Themengebiete „Algorithmentheorie“

- **Grundlagen**
  - Divide & Conquer Prinzip
  - Geometrisches D&C
- **Liniensegmentschnittproblem**
- **Voronoi Diagramm**
- **FFT**
- **Zufallszahlengeneratoren**
- **Randomisierter Primzahltest**
- **Verschlüsselungsverfahren**
  - Symmetrische
  - Public Key Verfahren
    - Multiplikatives Inverses
- **Listen**
  - Skiplisten
  - Randomisierte Skiplisten
- **Suchbäume**
  - Heaps
  - Treaps
  - Randomisierter Suchbaum
- **Amortisierung**
  - Aggregatmethode
  - Mankkontomethode
  - Potentialfunktionmethode
- **Dynamische Tabellen**
- **Priority Queues**
  - Listen
  - Heaps
  - Binomialqueues
  - Fibonacci Heaps
- **Union-Find Strukturen**
  - Als verkettete Listen
  - Als Baum
    - Pfadverkürzung
    - Ackermannfunktion
- **Greedy Prinzip**
- **Präfix-Codes**
  - Huffman Code
- **Dijkstras kürzeste Wege**
  - Optimalitätsprinzip
- **Minimalspannende Bäume**
  - Färbungsverfahren von Tarjan
  - Algorithmus von Kruskal
  - Algorithmus von Prim
  - Matroide
- **Bin-Packing**
  - Online Verfahren
    - Next Fit
    - First Fit
    - Best Fit
  - Offline Verfahren
    - First Fit Decreasing
    - Next Fit Decreasing
  - C-Competitivität
- **Online Algorithmen**
  - Seitenaustauschstrategien
  - Gegenspielertypen
  - **Marking**

- **Dynamische Programmierung**
  - Matrixkettenprodukt
  - Notizblockmethode
  - Suchbäume
  - Bottom-Up und Top-Down Ansatz
  - Editierdistanz
  - Sequence Alignment
- **Textsuche**
  - Knuth Morris Pratt
  - Boyer Moore
  - Datenstrukturen
    - Trie
    - Suffix-Trie
    - Suffixbäume
    - Algorithmus von McCreight
    - Suffix-Arrays
    - Präfixsuche im Suffix-Array
      - LCP
      - RLCP/LLCP
      - L_wSearch
- **Textcodierungsverfahren**
  - Huffmann Code
  - Lempel Ziv
- **Heuristische Verfahren**
- **Genetische Algorithmen**
  - Simple genetischer Algorithmus
  - Kombinatrorische Optimierungsprobleme
    - Sub-Set Sum Problem
    - Maximum Cut Problem
  - Schema und Schematheorem
  - Parameterlose Gas
    - **Verfahren von Harik Lobo**
- **Künstliche Neuronale Netze**
  - Hopfield Netz
    - Multi-Flop Problem
    - TSP
  - Selbstorganisierende Karten
    - ETSP
  - Geschichtetes Feed-Forward Netzwerk (MLP)
  - Parameterlernen
  - Gradientenbestimmung mit Backpropagation
- **Ameisenalgorithmen**
  - TSP
  - Quadratisches Zuordnungsproblem
  - Sintflutalgorithmus

Fragenkatalog Algorithmentheorie

1. **Erkläre die drei Schritte des Divide and Conquer Prinzip?** [1. Divide-Schritt: Teile das Problem in zwei oder mehr Teilprobleme, wenn N> einer konstanten c ist. Sonst löses das Problem direkt in O(1)| 2. Conquer-Schritt: Löse die Teilprobleme genauso rekursiv| 3. Merge-Schritt; Füge die Teillösungen zur Gesamtlösung zusammen]
2. **Erkläre Geometrisches D&C anhand des Closest Pair Problems?** [Divide: Punktemenge in linke rechte Seite aufteilen| Conquer: Rufe die Funktion rekursiv auf bis das Problem klein genug ist| bestimme auf jeder Seite das closest pair. Sei d = min {dl,dr} (also das kleinere Closest Pair der beiden zu mergenden Teile), dann werden nur die Punkte in dem Streifen der Breite 2d um die Schnittlinie betrachtet. Diese werden nach aufsteigenden y-Koordinaten sortiert. Anschließend werden zu einem gegebenen Punkt nur diejenigen betrachtet, deren y-Abstand kleiner als d ist. Genau diese Punktpaare sind die einzigen, die beim Merge-Schritt ein kleineres Closest-Pair erzeugen können als die der beiden Teile => Laufzeit ist $O(n \log^2 n)$ bzw $O(n \log n)$ für Mergeschritt (da sortieren nach y-Koordinaten)]
3. **Was bedeutet Liniensegmentschnitt?** [Finden aller sich schneidender Liniensegmente (in Algo.Theo. nur horizontale und vertikale Segmente, vgl. geometrische Algorithmen -> beliebige Ausrichtung)]
    a. **Laufzeit?** [Naiv $O(n^2)$, D&C $O(n \log n) + k$]
    b. **Was ist das besondere beim Zeitaufwand?** [Outputsensitiv, d.h. Laufzeit hängt von der Größe der Ausgabe ab.]
    c. **Wie funktioniert der Algorithmus?** [Naiv: Jedes mit jedem kreuzen| D&C: Horizontale Segmente durch linken und rechten Endpunkt repräsentierten => Menge lässt sich aufteilen (Divide-Schritt). Somit besteht eine Menge bei einem Teilungsschritt aus vertikalen Segmenten und einer Menge von linken und rechten Endpunkten von horizontalen Segmenten. Diese Teilung muss durchgeführt werden bis jede Menge nur noch einem der folgenden Basisfälle entspricht: 1. Fall: S enthält nur ein Element => 1.1. S enthält einen linken Endpunkt (eines horizontalen Segments), 1.2 S enthält einen rechten Endpunkt (eines horiz. Seg.) und 1.3 S enthält ein vertikales Segment. Andernfalls muss S weiter geteilt werden und zwar so, dass S in zwei gleich große Mengen S_1 und S_2 geteilt wird.| Conquer: Schnitte beider Mengen berichten| Merge: 1.Fall: In der linken Hälfte ist ein linker Endpunkt eines horizontalen Segmentes deren rechter Endpunkt nicht im rechten Teil liegt. In diesem Fall müssen alle Schnitte mit vertikalen Segmenten des rechten Teiles berichtet werden. Dazu sortiert man die vertikalen Segmente des rechten Teiles nach aufsteigenden unteren Endpunkten. 2.Fall (symmetrisch): rechter Endpunkt im rechten Teil dessen linker Endpunkt nicht im linken Teil liegt, etc.. Alle anderen Fälle wurden in vorherigen Merge-Schritten schon berichtet (z.B. linker Endpunkt in linkem Teil und rechter Endpunkt in rechtem Teil, linker und rechter Endpunkte beide in linkem Teil, etc....)]
4. **Was ist das Voronoi Diagramm?** [Eine Unterteilung der Ebene in Regionen gleicher nächster Nachbarn]
    a. Wie funktioniert hier der D&C Ansatz? [Divide: Einteilung der Punktmenge in zwei Hälften| Conquer: Rekursion| Abbruchbedingung für Divide: Das Voronoi Diagramm eines einzelnen Punktes ist die gesamte Ebene| Merge: Verbindugn zweier Teildiagramme durch einen Kantenzug (Am Anfang einfach die Mittelsenkrechte der Verbindung der obersten zwei Punkte, beim Auftreffen z.B. auf eine Kante des linken Teiles ab dort Mittelsenkrechte zwischen nächstem Punkt der linken Hälfte und des gleichen rechten Punktes, überhängende Äste abschneiden, usw.]
5. **Nenne ein typisches Anwendungsbeispiel indem die FFT verwendet wird?** [Multiplikation zweier Polynome in $O(n \log n)$ statt in $O(n^2)$ naiv]
    a. **Welche drei Darstellungen eines Polynoms haben wir besprochen?** [Koeffizientendarstellung, Nullstellenprodukt, Punkt/wert Darstellung]
    b. **Welche Vor- und Nachteile haben die verschiedenen Darstellungen?** [In der Koeffizientendarstellung kann man sehr schnell addieren in O(n)| In der Punkt-/Wertdarstellung kann man sehr schnell Multiplizieren in O(n)| In der Nullstellendarstellung kann man sehr schnell Multiplizieren| KD: schwere Multiplikation| ND: schwere Addition| PWD: Addition einfach wenn x-Koord. gleich (dann nur aufaddieren der y-Koord), ansonsten schwer]
    c. **Wie kann man das Produkt zweier Polynome effizient berechnen? (Ablauf)** [Die 2 Polynome vom Grad < n (n-1 = MAX(Grad 1, Grad 2)), d.h. mit n Koeffizienten, an 2n

beliebigen x-Stellen (aber bei beiden Polynomen an den gleichen Stellen) auswerten, man erhält dann 2n Punkt/Wert Paare➔Punktweise Multiplikation (x-Werte bleiben gleich, nur y-Werte werden multipliziert) ➔Rücktransformation in Koeffizientendarstellung (Interpolation). Man Erhält ein Polynom vom Grad 2n-2 mit 2n-1 Koeffizienten]

d. **Für was benötigt man nun die FFT?** [Sie wird benötigt, um ein Polynom in der Koeffizienten-Darstellung in die Punkt/Wert-Darstellung umzuwandeln, und zwar in der Art, dass man die Werte an den Stellen der n-ten komplexen Einheitswurzel (siehe später) berechnet, d.h. die x-Koordinaten der Punkt/Wert-Darstellung sind die Einheitswurzeln. Nur durch die Eigenschaften der Einheitswurzeln wird ein schneller D&C-Ansatz erst möglich. In unserem Falle muss für die Multiplikation von 2 Polynomen die FFT beider Polynome berechnet werden.]

e. **Was ist der Interpolationsschritt?** [Rücktransformation von Punkt-/Wert Darstellung in Koeffizientendarstellung. Dies geschieht über ein lineares Gleichungssystem der Art Va=y. Die Matrix V besteht hierbei aus Gleichungen die y_i als Lösung haben. Da man den Koeffizienten-Vektor a ermitteln will, muss man beide Seiten von links her mit der Inversen von V multiplizieren, so dass $a = V^{-1}y$. Die Inverse V^{-1} läßt sich durch besondere Eigenschaften der Einheitswurzeln sehr einfach ermitteln. Auf diese Weise lässt sich dann der Koeffizientenvektor a berechnen, der in unserem Fall die Koeffizienten des Polynomproduktes enthält.]

f. **Was ist die n-te komplexe Haupteinheitswurzel?** [Trägt man in ein Koordinatensystem, bei dem die y-Achse der imaginäre Anteil ist, und die x-Achse den reellen Anteil darstellt, einen Kreis ein, und zwar derart, dass die reelle Ausdehnung 1 ist und die imaginäre i, so ist die n-te komplexe Einheitswurzel der n-te Teil dieses Kreises, wenn man ihn in n Kuchenstücke zerteilt. Diese Einheitswurzeln lassen sich durch folgende Formel einfach berechnen: $w_n = e^{2\pi i/n}$. i = sqrt(-1). Bei z.B. n=4 sind diese: 1, i, -1, -i. Einheitswurzeln bilden eine multiplikative Gruppe, d.h. es gibt ein 1. Eins-Element (?), 2. wenn 2 Einheitswurzeln multipliziert werden, ergibt dies wieder eine Einheitswurzel der gleichen Menge (?)]

g. **Kürzungslemma, Halbierungslemma, Summationslemma ?** [Kürzungslemma: Wenn oben und unten an der Einheitswurzel der gleiche Wert steht , kann man diesen kürzen (da $w_n=e^{2*pi*i/n}$)| Halbierungslemma: Die Menge der Quadrate der 2n komplexen 2n-ten Einheitswurzeln ist gleich der Menge der n komplexen n-ten Einheitswurzeln. Beweis: $(w_{2n}{}^k)^2 = w_{2n}{}^{2k} = w_n{}^k$.| Summationslemma: Für alle n>0, j≥0 mit „n kein Teiler von j“ gilt: Summe k=0 bis n-1 von $w_n{}^{jk}$ ist 0. Wenn z.B. j=1: Alle Einheitswurzeln bis zur n-1 ten aufsummiert ergeben 0 (Beispiele: n=4, j=1: 1+i-1-i=0; n=4, j=2: 1-1+1-1; n=4, j=3: 1-i-1+i. Achtung man durchläuft die n-ten komplexen Einheitswurzeln eventuell mehrmals!) (Beweis mit geometrischer Reihe)]

h. **Unterschied FFT ⇔ DFT?** [FFT ist DFT mit D&C, d.h. FFT ist ein Algorithmus zur Berechnung der DFT mittels D&C]

i. **Wie funktioniert der D&C Ansatz bei der DFT? Algorithmus?** [Die Eingabe ist der Koeffizientenvektor eines Polynoms, von dem man die DFT (Ausgabe) berechnen will. Der D&C-Ansatz besteht darin, dass man die Koeffizienten nach geraden und ungeraden Positionen aufteilt, und man den FFT-Algorithmus für beide entstandenen Koeffizientenmengen aufruft (Rekursion). Die entstandenen beiden Ergebnisse muss man nun in einem Merge-Schritt wieder zusammenfügen. Durch Eigenschaften der Einheitswurzeln lässt sich dies zeitsparend in einer Schleife realisieren, die von 0 bis n/2-1 läuft (n = übergebene Koeffizienten dieses Rekursionsschrittes) (Funktionsweise ???).]

