

Mathe-1

Charlotte P., Lena W., Vera C., Christian K. | 12. Juni 2018

ITI WAGNER & IPD TICHY

$$\begin{aligned}
 & \sum_{m=1}^{\infty} q_m(\omega) \int_0^{L_0} \left\{ (1 + i\eta) \frac{d^2}{dx^2} \left[k(x) \frac{d^2 \psi_m(x)}{dx^2} \right] - \omega^2 \psi_m(x) \right. \\
 & \quad \times \left. \left[\rho_l(x) + \frac{\pi}{4} \rho_f b^2(x) \Gamma(\beta(x, \omega), \alpha(x)) \right] \right\} \psi_n(x) dx \\
 & = \omega^2 \int_0^{L_0} \left\{ \hat{\theta}_B(\omega)(x + L_0) \left[\rho_l(x) + \frac{\pi}{4} \rho_f b^2(x) \Gamma(\beta(x, \omega), \right. \right. \\
 & \quad \alpha(x)) \left. \right] + \frac{\pi}{4} \rho_f b^2(x) \Delta \left(\beta(x, \omega), \frac{1}{b(x)} \left| \sum_{m=1}^{\infty} q_m(\omega) \psi_m(x) \right. \right. \right. \\
 & \quad \left. \left. + \hat{\theta}_B(\omega)(x + L_0) \right|, \alpha(x) \right) \\
 & \quad \times \left. \left[\sum_{m=1}^{\infty} q_m(\omega) \psi_m(x) + \hat{\theta}_B(\omega)(x + L_0) \right] \right\} \psi_n(x) dx. \quad (10)
 \end{aligned}$$

- 1 Big Integer
- 2 Exponentiation by squaring
- 3 Kombinatorik
- 4 Spieltheorie

- die maximale Zahl ist größer als integer?
- nehme long long
- die Zahl ist größer als long long
- ????????????????????????????????????? (Panik)

- die maximale Zahl ist größer als integer?
- nehme long long
- die Zahl ist größer als long long
- ?????????????????????????????????????? (Panik)

- die maximale Zahl ist größer als integer?
- nehme long long
- die Zahl ist größer als long long
- ??? (Panik)

- die maximale Zahl ist größer als integer?
- nehme long long
- die Zahl ist größer als long long
- ?????????????????????????????????????? (Panik)

- `import java.math.BigInteger`
- Konstruktor: `BigInteger(String val)`
- Methoden:
 - `BigInteger add(BigInteger val)`
 - `BigInteger multiply(BigInteger val)`
 - `BigInteger subtract(BigInteger val)`
 - ...

- Addition, Subtraktion in $\mathcal{O}(n)$
- Multiplikation in $\Theta(n^{\log_2 3})$ (Karatsuba)

C++? Selbst implementieren!

- Addition: Die Tafel ist da \longrightarrow
- Multiplikation (z.B. Karazuba-Multiplikation)

Karatsuba-Ofman Multiplikation^[1962]

Beobachtung: $(a_1 + a_0)(b_1 + b_0) = a_1 b_1 + a_0 b_0 + a_1 b_0 + a_0 b_1$

Function `recMult(a, b)`

assert a und b haben n Ziffern, sei $k = \lceil n/2 \rceil$

if $n = 1$ **then return** $a \cdot b$

Schreibe a als $a_1 \cdot B^k + a_0$

Schreibe b als $b_1 \cdot B^k + b_0$

$c_{11} := \text{recMult}(a_1, b_1)$

$c_{00} := \text{recMult}(a_0, b_0)$

return

$c_{11} \cdot B^{2k} +$

$(\text{recMult}((a_1 + a_0), (b_1 + b_0)) - c_{11} - c_{00}) B^k$

$+ c_{00}$

```
int exp(int x, int n) {  
    int result = 1;  
    for (int i = 0; i < n; i++) {  
        result *= x;  
    }  
    return result;  
}
```

Bei ICPC gehen wir davon aus, dass Multiplikation zweier Zahlen in $\mathcal{O}(1)$ liegt, also naive Exponentiation in $\mathcal{O}(n)$

Beobachtung:

$$x^n = \begin{cases} (x^2)^{n/2} & \text{für } n \text{ gerade} \\ x * (x^2)^{(n-1)/2} & \text{für } n \text{ ungerade} \end{cases} \quad (1)$$

