7-3 随机产生 m 个整数。

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。

答：**算法思路**

使用集合存储已生成的数：集合的特性是自动去重，每次生成随机数时检查是否已存在于集合中，若不存在则加入。

生成随机数：利用随机数生成器生成 1~n 之间的整数，直到集合中元素个数达到 m。

处理边界情况：若 $m>n$，则无法生成（题目中应保证 $m\leq n$，否则返回空或报错）。

代码实现如下：

```cpp
vector<int> generateUniqueRandomNumbers(int m, int n) {
    if (m > n) return {}; // 无法生成，返回空
    set<int> nums;
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(1, n);
    while (nums.size() < m) {
        int num = dis(gen);
        nums.insert(num);
    }
    return vector<int>(nums.begin(), nums.end());
}
```

7-4 集合大小的概率算法。

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。

答：（1）如下

XIAMEN UNIVERSITY

7.4 (1) 由题可得 P(第k次选取首次重复) $= \frac{C_n^{k-1}(k-1)!(k-1)}{n^k}$

则 $E(k)=\sum_{k=1}^{n}\frac{C_n^{k-1}(k-1)!(k-1)k}{n^k}$

$\because n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left[1+\frac{1}{12n}+\theta\left(\frac{1}{n^2}\right)\right]$

$\therefore E(k)=\sqrt{\frac{\pi n}{2}}-\frac{1}{3}+\theta\left(\frac{1}{\sqrt{n}}\right)$

$\because \beta\sqrt{\frac{\pi}{2}}=1.253$

$\therefore \beta=\frac{1.253}{\sqrt{\frac{\pi}{2}}}$ 即 $\beta\sqrt{n}=\sqrt{\frac{\pi n}{2}}-\frac{1}{3}$

(2)

（2）**算法思路**：

模拟随机选取：重复模拟从集合 X 中随机选元素，记录首次重复的次数 k，进行 t 次试验求平均 $\bar{k}$。

估计集合大小：利用 $n \approx (\bar{k}/\beta)2$ 计算 n 的估计值，其中 $\beta \approx 1.253$。

代码如下：

```
function estimate_set_size():
    t = 1000   #  试验次数
    sum_k = 0
    for i in 1..t:
        seen = empty set
        k = 0
        while True:
            x = random_element()   #  模拟从 X 中随机选元素
            k += 1
            if x in seen:
                sum_k += k
                break
            seen.add(x)
    avg_k = sum_k / t
    beta = 1.253
    n_estimate = (avg_k / beta) ** 2
    return n_estimate
```

7-5　生日问题。

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。

答：首先化简原计算公式：

365! /(340!*365^25)=(341/365)*(342/365)*……(364/365)

以 24 次随机试验为一次事件。第(365-K)次试验随机值取值范围是［1，365］，若实验

值在［1，K］则为真，否则为假。若 24 次试验值都为真，则此次事件为真，否则为假，代码如下：

```cpp
int solution() {
    static default_random_engine eng;
    static uniform_int_distribution<int> dis(1, 365);
    return dis(eng);
}

int main() {
    int t = 10;
    for (int i = 1; i <= t; ++i) {
        int total = i * 1e7;
        int cnt = total;
        for (int j = 0; j < total; ++j) {
            for (int k = 340; k <= 364; ++k) {
                if (solution() > k) {
                    cnt--;
                    break;
                }
            }
        }
        cout << "total: " << total << endl << "answer: " << double(cnt) / total << endl;
    }
    return 0;
}
```

7-9　n 后问题解的存在性。

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？

答：

(1) 用 x_{ij} 表示在 (i,j) 处皇后的状态，当 $x_{ij}=1$ 时表示在 (i,j) 处放置了一个皇后，为0不放

则可化为 0-1 线性规划问题 $\max \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}$

$\sum_{j=1}^{n} x_{ij} \leq 1 \quad 1\leq i\leq n$，$\sum_{i=1}^{n} x_{ij} \leq 1 \quad 1\leq j\leq n$

