Aufgabensammlung ADS-Repetitorium WS 25/26

Dynamische Programmierung – Rückblick

Aufgabe 1: SubsetSum

Gegeben sind eine Liste $A = \langle o_1, o_2, \ldots, o_n \rangle$ von n natürlichen Zahlen sowie ein $b \in \mathbb{N}$. Gefragt ist, ob die Summe von einem Teil der in A enthaltenen Zahlen genau b ergibt. Formal ist also gefragt, ob ein $I \subseteq \{1, 2 \ldots, n\}$ mit $\sum_{i \in I} o_i = b$ existiert. Ein naiver Ansatz wäre es, alle Teilmengen der in A enthaltenen Zahlen auszuprobieren. Da es jedoch 2^n viele solche Teilmengen gibt, hat dieser Ansatz eine Laufzeit von $\Omega(2^n)$. Wir wollen stattdessen ein dynamisches Programm angeben, dessen Laufzeit in $\mathcal{O}(n \cdot b)$ liegt.

(a) Wie sehen geeignete Teilprobleme aus, die hierfür gelöst werden müssen?

(b) Wir suchen nun zunächst eine rekursive Formulierung, mit der die Antwort auf die Teilprobleme berechnet werden kann. Welchem Teilproblem entspricht die Lösung der eigentlichen Aufgabe?

(c) Nun soll diese rekursive Lösung in einem dynamischen Programm berechnet werden. Wie groß ist der Speicherbedarf dieser Lösung?

(d) Ist es möglich, einen von n unabhängigen Speicherbedarf zu erreichen? Wenn ja, wie lässt sich das erreichen?

Aufgabe 2: Knapsack

Gegeben sei eine Menge von n Objekten. Jedes Objekt o_i besitzt einen ganzzahligen Wert v_i und ein ganzzahliges Gewicht $w_i \in \mathbb{N}$. Außerdem ist ein Maximalgewicht $W \in \mathbb{N}$ gegeben. Gesucht ist eine Teilmenge von Objekten U, sodass $\sum_{o_i \in U} w_i \leq W$, wobei $\sum_{o_i \in U} v_i$ maximiert wird. Lösen Sie dieses Problem mittels dynamischer Programmierung! Ihr Programm sollte eine Laufzeit von $\mathcal{O}(n \cdot W)$ haben. Es reicht, wenn Sie den Wert $\sum_{o_i \in U} v_i$ einer optimalen Lösung U bestimmen.

Aufgabe 3: Längste Wege in azyklischen Graphen

Wir haben bereits gesehen, dass das Problem Längster Wege in allgemeinen Graphen nicht die Eigenschaft optimaler Teilstrukturen aufweist. Wir beschränken uns daher nun auf azyklische gerichtete Graphen. Es wird ein solcher Graph $G = (V, E)$ mit $E \subseteq V^2$ sowie eine Kantengewichtsfunktion $w : E \mapsto \mathbb{R}^+$ gegeben. Außerdem werden Start- und Endknoten $s, t \in V$ übergeben. Gesucht ist ein längster Weg von s nach t.

(a) Wir wollen zunächst zeigen, dass dieses eingeschränkte Problem die optimale-Teilstruktur-Eigenschaft hat. Dazu wollen wir nun beweisen, dass der längste s-t-Weg für einen Knoten $v \in V$ aus einem längsten s-v-Weg sowie einem längsten v-t-Weg besteht.

Hinweis: Es könnte hilfreich sein, sich nochmal zu überlegen, wieso das Problem *kürzester* Wege aus optimalen Teilstrukturen besteht.

(b) Was sind die Teilprobleme, die in einem dynamischen Programm gelöst werden müssten? Wie können diese Teilprobleme rekursiv gelöst werden?

(c) In welcher Reihenfolge müssen die Teilprobleme gelöst werden?

Aufgabe 4: Vereinigung von Intervallen

Gegeben sei eine Liste $R = \langle [x_1, y_1], \ldots, [x_n, y_n] \rangle$ von gegebenenfalls überlappenden Intervallen. Geben Sie einen Algorithmus an, der die Gesamtlänge der Vereinigung von den Intervallen in R angibt. Der Algorithmus soll eine asymptotische worst-case Laufzeit von $O(n \log n)$ haben!