j. **Zeitaufwand der Einzelnen Schritte? Gesamt?** [FFT: O(n log n)| Multiiplikation O(n)| Interpolation: O(n log n)| Gesamt: O(n log n)]

6. **Welche Methoden kennen Sie zur Generierung von Zufallszahlen?** [Lineare Kongruenzmethode(Lehmer)| Verbesserung nach Schrage (korrekte überlauffreie Implementation für z.B. 32Bit-Rechner)| Gemischt kongruente Generatoren]

a. **Wie funktioniert die lineare Kongruenzmethode (Lehmer Generator)?** [Startwert Z_0, $Z_{n+1} = f(Z_n)$| f(z)=a*z mod m| a muss eine Primitivwurzel sein, damit man eine volle Periode von Zahlen erhält]

b. **Wie kann man Zufallszahlengeneratoren beurteilen?** [T1: Die Funktion f(z) soll eine komplette Periode erzeugen, also eine Sequenz in der alle Zahlen vorkommen➔a muss Primitivwurzel sein| T2: Die Zahlenfolge soll zufällig sein| T3:

Der Algorthmus muss effizient sein und in bezug auf einen 32-Bit Rechner korrekt arbeiten]

c. **Was ist das Problem beim Lehmergenerator und wie probiert Schrage dies zu lösen?** [Leider kann es Multiplikationen in f(z) geben die den Integerbereich verletzen und somit einen Überlauf produzieren. Schrage schränkt die ausführbaren Multiplikationen einfach auf diejenigen ein, die im Integerbereich liegen, in dem er einige trickreichen Umformungen der Zufallsfunktion anwendet.]

d. **Was ist der Unterschied bei den Gemischt kongruenten Generatoren zum Lehmer Generator?** [Die Zufallsgenerierende Funktion sieht etwas anders aus. Beim Lehmer-Generartor war die Funktion a*z mod m, bei Gemischt-Kongruenten Generatoren ist diese (a*z+b) mod m. Häufig wählt man für m hier 2^k. Allerdings gibt es 2 Probleme bei dieser Art von Funktionen: 1. Die Zufallszahl wechselt immer zwischen geraden und ungeraden Werten, 2. allgemein hat der Zyklus der letzten l Bits Periode 2^l.]

7. **Welche beiden Klassen von randomisierten Algorithmen gibt es?** [Las Vergas (wahrscheinlich schnell, immer korrekt)|Monte Carlo (immer schnell, wahrscheinlich korrekt)]
8. **Was ist die Besonderheit beim randomisierte Quicksort gegenüber dem Normalen?** [Zufällige Wahl des Pivotelementes ➔ Eliminiert Worst-Case bei bereits Sortierten Folgen, es kann aber trotzdem sein, auch wenn mit sehr geringer Wahrscheinlichkeit, dass der Algorithmus bei einer Eingabefolge in $O(n^2)$ läuft.]
9. **Vorteil des randomisierten Primzahltests gegenüber der deterministischen Version?** [Der deterministische Primzahltest teilt einfach jede zu testende Zahl n durch alle ungerade Zahlen die kleiner sind als die Quadratwurzel von n. ➔ Für große Zahlen ungeeignet| Der randomsierte Primzahltest ist ein Monte Carlo Algorithmus, was der Hauptvorteil ist, da solche Algorithmen besonders schnell sind.]

   a. **Auf was beruht dessen Prinzip?** [Kleiner Fermat wenn p eine Primzahl ist und 0<a<p, dann ist a^{p-1} mod p = 1]

   b. **Was sind Carmichaelzahlen? Probleme?** [Carmichael Zahlen sind Pseudoprimzahlen bei denen der kleine Fermat auch eine 1 liefert. Das passiert dann, wenn a teilerfremd mit p ist. Der Fermat ist aber trotzdem korrekt, denn er verlangt ja eine Primzahl p, aber wir wollen ja gerade das testen. Die kleinste der Carmichaelzahlen ist die Zahl 561 = 3*11*17. Wir erhalten also immer dann ein falsches Ergebnis, wenn a so gewählt wird, dass es nicht Primfaktor oder Vielfaches davon von unserer gesuchten Zahl p ist, dann erhalten wir ein falsches Ergebnis. Die Wahrscheinlichkeit ist leider sehr hoch. Wir brauchen eine Verbessung, die „Nichttrivialen Quadratwurzeln]

   c. **Was sind nichttriviale Quadratwurzeln?** [Ist p eine Primzahl und a eine Zahl zwischen 0 und p. dann hat die Gleichung a^2 mod p = 1 genau die Lösungen a = 1 und a = p-1, wenn jetzt a^2 mod p = 1 gilt und aber weder a =1 noch a =p-1 gilt und p nur irgendeine potentielle Primzahl ist, dann nennt man a eine nichttriviale Quadratwurzel mod p]

   d. Schnelle Exponentiation? [Schnelle Berechnung von a^n: Falls n gerade ist, teilen wir einfach die Exponenten auf in $a^{n/2} * a^{n/2}$. Fall n ungerade ist teilen wir auf in $a^{(n-1)/2} * a^{(n-1)/2} * a$ (somit ist eine Rekursion möglich, was heißt dass sich das Problem in jedem Schritt nahezu halbiert, denn 1. $a^{(n-1)/2}$ braucht nur einmal berechnet zu werden, 2. der Exponent hat sich halbiert) ➔ Das ganze sooft. Bis man einfach berechnen kann ➔ Laufzeit $O(\log^2 a^n \log n)$]

   e. Wie gut ist der aus allen Bedingungen zusammengefasste Algorithmus? [Der Algorithmus versagt für höchstens (n-9)/4 Zahlen (wenn n nicht prim ist), also für fast ein Viertel (d.h. wenn wir eine nicht-Primzahl haben, dann gibt es höchstens ¼ Zahlen in dem Bereich zwischen 0 und n, für die der Test sagt sie wäre prim)➔ Wenn man den Algorithmus nun 50 mal ausführt geht die Irrtumswahrscheinlichkeit gegen 0, die Wahrscheinlichkeit für einen Hardwarefehler ist bereits dann deutlich höher (Bemerkung: Wenn der Test sagt eine Zahl ist nicht prim, dann ist sie auf jeden Fall nicht prim, sagt er allerdings sie sei prim, stimmt dies nur zu einer gewissen Wahrscheinlichkeit, welche sich aber durch wiederholtes anwenden des Tests stark erhöhen lässt)]

10. **Wie funktioniert die traditionelle Verschlüsselung mit symmetrischem Schlüssel?** [Es gibt genau einen Schlüssel pro Partei mit dem man sowohl ver- als auch entschlüsseln kann.] Probleme: Schlüssel müssen irgendwann ausgetauscht werden (Sicherheitsrisiko); Man benötigt für jeden Kommunikationspartner einen neuen Schlüssel]

a. **Welche Probleme treten dabei auf?** [Probleme: Schlüssel müssen irgendwann ausgetauscht werden (Sicherheitsrisiko); Man benötigt für jeden Kommunikationspartner einen neuen Schlüssel (d.h. für Nachrichten zwischen n Parteien sind n(n-1)/2 Schlüssel erforderlich)]
b. **Was ist der entscheidende Vorteil?** [Sehr schnell, sowohl encode als auch decode]

11. **Wie funktioniert das Public Key Verfahren? Was ist RSA?** [Jeder Teilnehmer besitzt zwei Schlüssel, einen öffentlichen und einen privaten Schlüssel. Beim Versenden einer Nachricht wird diese mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Entschlüsseln kann man diese Datei aber nur mit dem privaten Schlüssel des Empfängers und diesen hat nur er selber. Es gilt S(P(m))=P(S(m))
    a. **Was ist eine digitale Signatur?** [Dabei geht es nicht um die Geheimhaltung einer Nachricht sondern um die Authentizität der Nachricht. Die Signatur kann zB ein Teil der Nachricht oder auch eine Checksumme/Hashwert (sogenannter Message-Digest) daraus sein. Diese wir beim Versand mit dem eigenen privaten Schlüssel verschlüsselt, der Empfänger kann nun mit dem öffentlichen Schlüssel des Senders die Signatur entschlüsseln und mit der selbst berechneten Checksumme/Teilstück/Hashwert des übermittelten Textes vergleichen. Die Art der Berechnung der Checksumme muss natürlich festgelegt sein.]
    b. **Was ist der private/öffentliche Schlüssel?** [Der privat Schlüssel ist nur dem Besitzer bekannt und liegt gesichert bei diesem auf dem Rechner. Der öffentliche Schlüssel (Public Key) kann entweder vom jeweiligen Partner angefordert werden oder in einem öffentlichen Keyserver abgerufen werden]
        i. **Wie berechnen sich diese?** [Man wähle zwei große Primzahlen p und q (ca. 200 Bits). n sei nun pq. Sei e nun eine kleine natürliche Zahl die teilerfremd zu (p-1)*(q-1) ist. Berechne die Multiplikative Inverse von e d*e ≡ 1 (mod (p-1)(q-1)) (d.h. d*e mod (p-1)(q-1) soll 1 sein). Öffentlicher Schlüssel: P = (e,n) Privater Schlüssel: S=(d,n)]
        ii. **Was ist die Multiplikative Inverse?** [d*e ≡ 1 (mod (p-1)(q-1)) gesucht ist die Multiplikativ inverse d von e bzgl. p und q|Die Multiplikative Inverses von 2 Zahlen a und b kann man parallel zur ggT-Berechnung durchführen, wobei man nach 2 ganzen Zahlen x und y sucht, so dass x*a+y*b = ggT(a,b). In unserem Fall muss a = (p-1)(q-1) und b=e sein. X ist dann unsere gesuchte Multiplikative Inverse]
    c. **Wie wird nun konkret verschlüsselt bzw. entschlüsselt, wenn man beide Schlüssel (öffentlich/privat) hat?** [Verschlüsselung: $P(M) = M^e \bmod n$| Entschlüsselung: $S(C) = C^d \bmod n$]

12. **Welche Arten von Listen gibt es?** [Skiplisten, einfach/doppelt verkettete Listen, Arrays, Priority Queues(Heap, Binomial-Queue., Fibonacci-Heap), Blattsuchbaum, Intervallsuchbaum, AVL-Baum, Union-Find, etc.]
    a. **Was sind (perfekte) Skiplisten?** [Eine Skipliste enthält gegenüber einer normalen sortierten Liste zusätzliche Zeiger, welche in größeren Abständen von einem Element auf ein weiter entferntes zeigen, diese Zeiger skippen also einige Elemente dazwischen, deshalb der Name. Wenn man z.B. jedes zweite Element mit einem zweiten Zeiger ausstattet (also zwei Zeigerebenen), dann verdoppelt sich die Suchgeschwindigkeit bereits. Bei perfekten Skiplisten setzt man die Niveahöhe nun auf log n und die erste Ebene ist direkt verzeigert, die 2. Ebene überspring immer einen Schlüssel, die 3. Ebene überspringt immer 4 Schlüssel, etc (die Struktur ist bei perfekten Skip-Listen genau definiert) die Suche geht dann logarithmisch schnell. Man hat ein Kopfelement am Anfang der Liste ohne Schlüssel (d.h. -∞) und ein Endelement am Ende mit Schlüssel +∞]
        i. **Wie kann man in Skiplisten suchen?** [Man fängt beim Kopfelement an im obersten Niveau und springt bis zu dem Element was gerade kleiner ist als das gesuchte. Dann geht man eine Ebene tiefer und setzt die Suche fort, wieder bis man bei einem Element ist, welche gerade noch kleiner ist usw. Irgendwann landet man bei dem gesuchten Element oder auch nicht falls es nicht vorhanden ist.]
        ii. **Welche Probleme gibt es beim Einfügen bzw. Entfernen von Elementen?** [Bei Perfekten Skip-Listen: Es geht leider O(n) Zeit für jede Einfüge oder entfernen Operation, da die gesamte Liste jedes Mal neu generiert werden muss, damit die Liste wieder perfekt ist, sonst würde sie sehr schnell degenerieren]

b. **Was sind randomisierte Skiplisten?** [Hierbei versucht man das eben genannte Problem durch Randomisierung zu eliminieren. Die Liste ist dabei nicht mehr perfekt]
c. **Wie funktioniert hier das Einfügen bzw. Entfernen?** [Beim Einfügen eines Elementes wird randomisiert eine Höhe erzeugt Nun muss man noch alle Zeiger die bisher über diese Stelle hinweggezeigt haben finden und diese mit dem richtigen Niveau des neuen Elementes verbinden]
    i. **Wie erzeugt man die Höhen beim Einfügen für ein neues Element randomisiert?** [Für jedes Niveau wird eine Münze geworfen, zeigt die Münze zB Kopf, wird ein Niveau hinzugefügt, bei Zahl stoppt die Erhöhung der Niveaus. Bei 5 mal hintereinander Kopf haben wir also Niveauhöhe 5.]
    ii. **Vorteile von den randomisierten Skiplisten?** [Jede Höhe kommt prozentual gleich viel vor wie bei perfekten Skip-Listen. Erwartete Suchkosten sind O(log n). Einfügen und entfernen in erwarteter Zeit O(log n). Erwartete Anzahl von Zeigern ist 2n bei Listenlänge n. Erwartete Niveauhöhe ist O(log n)