$\sum_{i+j=k} x_{ij} \quad 2\leq k\leq 2n$

$\sum_{i-j=k} x_{ij} \quad 1-n\leq k\leq n-1$

$x_{ij}\in\{0,1\} \quad 1\leq i,j\leq n$ 显然 $\max \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} \leq n$

下面对 $n\geq 4$ 构造上述线性规划问题的一个解，使 $\max \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} = n$，从而证 $n\geq 4$ n后问题有解

① $n\geq 4$ 且为偶且 $(n-2)\%6\neq 0$ 时，对 $1\leq j\leq \frac{n}{2}$，取 $x(j,2j)=1$，$x(\frac{n}{2}+j,2j-1)$

② 当 $n\geq 4$ 为偶且 $n\%6\neq 0$ 时，对于 $1\leq j\leq \frac{n}{2}$，取 $x(j,k(j))=1$，$x(n+1-j,n-k(j)+1)$

其中 $k(j)=(\frac{n}{2}+2j-3)\%n$

③ 当 $n>4$ 为奇，再取 $x(n,n)=1$，再转换为 $(n-1)\times(n-1)$ 棋盘问题，用 (1) 或 (2) 的方法构造其余值

按照上述方法构造 n 后问题解的算法如下：

```
void construct(int k) {
    if(k < 4)
        return;
    if(odd(k))
        x[k] = k--;
    if((k-2)%6 > 0)
        build1(k);
    else
        build2(k);
}

void build1(int k) {
    k /= 2;
    for(int i=1; i <= k; i++) {
        x[i]=2*i;
        x[k+i]=2*i-1;
    }
}

void build2(int k) {
    for(int i=1; i <= k/2; i++) {
        x[i] = 1+(2*(i-1)+k/2-1)%k;
        x[k+1-i] = k*(2*(i-1)+k/2-1)%k;
    }
}
```

容易验证，上述方法构造的解是 n 后问题的 0-1 线性规划解。故 $n \geq 4$ 时，n 后问题有解。

(2) **结论**：存在正数 δ>0，对所有 n≥4，随机算法成功概率至少为 δ。

分析：

随机算法（如随机放置皇后，逐步优化）在 n≥4 时，解空间 S(n)　=∅（由（1）知）。

每次随机选择布局，成功概率 p(n)=｜S(n)｜/n^n>0。取 δ=$\min_{n\geq4}$p(n)，由于 n=4 时 p(4)>0，且 p(n) 对 n≥4 恒正，故 δ 存在且 δ>0（例如，δ 可设为 p(4) 的最小值，确保下限非零）。

7-12　重复 3 次的蒙特卡罗算法。

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%。

答：（1）重复 3 次的蒙特卡罗算法各次正确的分布有 8 种不同情况：000，001，010，011，100，101，110，111。其中，011、101、110、111 这 4 种情况返回正确解。因此返回正确解的概率为

1/4*3/4*3/4+3/4*1/4*3/4+3/4*3/4*1/4+3/4*3/4*3/4=27/32

（2）如果 mc (x) 不是一致的，则 110 不能保证返回正确解，因此返回正确解的概率可能低到

1/4*3/4*3/4+3/4*1/4*3/4+3/4*3/4*3/4=45/64=0.7031，所以 mc3 (x) 的正确率有可能低于 71%。

7-14　由蒙特卡罗算法构造拉斯维加斯算法。

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。

答：算法如下

```
while True:
    resA = A()
    resB = B()
```

if resA == resB:

return resA

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	

答：**算法设计思路：通过随机抽样验证集合包含关系，高效判定集合相等**

代码实现如下：

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

bool areEqual(const vector<int>& S, const vector<int>& T) {
    unordered_set<int> setS(S.begin(), S.end());
    unordered_set<int> setT(T.begin(), T.end());

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(0, S.size() - 1);

    // 验证 S ⊆ T
    for (int i = 0; i < 100; ++i) {
        int idx = dis(gen);
        if (setT.find(S[idx]) == setT.end()) {
            return false;
        }
    }