Hinweis: Abbildung 1 zeigt ein Beispiel. Könnte eine gewisse Sortierung der Intervalle helfen?

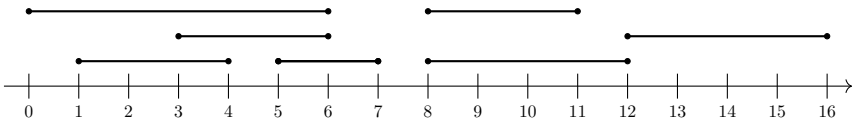


Abbildung 1: Mögliche Intervalle in R. Die Gesamtlänge der Vereinigung dieser Intervalle ist 15.

Aufgabe 5: Polynome evaluieren

Die Regel von Horner ist eine Möglichkeit Polynome zu evaluieren:

$$P(x) = \sum_{k=0}^{n} a_k x^k = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n) \ldots))$$

Folgender Pseudocode implementiert diese Regel:

Algorithmus 3: evaluatePolynomial$(a_0, \ldots, a_n, x)$

```
1 y = 0
2 for k = n downto 0 do
3 |  y = a_k + x · y
4 return y
```

(a) Welche asymptotische Laufzeit hat evaluatePolynomial? Begründen Sie Ihre Antwort kurz!

(b) Schreiben Sie einen Pseudocode, der naiv das Polynom evauliert, indem jedes x^k komplett neu berechnet wird. Schätzen Sie die Laufzeit Ihres Algorithmus asymptotisch scharf in Θ-Notation ab!

(c) Zeigen Sie die Korrektheit von evaluatePolynomial mittels Schleifeninvariante!

Aufgabe 6: Flugsicherheit

Im Flugverkehr müssen die Flugzeuge gewisse Abstände einhalten. Gegeben ist eine unsortierte Liste von Flugzeugen. Jedes Flugzeug a hat drei Attribute, nämlich $a.x$, $a.y$ und $a.z$. Diese Attribute geben die Koordinaten im Luftraum an. Sie sollen einen Algorithmus angeben, der true ausgibt, falls sich zwei Flugzeuge näher als den Abstand d kommen. Ihr Kommilitone hat einen Algorithmus entwickelt (siehe Algo 5), der dieses Problem lösen soll.

Algorithmus 5: planesTooClose(Plane[ ] planes, d)

```
1  mergeSort(planes) // Sortiere nach y-Koordinate
2  if planes.length < 2 then
3      return false
4  for i = 2 to planes.length do
5      f_1 = planes[i]
6      f_2 = planes[i − 1]
7      yDistance = |f_1.y − f_2.y|
8      if yDistance < d then
9          e = sqrt((f_1.x − f_2.x)^2 + (f_1.y − f_2.y)^2 + (f_1.z − f_2.z)^2)
10         if e < d then
11             return true
12 return false
```

(a) Welche Laufzeit hat dieser Algorithmus, wenn man davon ausgeht, dass eine Wurzeloperation $\mathcal{O}(1)$ Zeit benötigt?

(b) Ist der Algorithmus korrekt? Begründen Sie Ihre Antwort.

Aufgabe 7: Worst-Case-Laufzeiten für Algorithmen

Wir betrachten folgendes Problem. Aus einem Eingabefeld A von ganzen Zahlen sollen alle Tupel (i, j) mit $i < j$ ausgegeben werden, sodass A[i] + A[j] ein Vielfaches von 10 ist.

(a) Zeigen Sie, dass jeder Algorithmus, der dieses Problem löst, eine Worst-Case-Laufzeit von $\Omega(n)$ hat.

(b) Kann man eine asymptotisch größere (und damit bessere) untere Schranke angeben? Beweisen Sie Ihre Behauptung. Sie müssen dafür entweder zeigen, dass die Laufzeit für *alle* Listen in $\Omega(n)$ liegt bzw. es *eine* Familie von Listen der Länge n gibt, für die die Laufzeit größer ist.

Aufgabe 8: Sortieren mit Quicksort

(a) Gegeben ist die Ausgabe der Methode Partition des Quicksort Algorithmus. Rekonstruieren Sie die Eingabe. Konkret sollen Sie das Array $A = \langle \_, \_, 1, \_, \_ \rangle$ so vervollständigen, dass der Aufruf Partition(A, 1, 5) die Zahl 3 zurückgibt und nach dem Aufruf gilt, dass $A = \langle 1, 2, 3, 4, 5 \rangle$ ist.