Exponentiation by squaring, rekursive Implementierung

```
int exponentiationBySquaring(int n, int x) {  
    if (n < 0)  
        return exponentiationBySquaring(-n, 1/x);  
    if (n == 0)  
        return 1;  
    if (n == 1)  
        return x;  
    if (n % 2 == 0)  
        return exponentiationBySquaring(n/2, x*x);  
    return x*exponentiationBySquaring((n-1)/2, x*x);  
}
```

Exponentiation by squaring, iterative

Implementierung

```
int exponentiationBySquaring(int n, int x) {  
    if (n < 0) {  
        n = -n;  
        x = 1/x;  
    }  
    if (n == 0)  
        return 1;  
    int y = 1;  
    while (n > 1) {  
        if (n % 2 == 0) {  
            x = x * x;  
            n = n/2;  
        } else {  
            y = y * x;  
            x = x * x;  
            n = (n - 1) / 2;  
        }  
    }  
    return x*y;  
}
```

Da Multiplikation konstant viel Zeit benötigt, liegt die
Exponentiation $\mathcal{O}(\log(n))$

Hier kommt ein kleines Beispiel auf dem Tafel

- Baue eine Mauer aus bestimmten Ziegeln.
- jeder Ziegel ist 2 Einheiten breit und 1 Einheit hoch und kann beliebig gedreht werden.
- jede Mauer ist 2 Einheiten hoch und m Einheiten breit ($0 < m \leq 50$).
- Aufgabe: Wie viele Baumöglichkeiten?

Definition:

$$f(0) = 0$$

$$f(1) = 1$$

$$n > 1 : f(n) = f(n-1) + f(n-2)$$

Also: 0, 1, 1, 2, 3, 4, 8, 13, 21, 34, 55, 89...

Sollte man erkennen!

Mit DP in $O(n)$

Binet's Formel

//TODO: soll die hier rein? testcase schreiben ab wann zu ungenau

Alternativ: vorberechnen während andere Aufgaben gelöst werden.

Achtung: Wird sehr schnell sehr groß.

Definition

”Combinatorics is a branch of discrete mathematics concerning the study of countable discrete structures“^a

^aCompetitive Programming 3

Bei ICPC-Aufgaben erkennbar an:

- ”Wie viele Moeglichkeiten gibt es, ..?”
- ”Berechne die Anzahl an X.”
- Alles, was mit Zaehlen zu tun hat

Definition

"Combinatorics is a branch of discrete mathematics concerning the study of countable discrete structures"^a

^aCompetitive Programming 3

Bei ICPC-Aufgaben erkennbar an:

- "Wie viele Möglichkeiten gibt es, ..?"
- "Berechne die Anzahl an X.."
- Alles, was mit Zählen zu tun hat

Aufgabe

Lisa macht ein Austauschsemester in Australien. Um fuer einen Mathetest zu lernen, loest sie Rechen-Aufgaben, die ihr eine Kommilitonin diktiert hat. Leider hat die Kommilitonin nicht gesagt, wie die Aufgaben geklammert sind.

Gegeben die Anzahl an Faktoren, wie viele verschiedene Wege gibt es diese zu klammern?

Beispiel:

- Gegeben: $\{a, b, c, d\}$
- Gesucht: Moeglichkeiten fuer Klammerung
- $a(b(cd))$, $(ab)(cd)$, $((ab)c)d$, $(a(bc))d$, $a((bc)d)$

Aufgabe

Lisa macht ein Austauschsemester in Australien. Um fuer einen Mathetest zu lernen, loest sie Rechen-Aufgaben, die ihr eine Kommilitonin diktiert hat. Leider hat die Kommilitonin nicht gesagt, wie die Aufgaben geklammert sind.

Gegeben die Anzahl an Faktoren, wie viele verschiedene Wege gibt es diese zu klammern?

