Vorwort

- Schaffe es nicht in die Kamera zu schauen -> Entschuldigt bitte
- Wenn ich Fragen stelle -> Ratet gerne
    - Möchte, dass ihr dabei bleibt und mitdenkt
    - Ob es richtig ist, ist unwichtig
- **Voraussichtliche Länge**

Allgemeines zu Algorithmen

Was ist ein Algorithmus

- eindeutige Folge von Anweisungen, mit endlicher Anzahl an Schritten, um ein Problem zu lösen.

Beispiel aus der Schule:

Multiplikation zweier mehrstelliger Zahlen

```
  4 2 * 6 8
-----------
    2 4
+     1 2
+     3 2
+       1 6
-----------
    2 8 5 6
```

Eigenschaften von Algorithmen

Endlichkeit Endliche Länge der Beschreibung des Algorithmus und endlich großer Ressourcenverbrauch (Zeit und Speicher) bei der Ausführung

Komplexität (Aufwand an Rechenzeit und Speicherplatz)

Terminiert (Liegt ein Ergebnis nach endlich vielen Schritten vor?)

Determiniertheit (gleiche Ausgabe bei gleicher Eingabe. Können andere innere Zustände durchlaufen)

Deterministisch

- Bei gleicher Eingabe, gleiche Ausgabe und gleiche Folge an Zuständen wird durchlaufen

Anmerkung:

- Deterministisch ist nicht Determiniertheit

- Ein deterministischer Algorithmus ist immer determiniert, d. h., er liefert bei gleicher Eingabe immer die gleiche Ausgabe.
- Die Umkehrung aber gilt nicht: So gibt es Algorithmen, die nicht-deterministisch, aber trotzdem determiniert sind

Determinismus = deterministisch != determiniert

Nicht-Deterministisch

- Bei gleicher Eingabe, gibt es mehere Möglichkeiten in den nächsten Zustand überzugehen. Der Maschiene wird keine Vorgabe gemacht, welcher Weg zu wählen ist.
- Konzept der theoretischen Informatik
- Theoretisches Modell, **meist** nicht praktisch realisiertbar

Warum macht man das, wenn es sich meist nicht umsetzen lässt?

- Es ist leichter einen nicht-deterministischen Algorithmus zu finden als einen deterministischen
- Zweck von nichtdeterministischen : Komplexität von Problemen beschränken
    - Wenn man einen nichtdeterministischen Algorithmus angeben kann, ist das Problem leichter als wenn die nicht möglich ist.
- Wichtige Frage der Informatik: unter welchen umständen kann man nichtdeterministische Algorithmen simulieren?

Veranschaulichung von nichtdeterministischen Algorithmen:

probabilistischer Algorithmus

Beispiel Las Vegas Algorithmus:

- Nichtdeterministisch determiniert
- Wenn der Algorithmus terminiert, liefert dieser ein konkretes und gleiches Ergebnis zurück
- Zeitkomplexität hängt von einer Zufallsvariabel ab

Code Beispiel Las Vegas

- Was fällt auf?
    - Gleiches Ergebnis wenn zu einem Ergebnis gekommen wird
        - Wenn terminiert, determiniert
    - Unterschiedliche Laufzeit und Zwischenschritte
        - Nicht-deterministisch

Rekursive und Iterative Algorithmen

Iterative Algorithmen

(das, was wir bisher die ganze Zeit gemacht haben)

Lat. Iterare = wiederholen

Mehrfach auszuführende Arbeitsschritte werde durch Schleifen umgesetzt

Zu beachten: Endlosschleifen können entstehen

Rekursive Algorithmen

Exkurs: Ada Lovelace

Geboren Augusta Ada Byron 10. Dezember 1815; gestorben mit 36 am 27. November 1852

Gilt als erste Programmieren

- Hat unter anderem ein Programm für die (theoretische) „Analytical Engine“ zum Berechnen der Bernoulli Zahlen
    - Note G berechnen von B_7 (8. Bernoulli Zahl)
- Note G beschrieb ein rekursives Programm
- Rekursive Algorithmen sollten Lovelace Algorithmen heißen

rekursion

Lat. Recurrere = wiederholen

Eine Methode ruft sich selbst auf

Direkte Rekursion: Eine Methode ruft sich selbst auf

Indirekte Rekursion: a() ruft b() auf, b() ruft c() auf, c() ruft d() auf und d() ruft a() auf

Wie sieht eine rekursive Methode aus?