(b) Beweisen Sie die Korrektheit von Partition mittels Schleifeninvariante.

(c) Geben Sie für jede natürliche Zahl n eine Instanz der Länge n an, sodass QuickSort $\Omega(n^2)$ Zeit benötigt. Begründen Sie ihre Behauptung.

(d) Was müsste Partition (in Linearzeit) leisten, damit QuickSort Instanzen der Länge n in $\mathcal{O}(n \log n)$ Zeit sortiert? Zeigen Sie, dass Partition mit der von Ihnen geforderten Eigenschaft zur gewünschten Laufzeit von QuickSort führt.

Aufgabe 9: Rekursive Gleichung aufstellen

Gegeben sei folgender Algorithmus:

Algorithmus 6: RecursiveAlgo(int A[ ], int l= 1, int r=A.length)

```
1 if l < r then
2     m = ⌊(l + r)/2⌋
3     RecursiveAlgo(A, l, m)
4     RecursiveAlgo(A, m + 1, r)
5     InsertionSort(A, l, r)
```

(a) Stellen Sie eine Rekursionsgleichung T für den gegebenen Algorithmus auf.

(b) Finden Sie eine Funktion f, für die $T \in \Theta(f)$ gilt.

Aufgabe 10: Bernoullis Spielzeuge

Eine Supermarktkette verschenkt ein zufälliges Spielzeug, wenn Sie einen Einkauf in Höhe von über 10 Euro tätigen. Insgesamt gibt es n unterschiedliche Spielzeuge, die Sie gerne sammeln möchten. Mit wie vielen Einkäufen müssen Sie rechnen, bis Sie alle Spielzeuge gesammelt haben?

Aufgabe 11: Löschen in einer Hash-Tabelle

Gegeben sei eine Hashtabelle H mit einer Hash-Funktion $h(x, i)$. Es wird offene Adressierung verwendet.

(a) Beschreiben Sie in Worten, wie der Algorithmus Search(**int** k) aus der Vorlesung funktioniert.

(b) Ein Element k soll aus der Tabelle gelöscht werden. Warum sollte man den Wert nicht mit der folgenden Befehlsfolge löschen?
j = Search(k)
H[j] = -1

(c) Implementieren Sie die Operation Delete(**int** k), die einen Schlüssel aus der Tabelle löscht, ohne dass das Problem aus b) auftritt. *Tipp*: Verwenden Sie einen besonderen Wert, um gelöschte Zellen zu markieren.

(d) Welche Änderung muss nun in den Methoden Insert(**int** k) und Search(**int** k) vorgenommen werden?

(e) Beschreiben Sie kurz die Auswirkungen Ihrer Änderungen auf die Laufzeit der Operationen.

Aufgabe 12: Löschen in Binärbäumen

Sie haben bereits kennengelernt, wie man einen Knoten aus einem Binärbaum löscht. Implementieren Sie Delete in Pseudocode!

(a) Starten Sie mit einer Hilfsfunktion Transplant, die den Baum T und zwei Knoten u, v als Argumente entgegen nimmt und den Teilbaum u mit den Teilbaum v ersetzt. Das heißt nach einem Aufruf von Transplant hat u's *parent* v als Kind an der richtigen Stelle. Achten Sie auf das Setzen der richtigen *parent* Pointer!

(b) Nutzen Sie Transplant als Unterroutine, um Delete zu implementieren!

Aufgabe 13: Finden von relevanten Intervallen

Angenommen Sie haben einen Sensor, der n Werte über eine Zeit in unregelmäßigen Abständen gemessen hat und diese Werte chronologisch in ein Feld A geschrieben hat. Ein Eintrag eines Feldes entspricht einem Tupel (t_i, x_i), wobei t_i die Zeit ist, an dem der Messwert x_i vom Sensor gemessen wurde. Sie wollen nun einen bestimmten Zeitraum $[t_s, t_e]$ an Messwerten abfragen. Geben Sie einen Algorithmus an, der in $o(n)$ ein Tupel von Indize (i, j) zurückgibt, sodass alle Messungen, die im Zeitraum $[t_s, t_e]$ getätigt wurden im Teilfeld $A[i..j]$ stehen.
Hinweis: Weder t_s noch t_e müssen als Werte in A existieren.

