Insel. Ein Straßennetz, in dem jede Stadt von jeder anderen Stadt erreichbar ist, heiße *zusammenhängend*. Uns interessiert die Wahrscheinlichkeit $P(n)$, dass ein aus dem in der Aufgabenstellung beschriebenen Verfahren entstehendes Straßennetz mit n Städten zusammenhängend ist. Offensichtlich gilt $P(1) = 1$ für $n = 1$ Stadt.

Teil (i). Im Falle von $n = 7$ gibt es $\binom{7}{2} = \frac{7 \cdot 6}{2} = 21$ Paare von Städten und damit mögliche Straßen. Diese Anzahl ist klein genug, um in einer Brute-Force-Lösung alle $2^{21} = 2\,097\,152$ Teilmengen dieser 21 Straßen aufzuzählen und zu überprüfen, ob sie zusammenhängend sind. Es sind aber nicht alle Straßennetze gleichwahrscheinlich. Da die Wahrscheinlichkeit für jede Straße 40% beträgt, ist die Wahrscheinlichkeit für ein bestimmtes Straßennetz mit k Straßen durch

$$\left(\frac{2}{5}\right)^k \left(1 - \frac{2}{5}\right)^{m-k} = \frac{2^k \cdot 3^{m-k}}{5^m}$$

gegeben, wobei $m = \binom{n}{2}$ die Anzahl der möglichen Straßen bezeichnet. Durch Summation dieser Wahrscheinlichkeiten über alle zusammenhängende Straßennetze erhalten wir die gewünschte Wahrscheinlichkeit $P(n)$.

Um für ein Straßennetz zu überprüfen, ob es zusammenhängend ist, können wir es als ungerichteten Graph darstellen, eine Tiefensuche von einem beliebigen Knoten ausgehend starten und überprüfen, ob die Tiefensuche alle Knoten besucht. Es wird Zeit $\mathcal{O}(n + m)$ pro Tiefensuche benötigt und es sind 2^m Tiefensuchen erforderlich. Bei Vorberechnung der Potenzen von 2, 3 und 5 ergibt sich mit $m \in \mathcal{O}(n^2)$ eine Gesamtlaufzeit in $\mathcal{O}\left((n + m) \cdot 2^m\right) = \mathcal{O}\left(n^2 \cdot 2^{n(n-1)}\right)$.

Für die Implementierung empfiehlt es sich, beim Summieren immer nur $2^k \cdot 3^{m-k}$ zu addieren und erst ganz am Ende durch 5^m zu teilen. Das Ergebnis kann sogar als vollständig gekürzter Bruch ausgegeben werden, indem man Zähler und Nenner durch den größten gemeinsamen Teiler teilt, der mit dem euklidischen Algorithmus effizient bestimmt werden kann. So wird Gleitkommaarithmetik vermieden und das Ergebnis ist exakt. Schließlich lautet die Lösung für $n = 7$:

$$P(7) = \frac{66\,640\,974\,430\,912}{95\,367\,431\,640\,625} \approx 69,88\,\%$$

Am Ende dieses Dokuments befindet sich eine Implementierung in C++.

Teil (ii). Die Brute-Force-Lösung ist für $n = 25$ deutlich zu langsam, selbst wenn uns die Rechenleistung sämtlicher Computer auf der Welt zur Verfügung stünde. Es gibt nämlich $\binom{25}{2} = 300$ mögliche Straßen und somit $2^{300} > 10^{90}$ Straßennetze. Das sind mehr Straßennetze als Atome im sichtbaren Universum.

Erstaunlicherweise können wir eine Rekursionsgleichung finden, sodass sich $P(n)$ mit dynamischer Programmierung in quadratischer Laufzeit $\mathcal{O}(n^2)$ berechnen lässt. Der Leser mag sich fragen, ob wir $P(n)$ sogar für $n = 50\,000$ in akzeptabler Zeit berechnen können, schließlich haben wir eine $\mathcal{O}(n^2)$-Lösung. Die Lösung hat aber einen großen Haken: Es gibt gigantische Zahlen als Zwischenergebnisse, die nicht mehr in Zeit $\mathcal{O}(1)$ verarbeitet werden können und sehr viel Speicherplatz benötigen. Das liegt an den Binomialkoeffizienten und Potenzen. Wir werden sehen, dass dies schon für $n = 25$ eine Herausforderung darstellt.

Zentral für die Lösung ist der Beweis des folgenden Satzes.

Satz. Es gilt $P(1) = 1$ und $P(n) = 1 - \sum_{k=1}^{n-1} \binom{n-1}{k-1} P(k) \left(\frac{3}{5}\right)^{k(n-k)}$ für $n \geq 2$.

Beweis. Induktion über die Anzahl n der Städte. Der Induktionsanfang $n = 1$ ist bereits gezeigt. Die Rekursionsgleichung sei nun erfüllt für $1, 2, \ldots, n-1$ Städte nach Induktionsvoraussetzung. Wir berechnen die Wahrscheinlichkeit $1 - P(n)$, dass ein Straßennetz mit n Städten nicht zusammenhängend ist und betrachten dazu die Menge U aller von Stadt 1 erreichbaren Städte inklusive Stadt 1. Sei V die Menge aller Städte. In einem nicht zusammenhängenden Straßennetz ist U eine echte Teilmenge von V. Sei k die Anzahl der Städte in U. Dann ist $k < n$ und nach Induktionsvoraussetzung ist $P(k)$ die Wahrscheinlichkeit, dass ein Straßennetz mit den Städten aus U zusammenhängend ist. Außerdem darf es keine Straße geben, die eine Stadt aus U mit einer Stadt aus $V \setminus U$ verbindet. Es gibt $n - k$ Städte in $V \setminus U$ und somit $k(n-k)$ solche Straßen, die nicht existieren dürfen. Sonst würde U im Widerspruch zur Annahme mehr als k Städte enthalten. Zudem ist

$$\left(\frac{3}{5}\right)^{k(n-k)}$$

die Wahrscheinlichkeit, dass all diese Straßen nicht existieren. Schließlich gibt es

$$\binom{n-1}{k-1}$$

Teilmengen von V mit $k-1$ Elementen und damit Möglichkeiten, U zu wählen. Man beachte, dass Stadt 1 immer in U liegt. Summation über alle möglichen k und Multiplikation der Wahrscheinlichkeiten[1] liefert dann

$$1 - P(n) = \sum_{k=1}^{n-1} \binom{n-1}{k-1} P(k) \left(\frac{3}{5}\right)^{k(n-k)}$$

und das ist offensichtlich äquivalent zur Behauptung. □

[1] Tritt das Ereignis A ein, wenn das Straßennetz mit den Städten aus U zusammenhängend ist und tritt das Ereignis B ein, wenn die $k(n-k)$ zwischen Städten aus U und $V \setminus U$ verlaufenden Straßen sämtlich nicht existieren, dann sind A und B stochastisch unabhängig. Außerdem sind die Ereignisse A bzw. B für verschiedene k disjunkt.

Da bei einer exakten Berechnung von $P(25)$ rationale Zahlen mit großen Zählern und Nennern entstehen und diese bei 64-Bit-Datentypen zu Überläufen führen, werde hier eine Lösung basierend auf Gleitkommaarithmetik präsentiert. Dennoch ist eine exakte Lösung mit viel Mühe und selbstgeschriebenen Funktionen für Berechnungen auf großen Zahlen möglich und lehrreich, aber nicht Gegenstand dieser Aufgabe. Für die Implementierung bietet sich eine Vorberechnung der Potenzen von $\frac{3}{5}$ und Binomialkoeffizienten an. Die Binomialkoeffizienten erfüllen die Rekursionsgleichung

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

für $n \geq k \geq 1$, wodurch eine Tabelle mit $\mathcal{O}(n^2)$ Einträgen in Zeit $\mathcal{O}(n^2)$ berechnet werden kann. Die Potenzen lassen sich mit der Gleichung

$$\left(\frac{3}{5}\right)^n = \frac{3}{5}\left(\frac{3}{5}\right)^{n-1}$$

leicht berechnen. Die obere Grenze ist hier $m = \max\limits_{1 \leq k < n} k(n-k)$. Eine Abschätzung ist mit AM-GM möglich oder direkt durch quadratische Ergänzung:

$$k(n-k) = \frac{n^2}{4} - \left(k - \frac{n}{2}\right)^2 \leq \frac{n^2}{4}$$

Somit ist $m \in \mathcal{O}(n^2)$ und die asymptotische Laufzeit wird durch die Vorberechnung nicht verschlechtert. Am Ende des Dokuments befindet sich eine Implementierung. Übrigens ist ein Städtenetzwerk aus 25 Städten mit an Sicherheit grenzender Wahrscheinlichkeit, nämlich $P(25) \approx 0,99988 = 99,988\%$, zusammenhängend.

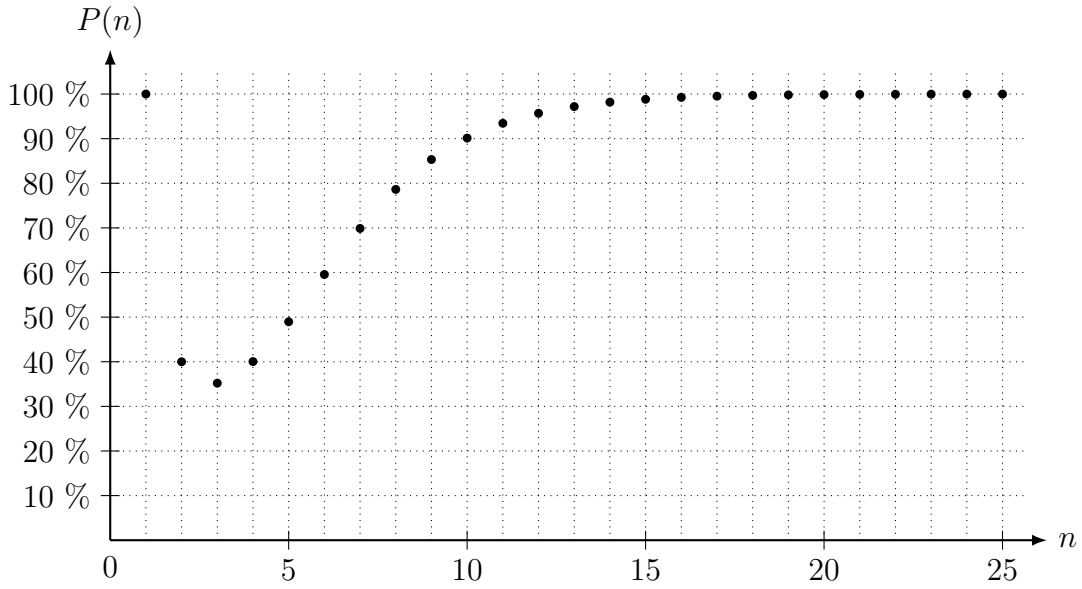

Abbildung 1: Wahrscheinlichkeiten $P(n)$ für $1 \leq n \leq 25$ Städte

Programm zu Teil (i):

```cpp
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;

ll pow2[30], pow3[30], pow5[30];
ll euclid(ll a, ll b) {
    return !b ? a : euclid(b, a % b);
}

pair<ll, ll> solve(int n) {
    if (n <= 1) return { 1, 1 };
    int m = n * (n - 1) / 2;
    pair<int, int> streets[m];
    int idx = 0;
    for (int u = 0; u < n; ++u) {
        for (int v = u + 1; v < n; ++v) {
            streets[idx++] = { u, v };
        }
    }

    ll rs = 0;
    bool vis[n], adj[n][n];
    for (int mask = 0; mask < (1 << m); ++mask) {
        int num = 0;
        for (int i = 0; i < m; ++i) {
            int u = streets[i].first;
            int v = streets[i].second;
            adj[u][v] = adj[v][u] = false;
            if (mask & (1 << i)) {
                ++num;
                adj[u][v] = adj[v][u] = true;
            }
        }

        stack<int> s; s.push(0);
        fill(vis, vis + n, false);
        while (!s.empty()) {
            int cur = s.top(); s.pop();
            vis[cur] = true;
            for (int next = 0; next < n; ++next) {
                if (!vis[next] && adj[cur][next]) s.push(next);
            }
        }

        bool connected = true;
        for (int i = 0; i < n; ++i) connected &= vis[i];
        if (connected) rs += pow2[num] * pow3[m - num];
    }

    ll gcd = euclid(rs, pow5[m]);
    return { rs / gcd, pow5[m] / gcd };
}
```

```
53 int main() {
54     pow2[0] = pow3[0] = pow5[0] = 1;
55     for (int n = 1; n < 30; ++n) {
56         pow2[n] = 2 * pow2[n - 1];
57         pow3[n] = 3 * pow3[n - 1];
58         pow5[n] = 5 * pow5[n - 1];
59     }
60 
61     for (int n = 1; n <= 7; ++n) {
62         auto rs = solve(n);
63         cout << "P(" << n << ") = " << rs.first << "/" << rs.second << endl;
64     }
65 
66     return 0;
67 }
```

Programm zu Teil (ii):

```
1  #include <bits/stdc++.h>
2  #define N 25
3  typedef long long ll;
4  using namespace std;
5  
6  int main() {
7      ll binomial[N][N];
8      for (ll n = 0; n < N; ++n) {
9          for (ll k = 0; k < N; ++k) {
10             if (k > n) binomial[n][k] = 0;
11             else if (n == 0 || k == 0) binomial[n][k] = 1;
12             else binomial[n][k] = binomial[n - 1][k] + binomial[n - 1][k - 1];
13         }
14     }
15 
16     double pw[N * N / 4 + 1]; pw[0] = 1.0;
17     for (ll i = 1; i <= N * N / 4; ++i) {
18         pw[i] = 0.6 * pw[i - 1];
19     }
20 
21     double dp[N + 1];
22     for (ll n = 1; n <= N; ++n) {
23         dp[n] = 1.0;
24         for (ll k = 0; k < n; ++k) {
25             dp[n] -= binomial[n - 1][k - 1] * pw[k * (n - k)] * dp[k];
26         }
27 
28         cout << fixed << setprecision(8)
29              << "P(" << n << ") = " << dp[n] << endl;
30     }
31 
32     return 0;
33 }
```