Methode(n){

Base Case

(Beispiel: Wenn n < 1 -> return n)

Rekursiver Aufruf

(Beispiel: Methode(n-1))

}

Wann kann man Rekursion anwenden?

Problem lässt sich in kleinere Probleme aufteilen, die sich rekursiv lösen lassen

Der Base Case ist klar und einfach zu identifizieren

Zu Beachten:

- Jeder rekursive Aufruf wird auf den Stack gelegt
- Der Stack ist reservierter Speicher im Arbeitsspeicher
    - endlich
- Wenn der Speicher voll ist: Stackoverflow -> Programmabbruch

Codebeispiel Rekursion

Berechnung von x! (x factorial)

3! = 3*2*1 = 6

5! = 5*4*3*2*1 = 120

Siehe Bilder factorial code, factorial diagram

Vergleich Rekursion und Iterativ

Vor- und Nachteile von Rekursion und Iteration

Vorteile von Rekursion:

- Schlicht und Elegant
    - Weniger Code Zeilen
- Lösen komplexer Probleme
    - Besonders bei Baum oder Hierarchischen Strukturen
- Divide and Conquer wird implementiert
    - Effizient für viele Algorithmen

Nachteile von Rekursion:

- Stack Overflow

  Frage: Bei dem Beispiel Faktorial zu berechnen: Was ist das erste n, welches einen Stack Overflow triggert? 9637

- Mehr Arbeitsspeicher wird benötigt
- vielen Funktionsaufrufe und Stack-Management
    - frisst Performance und kann langsamer sein
- Schwer zu Debuggen

Wann kann man Rekursion gut verwenden?

- „Rekursion sollte man möglichst vermeiden“ ~ Goethe
- Das Problem hat eine Rekursive Struktur
- Eine Rekursive Lösung ist simpler und intuitiver

- Die tiefe der Rekursion ist bekannt
    - !!Stack Overflow!!
- Performance ist nicht das Hauptziel

Vorteile von Iteration:

- Effizienter Umgang mit Arbeitsspeicher
- Besser optimierbar -> bessere Performance
- Einfach zu debuggen

Nachteile von Iteration

- Komplexität des Codes
    - Besonders bei Hierarchischen Strukturen
    - Schwer verständlich
- „off-by-one-error“
    - Starte ich die for Schleife mit i=0 oder i=1
- State Management über Iteration kann zu Fehlern führen
    - State Management beschreibt das speichern, lesen und verändern von Werten

Wann kann man Iteration gut verwenden?

- Immer!
- Für Performance optimieren
- Große Anzahl an Input
- Effiziente Nutzung des Arbeitsspeichers ist wichtig

Es ist bewiesen, dass sich jeder rekursive Algorithmus auch iterativ ausdrücken lässt.

O-Notation und Komplexität

Bild O-notation

- Notation zur Beschreibung der Komplexität von Algorithmen
    - vergleichbar
- Darstellung abhängig von der Eingabegröße
- Gut: O(log(n)) und kleiner
- Okay: O(n)
- Ganz okay O(n*log(n))
- Eieieiei: O(n^2); O(n^3)
- Grenze des praktisch machbaren: O(n^k)

Zwei Arten von Komplexität: Zeit und Speicher (time and space)

Zeitkomplexität:

- Zeit die ein Algorithmus braucht um durchzulaufen
- Zeit ist Funktion der Eingabelänge (z.B. O(n^2)), nicht die Laufzeit

Speicherkomplexität

- Besteht aus mehreren Teilen
    - Eingabe Speicher (Input Space)
        - Codezeilen, Daten die sortiert werden
        - In Praxis oft weggelassen
    - Hilfsspeicher (Auxiliary space and temporary memory)
        - Zusätzlicher Speicher den der Algorithmus zur Durchführung baucht
        - Beispiel: Array soll sortiert werden, bereits sortierte Teile werden in einen extra Array geschrieben
    - Call Stack, Rekursionsspeicher
        - Speicher der bei Rekursion benötigt wird
    - Data structure overhead
        - Zusätzlicher Speicher bei Datenstrukturen, der benötigt information über die Daten zu speichern und nicht die Daten selber
        - Beispiel: Linked List: speicher auch wo das nächste Element liegt