Aufgabe 14: Elemente ausgeben

Gegeben seien zwei unsortierte Arrays mit ganzen Zahlen A und B mit jeweils n Elementen. Die Elemente eines Arrays sind dabei paarweise verschieden. Entwickeln Sie einen Algorithmus, dessen worst-case Laufzeit $\mathcal{O}(n \log n)$ ist, der alle Elemente von A ausgibt, die **nicht** in B vorkommen. Finden Sie auch einen Algorithmus, der das Problem in (erwartet) $\mathcal{O}(n)$ löst?

Aufgabe 15: Anzahl der einfachen Pfade im Graph

Verwenden Sie den Tiefensuche-Algorithmus, um die *Anzahl* der einfachen Pfade zwischen zwei Knoten in einem azyklischen, gerichteten und ungewichteten Graphen in $\mathcal{O}(|V| + |E|)$ Zeit zu berechnen. Verwendet Ihr Algorithmus das Prinzip dynamischer Programmierung oder ist er ein Greedy-Algorithmus?

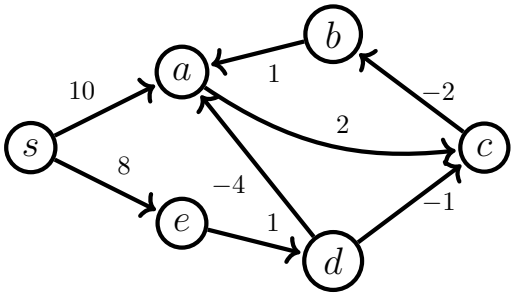
Aufgabe 16: Kürzeste Wege mit negativen Kanten

Gestern haben wir festgestellt, dass die Ergebnisse von Dijkstra auf Graphen mit negativen Kanten unter Umständen nicht die kürzesten Wege repräsentieren. In dieser Aufgabe wollen wir einen Algorithmus finden, der auf einem Graphen $G = (V, E)$ mit der Gewichtsfunktion $w : V \times V \to \mathbb{R}$ einen kürzesten Weg zwischen zwei Knoten s und t findet. Wir nehmen an, dass der Graph G keine von s erreichbaren, negativen Kreise enthält.

(a) Zeigen Sie, dass das Problem eine optimale Substruktur aufweist. Sie müssen also zeigen, dass der kürzeste Weg zwischen s und t aus kleineren Teillösungen desselben Problems berechenbar ist. In welchen Fällen ist es besonders einfach, den kürzesten Weg zwischen s und t zu berechnen?

(b) Wie viele Kanten kann jeder kürzeste Weg im Graphen $G = (V, E)$ maximal haben? Angenommen, der Distanzwert $v.d$ ist für alle $v \in V$ mit ∞ intialisiert und $s.d = 0$. Was passiert auf jeden Fall, wenn Sie die Relax-Methode (siehe Dijkstra) nun auf *jede* Kante in beliebiger Reihenfolge aufrufen? Was passiert, wenn Sie dies erneut tun?

(c) Wir betrachten nun eine zweidimensionale Tabelle T. Jede Spalte steht für einen Knoten (in beliebiger Reihenfolge), und es gibt $|V| - 1$ Zeilen. Die Zelle $T(i, j)$ enthält die *Länge* des kürzesten Weges von s zum i-ten Knoten, nachdem j Mal die Relax-Methode auf *alle* Kanten aufgerufen wurde. Stellen Sie die Tabelle für folgenden Graphen auf und füllen Sie sie zeilenweise aus. Relaxieren Sie die Kanten in alphabetischer Reihenfolge nach ihrem Startknoten.



Iteration	$s.d$	$a.d$	$b.d$	$c.d$	$d.d$	$e.d$
0	0	∞	∞	∞	∞	∞
1						
2						
3						
4						
5						

(d) Geben Sie nun den Algorithmus in Pseudocode an. Welche Laufzeit hat er?

(e) Modifizieren Sie Ihren Algorithmus so, dass er auch die optimale Lösung selbst, also den kürzesten Weg berechnet. Welche Methode aus der Vorlesung können Sie dafür verwenden?

(f) Unter welchen Umständen kann der Algorithmus vorzeitig abgebrochen werden? Wie kann mithilfe des Algorithmus ein negativer Zykel detektiert werden?

