

算法分析题:

7-3

7-3 试设计一个算法，随机地产生范围在 $1 \sim n$ 的 m 个随机整数，且要求这 m 个随机整数互不相同。

可以先利用 set 构建 1 到 n 的数字集合，再通过循环从集合中随机选取元素并移除，确保生成 m 个互不相同、范围在 1 到 n 的随机整数。具体算法的 python 代码和测试结果如下：

```
*7-1.py - C:/Users/86158/Desktop/算法设计/code/7-1. Python 2.7.16 Shell
File Edit Format Run Options Window Help File Edit Shell Debu
import random
def generate_unique_random(n, m):
    result = []
    # 创建 1 到 n 的数字集合
    numbers = set(range(1, n + 1))
    for _ in range(m):
        # 随机选一个并移除，保证不重复
        num = random.sample(numbers, 1)[0]
        numbers.remove(num)
        result.append(num)
    return result
# 假设n=10, m=5
n = 10
m = 5
print(generate_unique_random(n, m))
>>>
===== RESTART
[7, 4, 5, 6, 8]
>>>
===== RESTART
[5, 1, 4, 6, 9]
>>>
===== RESTART
[9, 10, 5, 8, 1]
>>>
===== RESTART
[3, 9, 6, 8, 2]
>>>
===== RESTART
[8, 4, 1, 2, 9]
>>>
===== RESTART
[3, 8, 2, 1, 6]
>>>
===== RESTART
[6, 4, 5, 7, 1]
>>>
===== RESTART
[2, 3, 6, 7, 1]
>>>
===== RESTART
[4, 1, 10, 8, 3]
>>>
===== RESTART
[10, 5, 6, 9, 8]
>>>
===== RESTART
[5, 6, 4, 8, 2]
>>>
===== RESTART
[4, 9, 3, 2, 10]
>>>
===== RESTART
[10, 5, 2, 7, 1]
>>> |
```

7-4

7-4 设 X 是含有 n 个元素的集合，从 X 中均匀地选取元素。设第 k 次选取时首次出现重复。

- (1) 试证明当 n 充分大时， k 的期望值为 $\beta\sqrt{n}$ 。其中， $\beta\sqrt{\pi/2} = 1.253$ 。
- (2) 由此设计一个计算给定集合 X 中元素个数的概率算法。

7.4.1) 证明:

设 $X = \{x_1, x_2, \dots, x_n\}$, 定义 A_i 为“前 i 次选取元素互不重复”的事件
~~第 i 次选由 $P(A_i)$~~ 第 i 次选必然不重复, 即 $P(A_1) = 1$

$$\text{则 } P(A_i | A_{i-1}) = \frac{n-i+1}{n}$$

记 B_k = “首次重复在第 k 次”

等价于前 $k-1$ 次不重复, 第 k 次与前 $k-1$ 次中任一次重复

$$P(B_k) = P(\bar{A}_k | A_{k-1}) \cdot P(A_{k-1})$$

其中 $P(\bar{A}_k | A_{k-1}) = 1 - P(A_k | A_{k-1}) = \frac{k-1}{n}$, 且

$$P(A_{k-1}) = \prod_{i=2}^{k-1} P(A_i | A_{i-1}) = \prod_{i=2}^{k-1} \frac{n-i+1}{n} = \frac{n!}{(n-k+1)! \cdot n^{k-1}}$$

$$\therefore P(B_k) = \frac{\frac{n!}{(n-k+1)! \cdot n^{k-1}} \cdot \frac{k-1}{n}}{1} = \frac{(k-1) \cdot n!}{(n-k+1)! \cdot n^k}$$

$$k \text{ 期望值 } E(k) = \sum_{k=1}^n k \cdot P(B_k)$$

Stirling 公式近似后 $(n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left[1 + \frac{1}{12n} + O\left(\frac{1}{n^2}\right)\right])$

有 $E(k) = \sqrt{\frac{\pi}{2}} \cdot \frac{1}{3} + O\left(\frac{1}{\sqrt{n}}\right)$, 当 n 充分大时, 后两项舍去,