Bild O-notation (wird gleich sehr relevant)

Anmerkung:

- Komplexität kann man für eigene Algorithmen ausrechen
- Rechenregeln bei verschachtelten Algorithmen
- Online Tools die es ausrechen

polynomieller Algorithmus

- Zeitkomplexität
- Es gibt eine natürliche Zahl k mit $t \in O(n^k)$
- Polynomialzeit gilt als Grenze zwischen praktisch lösbaren und unlösbaren
    - In der Praxis nicht ganz trennscharf

P-NP-Problem

P: Klasser aller Probleme, die sich auf einer deterministischen sequentiellen Maschine in Polynomialzeit lösen lassen

NP: Die Klasse aller Probleme, die sich von einer nichtdeterministischen Maschine in Polynomialzeit lösen lassen

Klar: $P \subseteq NP$

Frage: Sind NP und P voneinander echt verschieden

- Kann man auch die schwersten Probleme der Klasse NP mit deterministischen Maschinen effizient lösen?

Ist das 4. Der 7 Millennium-Probleme (bisher nur das 5. Gelöst)

Listen Sortieren

Geben ist eine unsortierte Liste mit Zahlen

Bubble Sort

- Vergleicht das aktuelle Element mit dem darauf folgenden
    - Wenn das folgende Element größer ist, wird das folgende Element zum aktuellen Element
    - Wenn das folgende Element kleiner ist, werden die Positionen vom aktuellen und folgenden Element getauscht, das aktuelle Element bleibt das aktuelle Element
- Im Ergebnis: das größte Element wandert an den Rand der Liste

Visualisierung mit 20 Elementen: https://www.sortvisualizer.com/bubblesort/

Code Beispiel Bubblesort (siehe Bild)

Komplexität:

Average Complexity: O(n2)

Best Case: O(n)

- (sortiert in aufsteigender Weise)

Worst Case: $O(n^2)$

- (sortiert in absteigender Weise)

Space Complexity O(1)

Quicksort

Nicht-deterministisch determiniert

Bild Quicksort 2

- Es wird ein zufälliges Element gewählt (Pivot Element)
- Alle kleinere Elemente werde auf die Linke Seite von Pivot geschoben, alle größeren auf die rechte
    - 2 Teilbereiche entstehen
    - Das Pivot Element steht an der richtigen Stelle
- Das gleiche Muster wird rekursiv auf die beiden neuen Teilbereiche angewendet

Diese Art der Problemlösung wird Divide and Conquer[1] (Teile und herrsche) genannt.

Vorteil: Durch das Teilen der Liste können die verschiedenen Teilbereiche gleichzeitig berechnet werden

Visualisierung mit 20 und 1000 Elementen: https://www.sortvisualizer.com/quicksort/

Codebeispiel Quicksort (siehe Bild)

Average Complexity: O(n × log n)

Best Case: O(n × log n)

Worst Case: O(n2)

- Bereits sortiert
- Sortiert in umgekehrter Reihenfolge
- Alle Elemente sind Gleich
- Pivot-Element immer das gröchste/kleinste (extrem unwahrscheinlich)

Space Complexity: O(n)

[1] Aus dem lateinischen „Divide et impera“ (Ursprung 15. Oder 16. Jahrhundert). Die Redewendung empfiehlt eine zu beherrschende Gruppe in Untergruppen mit widerstrebenden Interessen zu unterteilen. Dadurch soll erreicht werden, dass sich die Untergruppen gegeneinander wenden, statt sich gemeinsam gegen den gemeinsamen Unterdrücker aufzulehnen. Dieses Prinzip fand schon in der römischen Außenpolitik oder im europäischen Imperialismus und Kolonialismus angewendet.