Beispiel:

- Gegeben: $\{a, b, c, d\}$
- Gesucht: Moeglichkeiten fuer Klammerung
- $a(b(cd))$, $(ab)(cd)$, $((ab)c)d$, $(a(bc))d$, $a((bc)d)$

Wie viele Möglichkeiten gibt es, k Objekte aus einer Menge von n verschiedenen Objekten zu ziehen?

$$C(n, k) = \binom{n}{k} = \frac{n!}{(n-k)! \times k!}$$

Rekursive Definition:

$$C(n, 0) = C(n, n) = 1$$

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$

Wie viele Möglichkeiten gibt es, k Objekte aus einer Menge von n verschiedenen Objekten zu ziehen?

$$C(n, k) = \binom{n}{k} = \frac{n!}{(n-k)! \times k!}$$

Rekursive Definition:

$$C(n, 0) = C(n, n) = 1$$

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$

Wie viele Möglichkeiten gibt es, k Objekte aus einer Menge von n verschiedenen Objekten zu ziehen?

$$C(n, k) = \binom{n}{k} = \frac{n!}{(n-k)! \times k!}$$

Rekursive Definition:

$$C(n, 0) = C(n, n) = 1$$

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$

Tipps:

- Meist interessieren nicht alle Werte von $C(n, k)$
 - Implementierung deshalb mit top-down
- Fakultät kann sehr gross werden
 - benutze BigInteger
 - bei grossem k : $C(n, k) = C(n, n - k)$

Definition:

$$\begin{aligned} \text{Cat}(n) &= \frac{1}{n+1} \binom{2n}{n} \\ &= \frac{(2n)!}{(n+1) \times n! \times n!} = \frac{(2n)!}{(n+1)! \times n!} \\ \text{Cat}(n+1) &= \frac{(2n+2) \times (2n+1)}{(n+2) \times (n+1)} \times \text{Cat}(n) \end{aligned}$$

$Cat(n)$ entspricht zum Beispiel:

- Anzahl verschiedener Binaer-Baeume mit n Knoten
- Anzahl korrekter Klammerausdruecke mit n Klammerpaaren
- Anzahl verschiedener Moeglichkeiten, $n + 1$ Faktoren korrekt zu klammern

Aufgabe

Lisa macht ein Austauschsemester in Australien. Um fuer einen Mathetest zu lernen, loest sie Rechen-Aufgaben, die ihr eine Kommilitonin diktiert hat. Leider hat die Kommilitonin nicht gesagt, wie die Aufgaben geklammert sind.

Gegeben die Anzahl an Faktoren, wie viele verschiedene Wege gibt es diese zu klammern?

Loesung:

- Sei n die Anzahl an Faktoren
- $Cat(n - 1)$ loest die Aufgabe

Aufgabe

Lisa macht ein Austauschsemester in Australien. Um fuer einen Mathetest zu lernen, loest sie Rechen-Aufgaben, die ihr eine Kommilitonin diktiert hat. Leider hat die Kommilitonin nicht gesagt, wie die Aufgaben geklammert sind.

Gegeben die Anzahl an Faktoren, wie viele verschiedene Wege gibt es diese zu klammern?

Loesung:

- Sei n die Anzahl an Faktoren
- $Cat(n - 1)$ loest die Aufgabe

Zusammenfassung - Kombinatorik bei ICPC

Die Lösung fuer eine Kombinatorik-ICPC-Aufgabe ist meist eine kurze rekursive Formel, oft in Verbindung mit Greedy oder DP. Der Aufwand liegt nicht in der Implementierung, sondern im Aufstellen der Formel.

- Kombinatorik-Aufgaben von **einer** Person bearbeiten lassen
 - bestenfalls mit guten mathematischen Kenntnissen
- Sobald die Formel fertig ist, Loesung coden und abgeben!

Zusammenfassung - Kombinatorik bei ICPC

Die Lösung fuer eine Kombinatorik-ICPC-Aufgabe ist meist eine kurze rekursive Formel, oft in Verbindung mit Greedy oder DP. Der Aufwand liegt nicht in der Implementierung, sondern im Aufstellen der Formel.

- Kombinatorik-Aufgaben von **einer** Person bearbeiten lassen
 - bestenfalls mit guten mathematischen Kenntnissen
- Sobald die Formel fertig ist, Loesung coden und abgeben!

Zusammenfassung - Kombinatorik bei ICPC

Die Lösung fuer eine Kombinatorik-ICPC-Aufgabe ist meist eine kurze rekursive Formel, oft in Verbindung mit Greedy oder DP. Der Aufwand liegt nicht in der Implementierung, sondern im Aufstellen der Formel.

