ArrangeToNumber(int ans_size, int n)
	{
		vector<int>ans(ans_size);
		//为temp赋值1-n，为之后在合适的位置放数字使用
		vector<int>temp(ans_size);
		for (int i = 0; i < ans_size; i++) {
			temp[i] = i + 1;
		}

		int position = 0;
		int fac_res = 0;
		for (int i = 0; i < ans_size; i++) {
			fac_res = factorial(ans_size - 1 - i);
			position = n / fac_res;
			n %= fac_res;
			ans[i] = temp[position];
			temp.erase(temp.begin() + position);
		}

		return ans;
	}
```

> 3、由当前排列得到下一排列

```
vector<int>NextArrange(vector<int>nums)
	{
		int nums_size = nums.size();

		//找到nums[i]<nums[i+1]
		int i = nums_size - 2;
		for (; i >= 0; i--) {
			if (nums[i] < nums[i + 1]) {
				break;
			}
		}

		//找到nums[i]<nums[j]
		int j = nums_size - 1;
		for (; j > i; j--) {
			if (nums[j] > nums[i]) {
				break;
			}
		}

		swap(nums[i], nums[j]);

		//反转i之后的数字
		vector<int>ans(nums);
		int k = i + 1;
		for (; k < nums_size; k++) {
			ans[k] = nums[nums_size + i - k];
		}

		return ans;
	}
```

算法实现2-9双色汉诺塔问题

> 证明重点在于证明之前单色汉诺塔的算法不会违反规则（3）
>
> 假设起始盘是a，终点盘是b
>
> 我们将最后一个圆盘n移动到塔座B上，然后考虑把塔座C上的n-2个圆盘移动到塔座A上，之后我们将C上的圆盘n-1移动到B上，再把塔座A上的n-2个圆盘移动塔座B上。所以我们可以看到在塔座B上会出现移动的时候圆盘会接触，在塔座B上，n-1和n是不同颜色相邻，所以之前的算法不会违反规则（3）.

```
class Solution {
public:
        void move(int n, char src, char dst)
        {
                cout << n << " " << src << " " << dst << " " << endl;
        }

        void hanoi(int n, char src, char tmp, char dst)
        {
                if (n == 1) {
                        move(n, src, dst);
                        return;
                }
                hanoi(n - 1, src, dst, tmp);
                move(n, src, dst);
                hanoi(n - 1, tmp, src, dst);
        }
};
```