Nebenbemerkung:

- Quicksort verändert die ursprüngliche Reihenfolge der Elemente
- Wenn man einen Datensatz mit Vorname, Nachname und Geburtstag hat, der nach Nachname sortiert ist. Diesen mit Quicksort nach Geburtsdatum sortiert, kann das dazu führen, dass bei gleichem Geburtstag die Einträge nicht mehr nach Nachname sortiert sind
    - Bei bubblesort ist dies nicht der Fall

Unterschied zwischen CPU und GPU

Anmerkung: sehr grundlegend und vereinfacht

CPU: Von Neumann Architektur

Von 1945

(Idee der Turing Maschine: 1936; Entschlüsselung der Enigma: Januar 1940)

Grundlage für die Arbeitsweise heutiger Computer

Beschreibt design Architektur für einen digitalen Computer

Besteht aus folgenden Teilen:

- Zentrale Logische und Arithmetische Einheit
    - Führt logische und mathematischen Operationen
- Kontrolleinheit
    - Organisiert die Reihenfolge der Operationen
- Logische Einheit und Kontrolleinheit bilden die CPU
- Speichereinheit
    - Speichert Anweisungen und Daten
- Eingabe und Ausgabe Gerät um Daten zwischen der Speichereinheit und dem Speichermedium außerhalb zu übertragen

Unterschiede an Bildern erklärt

Bilder CPU GPU

Begriffserklärung CPU (Central Processing Unit):

ALU: Arithmetisch-logische Einheit

Cache:

- Reduziert die Kosten um auf Daten zuzugreifen im Vergleich zum Hauptspeicher
    - Auf der Hauptspeicher zuzugreifen ist zehn bis hunderte Male langsamer
    -

DRAM (Dynamic Random Access Memory):

- Daten werden in Kondensatoren gespeichert
- Müssen zyklisch aufgefrischt werden, damit sie nicht verloren gehen
    - Alle paar millisekunden

Begriffserklärung GPU (Central Graphics Unit):

- DRAM (manchmal VRAM (Video RAM genannt): Bei der GPU darauf spezialisiert Grafiken und Texturen zu speichern
- Gelbes Kästchen: Kontrolleinheit
- Orangenes Kästchen: Cache
- Grünes Kästchen: ALU

Stärker der CPU

- Vielseitig
    - Kann verschiedenste komplexe Aufgaben ausführen
        - Ausführen von Programmen, Betriebssystem
- Gute Single-Thread Leistung
    - Thread wird von einem Prozess erstellt
    - Der Code im Thread wird dann von einem Core bearbeitet
    - Programm: Itereration durch einen Array -> 1 Thread -> single thread
    - Parallelsierung von Quicksort -> Multithreadding
        - Mehere Threads die gleichzeitig bearbeitet werden
- Multitasking
    - Kann mehere Programm gleichzeitig ausführen
        - Aufgaben werden in Threads unterteilt
        - Bei mehreren Kernen können mehere threads parralel ausgeführt werden
- Geringe Latenz

Stärken der GPU

- Parralele Verarbeitung
    - Kann tausende von Berechnungen gleichzeitig ausführen
    - Besonders effizient wenn es repetative Berechnungen sind
        - Matrizen Multiplikation z.B.: Deep learning, Simulationen, rendern von Videos
- Gut beim Rendern von Grafiken
- Hoher Datendurchlauf
    - Aufgrund der hohen Anzahl an Kernen und Rechendichte
    - Gut geeignet für Matrizen Rechnungen

Matrizen Rechnungen

Was ist eine Matrize?

Darstellung: siehe Bild Matrizen1

- Man kann Matrizen als 2D-Arrays betrachten
    - Nicht bei jeder Matrizen ergibt jede Rechenoperation Sinn
- Dargestellte Matrizen hat die Dimension (m*n)
    - Siehe Bild Matritzen2 und 3

Matrizen Multiplikation

- Wichtigeste Matrizen Rechnung bei Computern

Einfacher Rechenweg

Siehe Bilder Matrizen 4 und 5

- Spalte der ersten Matrix * Zeile der zweiten Matrix -> eine Koordinate der neuen Matrix
- Bei Matrizen mit unterschiedlichen Dimensionen -> mit 0 auffüllen