Aufgabe 17: Palindrome Subsequenzen

Ein *Palindrom* ist eine Zeichenkette, die von vorne und von hinten gelesen das gleiche ergibt, zum Beispiel das Adjektiv „soldlos“. Eine *Subsequenz* ist ein String, der nach Weglassen beliebig vieler Zeichen aus einem String hervorgeht, beispielsweise das Wort „Baum“ aus „Brauchtum“. Wir suchen nun einen effizienten Algorithmus, der eine längste Subsequenz einer Zeichenkette $s = s_0 \dots s_n$ findet, die gleichzeitig ein Palindrom ist. Die längste palindrome Subsequenz in „Amortisierte Laufzeit“ ist „tieteit“.

(a) Erklären Sie kurz, wie ein Brute-Force-Algorithmus vorgehen würde, um das Problem zu lösen. Was ist die Laufzeit dieses Algorithmus?

(b) Gegeben sei eine Zeichenkette $s = s_0 \dots s_n$ und ihre längste palindrome Subsequenz $p = p_0 \dots p_m$. Beschreiben Sie wie p aus einer kleineren Instanz desselben Problems hervorgeht. Betrachten Sie dazu einen Teilstring s' von s, und erklären Sie, wie p zu s' steht.

(c) Wir definieren $l(i, j)$ als die Länge der längsten palindromen Subsequenz im Substring $s_i \dots s_j$. Was ist $l(i, i)$ und $l(i, i+1)$? Dies sind die Basisfälle und sind einfach anzugeben. Überlegen Sie sich nun, wie Sie für allgemeine i, j mit $i < j$ den Wert $l(l, j)$ berechnen können. *Tipp*: Machen Sie eine Fallunterscheidung nach $s_i = s_j$ bzw. $s_i \neq s_j$ und greifen Sie auf $l(i', j')$ zu, wobei $i' < i$ oder $j' < j$.

(d) Legen Sie eine Matrix, die $l(i, j)$ für alle $0 \leq i \leq j \leq n$ repräsentiert, für die beiden Sequenzen „anna“ und „graphalgo“ an und füllen Sie sie aus. Wie lang sind die längsten palindromen Subsquenzen in den Wörtern? Wo steht der Wert der Lösung in der Matrix?

(e) Formulieren Sie jetzt einen Algorithmus, der eine solche Matrix automatisch ausfüllt und den *Wert* der Lösung zurückgibt.

(f) Jetzt möchten Sie nicht nur den Wert ermitteln, sondern auch die längste palindrome Subsequenz selbst. Beschreiben Sie, wie Sie mit einer zweiten Matrix die längste palindrome Subsequenz ermitteln können.

(g) Zeichnen Sie auch die ergänzte Matrix für die Wörter „anna“ und „graphalgo“. Was ist die jeweils längste palindrome Subsequenz?

(h) Ändern Sie Ihren Algorithmus so, dass er die längste palindrome Subsequenz zurückgibt.

(i) Überlegen Sie sich, wie Sie Ihren Algorithmus ändern können, sodass er den längsten palindromen *Substring* findet. Im Wort „stirnlappenbasilisk“ ist der längste palindrome Substring „silis“, während die bisher betrachtete palindrome Subsequenz „silappalis“ ist. Formulieren Sie die Matrix-Rekurrenzen aus Teilaufgabe c) um und beschreiben Sie in Worten, wo sich in der Matrix jetzt der Wert der Lösung befindet und wie Sie die Lösung rekonstruieren können.

(j) Falls Sie noch Zeit haben, geben Sie einen Brute-Force-Algorithmus an, der eine längste palindrome Subsequenz findet.