故 $E(k) \approx \sqrt{\frac{\pi}{2}} \approx 1.2539n$ 得证

(2) 由以上结论, 当 n 充分大时, $E(k) \approx \sqrt{\frac{\pi}{2}} \cdot n$, 变形得 $n \approx \frac{2}{\pi} k^2$

可设计如下算法:

具体代码:

```
import random

def estimate_set_size(X, trials=1000):
    total_k = 0
    n_estimates = []
    for _ in range(trials):
        seen = set()
        k = 0
        while True:
            elem = random.choice(list(X))
            if elem in seen:
                total_k += (k + 1)  # 首次重复在 k+1 次
                break
        n_estimates.append(total_k)
```

```

        seen.add(elem)
        k += 1
    avg_k = total_k / trials
    return (2 / 3.1415927) * avg_k ** 2 #  $\pi$  近似 3.1416
# 测试: 假设 X 是含 n 个元素的集合
X = set(range(1000)) # 实际中 X 是未知集合, 这里模拟
print(estimate_set_size(X))

```

测试结果:

```

>>>
===== RESTART: C:/Users/86158/Desktop/算法设计/code/7-4.py =====
1070.15782154
>>>

```

7-5

7-5 试设计一个随机化算法计算 $365!/340!365^{25}$, 并精确到 4 位有效数字。

① ~~stirling~~ stirling 近似

7.5 由 $n=365$ 比较大, 可用阶数较低的stirling公式近似

即 $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$, $\ln(n!) = \frac{1}{2}(\ln(2\pi n) + n \ln n - n)$

对数转换: $\ln\left(\frac{365!}{340!365^{25}}\right) = (\ln(365!) - \ln(340!) - 25 \ln(365))$

$$\approx \frac{1}{2} \ln(2\pi \cdot 365) + 365 \ln 365 - 365 - \left[\frac{1}{2} \ln(2\pi \cdot 340) + 340 \ln 340 - 340 - 25 \ln(365) \right]$$

$$= \frac{1}{2} \ln \frac{365}{340} + 365 \ln 365 - 340 \ln 340 - 25(1 + \ln 365)$$

计算该公式可得 $\ln\left(\frac{365!}{340!365^{25}}\right) = -0.8409$ $e^{-0.8409} = 0.4313$

② 随机化算法

由 $\frac{365!}{340!} = 365 \times 364 \times \dots \times 341 = \prod_{k=341}^{365} k$

可将 $\prod_{k=341}^{365} k$ 视为从区间 $[341, 365]$ 取数的期望, 通过蒙特卡洛

采样估算: 随机生成 N 个 $[341, 365]$ 内的整数 x_i , 计算均值

$\bar{x} = \frac{1}{N} \sum x_i$, 则 $\prod_{k=341}^{365} k \approx \bar{x}^{25}$, 具体算法如下:

Python 代码:

```

import random
def monte_carlo_product():
    product = 1.0

```



```

for _ in range(25):
    # 随机选 341~365 的整数
    x = random.randint(341, 365)
    product *= x
return product / (365**25)
# 多次采样取平均 (提升精度)
N = 10000
total = 0
for _ in range(N):
    total += monte_carlo_product()
result = total / N
print(f"蒙特卡洛近似结果: {result:.4g}")

```

蒙特卡洛近似结果: 0.4334

7-9


7-9 如果对于某个 n 值, n 后问题无解, 则算法将陷入死循环。

(1) 证明或否定下述论断: 对于 $n \geq 4$, n 后问题有解。

(2) 是否存在正数 δ , 使得对所有 $n \geq 4$ 算法成功的概率至少是 δ ?

7-9

(1) ~~证明~~ 有解, 证明:



当 $n=4$ 时, 存在经典解 (皇后位置 (1,2), (2,4), (3,1), (4,3))

$n > 4$ 可用构造法扩展解

当 n 为偶数时, 可将皇后按列交替放在 $(i, 2i \bmod n)$ 的地方, 这情况很合适
 由于 $i \neq 2i \bmod (n+1)$, 故 $x=y$ 的 ~~反对~~ 角线上一定无皇后

当 n 为奇数时
 则此时拓展到 $n+1$ 行, 只需在原基础上, 在 $(n+1, n+1)$ 的位置摆放皇后, 就也是一个合法的解

故 $n > 4$ 后无论 n 为奇偶都有解, 原命题得证。

(2)