13. **Was sind Suchbäume?** [Es gibt z.B. natürliche Suchbäume in denen kann man nach einem gespeicherten Element in logarithmischer Zeit suchen, man spricht auch von Binärsuchbaumen. Bei diesen Bäumen hat das linke Kind einen kleineren, und das rechte Kind einen größeren Knotenwert als der Vater]
14. **Was sind Heaps?** [Ähnlich wie Suchbäume, nur dass beide Kinder einen kleineren oder gleichen Wert haben wie ihr Vater]
    a. **Was sind Treaps?** [Ein Treap ist eine Kreuzung aus Heap und Tree. Ein Knoten x hat nun 2 Komponenten, einmal den Schlüssel x.key und eine Knotenpriorität x.priority. Von oben nach unten sind die Prioritäten im Baum geordnet und von links nach rechts die X Werte.]
        i. **Welche 2 Bedingungen muss ein Treap erfüllen?** [Erstens die sogenannte Suchbaumbedingung, d.h. das linke Kind eines Vaterknotens hat einen kleineren X Wert als der Vater und das rechte Kind hat einen größeren x Wert.| Zweitens muss die Heapbedingung erfüllt sein, d.h. die Prioritäten aller Kinder müssen jeweils höher sein, als der zugehörige Vaterknoten]
        ii. **Wie werden Elemente eingefügt bzw. entfernt?** [Einfügen: Einfügen an der richtigen Stelle wie bei natürlichem Suchbaum (Stelle an der das Element nach dem Suchen nach dem Key endet) und dann durch rotation die Heapbedingung (Priority) wiederherstellen.| Entfernen: Umgekehrt wie beim Einfügen, der zu entfernende Knoten wir bis auf Blattebene hinunterrotiert (laut Priority) und dann entfernt, es dürfen natürlich nur Schlüssel mit kleinerer Priorität im Gegenzug hochrotiert werden.]
    b. **Was ist ein randomisierter Suchbaum/Treap?** [Dabei geht es um das Wählen der Priorität beim Einfügen, die wird beim randomisierten Treap zufällig gewählt. Die Prioritäten sind unabhängig und gleichverteilt zwischen [0..1]. Die erwartete Struktur des randomisierten Treaps entspricht der eines natürlichen randomisierten Suchbaumes. Das heisst die Höhe ist in o(log n). Die Kosten zum einfügen und entfernen sind auch in O(log n)]
    c. **Wie wird das mit den Schlüsseln praktisch realisiert?** [Erzeuge Dualdarstellung der den Schlüsseln zugewiesenen Prioritäten nach Bedarf und Bitweise Stück für Stück]
15. **Was ist die Idee der Amortisierung?** [Für billige Operationen mehr zahlen um das Eingesparte für teurere Operationen zu verwenden. Man verteilt damit die Gesamtkosten auf alle Operationen gleichmäßiger. Man spricht nun neben worst-, average- und best-case auch noch vom amortisierten Worst-Case]
    a. **Welche 3 verschiedenen Methoden gibt es dabei?** [Aggregatmethoe, Bankkontomethode und Potentialfunktionsmethode]
        i. **Was ist die Aggregatmethode?** [Die Aggregatmethode kann man sehr gut am Dualzähler erklären, wann immer eine eins bei einem Zähler hinzuaddiert wird, ändert sich der Zähler, allerdings ändern sich nicht immer gleich viele Bits, bei 01111 springt der Zähler z.B. auf 10000, also 4 Bitwechsel gleichzeitig, bei 1000 springt er nur auf 1001, also nur ein Bitwechsel. Jede 2. Operation kostet also nur einen Bitwechsel, wobei jede 4 Operation 2 Bitwechsel kostet usw. Summiert man alle Bitwechsel wieder auf, ergeben sie amortisiert 2n Bitwechsel, also 2 pro Operation im Schnitt. Bei der aggregatmethode werden also erst alle Kosten zusammengezählt und dann durch die Anzahl der Operation geteilt, man erhält dann die amortisierten Kosten für eine Operation]

ii. **Was ist die Bankkontomethode?** [Die Bankkontomethode ist nun irgendwie so eine Erweiterung der Aggregatmethode, man sagt sich nun beim Dualzähler, dass jede Operation 2 Bitwechsel kosten soll, hat man nun eine Operation die nur einen Bitwechsel kostet, so kommt diese gesparte auf eine Konto, und sobald wir eine Operation haben die mehr als 2 kostet, wird vom Konto wieder abgebucht]

iii. **Was ist die Potentialfunktionsmethode?** [Man gibt jedem aktuellen Zustand ein bestimmtes Potential, was beim Dualzähler z.B. die Anzahl der Einsen sein kann. Diese Funktion die dieses Potential misst wird Potentialfunktion genannt. Diese Funktion muss folgende Eigenschaften haben: 1. Sie muss beim Start-Zustand 0 liefern und 2. Sie muss für alle Nachfolgezustände größer oder gleich 0 sein. Um nun die amortisierten Kosten einer Operation zu berechnen muss man die tatsächlichen Kosten zu dem Unterschied des Potentials vor und nach der Operation addieren, also $a_i = t_i + \Phi_i - \Phi_{i-1}$. Da eine Funktion wie z.B. #Einsen schwer handhabbar ist, versucht man dies auf eine andere Art&Weise auszudrücken, wie z.B. B_{i-1} #Einsen im i-1-ten Schritt und b_i = #Endender Einsen im i-1-ten Schritt. Nun hofft man das wenn man in der obigen Formel t_i Φ_i und Φ_{i-1} durch diese neuen Funktionen ausdrückt, sich die variablen Teile herauskürzen und nur noch eine Konstante übrigbleibt. Bemerkung: Das finden einer geeigneten Potentialfunktion ist sehr schwierig!.]

b. **Was sind Dynamische Tabellen?** [Bei Tabellen haben wir immer das Problem, die richtige Größe am Anfang zu wählen, eine statische Tabelle lässt zB n Elemente zu, kommt ein weiteres Element hinzu, dann passt dieses nicht mehr hinein. Diese Problem versucht man bei dynamischen Tabellen zu beseitigen, zusätzlich muss aber noch gelten, dass immer ein konstanter Anteil der Tabelle belegt ist und das die Kosten für n Einfüge- bzw. Entfernen Operation in O(n) liegen (d.h. amortisiert konstant sind)]

i. **Welche Strategien gibt es für das Einfügen bzw. Entfernen von Elementen?** [Will man nun ein Element einfügen oder entfernen, gibt es zwei Szenarien, entweder die Tabelle ist beim Einfügen bereits voll, dann muss man diese Expandieren, oder falls noch Platz ist, kann man direkt einfügen| Entfernen kann man theoretisch immer, allerdings macht es ab einem bestimmten Belegungsfaktor keinen sinn mehr, die Tabelle unnötig groß zu halten, d.h. Sie wird wieder kontrahiert. Damit nun nach einer Expansion gefolgt von einer Entferne Operation nicht gleich wieder kontrahiert wird, sagt man z.B. Expansion heisst Verdoppelung der Tabelle, aber kontrahiert wird erst wenn der Belegungsfaktor nur noch ¼ beträgt und zwar dann um die Hälfte]

ii. **Amortisierte Analyse auf „Dynamische Tabellen?** [Wie wir gesehen haben, sind nicht alle Einfüge bzw. Entferneoperationen gleich teuer, eine gute Möglichkeit also die amortisierten Kosten einer Operation zu berechnen. Dies geschieht über die Potentialfunktionsmethode. Das finden einer geeigneten Potentialfunktion ist allerdings nicht gerade trivial. Wenn man nun die Verschiedenen Einfüge und Entfernen Fälle durchrechnet kann man sagen, dass die Amortisierten kosten pro Operation mit „3“ abgeschätzt werden können, also konstant sind. (Bemerkung: Verwendete Potentialfunktion war $\Phi(T) = 2k-s$, wobei T die Tabelle ist (Zustand von dem man das Potential wissen will), k = #Elemente in der Tabelle, s = Größe der Tabelle)]

16. **Welche Arten von Vorrangswarteschlangen (Priority Queues) kennen Sie?** [Listen (einfach- oder doppelt-verkettet), Heaps, Binomial Queues und Fibonacci Heaps]

a. **Was sind Binomialqueues? Rekursive Definition?** [Eine Binomialqueue ist die Vereinigen Heapgeordneter Binomialbäume unterschiedlicher Ordnung, wobei eine Binomialbaum B_0 ist einfach nur ein Knoten, B_{n+1} sind zwei Binomialbäume verbunden mit einer Kante, wobei die eine der beiden Wurzeln eines Binomialbaumes die neue Wurzel ist. Es entstehen also leicht nach links degenerierte Bäume]

i. **Welches Knotenformat hat ein Eintrag in einer Binomialqueue?** [Die Binomialqueue besteht ja aus lauter Binomialbäumen unterschiedlicher Ordnung, Jeder Knoten eines Baumes, einschließlich der Wurzel hat das gleiche Knotenformat, nämlich ein Zeiger auf den „Parent, was bei der Wurzel

die nachfolgende Wurzel des nächsten Binomialbaumes ist. Im „Childzeiger“ steht ein Kind gespeichert, der „Sibling Zeiger“ zeigt auf eines der Geschwister. Es gibt nun noch einen Wert „Entry“ in dem der Schlüssel des Knotens gespeichert ist und einen Eintrag für „Degree“ welcher den Grad des Baumes wiedergibt (Anzahl Kinder des Knotens, so bekommt z.B. bei einem MELD die eine Wurzel ein Kind mehr)]

Welche Operationen können darauf ausgeführt werden? [Bestimmten des Minimums (in O(1)) da dieses als separater Zeiger mitgeführt wird| Verschmelzen zweier Bäume (in O(log n))➔ Wie binäre Addition| Einfügen eines Elementes (in O(log n) geschieht über das erstellen eines Binomialbaumes mit nur einem Knoten und anschließendem verschmelzen| Deletemin geht in O(log n) Zeit und zwar indem man den gesamten Baum der am Minimum hängt herausnimmt, und dessen Söhne in umgekehrter Reihenfolge wieder mit der Binomialqueue verschmilzt| Bei der Decreasekey Operation (O(log n)) muss man nur, nach dem Herabsetzen des Wertes eines Knotens, die Heapbedingung in einem einzelnen Baum durch nach oben steigen lassen des Knotens (Rotationen) wieder herstellen]

ii. **Was passiert wenn man zwei Binomialqueues miteinander verschmilzt?** [wie Binäre Addition, d.h. das z.B. 3 B_0-Bäume einen B_0-Baum und einen B_1-Baum (entspricht Übertrag bei Addition) ergeben würden]

b. **Was sind Fibonacci Heaps?** [Ein Fibonacci Heap Q ist eine Kollektion Heapgeordneter Bäume, man bezeichnet die Fibonacci Heaps auch als Lazy Meld Version der Binomialqueues, weil Sie nämlich zwei Bäume gleicher Ordnung nicht sofort verschmelzen sondern erst bis zur nächsten Deletemin-Operation gewartet wird]

i. **Warum heißen sie so?** [Wenn v eine Wurzel eines Teilbaumes des Fibonacci Heaps ist und der Baum den Rang k hat, dann besitzt der Baum mindestens F_{k+2} Knoten, wobei F_k die k-te Fibonacci Zahl ist, d.h. hat ein Knoten k Kinder dann ist er Wurzel eines Teilbaumes mit mindestens F_{k+2} Knoten]

ii. **Wie sieht das Knotenformat eines Fibonacci Heaps aus, was besagen die einzelnen Einträge?** [Im Gegensatz zum Binomialqueue Knotenformat haben wir statt eines Siblingzeigers nun einen Zeiger auf den jeweils rechten und linken nachbarn (dadurch haben wir die Kinder in einer zirkulären doppelt-verketteten Liste abgespeichert), des weiteren haben wir ein Flag „Mark“, in welchem markiert werden kann, wenn z.B. ein Kind entfernt wurde]

iii. **Was bringt uns die Erweiterung mit linker und rechter Nachbar?** [Man kann ein Element in konstanter Zeit entfernen und man kann zwei Listen in konstanter Zeit vereinigen]

iv. **Was sind Cascading Cuts?** [Will man einen Schlüssel verringern (Decreasekey) stimmt unter Umständen die Heapbedingung nicht mehr, in diesem Fall schneidet man den Knoten vom Vater, mit dem gesamten Unterbaum, ab, hängt ihn in die Wurzelliste und setzt das „Mark“ Flag beim Vater, sollte der Vater schon markiert sein, wird auch er in die Wurzelliste geschrieben und dessen Vater markiert ➔ Casading Cuts]

v. **Warum sind Fibonacci Heaps so gut?** [Weil alle Operationen ausser Deletemin und Delete in O(1) gehen, die Enfernungen gehen in O(log n) amortisierter Worst Case]

vi. **Warum kann es nicht sein, dass alle Operationen in O(1) gehen?** [Weil man sonst in O(n) sortieren könnte. N mal insert und n mal deletemin ergeben aber n log n, was der unteren Schranke für das Sortieren entspricht]

vii. **Durch welche Schranke ist der maximale Rang eines Fibonacci Heaps beschränkt?** [$\text{rang}(Q) \leq 2 \log n$, der maximale Rang k eines Knotens v in einem Fibonacci Heap Q mit n Knoten liegt also in O(log n). Das haengt wiederum damit zusammen, dass die maximale Anzahl der Knoten in einem Teilbaum des Fib.-Heaps F_{k+2} ist]