- Kombinatorik-Aufgaben von **einer** Person bearbeiten lassen
 - bestenfalls mit guten mathematischen Kenntnissen
- Sobald die Formel fertig ist, Lösung coden und abgeben!

Zusammenfassung - Kombinatorik bei ICPC

Die Lösung fuer eine Kombinatorik-ICPC-Aufgabe ist meist eine kurze rekursive Formel, oft in Verbindung mit Greedy oder DP. Der Aufwand liegt nicht in der Implementierung, sondern im Aufstellen der Formel.

- Kombinatorik-Aufgaben von **einer** Person bearbeiten lassen
 - bestenfalls mit guten mathematischen Kenntnissen
- Sobald die Formel fertig ist, Loesung coden und abgeben!

Gaenige Formeln sollte man kennen...

Zusatz-Tipp!

Gaenige Formeln sollte man kennen...
...oder ausprobieren!

On-Line Encyclopedia of Integer Sequences

Unter <http://oeis.org/> kann man die ersten Lösungen fuer kleine Probleminstanzen eingeben und so pruefen, ob bereits eine Formel fuer diese Folge existiert.

- Nullsummenspiel
- Genau ein Gewinner
- Alle spielen perfekt
- Gibt es eine Gewinnstrategie?

- Nullsummenspiel
- Genau ein Gewinner
- Alle spielen perfekt
- Gibt es eine Gewinnstrategie?

- Nullsummenspiel
- Genau ein Gewinner
- Alle spielen perfekt
- Gibt es eine Gewinnstrategie?

- Nullsummenspiel
- Genau ein Gewinner
- Alle spielen perfekt
- Gibt es eine Gewinnstrategie?

- sechs Münzen, zwei Spieler
- immer abwechselnd maximal drei Münzen nehmen
- Wer die letzte Münze nimmt, gewinnt

- sechs Münzen, zwei Spieler
- immer abwechselnd maximal drei Münzen nehmen
- Wer die letzte Münze nimmt, gewinnt

- sechs Münzen, zwei Spieler
- immer abwechselnd maximal drei Münzen nehmen
- Wer die letzte Münze nimmt, gewinnt

- Knoten: aktueller Spieler und Spielsituation
- Kanten: legale Spielzüge
- Wurzel: Spielsituation beim Start
- Blätter: Geben Bewertung (-1 oder 1) an

- Knoten: aktueller Spieler und Spielsituation
- Kanten: legale Spielzüge
- Wurzel: Spielsituation beim Start
- Blätter: Geben Bewertung (-1 oder 1) an

- Knoten: aktueller Spieler und Spielsituation
- Kanten: legale Spielzüge
- Wurzel: Spielsituation beim Start
- Blätter: Geben Bewertung (-1 oder 1) an

- Knoten: aktueller Spieler und Spielsituation
- Kanten: legale Spielzüge
- Wurzel: Spielsituation beim Start
- Blätter: Geben Bewertung (-1 oder 1) an

- Knoten: aktueller Spieler und Spielsituation
- Kanten: legale Spielzüge
- Wurzel: Spielsituation beim Start
- Blätter: Geben Bewertung (-1 oder 1) an

- Min-Max-Strategie: Gewinn mit größtem Unterschied

$$\minmax(s, k) = \begin{cases} k.Bewertung & \text{für } k \text{ Blatt-Knoten} \\ \min \{ \minmax(k') \mid k' \text{ Kindknoten} \} & \text{falls } s = \min \\ \max \{ \minmax(k') \mid k' \text{ Kindknoten} \} & \text{falls } s = \max \end{cases} \quad (2)$$

- Spieler-IDs können auch weggelassen werden:

$$\minmax(k) = \begin{cases} k.Bewertung & \text{für } k \text{ Blatt-Knoten} \\ -\min \{ \minmax(k') \mid k' \text{ Kindknoten} \} & \text{sonst} \end{cases} \quad (3)$$

- Min-Max-Strategie: Gewinn mit größtem Unterschied

$$\minmax(s, k) = \begin{cases} k.Bewertung & \text{für } k \text{ Blatt-Knoten} \\ \min \{ \minmax(k') \mid k' \text{ Kindknoten} \} & \text{falls } s = \min \\ \max \{ \minmax(k') \mid k' \text{ Kindknoten} \} & \text{falls } s = \max \end{cases} \quad (2)$$