7-12

7-12 设 $mc(x)$ 是一致的 75% 正确的蒙特卡罗算法，考虑下面的算法：

```
mc3(x) {  
    int t, u, v;  
    t = mc(x);  
    u = mc(x);  
    v = mc(x);  
    if ((t == u) || (t == v))  
        return t;  
    return v;  
}
```

(1) 试证明上述算法 $mc3(x)$ 是一致的 $27/32$ 正确的算法，因此是 84% 正确的。

(2) 试证明如果 $mc(x)$ 不是一致的，则 $mc3(x)$ 的正确率有可能低于 71%。

7-12

(1) 重复 3 次的蒙特卡罗算法各次正确的分布如下：

000, 001, 010, 011, 100, 101, 110, 111, 8 种情况

其中 011, 101, 110, 111 这 4 种返回正确解，故返回正确解

概率为 $\frac{1}{4} \times \frac{3}{4} \times \frac{3}{4} + \frac{3}{4} \times \frac{1}{4} \times \frac{3}{4} + \frac{3}{4} \times \frac{3}{4} \times \frac{1}{4} + \frac{3}{4} \times \frac{3}{4} \times \frac{3}{4} = \frac{27}{32} \approx 84\%$

(2) 如果 $mc(x)$ 不是一致的，则 110 不能保证返回正确解，

$\frac{1}{4} \times \frac{3}{4} \times \frac{3}{4} + \frac{3}{4} \times \frac{1}{4} \times \frac{3}{4} + \frac{3}{4} \times \frac{3}{4} \times \frac{1}{4} = \frac{45}{64} \approx 0.7031$

即解正确概率可能低至 0.7031，在 71% 以下。

7-14

7-14 设算法 A 和 B 是解同一判定问题的两个有效的蒙特卡罗算法。算法 A 是 p 正确偏真算法，算法 B 是 q 正确偏假算法。试利用这两个算法设计一个解同一问题的拉斯维加斯算法，并使所得到的算法对任何实例的成功率尽可能高。

算法设计如下：python 伪代码

```
LasVegas(x):  
    while True:  
        a = A(x) // 调用偏真算法 A  
        b = B(x) // 调用偏假算法 B
```

```
    if a == "是" 且 b == "是":  
        // A 说“是”, B 也说“是” → 仅当真实为“是”(因 B 对“否”实例高概率说“否”, 若真实“否”,  
B 说“是”概率  $\leq 1-q < 1/2$  )  
        return "是"  
  
    if a == "否" 且 b == "否":  
        // A 说“否”, B 也说“否” → 仅当真实为“否”(因 A 对“是”实例高概率说“是”, 若真实“是”,  
A 说“否”概率  $\leq 1-p < 1/2$  )  
        return "否"  
  
    // 否则 (A 和 B 结果矛盾, 说明至少一个算法出错, 重试)
```

算法实现题:

7-3

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

input.txt

3

2 3 7

7 2 3

输出文件示例

output.txt

YES

设计算法 C++代码如下

```
#include <iostream>
#include <vector>
#include <unordered_set>
#include <random>
#include <fstream>
using namespace std;

// 拉斯维加斯算法：判定集合 S 和 T 是否相等
bool lasVegasEquality(const unordered_set<int>& S, const unordered_set<int>& T, int
sampleTimes = 100) {
    // 大小不等直接不相等
    if (S.size() != T.size()) return false;
    int n = S.size();
    if (n == 0) return true; // 空集相等

    // 随机数生成器
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(0, n - 1); // 生成 [0, n-1] 索引

    // 转换为数组，方便随机抽样
    vector<int> vecS(S.begin(), S.end());
    vector<int> vecT(T.begin(), T.end());

    // 多次抽样验证 (sampleTimes 次)
    for (int i = 0; i < sampleTimes; ++i) {
        // 从 S 随机选元素，检查是否在 T
        int idx = dis(gen);
        int x = vecS[idx];
```

```

        if (T.find(x) == T.end()) return false;

        // 从 T 随机选元素，检查是否在 S
        idx = dis(gen);
        int y = vecT[idx];
        if (S.find(y) == S.end()) return false;
    }

    // 抽样均通过，判定相等
    return true;
}

int main() {

    ifstream input("input.txt");
    int n;
    input >> n;

    unordered_set<int> S, T;
    for (int i = 0; i < n; ++i) { int x; input >> x; S.insert(x); }
    for (int i = 0; i < n; ++i) { int y; input >> y; T.insert(y); }
    input.close();

    // 调用拉斯维加斯算法（可调整抽样次数，如 100 次）
    bool isEqual = lasVegasEquality(S, T, 100);

    ofstream output("output.txt");
    output << (isEqual ? "YES" : "NO") << endl;
    output.close();

    return 0;
}

