第七章算法实现题

学号：2209060322　　姓名：梁桐　　班级：计算机 2203

题目：

7-2　素数测试问题。

问题描述：试设计一个素数测试的偏真蒙特卡罗算法，对于测试的整数 n，所述算法是一个关于 $\log n$ 的多项式时间算法。结合教材中素数测试的偏假蒙特卡罗算法，设计一个素数测试的拉斯维加斯算法（见算法分析题 7-14）。

算法设计：设计一个拉斯维加斯算法，对于给定的正整数，判定其是否为素数。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 1 个正整数 p。

结果输出：将计算结果输出到文件 output.txt。若正整数 p 是素数，则输出“YES”，否则输出“NO”。

输入文件示例	输出文件示例
input.txt	output.txt
103	YES

算法思想：

该算法的基本思想是基于偏真蒙特卡罗算法和拉斯维加斯算法的组合，
使用重复的蒙特卡罗素数测试来判断一个正整数是否为素数。

1. 蒙特卡罗算法

蒙特卡罗算法的核心思想是通过随机化来进行概率性判断。在素数测试问题中，

蒙特卡罗算法通过对一个整数 N 进行随机性检测来判断 N 是否为素数。

具体来说，利用费马小定理 Fermat's Little Theorem 进行判断。

根据费马小定理，如果 a 是一个小于 N 的整数，且 a^(N-1) ≡ 1 (mod N) 成立，

那么 N 可能是素数，反之，则 N 不是素数。

2. 拉斯维加斯算法

拉斯维加斯算法是一种概率算法，与蒙特卡罗算法类似，但不同之处在于，

拉斯维加斯算法总是能给出正确的答案，只是运行时间是随机的。

拉斯维加斯算法通过重复调用蒙特卡罗测试，并根据多次测试的结果来提高判断的准确性。

如果多次测试结果显示 N 很可能是素数，则认为 N 是素数。

3. 算法流程

输入：从文件 input.txt 读取一个正整数 N。

蒙特卡罗测试：通过随机选择多个 a 值，利用费马小定理对 N 进行素数测试。

重复测试：为了提高测试的准确性，算法会执行多次蒙特卡罗素数测试（默认重复 100 次），

计算通过测试的比例。

输出：根据重复测试的结果，如果通过测试的比例大于精度阈值

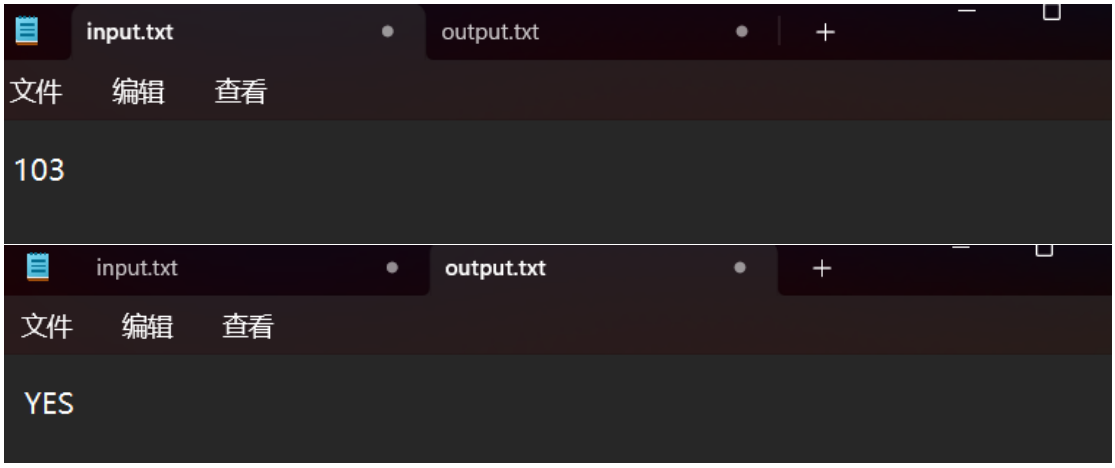
则认为 N 是素数，输出 YES；否则输出 NO。

代码：

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
#include <ctime>
using namespace std;