Matrix mit vektor multipliziert:

- Zeile der Matrix wird mit komplettem Vektor multipliziert

Anwendungsbeispiele

Transformationen

Problem: Objekt im dreidimensionalen Raum, das man verschieben, rotieren und skalieren möchte

Lösung: dreidimensionale Transformationsmatrix

Nun folgen affine tranformationen -> beibehalten von Parallelität und Kollinearität (Punkte auf gemeinsamer Geraden)

Bild: Matrizen 8

Bildbeschreibung:

- erste Matrix: Orginalkoordinaten
- zweite Matrix: transformationsmatrix
- T: Position
- M: Rotation
- S: Skalierung
- Zahlen: fixed -> nur 12 Freiheitsgrade

Bilder Matrizen 9-15

9: zu einer neuen Position bewegen

11: Rotation um die z-Achse lässt alle Punkte mit dem gleichen z unverändert

13: Rotation um die x-Achse

14: Rotation um die y-Achse

15: Stauchen oder Zerren entlang jeder Achsee; bei negativen sx, sy, sz wird gespiegelt

Um Rechenleistung zu sparen kann man Transformationsmatrizen auf verschiedene Art zusammenfassen

Das beschriebene ist eine 4x4x4 Matrix Multiplikation

Berechnung dieser:

Normal ($O(n^3)$): 64 Skalare Multiplikationen

Strassmannn Algorithmus (1969) ($O(n^{\log_2(7)})$: 49 Skalare Multiplikationen

Algorithmus von Googles Alpha Evolve (2025): 48 Skalare Multiplikationen

Maschinelles lernen

- Deep learning: riesige Datenmengen müssen gleichzeitig verarbeitet werden
- Neuronale Netzen sind gewichtete Verbindungen zwischen Knoten
    - Darstellbar als Matrix
    - Daten werden als Vektoren oder Matrizen eingespeist
    - Auf jeder Ebene findet Matrix Multiplikation statt
        - Lernen passiert durch das Anpassen der Gewichtung durch Matrix Rechnungen, bis das Modell einen verlässlichen Output hat

Verschlüsselung

Caeser Verschlüsselung

- einfaches symmetrisches Verschlüsselungsverfahren
    - symmetrisches: beide Teilnehmer verwenden den selben Schlüssel
- Klartextbuchstabe wird auf geheimtextbuchstabe abgebildet
    - Durch zyklische Verschiebung nach rechts
- Anzahl der verschobenen Zeichen bildet den Schlüssel
    - Entschlüsselung durch zyklische Verschiebung nach rechts

Verschiebung um 3 Zeichen mit dem Schlüssel C:

(siehe Bild Caeser1-2)

```
Klar:    A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Geheim:  D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
```

Mathematische Darstellung:

$Verschlüsseln_K(P) = (P+K)\%26$

$Entschlüsseln_K(C) = (C-K)\%26$

P: Klartextbuchstabe

K: Verschiebung

C: Geheimbuchstaben

Bei einem Alphabet mit 26 Zeichen

- Caeser: 1 privte Key den sich beide Teilen
    - Muss vorher abhör- und manipulationssicher ausgetauscht werden

Besser: asymmetrische Verschlüsselung

- erstes veröffentlichte Verfahren war RSA Verschlüsselung von 1977

Public-Key-Verschlüsselungsverfahren

Asymmetrisches Verschlüsselungsverfahren

- Ein Empfänger hat zwei Schlüssel
- ein öffentlicher Schlüssel wird benutzt um Klartext in einen Geheimtext umzuwandeln
- mit dem privaten Schlüssel kann der Klartext wiedergewonnen werden
- der öffentliche Schlüssel muss dem Inhaber des privaten Schlüssels zweifelsfrei zugeordnet werden

RSA

Rivest–Shamir–Adleman

Asymmetrisches Verschlüsselungsverfahren

Erstveröffentlichung 1977

Deterministischer Algorithmus

- Für bestimmte Angriffe anfällig
- Wird heute nicht mehr in der Ursprünglichen Form verwendet

Verschlüsselung basiert auf Einwegpermutation mit Falltür (engl. trapdoor one-way permutation, kurz TOWP)