17. **Was sind Union-Find Strukturen und warum heißen sie so?** [Eine Struktur die besonders gut für die Operationen Union und Findset (in welchem Set ein Element liegt) ist]

a. **Was ist ein typisches Beispiel indem Union-Find Strukturen verwendet werden können?** [Zusammenhangstest, sprich herausfinden ob zwei Elemente in der gleichen Zusammenhangskomponente liegen ➔ Das geht indem man für jeden Knoten zunächst ein „Set“ definiert, findet man zwei Knoten, die über eine Kante

verbunden sind, aber noch nicht im gleichen set sind, werden diese beiden Sets vereinigt. Wenn man nun das Ergebnis hat kann man den Zusammenhangstest leicht erreichen, in dem man 2 find-set-Operation macht, und die Ergebnisse vergleicht]

b. **Wie sieht die Struktur als verkettete Liste aus?** [Man verwendet dazu eine verkette Liste ➔ Jedes Element zeigt auf seinen Nachfolger und auf die Wurzel. Die Vereinigung kann dabei ganz einfach dadurch geschehen, dass die Listen aneinander gehängt werden, das einzige was jetzt im Prinzip Zeit kostet, ist die Wurzelknoten umzuhängen, dies kann man auf jeden Fall minimal halten indem man immer die kürzere an die Längere Liste hängt, dazu muss allerdings noch die Listenlänge als Parameter mitgeführt werden]
   i. **Welche Aufgabe hat der „Repräsentant“?** [Der Repräsentant eines Sets ist immer die Wurzel, haben zwei knoten die gleiche Wurzel, sind sie im gleichen set]
   ii. **Welche Strategien gibt es beim Vereinigen?** [1. Der Repräsentant der einen Menge wird der direkte Vater des Reräsentanten der anderen Menge ➔ Diese Variante hat aber einen sehr schlechten Worst-Case, indem jeweils ein Element vereinigt wird, also wenn immer die längere Liste an die kürzere gehängt wird. Hierbei muss man immer alle Zeiger der längeren Liste auf den neuen Repräsentanten ändern| 2, Immer den kleineren an den größeren hängen (genannt: Vereinigung nach Größe)]

c. **Wie werden Union Find Strukturen als Baum repräsentiert?** [Jedes Set wird in einem Baum gespeichert, Die ganze Struktur ist dann ein Wald von Bäumen]
   i. **Welchen Vorteil bringt die Pfadverkürzung?** [Bei der Findsert Operation, werden jeweils alle Knoten in einem Baum direkt an die Wurzel gehängt. Vereinigt man nun nach Größe und mit Pfadverkürzung benötigt man dazu Θ (m*a(m,n)) Schritte]
   ii. **Was hat die Ackermann Funktion mit der Laufzeit für die Strategie „Vereinigung nach Größe und Pfadverkürzung“ zu tun?** [In der Laufzeit steckt der Wert a(m,n) drin, das ist die Inverse der sogenannten Ackermann Funktion]

18. **Was ist das Greedy-Prinzip?** [Das Gierige Prinzip ☺ ➔ Treffe in jedem Verfahrensschritt diejenige Entscheidung die im Moment am besten ist (ohne in die Zukunft zu schauen)]
   a. **Welche 3 Ergebnistypen kann es geben?** [Optimale Lösung| Lösung sehr nah am Optimum| beliebig schlechte Lösung]
   b. **Nenne 2 Beispiele bei denen Greedy beliebig schlecht abschneidet?** [Land Absurdia: Hier gibt es Münzen in sehr ungünstiger stückelung ➔ 41 Euro Münze, 20 Euro und 1 Euro, will man nun einen Betrag von 60 Euro bezahlen, wählt das Greedy Prinzip automatisch als erste Münze die 41 Euro Münze, das diese dem Ziel in diesem Schritt am Nächsten kommt, leider muss Greedy dann 19 mal ein Euro nehmen, was sehr schlecht ist, besser wäre natürlich 3 x 20 Euro| Auch beim TSP Problem kann Greedy beliebig schlecht abschneiden ➔ Greedy kann sich hier sehr schnell verrennen, nämlich dann wenn es einen Pfad gibt der für eine optimale Lösung erfordert, dass man einen etwas größeren Weg geht um danach eine Abkürzung abzulaufen. Greedy findet diese Abkürzung aber nicht, da es stets den kürzesten Weg verfolgt.]
   c. **Nennen Sie zwei Beispiele bei denen Greedy stets die optimale Lösung liefert?** [Aktivitätenauswahlproblem: Man hat eine Menge von Aktivitäten die eine bestimmte Ressource benötigen, z.B. einen Hörsaal ➔ Zwei Aktivitäten sind dann Kompatibel, wenn Sie sich bezüglich ihrer Ressource nicht überschneiden ➔ Die Idee ist es nun die nicht verwendete Zeit zu maximieren, dadurch bekommt man möglichst viele Aktivitäten in das Zeitfenster➔ Sortieren der Aktivitäten nach aufsteigendem Endzeitpunkt: Nun durchläuft man die sortierte Liste und wählt die nächste legale Aktivität (Startzeitpunkt > Endzeitpunkt der letzten Aktivität) ➔ Die erste Aktivität ist also diejenige mit dem frühesten Endzeitpunkt]
   d. **Wann ist ein Greedy Verfahren optimal?** [Wenn man eine optimale Teillösung hat und man trifft eine lokale optimale Wahl, dann gibt es eine globale optimale Lösung die diese Wahl enthält (Optimalitätsprinzip) ➔ Eine Teillösung einer optimalen Lösung ist eine optimale Lösung des Teilproblems]
   e. **Was ist der Huffman Code?** [Beim Huffman Code geht es um die Codierung von Texten um diese zu komprimieren. Dabei wird der Text zunächst analysiert und geschaut, welches Zeichen wie häufig vorkommt, man erhält also eine Liste von Zeichen mit deren Häufigkeiten. Daraus generieren wir einen binären Code-Baum. In

den Blättern stehen die zu kodierenden Zeichen. Ein Pfad zu einem Blatt repräsentiert die Kodierung des Zeichens (an den Kanten steht 0 oder 1). Der Baum wird so aufgebaut, das seltene Zeichen eine längere Kodierung (Bit-Folgen) haben als häufige Zeichen.]

i. **Was unterscheidet den Huffman Code z.B. vom ASCII Code?** [Beim ASCII Code gibt es keinerlei Gewichtung der Zeichen, jedes benötigt 8 Bit zur Speicherung egal wie häufig es vorkommt]
ii. **Was ist ein Präfix Code?** [Ein Präfixcode ist ein Code Variabler Länge ganz im Gegenteil z.B. zum ASCII Code welcher ein Code fester Länge ist, es darf nun aber nicht passieren, dass eine Codewort selbst Präfix eines anderen Codes ist. Bei Telefonnummern ist das letztendlich auch so, es kann keiner Nummer 1234 geben und dann noch eine 12345 unter der gleichen Vorwahl. Der Huffman-Code ist z.B. ein Präfix-Code.]
    1. **Was ist dabei beim Dekodieren zu beachten?** [Man muss immer am Anfang anfangen.]
iii. **Welche Eigenschaften hat der Codebaum eines optimalen Präfixcodes?** [Wenn die Menge A n Elemente hat, dann besteht der Baum genau aus n Blättern und n-1 Knoten| Einen Code Baum nennt man übrigens auch „Trie“ von Retrieval]

f. **Wie wird der Codebaum beim Huffman Code aufgebaut?** [Zunächst werden die Zeichenhäufigkeiten ermittelt. Anschliessend werden die Zeichen in einer Priority-Queue gespeichert bei der die Priorität der Zeichenhäufigkeit entspricht (alle Zeichen zunächst in der Wurzelliste). Im nächsten Schritt werden aus dieser Queue die Zeichen (bzw. Bäume) mit minimaler Häufigkeit entfernt (2 * delete-min), an einen neuen Knoten angehängt (dieser Knoten erhält als Priorität die Summe der Prioritäten der an ihn gehängten Knoten), und wieder in die Queue eingefügt. Wiederhole diesen Schritt n-1 –Mal. Nach Ausführung haben wir einen optimalen Codebaum für die gegebenen Zeichenhäufigkeiten]
    i. **Was ist das Problem beim Dekodieren, was wird benötigt?** [Der Codebaum muss übermittelt werden, im Gegensatz zum Verfahren von Lempel-Ziv (siehe viel weiter unten unter Textcodierungsverfahren)]

19. **Wie funktioniert der Algorithmus von Dijkstra?** [Man redet auch von Dijkstras kürzeste Wege. Man möchte den kürzesten Weg in einem Distanzgraphen von einem Knoten v aus V zu jedem anderen Knoten s aus V finden. Stichwort: OSPF (Rechnernetze). Das Ganze läuft nun so, man nimmt denjenigen Knoten zu dem man die Distanzen finden will und schreibt ihn in eine Liste. Nun betrachtet man sämtliche Nachbarn des Knoten und sucht sich denjenigen mit der Kürzesten Distanz zu v aus. Dieser neue Knoten wird ebenfalls in die Knotenliste geschrieben, nun werden wieder alle Nachbarn der knoten aus der Knotenliste betrachtet und wieder der Nachbarknoten gewählt, welcher die kürzeste Distanz zu v hat, dies kann einer der Nachbarn sein, die vorher schon betrachtet wurden oder auch einer der neuen Nachbarn die über den zweiten Punkt hinweg den kürzesten Weg haben. In der Knotenliste wird also jeder Knoten mit seiner kürzesten Distanz zu v gespeichert]
    a. **Was ist hier das Optimalitätsprinzip?** [Hat man einen kürzesten Weg von v zu einem anderen Knoten, dann ist jede Teilstrecke auf diesem Weg bzgl. der passierten Knoten auch optimal]
    b. **Welche 2 Darstellungsformen eines Graphen haben wir hierbei behandelt?** [Adjazenzliste (zu jedem Knoten werden in einer Liste seine Nachbarn gespeichert) und Adjazenzmatrix (Abstände der Knoten stehen in einer symmetrischen n x n Matrix, wenn n die Anzahl der Knoten ist)]
    c. **Es gibt dabei zwei Verschiedene Implementierungen, wie unterscheiden sich diese?** [Bei der ersten Version werden die Randknoten nicht explizit gespeichert, sondern es gibt nur ein 4 Tupel indem steht (Knotennr, Entfernung zu v, Vorgänger, gewählt) ➔ Die Laufzeit beträgt $O=(n^2)$ und ist sehr gut für Graphen mit vielen Kanten geeignet, hat man hingegen einen Graphen mit sehr wenigen Kanten, nimmt man besser einen Fibonacci Heap, in welchem die Randknoten gespeichert werden. Die Laufzeit beträgt dann nur noch $O(|E| + |V| \log |V|)$, sehr gut bei kleiner Kantenzahl]
20. **Was ist ein Minimalspannender Baum?** [Ein MST ist derjenige Baum der alle Knoten eines Graphen kreisfrei verbindet und dabei minimales Gesamtgewicht hat]
    a. **Wie funktioniert das Färbungsverfahren von Tarjan?** [Es gibt eine rote und eine grüne Regel, diese Regeln färben den Graphen nach und nach ein, eine grüne Kante ist Teil des MST, eine rote Kante ist „weggestrichen“.]

i. **Welche beiden Regeln hat dieses? Wie lauten sie?** [Grüne Regel: Finden einen beliebigen Schnitt des Graphen. Wenn dieser noch keine grüne Kante und noch mindestens eine ungefärbte Kante hat, dann färbe die noch ungefärbte Kante minimalen Gewichts grün| Rote Regel: Betrachte Kreise im Graphen, der noch keine rote und mindestens eine ungefärbte Kante enthält und färbe die Kante mit maximalem Gewicht rot]
    1. **Was ist nun ein Schnitt?** [Ein Schnitt teilt einen Graphen in zwei Teile. Wenn G = (V,E) dann ist der Schnitt eine Partition (V,V\S). Eine Kante kreuzt den Schnitt, wenn der eine Endpunkt in der einen, und der andere in der anderen Menge liegt. Der Begriff Schnitt wird für die grüne Regel gebraucht.]
    2. **Was ist nun ein Kreis?** [Eine Folge von knoten, bei welcher jeweils zwei Knoten durch eine Kante miteinander verbunden sind und außer dem Anfangs-/ und dem Endknoten jeder Knoten nur einmal vorkommt. Dieser Begriff wird für die rote Regel benötigt.]
ii. **Wann terminiert der Algorithmus?** [wenn keine der Regeln mehr anwendbar ist, verbessern kann man dies in dem man den Algorithmus Terminiert, wenn die Grüne Regel bzw. rote Regel nicht mehr anwendbar ist, denn dann hat man ja bereits alle Kanten des MST gefunden (grüne Regel => grüne Kanten sind die Lösung, rote Regel => nicht rot gefärbte Kanten sind die Lösung)]

b. **Wie funktioniert der Algorithmus von Kruskal?** [Alle Kanten eines Graphen werden nach aufsteigendem Gewicht sortiert. Man nehme nun die Kanten der Reihe nach (aufsteigend) und teste ob beide Endknoten einer Kante in demselben grünen Baum liegen, falls ja wird sie rot gefärbt, denn es würde ein Zyklus erzeugt, falls nein, kann sie grün gefärbt werden. Insbesondere ist diejenige Kante mit dem kleinsten Gewicht grün gefärbt, da Sie ganz am Anfang eingefärbt wird.]
    i. **Wie unterscheidet er sich vom Verfahren von Tarjan?** [Eigentlich nur durch die Vorsortierung der Kanten, die Regeln sind im Prinzip isomorph. Er endet auf jeden Fall erst, wenn alle Kanten durchlaufen sind]
    ii. **Wie kann man den Algorithmus von Kruskal in einer Union-Find Struktur implementieren?** [Nach der Sortierung der Kanten, kann man die Grünen Bäume so behandeln: FIND: finde die beiden Bäume in denen die beiden Endknoten der zu färbenden Kante liegen (Test ob sie im gleichen grünen Baum liegen)| UNION: Vereinige die beiden gefundenen Bäume]
c. **Wie funktioniert der Algorithmus von Prim?** [Man nimmt einen beliebigen Startknoten und betrachtet ihn als Baum (soweit ist der Algorithmus auf jeden Fall korrekt, da die Knoten ja alle im MST liegen) ➔ Nun wird jeweils eine Kante hinzugenommen die einen Endknoten im grünen Baum hat, keinen Zyklus erzeugt und minimal ist]
    i. **Wie unterscheidet er sich vom Verfahren von Tarjan?** [Es gibt keine rote Regel, es ist vielmehr der klassische MST Greedy Algorithmus aus dem Grundstudium (D.A.S.)]
d. **Was ist ein Matroid?** [Matroide sind Mengensysteme die gleichzeitig Unabhängigkeitssysteme sind und die eine gewisse Austauscheigenschaft haben. Das Interessante ist nun, dass gerade die Greedy Algorithmen immer eine optimale Lösung auf einem Mengensystem liefert, wenn dieses ein Matroid ist. Siehe hierzu auch die genaue Definition eines Unabhängigkeitssystems und eines Unabhängikeitssystemes mit Austauscheigenschaft]