long Pow(int a, int b) // 计算 a 的 b 次方
{
    long s = 1;
    for (int k = 1; k <= b; k++)
    {
        s *= a;
    }
    return s;
}

bool RandomPrimalityTest(int N) // 蒙特卡罗素数测试
{
    srand((unsigned)time(NULL)); // 初始化随机种子
    int a = rand() % N; // 随机产生小于 N 的整数
    if (a == 0) a = N - 1;
    long b = Pow(a, N - 1) - 1;
    if (b % N == 0) return true;
    else return false;
}

void RepeatCall(int N, int n = 100) // 重复调用 RandomPrimalityTest 函数 n 次
{
    int cnt = 0;
    const double precision = 0.90; // 定义精度
    for (int k = 1; k <= n; k++)
    {
        if (RandomPrimalityTest(N))
            cnt++;
    }
    double e = cnt * 1.0 / n;
    if (e >= precision)
        cout << "YES" << endl; // 如果概率大于等于精度，认为是素数
    else
        cout << "NO" << endl; // 否则认为不是素数
}
```

```
int main()
{
    ifstream in("input.txt"); // 读取输入文件
    ofstream out("output.txt"); // 写入输出文件

    if (!in.good())
    {
        out << "输入异常!" << endl;
        return 0;
    }

    int N;
    in >> N; // 从文件读取正整数 p
    RepeatCall(N); // 调用素数测试函数

    in.close();
    out.close();
}
```

运行结果：



题目：

7-3 集合相等问题。

问题描述：给定两个集合 S 和 T，试设计一个判定 S 和 T 是否相等的蒙特卡罗算法。

算法设计：设计一个拉斯维加斯算法，对于给定的集合 S 和 T，判定其是否相等。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 1 个正整数 n，表示集合的大小。接下来的 2 行，每行有 n 个正整数，分别表示集合 S 和 T 中的元素。

结果输出：将计算结果输出到文件 output.txt。若集合 S 和 T 相等则输出“YES”，否则输出“NO”。

输入文件示例	输出文件示例
input.txt	output.txt
3	YES
2 3 7	
7 2 3	

思路分析：

该算法设计的目的是通过蒙特卡罗方法判断两个集合 S 和 T 是否相等。

具体来说，算法采用随机选择集合 S 中的元素，并检查该元素是否出现在集合 T 中。

通过多次随机检查，判断两个集合是否相等。若多次检查都通过，则认为两个集合相等；

若某次检查未通过，则可以断定两个集合不相等。

算法步骤：

1. 输入数据：
   从文件 input.txt 中读取数据，第一行是集合大小 n，
   接下来的两行分别包含集合 S 和集合 T 中的 n 个元素。

2. 随机选择元素：
   在 f1() 函数中，随机从集合 S 中选择一个元素，
   然后在集合 T 中查找是否存在该元素。
   如果集合 T 中存在该元素，则返回 true，反之返回 false。

3. 重复验证：
   f2() 函数重复调用 f1() 函数多次（次数由 log(1 / e) 决定，
   其中 e 为容错值）。如果 f1() 在任意一次调用中返回 false，
   即集合 T 中找不到集合 S 中的某个元素，则立即返回 false，
   表明集合不相等。
   如果所有的检查都通过，即 f1() 在所有调用中都返回 true，
   则认为集合 S 和集合 T 相等，返回 true。

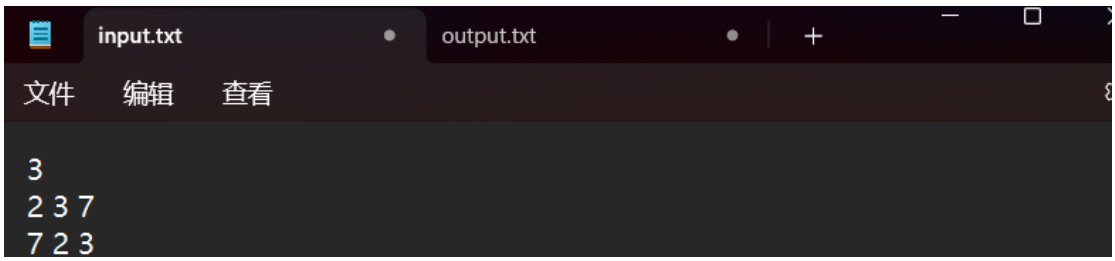
4. 输出结果：
   若 f2() 函数返回 true，则输出 YES，表示集合相等；
   否则，输出 NO，表示集合不相等。

代码：