- Funktioniert nur aufgrund der eigentschaften von TOWPs

Einwegfunktionen

Eine mathematische Einwegfunktion f muss folgende Eigenschaften aufweisen

- Es ist einfach für eine Funktion f(x)=y mit gegebenem x das y zu berechnen
    - Einfach = in Polynomialzeit berechenbar
- Aus dem y auf das ursprüngliche x zu schließen ist schwer
    - Schwer: kein probabilistischer Algorithmus kann x in Polynomialzeit berechnen

Beispiel: Multiplizieren zweier großer Primzahlen hat das Ergebnis y. Primfaktorzerlegung von y ist nun nicht in Polynomialzeit lösbar. (Annahme)

Problem der Einwegfunktionen:

- Unbekannt ob Einwegfunktionen exisiteren
- Beweis, dass es Einwegfunktionen gibt, würde den Beweis für das P-NP-Problem darstellen

Einwegfunktionen mit Falltür

Es ist möglich diese effizient umzukehren, wenn man eine gewisse Zusatzinformation besitz

Beispiel:

Die Multiplikation zweier sehr großer Primzahlen hat das Ergebnis: 4.700.140.157

Zusatzinformation: Eine der Primfaktoren ist: 52067

Also kann man rechnen: 4.700.140.157/52067 = 90271

Das Ergebnis ist der zweite Primfaktor

Bijektivität

(siehe Bild Bijection)

- Eine bijektive Funktion ordnet jedem Element der Definitionsmenge genau ein Element der Zielmenge zu
- Eine Bijektion auf sich selbst nennt man Permutation

Beispiel für RSA-Verschlüsselung

Anmerkung: erst zeige ich die Anwendung der Verschlüsselung und anschließend die Generierung der Schlüssel

Öffentlicher Schlüssel: (e, N)

Privater Schlüssel: (d, N)

Öffentlicher Schlüssel: (5, 14)

Privater Schlüssel: (11, 14)

- 11 ist der geheime Teil des privaten Schlüssels. Ein Verlust des Schlüssel würde alle Nachrichten entschlüsselbar machen

Klartex t m: „B“ -> 2

Verschlüsseln:

$C = M^e \% N$

$2^5 \% 14 = 32 \% 14 = 4$

Geheimtext c: „D“ -> 4

Entschlüsseln:

$c^d \% N = m$

$4^{11} \% 14 = 4194304 \% 14 = 2$

Entschlüsselte Nachricht: „B“ -> 2

Generieren der Schlüssel

- In der Realität sind es Primzahlen mit vielen hunderten Stellen

1.:

Wähle zwei Primzahlen p und q

p = 2, q = 7

2.:

Berechne N = p * q

N =14

3.:

Berechne die Anzahl an Teilfremden (coprimes) von 14

[1, ~~2~~, 3, ~~4~~, 5, ~~6~~, ~~7~~, ~~8~~, 9, ~~10~~, 11, ~~12~~, 13, ~~14~~]

Beispiel:

Primfaktoren von 14 = 2 * 7

Primfaktoren von 6 = 2 * 3

Folgerung: 2 und 7 können nicht Teilfremd sein, da sie die Primfaktoren von 14 sind. Und 6 kann nicht Teilfremd sein, weil einer der Primfaktoren sich mit 14 geteilt wird

Einfacher: Phi Funktion von N: $\Phi(N) = (p-1)*(q-1)$

$\Phi(N) = 6$

4.:

Wähle Zahl e

Bedingung 1: $1 < e < \Phi(N)$

Bedingung 2: Teilfremd mit N und $\Phi(N)$

Bedingung 1 erfüllen: 2, 3, 4, 5

2 un 3 sind Primfaktoren von 6; 4 teilt sich Primfaktoren mit 6

Bedingung 2 erfüllt: 5

5.:

Wähle d

Bedingung: d * e % Φ(N) = 1

5d % 6 = 1

5d könnte 5, 10, 15, 20, 25, 30 , 35

Modulo: 5, 4, 3, 2, 1, 0, 5

Überraschung! 5*6 %6 = 0 = 5*12 % 6

d = 11