21. **Was ist Bin Packing?** [Wir haben n Objekte der Größen 0 bis 1 und m Kisten der Größe 1, Ziel ist es nun alle n Objekte in eine möglichst geringe Anzahl m von Kisten zu verpacken]
    a. **Was ist der Unterschied zwischen online- und offline Bin-Packing verfahren?** [Bei einem Online Verfahren kommen die Kisten Online vom Förderband und müssen sukzessive Verpackt werden – Beim Offline Verfahren sind die Größen aller Objekte bekannt und es kann optimal verpackt werden]
    b. **Welche verschiedenen online-Verfahren gibt es?** [Next Fit, First Fit, Best Fit]
        i. **Erkläre Next Fit!** [Das aktuelle Objekt wird in die gerade noch offene Kiste gepackt, wenn es dort noch hineinpasst, ansonsten wird die Kiste verschlossen und eine neue, leere geöffnet um das Objekt zu verpacken]
        ii. **Erkläre First Fit!** [Packe das aktuelle Objekt in die erste Kiste in die es noch passt, falls das nicht geht, wird eine neue Kiste geöffnet. Die Kisten werden erst verschlossen, wenn Sie voll sind oder es keine Objekte mehr gibt.

Desweiteren kann man feststellen, dass es nie mehr als eine Kiste gibt, die weniger als Halbvoll ist, da man diese ja sonst ineinanderkippen könnte]

iii. **Erkläre Best Fit!** [Dieses Verfahren verpackt das aktuelle Paket nicht in die erstbeste Kiste mit ausreichend Platz, sondern in diejenige die danach am vollsten ist, sprich den kleinsten Freiraum hat]

c. **Wie stehen die Verfahren im Verhältnis zur optimalen Lösung?** [Next Fit braucht höchstens doppelt so viele Kisten, wie der optimale Offline Algorithmus| First Fit ist benötigt höchstens 1,7 mal so viele Kisten wie der optimale Algorithmus| Best Fit ähnlich zu First Fit]

d. **Wie sind die Laufzeiten? Wie schlecht können diese werden?** [Next-Fit ist linear schnell, da ja keine Suchoperation enthalten ist (jedes Objekt kann in O(1) verpackt werden)| First-Fit und Best Fit laufen Aufgrund dessen in O(n log n) (jedes Objekt benötigt O(log n))]

e. **Gibt es eine untere Schranke für die Anzahl der Kisten die ein Online Algorithmus gegenüber dem optimalen Offline-Verfahren benötigt?** [Ja und zwar bei 1,54 mal so vielen Kisten. Der beste bisher gefundene Online-Algorithmus liegt sogar nur bei knapp unter 1,59 mal so vielen Kisten]

f. **Welche verschiedenen Offline-Verfahren gibt es?** [Vollständiges durchsuchen des Problemraumes ist viel zu aufwendig, daher gibt es zwei sehr gute Verfahren: First Fit Decreasing und Best Fit Decreasing]

i. **Wie funktioniert First Fit Decreasing?** [Bei beiden Verfahren werden die Kisten zunächst absteigend nach Größe sortiert (Decreasing Schritt) Dann wird der Reihe nach verpackt nach dem First Fit Prinzip]

ii. **Wie funktioniert Best Fit Decreasing?** [Genau wie First Fit decreasing nur, dass halt nicht die erstbeste Kiste genommen wird sondern wieder diejenige die durch das Füllen am vollsten ist (Best Fit Prinzip)]

iii. **Wie stark weichen diese Verfahren vom Optimum ab?** [1,22 mal so viele Kisten + 4 Kisten wie der optimale Algorithmus]

g. **Wie kann man denn generell die obig erwähnten Abweichungen eines Online-Algorithmuses zu einem optimalen Offline-Algorithmus berechnen?** [Man wählt Objekte, die optimal verpackt werden können (alle Kisten randvoll). Die Objekte sind aber immer um ein epsilon (beliebig kleine Zahl) größer oder kleiner. Z.B. 1 Objekt der Größe ½ -2e, und 2 Objekte der Größe ¼+e. Diese können nun optimal in einer Kiste verpackt werden. Hat man nun z.B. 2 Objekte der 1. Größe und 4 der 2. Größe, dann würden diese z.B. bei First-Fit nicht optimal verpackt werden, da die ersten 2 Objekte beide in die 1. Kiste verpackt würden, und ein Freiraum von 4e erzeugen würde.]

22. **Was ist ein Online Algorithmus?** [Ein Verfahren für Probleme, bei denen nicht alle Informationen a priori Verfügbar sind]
23. **Wann ist ein Algorithmus c Competitiv?** [Wenn der zum Offline Algorithmus gehörige Online-Algorithmus höchstens um einen Faktor c (und einer konstanten Addition) schlechter ist.

a. **Online Algorithmen werden auch bei Caches/Speicher verwendet. Was für (Speicher)-seitenaustauschstrategien gibt es und wie funktionieren diese?** [LIFO (Last in First Out, im Prinzip ein Stack), FIFO (First in First out => Schlange), LRU (mit Timestamp => Am längsten nicht gebrauchtes Element (durch Angabe des Zeitpunktes) wird als nächstes entfernt), LFU (mit Counter => Am wenigsten gebrauchtes Element (kleinster Zugriffs-Zähler) wird als nächstes entfernt), FWF (Zugegriffene Seiten werden im Cache markiert, ist Speicher voll dann wird eine nicht markierte entfernt (vom Anfang her die nächste unmarkierte Seite, kann aber auch randomisiert (Marking) sein). Gibt es keine unmarkierte, dann werden alle Markierungen gelöscht (Flush-When-Full), und eine von diesen ersetzt)]

b. **Welche untere Schranke gibt es für einen Online-Seitenaustauschalgorithmus A?** [Er ist c competitiv und damit nur über einen Faktor vom optimalen abhängig]

c. **Welche 3 Gegenspielertypen gibt es?** [Man stelle sich einen Gegner vor, der einem die Bausteine beim Spiel Tetris unterschiebt, dabei gibt es denjenigen der Online gar nichts entscheidet und die Reihefolge schon von Anfang an festgelegt hat. Der adaptive online Gegenspieler greift nun aktiv ein und reagiert direkt und aktiv auf die Handlungen des Spielers. Der adaptive Offline Gegenspieler kennt die Spielzüge des Spielers von vorneherein und berechnet ein optimales Gegenspiel ➔ Vergleichbar mit dem ausspionieren eines Schlachtplanes auf welchem man entgegen planen kann und gut reagieren kann.]

d. **Wie funktioniert Marking?** [Entspricht dem verfahren Flush when Full, nur in randomisierter Form]
e. **Welche anderen Online-Algorithmen kennen Sie?** [Online Bin Packing, Load Balancing, Geometrische Online-Suche etc.]

24. **Was ist dynamische Programmierung?** [Man löst ein Problem indem man mehrere Teilprobleme berechnet aus denen sich das Gesamtproblem löst. Die Teillösungen werden abgespeichert, dadurch müssen ggf. notwendige Mehrfachberechnungen nicht mehr erfolgen]
    a. **Erkläre diese anhand der Fibonacci-Zahlen?** [Eines der besten Beispiele für dynamische Programmierung. Möchte man die n-te Fibonacci Zahl in der rekursiven Definition berechnen, werden im rekursionsbaum viele Berechnungen doppelt ausgeführt, da die n+1-te Fibonacci Zahl sich ja immer aus ihren 2 Vorgängern addiert. Man speichert beim dynamischen Programmieren nun sukzessive die Fibonacci Zahlen ab und muss beim Berechnen einer neuen Zahl nur die beiden Vorgänger aus der Tabelle holen, ohne diese neu zu berechnen]
    b. **Wie ist die Vorgehensweise beim dynamischen Programmieren?** [1. Das Problem muss rekursiv beschrieben werden|2. Bestimmen aller Teilprobleme T des Problems P| 3. Bestimmung einer Reihenfolge der Teilprobleme sodass jedes neu gelöste (Teil-)Problem durch die bereits vorhandenen Teillösungen berechnet werden kann| 4. Sukzessive Berechnung und Speicherung der Teilprobleme]
    c. **Erkläre dynamische Programmierung anhand des Kettenprodukts von Matrizen?** [Beim Lösen einer Kette von Matrizenmultiplikationen muss man nicht streng der Reihe nach vorgehen, das Problem ist zwar nicht kommutativ, aber assoziativ ➔ Das Ziel des Matrixkettenproduktes ist es nun eine möglichst gute Klammerung zu finden, bei der die Anzahl der skalaren Multiplikationen minimal bleibt. Durch Ausprobieren würde man $O(4^n)$ Schritte benötigen ➔ Dynamische Programmierung➔ Zunächst müssen wir das Problem rekursiv definieren ➔ Im Bottom-Up Ansatz]
    d. **Was ist das Prinzip der Notizblockmethode (Top-Down-Ansatz)?** [Speichern bereits berechneter Teilprobleme auf einem Notizblock. D.h. ein Teilproblem wird nur beim ersten Auftreten gelöst.]
    e. **Wie kann man optimale Suchbäume mit Hilfe dynamischer Programmierung konstruieren?** [Ein optimaler Suchbaum ist ein Suchbaum mit minimal möglicher gewichteter Pfadlänge. Das Prinzip ist ähnlich zum Matrixkettenprodukt, d.h. es werden optimale Splitpunkte gesucht, in dem man eine Rekursionsgleichung angibt. Der erste Splitpunkt entscheidet welcher Wert an der Wurzel steht, denn der linke Teilbaum enthält Knoten die kleiner als die Wurzel sind, der rechte Teilbaum alle die größer sind. (genaueres Vorgehen: keine Ahnung ☺]
    f. **Erkläre Vor und Nachteile der beiden Ansätze – Bottom-Up und Top-Down der dynamischen Programmiereung?** [Bottom-Up: Vorteile: kontrollierte effiziente Tabellenverwaltung, spart zeit; spezielle optimierte Berechnungsreihenfolge; Nachteile: weitgehende Umkodierung des Originalprogramms erforderlich; möglicherweise Berechnung nicht benötigter Werte; Top-Down: Vorteile: Originalprogramm wird nur gering oder nicht verändert; Nur tatsächlich benötigte Werte werden berechnet; Nachteile: separate Tabellenverwaltung benötigt Zeit; Tabellengröße oft nicht optimal]
    g. **Was versteht man unter der Editierdistanz?** [Die Editierdistanz ist die minimale Anzahl von Editieroperationen die eine Zeichenkette A in eine Zeichenkette B überführt| Es gibt die möglichen Editieroperationen Ersetzen eines Zeichens von A durch ein Zeichen von B ➔ Löschen eines Zeichens von A ➔ Einfügen eines Zeichens von B]
    h. **Wie wird sie berechnet?** [Das Problem wird vom Ende her angegangen, d.h. Fallunterscheidung der letzten Operation (Ersetzen, Löschen, Einfügen). Achtung: von Ersetzen redet man auch dann wenn beide Zeichen gleich sind.]
    i. **Wie sieht der rekursive Ansatz aus?** [folgende Anfangsbedingungen definieren: Editierdistanz 2 leerer Wörter ist 0 ($D_{0,0}$), Editierdistanz eines leeren Wortes mit einem der Länge j ist j ($D_{0,j}$), Editierdistanz eines Wortes der Länge i mit einem leeren Wort ist i ($D_{i,0}$); Jetzt muss man sich noch die Rekursionsgleichung definieren. Wie oben schon erwähnt beginnt man am Ende => die Kosten sind für: Ersetzen: $D_{i-1,j-1} + c(a_i,b_j)$ (Die c-Funktion ist Null, wenn $a_i=b_j$); Einfügen im ersten Wort: $D_{i-1,j} + 1$; Löschen im ersten Wort (bzw. Einfügen im zweiten Wort): $D_{i,j-1} + 1$; Somit ist $D_{i,j}$ das Minium der oben genannten Kosten für die 3 Operationen]

j. **Wo ist hierbei das Konzept der dynamischen Programmierung?** [Das Konzept steckt in der Matrix $D_{i,j}$ die oben nach und nach gefüllt wird. Das linke obere Element eines 2x2-Teilelementes der Matrix berechnet sich über das Minimum der anderen 3 Felder. Da die linkeste Spalte und die oberste Zeile durch die Anfangsbedingungen vorgegeben worden sind, kann jede Zeile immer von links her beginnend mit Werten gefüllt werden. Da hier auf jedes Element der Matrix doppelt zugegriffen wird, ist die dynamsiche Programmierung sinnvoll.]