Aufgabe 18: Der schönste Binärbaum der Welt

Sie sind auf Kiliani und laufen an einem Stand vorbei. Dort fällt Ihnen der schönste Binärbaum in die Augen, den Sie je gesehen haben. Unverzüglich gehen Sie den kürzesten Pfad zum Stand und sehen, dass dieser Binärbaum der Hauptpreis eines Spieles ist. Das Spiel funktioniert wie folgt:
Innerhalb des Standes sind n Pins nebeneinander aufgestellt. Auf jedem dieser Pins ist eine ganze Zahl (also auch negative) geschrieben. Sie erhalten einen Ball und müssen die Pins abwerfen, sodass Sie ihren Score maximieren. Dieser Score berechnet sich folgendermaßen:

- Sie treffen genau einen Pin: Sie erhalten den Wert des Pins aufaddiert zu ihrem bisherigen Score
- Sie treffen zwei benachbarte Pins: Sie erhalten das Produkt beider Pins zu ihrem bisherigen Score aufaddiert

Sie können nicht mehr als zwei benachbarte Pins treffen und Sie müssen nicht alle Pins abwerfen. Sobald Sie einen Pin abgeworfen haben, fällt dieser um und kann nicht erneut abgeworfen werden. Wie es der Zufall will können Sie ausgezeichnet werfen und treffen immer wohin Sie zielen.
Entwickeln Sie ein dynamisches Programm, das eine Sequenz $v_1, \ldots, v_n$ von Zahlen (die Werte der Pins) entgegen nimmt und den maximalen Score errechnet.

(a) Definieren Sie das Teilproblem $B(\cdot)$.

(b) Wie kann das Teilproblem, das Sie in Teilaufgabe (a) definiert haben berechnet werden? Geben Sie hierzu eine rekursive Gleichung an.

(c) Schreiben Sie einen Pseudocode, der die rekursive Gleichung in Teilaufgabe (b) implementiert! Ordnen Sie die asymptotische Laufzeit des Algorithmus ein und begründen Sie ihre Antwort! (eine genaue Berechnung ist nicht gefordert)

(d) Nutzen Sie nun *memoization*, um redundante Berechnungen in Ihrem rekursiven Algorithmus zu vermeiden! Welche Laufzeit hat der *memoized* Algorithmus nun?

(e) Implementieren Sie den Algorithmus nun bottom-up!

Aufgabe 19: Independent Sets in Bäumen

Eine unabhängige Menge in einem Graphen ist eine Teilmenge $U \subseteq V$, sodass keine Knoten in U miteinander benachbart sind. Im Independent Set Problem ist nach der größten unabhängigen Menge in einem Graphen G gesucht. Im Allgemeinen ist dieses Problem NP-schwer, doch in Bäumen lässt sich dieses Problem in Polynomialzeit lösen. Sei also ein Baum $T = (V, E)$ gegeben. Schreiben Sie ein dynamisches Programm, das in $\mathcal{O}(V + E)$ Zeit die Kardinalität eines größten Independent Sets in T findet.
Hinweis 1: Ein Baum als Graph besitzt keine eindeutige Wurzel. Sie können annehmen, dass der Baum T bereits an Wurzel r gewurzelt ist.
Hinweis 2: Nutzen Sie die rekursive Struktur eines Baums aus! Nutzen Sie zwei Tabelleneinträge pro Knoten, einmal für den Fall, dass Sie den Knoten nehmen und einmal für den Fall, dass Sie ihn nicht nehmen.

Aufgabe 20: Perfekte Binärbäume

Angenommen wir haben n Elemente und wissen, mit welcher Wahrscheinlichkeit diese angefragt werden. Wir wollen nun einen binären Suchbaum finden, der die erwartete Anfragezeit minimiert. Das heißt, gegeben eine Wahrscheinlichkeitsverteilung der n Elemente $p_1, p_2, \ldots, p_n$ mit der die Elemente angefragt werden. Wir wollen die Suchzeit $\sum_{i=1}^{l} p_i \cdot l_i$ minimieren, wobei l_i das Level von Element i ist. Im Folgenden wollen wir ein dynamisches Programm entwickeln, das dieses Problem löst.

(a) Wie verändert sich die erwartete Suchzeit eines Teilbaums, wenn dieser an einen weiteren Knoten gehängt wird?

(b) Angenommen Sie haben eine Methode, um die Wurzel eines optimalen Teilbaums zu berechnen. Gegeben sei ein Teilbaum mit folgender Sequenz von Elementen $v_i \ldots, v_r, \ldots, v_j$, wobei v_r die Wurzel eines optimalen Teilbaums ist. Wie können Sie den Rest des Teilbaums berechnen?

(c) Sei $D[i, j]$ der Wert einer optimalen Lösung mit den Elementen $v_i, \ldots, v_j$. Stellen Sie die Rekursionsgleichung für $D[i, j]$ auf!

(d) