Inhaltsverzeichnis

1 Vorspiel

1.1 Header

```
#include <bits/stdc++.h>
#define fst first
#define snd second

using namespace std;
typedef long long ll;
typedef __int128 lll;
typedef unsigned long long ull;
typedef long double ld;
typedef pair<ll, ll> pll;
typedef vector<ll> vll;
typedef vector<string> vs;

constexpr int oo = 0x3f3f3f3f;
constexpr ll ooo = 0x3f3f3f3f3f3f3f3fLL;
constexpr ld eps = 1e-7;
constexpr ld PI = 2.0 * acos(0.0);
```

2 Kombinatorische Algorithmen

2.1 Mengenpartitionen (T)

Der Algorithmus erzeugt alle k-Partitionen einer n-elementigen Menge.

```
void visit() {
    print_arr(A); // do something with A
}

vll A; // indicates in which partition an element is
```

```
void proc_f(ll mu, ll nu, ll sigma) {
    if (mu == 2) visit(); // do something
    else proc_f(mu - 1, nu - 1, (mu + sigma) % 2);
    if (nu == mu + 1) {
        A[mu] = mu - 1;
        visit();
        while (A[nu] > 0) {
            A[nu] = A[nu] - 1;
            visit();
        }
    } else if (nu > mu + 1) {
        if ((mu + sigma) % 2 == 1) A[nu - 1] = mu - 1;
        else A[mu] = mu - 1;
        if ((A[nu] + sigma) % 2 == 1) proc_b(mu, nu - 1, 0);
        else proc_f(mu, nu - 1, 0);
        while (A[nu] > 0) {
            A[nu] = A[nu] - 1;
            if ((A[nu] + sigma) % 2 == 1) proc_b(mu, nu - 1, 0);
            else proc_f(mu, nu - 1, 0);
        }
    }
}

void proc_b(ll mu, ll nu, ll sigma) {
    if (nu == mu + 1) {
        while (A[nu] < mu - 1) {
            visit();
            A[nu] = A[nu] + 1;
        }
        visit();
        A[mu] = 0;
    } else if (nu > mu + 1) {
        if ((A[nu] + sigma) == 1) proc_f(mu, nu - 1, 0);
        else proc_b(mu, nu - 1, 0);
        while (A[nu] < mu - 1) {
            A[nu] = A[nu] + 1;
            if ((A[nu] + sigma) == 1) proc_f(mu, nu - 1, 0);
            else proc_b(mu, nu - 1, 0);
        }
        if ((mu + sigma) % 2 == 1) A[nu - 1] = 0;
        else A[mu] = 0;
    }
    if (mu == 2) visit();
    else proc_b(mu - 1, nu - 1, (mu + sigma) % 2);
}
```

Das Beispiel zeigt die Initialisierung und den Aufruf.

```
int main() {
```

```
    A.push_back(-1); // ignore index 0
    ll m = 3, n = 5; // m = number of partitions, n = number of elements
    for (ll j = 1; j <= n - m; j++) A.push_back(0); // init with {0,0,0,1,2}
    for (ll j = 1; j <= m; j++) A.push_back(j - 1); // n-m 0s, then 0 to m-1
    proc_f(m, n, 0);
    return 0;
}
```

3 Numerische Algorithmen

3.1 Naiver Primzahltest

Der Algorithmus testet die Primalität einer Zahl in $O(2^{\frac{n}{2}})$.

```
bool is_prime(ll n) {
    if (n == 2) return 1;
    if (n <= 1 || n % 2 == 0) return 0;
    ll r = (ll) sqrt(n);
    for (ll d = 3; d <= r; d += 2) if (n % d == 0) return 0;
    return 1;
}
```

4 Graphentheoretische Algorithmen

4.1 Topologisches Sortieren

5 Mathematik

5.1 Kombinatorik

5.1.1 Stirlingzahlen zweiter Art

Die Stirlingzahlen zweiter Art $S_{n,k}$ oder $\left\{{n \atop k}\right\}$ bezeichnet die Anzahl von k-Partitionen einer n-elementigen Menge. Es gilt

$$S_{n,k} = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n \quad \text{und} \quad S_{n,k} = \begin{cases} 0, & n < k, \\ 0, & n \in \mathbb{N}, k = 0, \\ 0, & n = 0, k \in \mathbb{N}, \\ 1, & n = k = 0, \\ S_{n-1,k-1} + kS_{n-1,k}, & \text{sonst.} \end{cases}$$

5.2 Graphentheorie