k. **Wie funktioniert das Sequence alignment?** [z.B. finden zweier DNA Sequenzen die möglichst ähnlich sind, indem man Leerzeichen (bzw. Indels) einfügt. Hierbei wird ein Match (Übereinstimmung) mit +1 belohnt, wobei ein Missmatch mit –1 und ein Leerzeichen mit –2 bestraft wird. Match und Mismatch (Ersetzen) drückt man in einer Funktion s(a,b) aus, bei der a und b die Zeichen der zu vergleichenden Wörter repräsentieren. Löschen eines Zeichens im 1. Wort, bzw. Einfügen eines Zeichens im 2. Wort bedeutet einen Verlust um -c]

   i. **Rekursion?** [Der Algorithmus funktioniert ähnlich wie der der Editierdistanz. Man definiert Anfangsbedingungen und eine Rekursionsgleichung. Anfangsbedingungen: 2 leere Zeichenketten $S_{0,0}$=0, die erste leer die andere Länge j: $S_{0,j}$ = -jc, die erste Länge i die andere leer $S_{i,0}$ = -ic. Operationen: Ersetze-Operation: $S_{m-1,n-1} + s(a_m,b_n)$, Löschen in 1. Zeichenkette: $S_{m-1,n}$-c; Einfügen in 2. Zeichenkette: $S_{m,n-1}$-c; Um die Editieroperation die zur größten Ähnlichkeit führen zu finden, muss das Maximum dieser 3 Fälle berechnet werden, da wir ja mit Belohnung (positiv) und Bestrafung (negativ) rechnen. Nun haben wir wie bei der Editierdistanz wieder eine Matrix die durch S definiert ist. Anfangsbedingungen=1.Zeile und 1. Spalte, Auffüllen durch Rekursionsgleichung (2x2-Matrix betrachten). Um nun das Ergebnis aus dieser Matrix auslesen zu können, müssen wir eine erneute Matrix (Anfangsbed,Rek-Gl.) aufbauen, welche aus S heraus aufgestellt wird. Die Ausgabe ist dann eine optimale Spur mit Leerzeichen in beiden Zeichenketten.]

25. **Welche zwei Szenarios gibt es bei der Textsuche?** [Es gibt dynamische und statische Texte. Die Dynamischen sind z.B. Texteditoren. Statisches Texte sind fast alle anderen, wie z.B. Literaturdatenbank, Bibliotheksysteme➔ Man möchte nun ein bestimmtes Muster in dem Text wieder finden. Naiv geht dies, indem man das Muster einfach der Reihe nach an den Text anlegt und jeweils im einen Buchstaben inkrementiert ➔ O(m n)➔ Ziel: Größere Sprünge. Bei dynamsichen Text *kompiliert* man das Muster, und bei statischen Texten *kompiliert* man den Text.]

    a. **Erkläre das Verfahren von Knuth-Morris-Pratt?** [Wir versuchen nun also nicht Zeichen für Zeichen weiterzugehen wie beim naiven Verfahren, sondern größere Sprünge. Die Idee, die dahinter Steckt, ist es ein sogenanntes Next-Array Netx[j] zu berechnen, welches die Länge des längsten Anfangsstückes von P speichert, welches echtes Suffix von $P_{1..j}$ ist➔ Zu deutsch, wir brauchen das Gleiche nicht noch einmal vergleichen und können um diese Strecke springen (Muster wird an sich selber angelegt, und die weitest mögliche Sprungweite berechnet -> Next-Array)➔ Die Laufzeit des Verfahrens KMP ist nur O(m+n), d.h. jedes Zeichen wird einmal betrachtet, aber es geht noch schneller]

    b. **Erkläre das Verfahren von Boyer-Moore?** [Idee: Die Muster werden von links nach rechts angelegt, aber zeichenweise von rechts nach links verglichen. Wir haben nun eine sogenannte Vorkommensheuristik, die es uns ermöglicht Sprünge der Größe n zu machen, wann immer wir von rechts her im Text ein Zeichen haben, welches im Muster nicht vorkommt, können wir die gesamte Musterlänge weiterschieben. Kommt ein rechts angelegter Buchstaben vor, können wir nur bis zu der Stelle verschieben an der das Zeichen im Muster steht➔ Die Laufzeit wird sehr gut und zwar O(m+⌊n/m⌋)➔ Es gibt aber eine Worst-Case Folge und zwar ein Text aus lauter „0“ und ein Muster, welches aus einer 1 und nur „0“ besteht, man kann dann in jedem Schritt nur eines weiter gehen => Laufzeit: O(n*m)]

       i. **Der eben genannte Fall des Worst Case kann aber verbessert werden, wie?** [Wir analysieren unser Muster selber und suchen diejenige Position in unserem Muster an der das von rechts her nächste Vorkommen des Suffixes $P_{j+1...m}$ endet, dem nicht das Zeichen P_j vorangeht ➔ Beispiel: Banana wrw[3] ist z.B. 4 ➔ Der Worst Case Fall ist jetzt O(m+n/m)]

26. **Welche Eigenschaften sollte eine Datenstruktur zum Suchen in Texten haben? (5 Stück)** [Teilwortsuche in Zeit O(|a|), Anfragen an den Text, wie z.B. Längstes Teilwort, das

mindestens zweimal vorkommt || Präfixsuche ➔ Alle Stellen im Text mit einem bestimmten Präfix || Bereichssuche ➔ z.B. alle Wörter zwischen [abc,acc] || Speicherplatzbedarf und Konstruktionszeit sollten in linearer Komplexität sein]

a. **Was ist ein Trie?** [Ein Baum zur Repräsentation von Schlüsseln ➔ Trie kommt von ReTrieval, Hier (Zeichenkette): An den Kanten stehen einzelne Buchstaben des Alphabets, benachbarte Kanten verschiedene Zeichen, Blatt repräsentiert einen Schlüssel => Ein Kantenzug von der Wurzel zum Blatt ist ein Schlüssel (Kantenbeschriftungen ablesen), d.h. die Schlüssel stehen nicht in den Knoten!]
b. **Was ist ein Suffix-Trie**? [Ein Trie für alle Suffixe eines Wortes]
    i. **Welche der o.g. Eigenschaften unterstütz der Suffix-Trie?** [Die Zeichenkettensuche, das finden des längsten doppelt vorkommenden Wortes und die Präfix Suche]
c. **Was ist der Unterschied zwischen einem Suffixbaum und einem Suffixtrie?** [Der einzige Unterschied ist, dass unäre Knoten zusammengefasst werden (kontrahiert). Suffix-Baum ist also ein kontrahierter Suffix-Trie.]
d. **Wie werden die Suffix-Bäume intern repräsentiert?** [Jeder Knoten ist ein 4 Tupel, im ersten Eintrag steht die Position der Zeichenkette im Wort, der zweite Eintrag enthält die Position des Endes der Zeichenkette ($ steht hierbei für das Ende). Im dritten Eintrag steht ein Zeiger auf das erste Kind und im 4. Eintrag ein Zeiger auf ein Geschwister]
e. **Was bedeutet „$“?** [Teilzeichenkette enthält alles ab der aktuellen Position bis zum Schluss (Endemarkierung des Textes). Dieses Zeichen darf nicht Teil des Alphabetes des Wortes sein!]
f. **Welche Eigenschaften haben Suffix-Bäume?** [Kein Suffix ist Präfix eines anderen Suffixes ➔ Dafür brauchen wir das Abschlusszeichen „$“ || Eine Kante ist ein nichtleeres Teilwort des Wortes || Benachtbarte Kanten beginnen(!) mit verschiedenen Zeichen || Jeder innere Knoten hat mindestens zwei Söhne || Jedes Blatt entspricht einem nichtleeren Suffix des Textes]
g. **Erklären Sie die folgenden Begriffe**:
    i. **Partieller Weg?** [Weg von der Wurzel zu einem Knoten von T]
    ii. **Weg?** [Ein partieller Weg der in einem Blatt endet]
    iii. **Ort?** [Der Ort ist derjenige Knoten, der am Ende des mit α beschrifteten partiellen Weges ➔ Der Ort existiert nur dann, wenn der Endknoten nicht kontrakiert ist (d.h. wenn das letzte Zeichen von α am Ende einer Teilzeichenkette einer Kante steht)]
    iv. **Erweiterter Ort?** [Falls kein Ort exisitiert ist der erweiterte Ort derjenige Knoten bei dem die Kante endet. Also Ort der kürzesten Erweiterung von α deren Ort definiert ist.]
    v. **Kontraktierte Ort?** [Falls der Ort nicht existiert, ist der Vorgängerknoten des erweiterten Ortes der Kontraktierte Ort]
h. **Wie funktioniert die naive Suffixbaumkonstruktion?** [Fange mit einem leeren Baum an und füge nach und nach alle Suffixe ein ➔ Laufzeit ist $O(n^2)$, das Einfügen eines Suffixes kostet O(n-i), wobei n die Anzahl der Zeichen von α ist]
i. **Wie funktioniert der Algorithmus M von McCreight?** [Das Ziel des Algorithmus M von McCreight ist es, den erweiterten Ort von $head_{i+1}$ in konstanter amortisierter Zeit zu finden ➔ Dazu benötigt man zusätzliche Information ➔ Es gibt zusätzliche Suffixzeiger, welche jeweils von dem Ort eines Wortes $a\gamma$ zum Ort des Wortes γ zeigen.] **UNVOLLSTÄNDIG (fehlt relativ viel) !!!**
j. **Was sind die Vor- und Nachteile von Suffix-Bäumen?** [Schnelle Konstruktion , aber sehr komplexe Datenstruktur, die viele Bytes pro Zeichen benötigt. Außerdem hat man sehr komplexe Algorithmen.]
k. **Wie wird der Text in einem Suffix-Array abgespeichert?** [Eine sortierte Liste aller Suffixe eines Textes, zusätzlich werden Informationen über längste gemeinsame Präfixe gespeichert um die Suche zu beschleunigen]
l. **Welche Eigenschaften besitzen die Suffix-Arrays?** [Teilwortsuche in O(p + log n) Zeit |Zusätzlicher Speicherbedarf ist 2n | Konstruktion in O(n log n) im wort-case bzw O(n) im average case]
m. **Wie funktioniert Präfixsuche im Suffixarray?** [Alle Präfixe zu einem gesuchten Anfang W mit Länge p liegen im pos-Array hintereinander ➔ L_w Search: Suche die untere Grenze L_w für das Vorkommen von Präfix W mit |W|=p im pos-Array ➔ Diese

geht in O(p log n) Zeit, da immer p Buchstaben in jedem Schritt der binären Suche verglichen werden müssen]

n. **Kann man das noch verbessern?** [Ja und zwar zu P(p + log n) indem man Vergleiche spart zwischen Zeichen, deren Ergebnis schon bekannt ist ➔ Dazu nutzt man die zuvor genannte Extrainformation die im Array gespeichert wird und zwar das lcp Längste gemeinsame Präfix, dadurch müssen wir die ersten h Zeichen nicht mehr vergleichen ➔ Im Worst Case wird der Algorithmus NICHT besser]

o. **Der Algorithmus kann weiter verbessert werden, und zwar durch Angaben von statischen Werten wie Llcp und Rlcp. Wie berechnen sich diese und was bedeuten sie?** [Beim Algorithmus L_w-Search wird ja mittels binärer Suche der Bereich eingeschränkt in dem das zu suchende Präfix liegen muss. L ist das linke Ende und R das rechte Ende des abgesteckten Bereiches. Die Mitte wird als M bezeichnet welche (L+R)/2 ist.
M wird im Laufe des Algorithmus zu L, wenn der zu suchende Präfix zwischen M und R liegt. Im anderen Fall wird M zu R (zu suchender Präfix liegt zwischen L und M).
Nun erstellt man in der Tabelle Werte für L, R, Llcp, Rlcp.
In L steht der Index des vorhergehenden L-Indexes wenn M zu L werden würde. Für R gilt entsprechendes.
Llcp ist nun das längste gemeinsame Präfix zwischen dem Suffix an der neuen L-Grenze (=M) und dem an der vorherigen L-Grenze (L-Wert in der Tabelle). Rlcp entsprechend.
Die Werte für L, R, Llcp, Rlcp sind statisch, da der Ablauf der binären Suche genau festgelegt ist, und M nur entweder zur linken oder rechten Grenze werden kann.]

   i. **Wie kann man nun den Algorithmus L_w-Search mit Hilfe dieser Werte beschleunigen?** [Sei l = lcp($A_{pos[L]}$,W) und r=lcp($A_{pos[R]}$,W), wobei W das zu suchende Präfix ist. Initialisiert wird l und r, indem man L=0 und R=n-1 setzt. Im Laufe des Algorithmus werden l und r verändert. Wenn r≤l ist wird Llcp benutzt (im anderen Fall benutzt man Rlcp). Nehmen wir nun o.B.d.A. an das r≤l ist, dann gibt es 3 Fälle die man unterscheiden muss:
   Llcp[M] > l: In diesem Fall weis man, das $A_{pos[M]} =_{l+1} A_{pos[L]} \neq W$ gilt. Somit gehört L_w in die rechte Hälfte. Der l-Wert bleibt so wie er ist.
   Llcp[M] = l: Somit weis man das $A_{pos[M]} =_{l} W$. Da man nicht weis ab wann sich die beiden Strings unterscheiden, muss man die Zeichen l+1 bis l+j vergleichen, so dass gilt $A_{pos[M]} \neq_{l+j} W$. Jetzt weis man allerdings nicht mehr ob L_w links oder rechts liegt, deshalb setzt man r = l+j-1 bzw. l = l+j-1 (kommt darauf an, ob r kleiner oder größer als l war (siehe ganz oben)). Die #Zeichenvergleiche sind in diesem Schritt: ΔH+1 (ΔH=Differenz zwischen max{l,r} am Anfang und Ende einer Iteration)
   Llcp[M] < l: Somit gilt $W =_{l} A_{pos[L]}$ und $W =_{l'} A_{pos[M]}$ mit l'<l. Deshalb gehört L_w in die linke Hälfte, und r = Llcp[M]
   Durch die Nutzung von Llcp und Rlcp werden pro Iteration ΔH+1 Vergleiche benötigt (2. Bedingung). Da die Summe aller ΔH≤p ist, folgt folgender **Satz**: Die Anzahl der Symbolvergleiche beträgt höchstens $p + \lceil \log_2(n-1) \rceil$.]