```
#include<iostream>
#include<algorithm>
#include<fstream>
#include<ctime>
#include<cmath>
#define MAX_N 1000
using namespace std;

int setA[MAX_N], setB[MAX_N];
int size;
double epsilon = 0.00001;

bool isMatch() { // 判断一次
    srand((unsigned)time(NULL)); // 随机生成 0 到 size-1 之间的整数
    int randomIndex = rand() % size; // 随机索引
    for (int i = 0; i < size; i++) { // 查找 setB 是否包含
setA[randomIndex]
        if (setB[i] == setA[randomIndex])
            return true;
    }
    return false;
}

bool checkMatch() {
    int iterations = (int)ceil(log(1 / epsilon));
    for (int i = 1; i <= iterations; i++) { // 进行多次尝试
        if (!isMatch()) // 如果有一次不匹配，说明 setA 和 setB 不相等
            return false;
    }
    return true;
}

int main() {
    ifstream inputFile;
    inputFile.open("input.txt", ios::in);
    inputFile >> size; // 读取集合的大小
    for (int i = 0; i < size; i++)
        inputFile >> setA[i]; // 读取集合 A
    for (int i = 0; i < size; i++)
        inputFile >> setB[i]; // 读取集合 B

    ofstream outputFile;
    outputFile.open("output.txt", ios::out);
```

```
    if (checkMatch())
        outputFile << "YES" << endl;
    else
        outputFile << "NO" << endl;