    // 验证 T ⊆ S
    uniform_int_distribution<> disT(0, T.size() - 1);
    for (int i = 0; i < 100; ++i) {
        int idx = disT(gen);
        if (setS.find(T[idx]) == setS.end()) {
            return false;
        }
    }

    return true;
}

int main() {
    int n;
    cin >> n;
    vector<int> S(n), T(n);
    for (int i = 0; i < n; ++i) cin >> S[i];
    for (int i = 0; i < n; ++i) cin >> T[i];

    if (areEqual(S, T)) {
        cout << "YES" << endl;
    }
    else {
        cout << "NO" << endl;
    }

    return 0;
}
```

7-4 逆矩阵问题。

问题描述：给定两个 $n\times n$ 矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，试设计一个判定 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 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 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，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 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 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 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 中的元素。

结果输出：将计算结果输出到文件 output.txt。若矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 互逆，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
1 2 3	
2 2 3	
3 3 3	
−1 1 0	
1 −2 1	
0 1 −0.666667	

答：**算法设计思路**

矩阵互逆条件：若 A 和 B 互逆，则 \(AB = BA = I\)（单位矩阵）。利用蒙特卡罗算法，随机生成 n 维向量 x，验证 \(ABx = x\) 和 \(BAx = x\)。若多次验证均通过，则大概率互逆；若某次验证不通过，必不互逆。

优化矩阵乘法：直接计算 \(ABx = A(Bx)\) 和 \(BAx = B(Ax)\)，避免显式计算 AB 和 BA，每次向量乘法时间复杂度为 \(O(n^2)\)，符合题目要求。

随机验证：生成随机向量，多次验证（如 100 次），提高正确性概率。

代码实现如下：

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

typedef vector<vector<double>> Matrix;

bool multiplyABx(const Matrix& A, const Matrix& B, const vector<double>& x, vector<double>& y) {
    int n = A.size();
    vector<double> Bx(n, 0.0);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            Bx[i] += B[i][j] * x[j]; // Bx = B * x
        }
    }
    for (int i = 0; i < n; ++i) {
        y[i] = 0.0;
        for (int j = 0; j < n; ++j) {
            y[i] += A[i][j] * Bx[j]; // y = A * Bx = AB * x
        }
    }
    return true;
}

bool multiplyBAx(const Matrix& A, const Matrix& B, const vector<double>& x, vector<double>& y) {
    int n = A.size();
    vector<double> Ax(n, 0.0);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            Ax[i] += A[i][j] * x[j]; // Ax = A * x
        }
    }
    for (int i = 0; i < n; ++i) {
        y[i] = 0.0;
        for (int j = 0; j < n; ++j) {
            y[i] += B[i][j] * Ax[j]; // y = B * Ax = BA * x
        }
    }
    return true;
}
```

```cpp
bool areInverse(const Matrix& A, const Matrix& B, int n) {
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<double> dis(-1.0, 1.0);

    for (int t = 0; t < 100; ++t) { // 多次验证
        vector<double> x(n), y1(n), y2(n);
        for (int i = 0; i < n; ++i) {
            x[i] = dis(gen); // 生成随机向量
        }

        multiplyABx(A, B, x, y1); // 计算 ABx
        multiplyBAx(A, B, x, y2); // 计算 BAx

        bool valid = true;
        for (int i = 0; i < n; ++i) {
            // 检查是否接近 x（处理浮点误差）
            if (abs(y1[i] - x[i]) > 1e-6 || abs(y2[i] - x[i]) > 1e-6) {
                valid = false;
                break;
            }
        }
        if (!valid) {
            return false; // 一旦不通过，立即返回
        }
    }
    return true; // 多次验证通过，返回互逆
}

int main() {
    int n;
    cin >> n;
    Matrix A(n, vector<double>(n)), B(n, vector<double>(n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> A[i][j];
        }
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> B[i][j];
        }
    }

    if (areInverse(A, B, n)) {
        cout << "YES" << endl;
    }
    else {
        cout << "NO" << endl;
    }

    return 0;
}
```