27. **Nennen Sie Beispiele für Textcodierungsverfahren?** [Arithmetische Codierung, Huffman-Code, Lauflängen Codierung, Lempel-Ziv Codierung etc.]

   a. **Welche 2 Klassen von Kompressionsverfahren gibt es?** Verlustfreie: ZIP (WinZip, WinRAR, etc.), s.o. | Verlustbehaftete Codierung: JPeg, MPeg, MP3]

   b. **Erklären Sie das Prinzip der arithmetischen Codierung?** [Wir möchten Zeichen(ketten) nicht alle gleichmässig codieren, wie z.B. beim ASCII Code, bei welchem jedes Zeichen mit 8 Bit codiert ist, sondern wir möchten Zeichen oder auch Zeichenketten in Abhängigkeit ihrer Auftretenswahrscheinlichkeit mit vielen bzw, wenigen Bits codieren | Häufiger ➔ weniger Bits| Dazu ordnen wir den Zeichen Intervalle zwischen 0 und 1 zu in Abhängigkeit ihrer Auftretenswahrscheinlichkeit z.B. das Zeichen I das Intervall von 0.1 bis 0.4 möchten wir nun die Zeichenkette IN codieren, müssen wir das Intervall für das zusätzliche Zeichen weiter aufsplitten und zwar in z.B. 0.22 und 0.31 usw.]

      i. **Wie kehrt man diese um?** [möchte man nun zurücktransformieren ist dazu eine Zahl innerhalb des letzten Intervalls einer Zeichenkette abgespeichert bei uns z.B. 0.25 ➔ Der Decodierer schaut nun in die Tabelle und sieht dass 0.25 im Intervall von I liegt und weiterhin das Intervall von I ein weiteres Intervall hat indem 0.25 liegt, und das ist das des zweiten Zeichen N]

ii. **Welche Nachteile hat das Verfahren?** [Statisch, passt sich nicht an wechselnde Wahrscheinlichkeiten an| Berechnung sehr aufwendig]

c. **Erkläre kurz den Huffman-Code?** [Codieren einzelner Zeichen nach Ihrer Auftretenswahrscheinlichkeit ➔ Will man dabei einen Text kodieren, ohne vorher die Zeichenhäufigkeiten zu zählen kann man bei Texten ausnutzen das z.B. das „e“ bei uns am häufigsten auftritt, usw. (hatten wir viel weiter oben schon bereits besprochen, in dem wir einen Code-Baum aufgebaut haben)]

d. Erklären Sie das Verfahren von Lempel-Ziv? [Das Wörterbuch wird simultan mit der Codierung des Textes aufgebaut. Am Anfang habe wir für jedes vorkommende Zeichen einen festen Code, sobald eine neue Zeichenkette vorkommt, wird diese mit einem neuen Code codiert ➔ Umkehrbar!!! ➔ Die Rückkodierung geht jetzt genauso, ohne Kenntnis der Tabelle]

i. **Wie ergeben sich die neuen Code-Wörter beim Kodieren und Dekodieren?** [Kodieren: Zu dem letzten verwendeten Codewort nimmt man das erste Zeichen des aktuellen Codeworts hinzu. Dieses neue Code-Wort wird nun ins Wörterbuch eingetragen. Dekodieren: ähnlich zum Kodieren, wobei man hier ausnutzen kann, das wenn ein Code kommt der noch nicht im Wörterbuch ist, das dann beim Kodieren ein neues Code-Wort erstellt worden sein muss. Dieses muss aus dem vorletzten und dem letzten Code-Wort entstanden sein.]

ii. **Was sind die Vorteile?** [Die Schlüsseltabelle muss nicht übertragen werden nur die Codetabelle für den Initialzustand muss bekannt sein ➔ z.B. ASCII Tabelle | Das Wörterbuch passt sich dynamisch an den Text an| Codieren und Decodieren in linearer Zeit | Höhere Kompressionsraten als Huffman-Code]

28. **Warum benötigt man heuristische Verfahren?** [Weil man nicht immer den ganzen Problemraum absuchen kann, bzw. Eingabefolgen u.U. erst zur Laufzeit kennt ➔ Online-Algorithmen]
29. **Was sind genetische Algorithmen?** [Genetische Algorithmen sind Algorithmen die sich am Vorbild der Natur orientieren um gute Lösungen für ein Problem zu finden. Gemäß der Evolutionstheorie werden Lösungen immer wieder mutiert, gekreuzt und selektiert, dadurch entstehen immer bessere Lösungen]
30. **Welche Sucherverfahren für kombinatorische Optimierungsprobleme gibt es und wie gliedern sich diese?** [Die Allgemeinen Suchverfahren teilen sich in Analytische Verfahren, zu denen die direkten und indirekten Verfahren gehören, und sie teilen sich in enumerative Verfahren wie z.B. das dynamische Programmieren und sie teilen sich in geführte zufällige Suche zu welcher unsere Evolutionären/Genetischen Algorithmen gehören]
31. **Erklären Sie den simplen genetischen Algorithmus anhand der Begriffe: Individuum, Generation/Population, Bewertung, Fitness und Selektion?** [Ein Individuum ist eine Lösung des Problems, welche Üblicherweise über dem Alphabet [0,1] codiert ist | Eine Generation ist eine Liste aller Individuen die in einem Schritt entstanden sind | Bewertung ist eine Funktion die die Lösung bewertet, sie gilt es maximieren bzw. zu minimieren ➔ Fitnessfunktion enthält die Daten der Bewertung in normierter Form und zwar nicht-negativ und sie korreliert positiv mit der Anzahl der Reproduktionsmöglichkeiten| Selektion: Selektiere gute Lösungen in Abhängigkeit ihrer Fitness und weise ihnen Reproduktionsmöglichkeiten zu. Die Reproduktionsmöglichkeiten der einzelnen Individuen sollen so gewählt werden, dass wenn man sie aufsummiert gerade wieder die gleiche Populationsgröße herauskommt. Somit ist gewährleistet das die Individuen jeder Generation von der Anzahl her immer gleich bleiben. Insbesondere gilt z.B. eine Reproduktionsmöglichkeit von 2.5, das das Individuum in der nächsten Generation mit mindestens 2 Kopien vorkommt.]

a. **Erkläre anhand des “Rouletterades“ wie Kopien erstellt werden?** [Die Reproduktionsmöglichkeiten müssen zunächst auf z.B. 1 normiert werden, dann kann man von Reproduktionswahrscheinlichkeiten sprechen. Die Reproduktionswahrscheinlichkeit der Individuen werden in Abschnitten des „Rouletterades“ abgetragen, d.h. je nach dem wie groß die Reproduktionswahrscheinlichkeit ist desto größer ist der Abschnitt (360° = 1.0 => z.B. 0,4 = 144°). Nun wird über das alles ein gleichwinkliger Stern gelegt mit sovielen Strahlen wie Individuen in einer Population sind ➔ Die Strahlen sind alle in gleichem Abstand über eine zufällige Verschiebung wird der Stern etwas gedreht ➔ Nun zeigen unterschiedlich viele Strahlen des Sternes auf die Einzelnen Abschnitte. Abschnitte auf die kein Strahl zeigt werden nicht reproduziert. Welche auf die ein oder

mehrer Strahlen zeigen, werden entsprechend oft reproduziert und qualifizieren sich damit für eine Kreuzung]

   b. **Welche Möglichkeiten der Kreuzung gibt es?** [Qualifizierte Individuen können über eine 1-Punkt Kreuzung reproduziert werden, das heisst die DNA zweier Individuen wird ab einer zufällig gewählten position n vertauscht | Es gibt auch eine zweipunktkreuzug mit zwei solcher zufällig gewählten Vertauschungspunkte, sie markieren den Anfang und das Ende der Vertauschung | Bei der gleichmässigen Kreuzung werden die Bits über eine zufällige Bitmaske vertauscht]
   c. **Wie wird mutiert?** [Mit einer bestimmten Wahrscheinlichkeit werden Bits bei der Reproduktion gekippt]

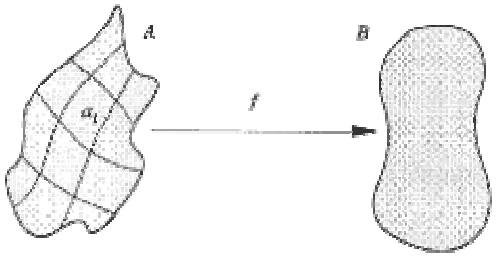
32. **Wie geht man an eine KOP Problemstellung heran?** [Wähle Codierung für potentielle Lösungen ➔ Wähle eine geeignete Bewertungs/Fitnessfunktion ➔ Wähle die Parameter, d.h. #Individuen, die Kreuzungs-/Mutationswahrscheinlichkeiten und die Abbruchbedingung] ➔ Implementation des Verfahrens]
   a. **Erkläre wie man das „Subset Sum-Problem“ mit Hilfe von GA löst? [**Gescht ist ein Subset von Zahlen aus w, welches aufsummiert gerade kleiner/gleich einer Konstanten c ist. Da es in diesem Fall u.U. unzulässige Lösungen gibt, die sehr nah an der optimalen Lösung sind, werden diese auch mit einer gewissen wahrscheinlichkeit reproduziert und nicht gleich gelöscht ➔ Mit 1000 Individuen und 20 Generationen erhält in 95% der Fälle das Optimum]
      i. **Wie geht man mit unzulässigen Lösungen um?** [Unzulässige Lösungen werden bei der Bewertung bestraft, es muss allerdings sichergestellt sein, dass die beste unzulässige Lösung nie besser als die schlechteste zulässige ist. Somit wird eine kurze Überlebenszeit von unzulässigen Lösung gewährleistet]
   b. **Erkläre wie man das „Maximum Cut Problem“ mit Hilfe von GA löst?** [Problem: einen Graphen an einer Stelle mit möglichst vielen Verbindungen schneiden. Die Gewichte der geschnittenen Kanten sollen aufaddiert maximal sein. Formulierung als Maximierungsproblem ➔ Bei 100 Individuen und 1000 Generationen ist die gefundene Lösung im Schnitt nur 5 % vom Optimum entfernt, dabei musste lediglich $4*10^{-24}$% des Suchraumes abgesucht werden ➔ Hervorragende Heuristik]
33. **Was ist ein Schema?** [Man kann mögliche Lösungen eines Problemes in einem Hypercube darstellen. Jeder Knoten ist dabei genau eine potentielle Lösung. Beschreibt man hingegen nur eine Fläche des Hypercubes, dann hat man ein Schema mit einem gewissen Freiheitsgrad. Ein Schema hat nun eine höhere Ordnung, je weniger Freiheitsgrade („*“) enthalten sind]
34. **Was besagt das Schema-Theorem?** [Bei einem Genetischen Algorithmus erhalten Schemata mit überdurchschnittlicher Fitneß, kurzer definierender Länge und niedriger Ordnung in den Folgegenerationen steigende Chancen, sich zu reproduzieren. ➔ Definition aus Schöneburg et al. (1994).
| Begriffe: Schema: Ein Schema ist definiert durch einen String der 0en, 1en und *ne enthält. Ein Hypercube hat mehrere Schemas (Flächen: 1**, 0**, *1*, *0*,**1,**0; Kanten: 00*,01*, etc; Punkte: 000,001), Definierende Länge: Differenz zwischen letzter und erster Position eines nicht „*“ Symbols (z.B. (**10**1*) => 7-3 = 4.) Ordnung eines Schemas: Anzahl Positionen die 0 oder 1 sind (z.B.: (**10**1*) => 3). Das Schema-Theorem besagt also, dass z.B. bei einem Hypercube die Folgegenerationen gegen das Schema einer Fläche konvergieren, was ja auch Sinn macht, da alle guten Lösungen zu einem bestimmten Schema gehören. Schlechte Lösungen sind in anderen Schemata zu finden, die aber nach Schema-Theorem unterdurchschnittliche Fitness haben. Berücksichtigt wird im Schema-Theorem die Selektion, 1-Punkt-Kreuzung und Mutation]
35. **Was ist der Vorteil von Parameterlosen GAs?** [Man muss diese nicht zuvor bestimmten, d.h. Codieren und „Knopf drücken“]
   a. **Wie stellt der Algorithmus die Parameter ein?** [Die Parameter werden automatisch bestimmt und zwar ebenfalls durch einen genetischen Algorithmus]
   b. **Wie geht dabei das Verfahren von Harik-Lobo?** [Für ein optimales Schema und für eine Generation t sollen die Werte für f (Fitness) und p_c (Kreuzung) so gewählt werden, dass sich die Wahrscheinlichkeit für dieses Schema in der nächsten Generation verdoppelt (ohne Garantie auf Richtigkeit ☺). Die Populationsgröße wird durch einen Wettbewerb mehrerer GAs durchgeführt.]
36. **Was steckt hinter dem Prinzip eines Künstlichen neuronalen Netzes?** [Das Prinzip eines biologischen neuronalen Netzes nachzuahmen]

