**2-25.线性时间选择select算法的元素划分**

**（1）**若每组取7个元素，则共有个组，其中除了元素个数少于7的组和包含中位数的的组外，至少有一半的组中有4个元素大于。因此，大于的元素个数至少为，故select算法的递归调用最多作用于个元素。假设任何少于126个元素的输入需要的时间，可得如下递归式：

下证：。

假设对某个适当大的常数和，有，若足够大，假设显然成立。同时，选择一个常数，使得，上述递归式中的项所对应的函数有上界。则：

下证：。

当时，不等式可转换成；

当时，。故只要选择常数，即可满足上述不等式。

综上：

**（2）**若每组取3个元素，与（1）类似，可以得到递归式如下：

解得。

此时，select算法不再是线性时间算法。事实上，当每组的划分k是不小于5的奇数时，select算法仍是线性时间算法。

**2-30.带权中位数问题**

**（1）**当时，带权中位数满足：

即

当且仅当是的不带权中位数时，上式成立。

**（2）**使用最坏情况下时间复杂度为的排序算法（如归并排序），对从小到大进行排序，然后对排好序的元素进行一次线性扫描，每次将元素对应的权重相加，直至权重和大于，此时的元素即为带权中位数。时间复杂度。

代码如下：

typedef struct element {

float x;

float w;

}elem;

elem select\_median(elem \*array, int n) {

//对elem.x进行归并排序

mergeSort(array, 0, n - 1);

int sum = 0;

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

sum += array[i].w;

if (sum > 0.5)

return array[i];

}

}

**（3）**基于线性时间选择算法select，先找到中位数的中位数，然后将数组划分为两部分：和，使得，，有。计算的权重之和weight，若weight==m（初始值为0.5），则即为带权中位数。否则，若weight>m，说明带权中位数在数组中，对数组重复上述步骤；若weight<m，说明带权中位数在数组中，令m=m-weight，对数组重复上述步骤。

代码如下：

typedef struct element {

float x;

float w;

} elem;

elem select\_median(elem\* array, int p, int r, int m) {

if (p == r)

return array[p];

int q = partition(array, p, r);

int weight = 0;

for (int i = p; i <= q; i++)

weight += array[i].w;

if (weight == m)

return array[q];

else if (weight > m)

return select\_median(array, p, q, m);

else

return select\_median(array, p + 1, r, m - weight);

}

**（4）**假设为最佳选择点，点为点右边的第一个点。则有：

（当不是个输入点之一时，）

由于

代入不等式可得

故

即

同理，取点为点左边的第一个点时，可以证明

故

即

综上所述：带权中位数是一维邮局问题的最优解。

**（5）**在二维的情形下，平面上的两个点和之间的距离定义为：

因此二维邮局问题：在平面上确定一点，使得下面的和式达到最小

可以转换为两个一维邮局问题，即分别求的带权的一维邮局问题的解和的带权的一维邮局问题的解，点就是二维邮局问题的解。

**2-31.Gray码的构造问题**

当n=1时，Gray码为0和1；

当n=2时，Gray码为00，01和11，10；

当n=3时，Gray码为000，001，011，010和110，111，101，100。

我们发现：可以将n位Gray码分成两组，第一组是依次在n-1位Gray码前面补0，第二组是先将n-1位Gray码的排列顺序颠倒，再在其前面补1。

因此，可以用分治策略设计一个算法构造任意的n位Gray码。为方便起见，可以将n位Gray码的01串视为二进制整数，并将第i个串存储在数组下标为i的位置。

代码如下：

#include <iostream>

#include <cmath>

using namespace std;

int count = 0;

int\* Gray(int n) {

int\* a = new int[(int)pow(2, n) + 1];

if (n == 1) {

a[1] = 0;

a[2] = 1;

return a;

}

int\* temp = Gray(n - 1);

for (int i = 1; i <= (int)pow(2, n - 1); i++)

a[i] = temp[i];

for (int i = (int)pow(2, n - 1) + 1, k = 0; i <= (int)pow(2, n); i++, k++)

a[i] = a[(int)pow(2, n - 1) - k] + (int)pow(2, n - 1);

delete[] temp;

return a;

}

void binary(int num, int n) {

count++;

int remainder;

if (num <= 1) {

if (count < n)

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

cout << 0;

cout << num;

return;

}

remainder = num % 2;

binary(num >> 1, n);

cout << remainder;

}

void printGray(int n) {

int\* a = Gray(n);

for (int i = 1; i <= (int)pow(2, n); i++) {

binary(a[i], n);

cout << endl;

count = 0;

}

}

int main(void) {

int n;

cin >> n;

printGray(n);

return 0;

}