```


7-4 逆矩阵问题。

问题描述：给定两个 $n \times n$ 矩阵 A 和 B ，试设计一个判定 A 和 B 是否互逆的蒙特卡罗算法（算法的计算时间应为 $O(n^2)$ ）。

算法设计：设计一个蒙特卡罗算法，对于给定的矩阵 A 和 B ，判定其是否互逆。

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 ，表示矩阵 A 和 B 为 $n \times n$ 矩阵。接下来的 $2n$ 行，每行有 n 个实数，分别表示矩阵 A 和 B 中的元素。

结果输出：将计算结果输出到文件 output.txt。若矩阵 A 和 B 互逆则输出“YES”，否则输出“NO”。

输入文件示例

input.txt

3

1 2 3

2 2 3

3 3 3

-1 1 0

1 -2 1

0 1 -0.666667

输出文件示例

output.txt

YES

算法设计：

矩阵 A 和 B 互逆的充要条件是： $AB = I$ 且 $BA = I$ 其中 I 是 $n \times n$ 单位矩阵。

蒙特卡罗思路：随机生成一个 $n \times 1$ 向量 x ，验证 $ABx = x$ 且 $B Ax = x$ 。若 A 和 B 不互逆，存在非零概率检测到矛盾

具体 C++代码如下：

```
#include <iostream>
#include <vector>
#include <random>
#include <fstream>
#include <cmath>
using namespace std;

const double EPS = 1e-6; // 浮点误差容忍度

// 矩阵乘法: A(n×n) * x(n×1) = y(n×1)
vector<double> matMulVec(const vector<vector<double>>& A, const vector<double>& x) {
    int n = A.size();
    vector<double> y(n, 0.0);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            y[i] += A[i][j] * x[j];
        }
    }
    return y;
}
```

```

// 验证向量是否近似相等（考虑浮点误差）
bool isVecEqual(const vector<double>& a, const vector<double>& b) {
    for (int i = 0; i < a.size(); ++i) {
        if (fabs(a[i] - b[i]) > EPS) return false;
    }
    return true;
}

// 蒙特卡罗算法判定矩阵互逆
bool isInverse(const vector<vector<double>>& A, const vector<vector<double>>& B, int
trials = 1) {
    int n = A.size();
    if (n == 0) return true;

    // 随机数生成器
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<double> dis(-1.0, 1.0); // 随机向量元素范围

    for (int t = 0; t < trials; ++t) {
        // 生成随机向量 x
        vector<double> x(n);
        for (int i = 0; i < n; ++i) {
            x[i] = dis(gen);
        }

        // 计算 ABx 和 BAx
        vector<double> Bx = matMulVec(B, x);
        vector<double> ABx = matMulVec(A, Bx);

        vector<double> Ax = matMulVec(A, x);
        vector<double> BAx = matMulVec(B, Ax);

        // 验证是否近似等于 x
        if (!isVecEqual(ABx, x) || !isVecEqual(BAx, x)) {
            return false;
        }
    }

    return true;
}

int main() {

```

```

    ifstream input("input.txt");
    int n;
    input >> n;

    vector<vector<double>> A(n, vector<double>(n));
    vector<vector<double>> B(n, vector<double>(n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            input >> A[i][j];
        }
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            input >> B[i][j];
        }
    }
    input.close();

    // 调用蒙特卡罗算法
    bool result = isInverse(A, B, 3); // 3 次抽样降低错误概率

    ofstream output("output.txt");
    output << (result ? "YES" : "NO") << endl;
    output.close();

    return 0;
}

```