a. **Welche Vorteile kann es haben?** [Man kann u.U. Lösungsansätze aus der Natur übernehmen. Neuronale Netzte sind geradezu prädestiniert für parallele Verarbeitung. NN sind redundat angelegt und lernfähig mit niedrigem Programmier-Aufwand]
b. **Was ist für ein KNN zu spezifizieren?** [Wie sich die Neuronen verhalten sollen, wie sie der Reihe nach berechnet werden und wie sie der Reihe nach aktiviert werden sollen. Einige Neuronen enthalten die Eingabewerte und wieder andere die Ausgabewerte | Die Netztopologie muss festgelegt werden, d.h. z.B, Rekurrentes Netz oder ein Feed-Forward Netz]
c. **Was ist ein Hopfield Netz?** [Ein Hopfieldnetz besteht aus d Neuronen, Aktivierungen der Neuronen, und Verbindungen von Neuronen. Das Netz wird als Graph repräsentiert: Neuronen sind die Knoten und können aktiviert (1-Wert) oder nicht aktiviert (0-Wert) sein. Kanten haben ein Gewicht und zwar derart, das das Netz etwas bestimmtes tut, d.h. die Funktionsweise oder anders ausgedrückt, das was es machen soll wird durch die Gewichte ausgedrückt. Die Ein-&Ausgabe erfolgt über die Neuronen (Aktivierung). Die Neuronen berechnen ihre Aktivierungen anhand der Aktivierungen der Nachbarknoten, und den Gewichten der Kanten zu den Nachbarknoten. Diese Updates erfolgen nach folgender Formel:

$$x'_j := \begin{cases} 0 & \text{falls } \sum_{i=1}^{d} x_i w_{ij} < 0 \\ 1 & \text{falls } \sum_{i=1}^{d} x_i w_{ij} > 0 \\ x_j & \text{sonst} \end{cases}$$

Diese Updates werden solange durchgeführt, bis das Netz einen stabilen Zustand erreicht hat. Das Ergebnis z.B. einer Boolschen Funktion kann dann an bestimmten Neuronen abgelesen werden.]
    i. **Anwendungsgebiete?** [Assoziativer Speicher, zur Berechnung boolescher Funktionen und zur kombinatorischen Optimierung]
    ii. **Was versteht man unter der Energie eines Hopfield Netzes?** [alle beidseitig aktivierten Kanten werden aufsummiert, aber jede Kante wird nur einmal betrachtet: $E(\mathbf{x}) := -\frac{1}{2}\mathbf{x}^{\mathrm{T}}\,\mathbf{W}\,\mathbf{x} = -\sum_{i<j} x_i\,w_{ij}\,x_j$]
    iii. **Welche Eigenschaften hat die Energie?** [Sie nimmt mit jedem Update des Hopfield Netzes ab]
    iv. **Wie löst man ein KOP mit Hilfe eines Hopfield Netzes? Vorgehensweise?** [Wähle Hopfield-Netz mit den Parametern des KOP als Gewichten und Lösung im Energie-Minimum | Starte Netz mit zufällig gewählten Aktivierungen | Berechne Folge von Updates, bis Stabilität erreicht | Lese Parameter aus dem Netz ab | Teste die Zulässigkeit und Optimalität der Lösung]
        1. **Beispiel: Multi-Flop-Problem?** [Gesucht ist ein Vektor der Größe n mit k vielen „1“ (k<n), das heisst Aktivierungen. Linear hört sich das trivial an, parallel lässt sich dies jedoch nur mit einem Hopfield Netz sehr elegant lösen. Man formuliert das Multiflop Problem dabei als Minimierungsproblem.
        2. **Beispiel: TSP?** [Man benutzt ein Hopfield-Netz mit n^2+1 Neuronen (n=#Städte). In der nxn Matrix wird die Ablaufreihenfolge durch die Städte repräsentiert. In den Kantengewichten werden Informationen zu Entfernungen zwischen den Städten und Werte zum Sicherstellen von zulässigen Lösungen kodiert. Eine Zulässige Lösung ist eine Permutation der Städte 1..n, das stellt sicher, dass keine Stadt zweimal besucht wird. Die optimale Lösung ist nun eine zulässige Lösung mit minimalem Kantengewicht. Ein Problem ist es beide Bedingungen, also eine Lösung zu finden die gleichzeitg eine zulässige Lösung ist, darzustellen ➔ Stetige Hopfield-Netze
d. **Was sind selbstorganisierende Karten?** [Selbstorganisierende Karten sind in den Bereich der Kohonen-Netze einzugliedern und sind somit Neuronale Netze. Eine grobe Klassifikation der Typen von neuronalen Netzen und ihrer Lernalgorithmen kann nach der Art des Lernens vorgenommen werden. So wird zwischen *überwachten* und *unüberwachtem* Lernen unterschieden. In der Situation des *überwachtem* Lernens stehen i. allg. eine Menge von Trainingsbeispielen mit einer bekannten Ausgabe zur Verfügung. Entsprechend der Ausgabe, die ein Netz für eine Trainingseingabe erzeugt, werden die Gewichte korrigiert. Beim *unüberwachten*

Lernen ist die gewünschte Ausgabe nicht unbedingt bekannt, das Netz muß folglich *selbstorganisierend* lernen. Selbstorganisierende Karten (wie auch der überwiegende Teil der unterschiedlichen Kohonen Netze) gehören zur letzteren Klasse. Entwickelt wurde das Modell der SOMs zu Beginn der 80er Jahre von Teuvo Kohonen, der damit an ältere Arbeiten von *C. von der Marlsburg* anknüpfte. Aufgabe dieses speziellen Netzes ist es, einen Eingaberaum zu kartieren, das heißt, jedem Neuron einen speziellen Bereich des Eingaberaumes 'zuzuweisen'. Dabei sollen benachbarte Neuronen auch ähnliche Werte aus dem Eingaberaum repräsentieren.



Mathematisch ausgedrückt bedeutet das, den Definitionsbereich A einer Funktion: $f : A \rightarrow B, A \subseteq \mathbb{R}^n, B \subseteq \mathbb{R}^m$ derart zu kartieren, daß für eine Eingabe aus dem Unterbereich $a_i \subseteq A$ genau ein Neuron zuständig ist (siehe Abb). Der Lernvorgang ist ein selbstgesteuerter, stochastischer Prozeß, der zur Festlegung der Netzgewichte führt.]

1. **Nenne ein paar abstrakte Anwendungs-Beispiele!** [Anpassung einer 50er-Kette an Punkte innerhalb eines Dreiecks. | Anpassung eines 15x15 Gitters an Punkte innerhalb eines Dreiecks]
2. **Beispiel: ETSP?** [Das Problem unterscheidet sich zum normalen TSP dadurch, dass die Städte nun Koordinaten in einem euklidischen Koordinatensystem haben. ➔ Lösung mit SOM. Idee: Verwende wachsenden Ring von Neuronen, der Ring legt sich nun wie bei den Selbstorganisierenden Karten definiert in jedem Verfeinerungsschritt um immer mehr Städte des ETSP (d.h. der Ring passt sich an die Städte an, ähnlich zu der 50er-Kette die sich an Punkte aus dem Dreieck anpasst, nur sind die Punkte jetzt die einzelnen Städte) | Komplexität $O(n^2)$. Tests mit bis zu 2392 Neuronen haben gezeigt, dass die Laufzeit nur linear skaliert und die gefundenen Lösungen weniger als 9% vom Optimum abweichen]
3. **Was ist ein geschichtetes Feed-Forward-Netzwerk (MLP)?** [Man nennt diese auch mehrlagiges Perzeptron. Wir haben mehrere Stufen der Abarbeitung (immer komplett eine Schicht wird abgearbeitet, bevor die nächste dran kommt), das Netz ist gerichtet. Jede Stufe enthält Neuronen. Es gibt Eingabe, Ausgabe Neuronen und versteckte, dazwischenliegende deren Funktion man von aussen nicht „sehen“ kann. Man kann die MLPs verwenden um z.B. Boolesche Funktionen zu berechnen, dies geht durch ein nur 2-Schichtiges MLP | Desweiteren können z.B. stetige, reele Funktionen und auch Ihrer Ableitungen auf einem kompakten Bereich beliebig genau approximiert werden ➔ 2-Schichtiges MLP]

ii. **Wie funktioniert das Parameterlernen?** [Gesucht sind Gewichte die die Fehlerfunktion minimieren]

iii. **Wie funktioniert die Gradientenbestimmung mit Backpropagation?** [Wiederhole für alle $n \in \{1,..,N\}$:
Berechne Netzfunktion $f(x^n,w)$ und dazugehörigen Fehler E in Vorwärtsrichtung, speichere Zwischenwerte im Netz.
Berechne partielle Ableitungen von E in Bezug auf alle Zwischenwerte in Rückwärtsrichtung und addiere Beiträge zum Gesamtgradienten. (hab ich nicht kapiert☺)]

37. **Welche Idee steckt hinter den Ameisenalgorithmen?** [Ameisen verstreuen bei Ihrer Suche nach einem Futterplatz Feromone. Am Anfang laufen die Ameisen recht zufällig bestimmte Wege und hinterlassen Ihre Feromone. Mit zunehmende Feromon Konzentrattion auf den Wegen, orientieren sich die Ameisen immer mehr (mit Wahrscheinlichkeitsfaktor) an der höheren Feromonkonzentrattion. Das Feromon verdunstet nach und nach und kann nur durch

neue darüberlaufende Ameisen aufgefrischt werden. Bei langen Wegen zum Futter ist daher die Ameisendichte von Anfang an geringer und daher laufen die Ameisen sehr schnell einen immer optimaleren Weg.]

a. **Erkläre an Hand des TSP?** [Am Anfang lässt man an jeder Stadt eine Ameise loslaufen. Jede Stadt hat nun Verbindungen zu jeder anderen Stadt. Die Ameisen laufen beliebige Permutationen von [1..n]. Nach dem Prinzip der Ameisen, werden nun besonders kurze Strecken gefunden und irgendwann laufen alle Ameisen genau den gleichen Zyklus ab, welcher dann die optimale Lösung ist.| Folgende Parameter gilt es einzustellen: Entscheidung der Ameise, eher nach dem Feromonlevel oder eher nach der Entfernung (Sichtbarkeit) zu einer Stadt (damit die Ameise nicht immer nur nach dem Feromonlevel entscheidet). Gedächtnis der Ameise. Einfluss neuer Information in Relation zur Initialisierung. Dann muss man noch die Geschwindigkeit der Verdunstung wählen]
b. **Erkläre das Quadratische Zuordnungsproblem QAP?** [Das Problem ist es, bestimmte Aktivitäten an Orte zu binden, die Orte haben gewisse Entfernungen und die Aktivitäten eine bestimmte Flussmenge. Vergleichbar ist das Ganze mit einer Shopping Mall. Viele Geschäfte mit hohem Flusspotential (das heisst vielen potientiellen Besuchern sind oft nahe beieinander, damit die Leute nicht so viel laufen müssen). Umgekehrt sind die mit weniger Flusspotential, weiter weg ➔ Typisches Minimierungsproblem ➔ Die Heuristik geht nun folgendermaßen: Ordne die Aktivitäten in der Reihenfolge ihrer Flusspotentiale absteigend und weise Ihnen sukzessive einen noch freien Ort mit minimaler Distanz zu. Das Problem kann mit einem Ameisen-Algorithmus relativ gut gelöst werden.]
c. **Was ist die Idee des Sintflutalgorithmus?** [Lösungen werden wie eine Landschaft mit unterschiedlichen Maxima repräsentiert. Das Ziel ist es nicht ein lokales Maxima zu finden, sondern das absolute. Dazu lässt man Ameisen auf der gesamten Ebene herumlaufen und füllt die Ebene nach und nach mit Wasser auf, die Ameisen laufen nun den ihnen am nächsten liegenden Berg hinauf. Hat man genügend Ameisen, dann sollte am Schluss hoffentlich zumindest eine auf dem Berg mit dem Maximum sitzen. Die anderen sind ertrunken (nicht optimale Lösungen)]