    inputFile.close();
    outputFile.close();
}
```

运行结果：



题目：

7-6　皇后控制问题。

问题描述：在 $n \times n$ 个方格组成的棋盘上的任一方格中放置一个皇后，该皇后可以控制其所在的行、列及对角线上的所有方格。对于给定的自然数 n，在 $n \times n$ 个方格组成的棋盘上最少要放置多少个皇后才能控制棋盘上的所有方格，且放置的皇后互不攻击？

算法设计：设计一个拉斯维加斯算法，对于给定的自然数 n（$1 \leqslant n \leqslant 100$）计算在 $n \times n$ 个方格组成的棋盘上最少要放置多少个皇后才能控制棋盘上的所有方格，且放置的皇后互不攻击。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 1 个正整数 n。

结果输出：将计算的最少皇后数及最佳放置方案输出到文件 output.txt。文件的第 1 行是最少皇后数；接下来的 1 行是皇后的最佳放置方案。

输入文件示例	输出文件示例
input.txt	output.txt
8	5
	0 3 6 0 0 2 5 8

代码：

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cmath>
#include <cstdlib>
```

```
using namespace std;

const unsigned long maxVal = 65536L;
const unsigned long factor = 1194211693L;
const unsigned long increment = 12345L;

class CustomRandom {
public:
    CustomRandom(void) {};
    unsigned long seedValue;
    CustomRandom(unsigned long seed) {
        if (seed == 0)
            seedValue = time(0);
        else
            seedValue = seed;
    }

    unsigned long generate(unsigned long range) {
        seedValue = factor * seedValue + increment;
        return (unsigned short)((seedValue >> 16) % range);
    }

    double generateFraction(void) {
        return generate(maxVal) / double(maxVal);
    }
};

template <class T>
void allocate2DArray(T** &arr, int rows, int cols) {
    arr = new T*[rows];
    for (int i = 0; i < rows; i++) {
        arr[i] = new T[cols];
    }
}

template <class T>
void deallocate2DArray(T** &arr, int rows) {
    for (int i = 0; i < rows; i++) {
        delete[] arr[i];
    }
    delete[] arr;
    arr = nullptr;
}
```

```
class NQueenSolver {
    friend bool solveNQueens(int size);

private:
    NQueenSolver(void) {};
    bool isSafe(int pos);
    bool findSolution(int step);
    int solve(int limit);
    bool checkSolution(int size);

    int boardSize, *positions, *validPositions, *solution, **board;
    int minConflicts, conflictCount;
    CustomRandom random;
};

bool NQueenSolver::isSafe(int pos) {
    if (positions[pos] > 0) {
        for (int i = 1; i < pos; i++) {
            if ((positions[i] > 0) && ((abs(pos - i) == abs(positions[i]
- positions[pos])) || positions[i] == positions[pos])) {
                return false;
            }
        }
    }
    return true;
}

bool NQueenSolver::checkSolution(int size) {
    int i, j, u, v, count = 0;
    for (i = 1; i <= size; i++)
        for (j = 1; j <= size; j++)
            board[i][j] = 0;

    for (i = 1; i <= size; i++) {
        if (positions[i] > 0) {
            for (j = 1; j <= size; j++) { board[i][j] = 1;
board[j][positions[i]] = 1; }
            for (u = i, v = positions[i]; u >= 1 && v >= 1; u--, v--)
board[u][v] = 1;
            for (u = i, v = positions[i]; u <= size && v >= 1; u++, v-
-) board[u][v] = 1;
            for (u = i, v = positions[i]; u >= 1 && v <= size; u--,
v++) board[u][v] = 1;
```

```
            for (u = i, v = positions[i]; u <= size && v <= size; u++,
v++) board[u][v] = 1;
        }
    }

    for (i = 1; i <= size; i++)
        for (j = 1; j <= size; j++)
            count += board[i][j];

    return (count == size * size);
}

int NQueenSolver::solve(int limit) {
    int k = 1;
    int count;
    conflictCount = 0;

    while (k <= limit) {
        count = 0;
        validPositions[count++] = 0;
        for (int i = 1; i <= boardSize; i++) {
            positions[k] = i;
            if (isSafe(k)) validPositions[count++] = i;
        }
        if ((positions[k++] = validPositions[random.generate(count)]) >
0) conflictCount++;
    }
    return conflictCount;
}

bool NQueenSolver::findSolution(int step) {
    if (step > boardSize) {
        if    (checkSolution(boardSize)    &&    (conflictCount    <=
minConflicts)) {
            solution[0] = conflictCount;
            for (int i = 1; i <= boardSize; i++) solution[i] =
positions[i];
            return true;
        } else return false;
    } else {
        for (int i = 0; i <= boardSize; i++) {
            positions[step] = i;
            if (i > 0) conflictCount++;
            if ((conflictCount <= minConflicts) && isSafe(step) &&
```

```
findSolution(step + 1))
                return true;
            if (i > 0) conflictCount--;
        }
        return false;
    }
}

bool solveNQueens(int size) {
    NQueenSolver solver;
    solver.boardSize = size;

    int *p = new int[size + 1];
    int *q = new int[size + 1];
    int *result = new int[size + 1];
    int **boardArray;

    allocate2DArray(boardArray, size + 1, size + 1);

    for (int i = 0; i < size; i++) p[i] = 0;
    solver.positions = p;
    solver.validPositions = q;
    solver.board = boardArray;
    solver.solution = result;
    solver.minConflicts = size;

    int stop = 3;
    int num = 0, maxAttempts = 100;
    if (stop > 15) stop = size - 15;

    while (true) {
        while (solver.solve(stop) == 0);
        if (solver.findSolution(stop + 1)) {
            num++;
            if (solver.conflictCount < solver.minConflicts) {
                solver.minConflicts = solver.conflictCount;
                num = 0;
            }
        }
        if (num > maxAttempts) {
            // 将结果输出到文件 output.txt
            ofstream outfile("output.txt");
            outfile << result[0] << endl;
            for (int i = 1; i < size; i++)
```

```
                outfile << result[i] << " ";
            outfile << result[size] << endl;
            outfile.close();
            return true;
        }
    }

    delete[] p;
    delete[] q;
    delete[] result;
    deallocate2DArray(boardArray, size + 1);

    return true;
}

int main() {
    // 从文件 input.txt 读取输入
    ifstream infile("input.txt");
    int n;
    infile >> n;
    infile.close();

    return solveNQueens(n);
}
```

运行结果：