- Spieler-IDs können auch weggelassen werden:

$$\minmax(k) = \begin{cases} k.Bewertung & \text{für } k \text{ Blatt-Knoten} \\ -\min \{ \minmax(k') \mid k' \text{ Kindknoten} \} & \text{sonst} \end{cases} \quad (3)$$

- wie gewohnt als Baum

```
struct Node {  
    vector<int> children;  
    int Bewertung;  
};
```

- Spieler-IDs können weggelassen werden

```
int minmax(Zustands-Knoten k):  
    if(k ist Blatt){  
        return k.getBewertung  
    } else {  
        for(alle Kindknoten kind von k){  
            res = - min(res, Bewertung(k))  
        }  
    }
```

- wie gewohnt als Baum

```
struct Node {  
    vector<int> children;  
    int Bewertung;  
};
```

- Spieler-IDs können weggelassen werden

```
int minmax(Zustands-Knoten k):  
    if(k ist Blatt){  
        return k.getBewertung  
    } else {  
        for(alle Kindknoten kind von k){  
            res = - min(res, Bewertung(k))  
        }  
    }
```

- Beispiel: Zahlen abwechselnd mit 2-9 multiplizieren
- erster über n (Grenze) gewinnt
- je acht Kinder: Viel zu viel
- optimal: Immer abwechselnd 9 und 2

- Beispiel: Zahlen abwechselnd mit 2-9 multiplizieren
- erster über n (Grenze) gewinnt
- je acht Kinder: Viel zu viel
- optimal: Immer abwechselnd 9 und 2

- Beispiel: Zahlen abwechselnd mit 2-9 multiplizieren
- erster über n (Grenze) gewinnt
- je acht Kinder: Viel zu viel
- optimal: Immer abwechselnd 9 und 2

- Beispiel: Zahlen abwechselnd mit 2-9 multiplizieren
- erster über n (Grenze) gewinnt
- je acht Kinder: Viel zu viel
- optimal: Immer abwechselnd 9 und 2

- mehrere Haufen mit Objekten
- zwei Spieler nehmen abwechselnd von einem Haufen
- Wer das letzte Objekt nimmt, gewinnt
- Für ein oder zwei Haufen mit Baum möglich

- mehrere Haufen mit Objekten
- zwei Spieler nehmen abwechselnd von einem Haufen
- Wer das letzte Objekt nimmt, gewinnt
- Für ein oder zwei Haufen mit Baum möglich

- mehrere Haufen mit Objekten
- zwei Spieler nehmen abwechselnd von einem Haufen
- Wer das letzte Objekt nimmt, gewinnt
- Für ein oder zwei Haufen mit Baum möglich

- mehrere Haufen mit Objekten
- zwei Spieler nehmen abwechselnd von einem Haufen
- Wer das letzte Objekt nimmt, gewinnt
- Für ein oder zwei Haufen mit Baum möglich

- Anzahl Objekte in Haufen binär mit xor verknüpfen
- Summe auf Null bringen, um zu gewinnen
- Immer möglich

- Anzahl Objekte in Haufen binär mit xor verknüpfen
- Summe auf Null bringen, um zu gewinnen
- Immer möglich

- Anzahl Objekte in Haufen binär mit xor verknüpfen
- Summe auf Null bringen, um zu gewinnen
- Immer möglich

- Theorem von Sprague-Grundy: Jedes neutrale Spiel äquivalent zu Standard-Nim-Spiel
- Grundy-Zahlen: kleinste Zahl, die nicht Grundy-Zahl von Nachfolgerstellung ist
- Gewinnstrategie: Grundy-Zahl möglichst immer auf 0 bringen

- Theorem von Sprague-Grundy: Jedes neutrale Spiel äquivalent zu Standard-Nim-Spiel
- Grundy-Zahlen: kleinste Zahl, die nicht Grundy-Zahl von Nachfolgerstellung ist
- Gewinnstrategie: Grundy-Zahl möglichst immer auf 0 bringen

- Theorem von Sprague-Grundy: Jedes neutrale Spiel äquivalent zu Standard-Nim-Spiel
- Grundy-Zahlen: kleinste Zahl, die nicht Grundy-Zahl von Nachfolgerstellung ist
- Gewinnstrategie: Grundy-Zahl möglichst immer auf 0 bringen