目 录

1　预排序

1.1　检验数组中元素的唯一性

蛮力算法：对数组中的元素对进行比较，直到找到了两个相等的元素，或者所有的元素对都已比较完毕。它的最差效率属于 $\Theta(n^2)$。

变治法的思想：先对数组排序，然后只检查它的连续元素。如果该数组有相等的元素，则一定有一对元素是相互紧挨着的。

实现代码如下：

```
bool presort_element_uniqueness(int *A, int len)
{
    int i;
    quick_sort(A, len);
    for(i = 0; i < len-1; ++i)
    {
        if(A[i] == A[i+1])
            return false;
    }
    return true;
}
```

算法分析：算法总时间为用于排序的时间加上用于检验连续元素的时间。前者至少有 $n\log n$ 次比较，而后者比较次数不会超过 $n-1$。

1.2　模式计算

在给定的数字列表中最经常出现的一个数值称为模式。

蛮力法的思想：在另一个列表中存储已经遇到的值和它们的出现频率。在每次迭代当中，通过遍历这个辅助列表，原始列表中的第 i 个元素要和已遇到的数值进行比较。如果遇到一个匹配数值，该数值的出现次数加一。否则将当前元素添加到辅助列表中，并把它的出现次数置为一。

基于蛮力法的算法，它的最差输入是一个没有相等元素的列表。在最差情况下，该算法的比较次数为 $C(n) = \frac{(n-1)n}{2} \in \Theta(n^2)$。

变治法的思想：先对输入排序，然后求出在该有序数组中邻接次数最多的等值元素，相应地就求出了模式。

```
int presort_mode(int *A, int len)
{
    int i = 0;
    int mode = -1;
    int max_length = 0;
    int length = 0;
    int temp;
    quick_sort(A, len);
```

```
        while(i < len)
        {
            length = 1;
            temp = A[i];
            while(i < len-1 && temp == A[i+1])
            {
                ++i;
                ++length;
            }
            if(length > max_length)
            {
                mode = temp;
                max_length = length;
            }
            ++i;
        }
        return mode;
    }
```

该算法的运行时间受限于排序时间。

1.3 查找问题

蛮力法的思想：顺序查找，最差情况下需要进行 n 次比较。

变治法的思想：预排序，然后应用折半查找。这个查找算法在最差情况下的总运行时间是 $\Theta(n \log n)$。

2 高斯消去法

2.1 算法实现代码

```
    void gauss_elimination(double **A, double *b, int row, int col)
    {
        int i,j,k;
        double temp;
        for(i = 0; i < row; ++i)
        {
            for(j = i+1; j < row; ++j)
            {
                if(A[i][i] != 0)
                    temp = A[j][i] / A[i][i];
                else
                    temp = 0;
                for(k = i; k < col; ++k)
                    A[j][k] = A[j][k] - temp * A[i][k];
                b[j] = b[j] - temp * b[i];
            }
        }
```

```
}
```

2.2 算法改进

上述算法存在一个问题：$A[i][i]$ 可能会非常小，导致比例因子 $A[j][i]/A[i][i]$ 非常大。

改进：每次都去找第 i 列系数的绝对值最大的行，然后把它作为第 i 次迭代的基点。这种修改称为部分选主元法。

改进算法的实现代码如下：

```
void better_gauss_elimination(double **A, double *b, int row, int col)
{
    int i,j,k;
    int max_row;
    double temp;
    for(i = 0; i < row; ++i)
    {
        max_row = i;
        for(j = i+1; j < row; ++j)
        {
            if(A[j][i] > A[max_row][i])
                max_row = j;
        }
        for(k = i; k < col; ++k)
        {
            temp = A[i][k];
            A[i][k] = A[max_row][k];
            A[max_row][k] = temp;
        }
        temp = b[i];
        b[i] = b[max_row];
        b[max_row] = temp;
        for(j = i+1; j < row; ++j)
        {
            temp = A[j][i] / A[i][i];
            for(k = i; k < col; ++k)
                A[j][k] = A[j][k] - temp * A[i][k];
            b[j] = b[j] - temp * b[i];
        }
    }
}
```

2.3 算法分析

算法的基本操作是乘法操作。乘法操作次数为

$$C(n) \approx \frac{1}{3}n^3 \in \Theta(n^3) \tag{1}$$

2.4 LU 分解

高斯消去法的现代商业实现以 LU 分解为基础。

矩阵 A 可以分解为下三角矩阵 L 和上三角矩阵 U。L 矩阵由主对角线上的 1 和高斯消去过程中行的乘数所构成的。U 矩阵是对 A 进行高斯消去后得到的结果。

解方程组 $\boldsymbol{Ax = b}$ 就等价于解方程组 $\boldsymbol{LUx = b}$。先设 $\boldsymbol{y = Ux}$，那么 $\boldsymbol{Ly = b}$，就能求出 y。然后求解方程组 $\boldsymbol{Ux = y}$，就能求出 x。

LU 分解的优点：只要得到了矩阵 A 的 LU 分解，无论对于什么样的右边向量 b，都可以利用矩阵 L 和矩阵 U 对其进行求解，不需要每次都进行高斯消去法。

LU 分解不需要额外的存储空间，我们可以把 U 的非 0 部分存储在 A 的上三角部分，把 L 的有效部分存储在 A 的主对角线的下方。

2.5 计算矩阵的逆

2.6 计算矩阵的行列式

3 平衡查找树

4 堆和堆排序

5 霍纳法则

6 二进制幂

7 问题化简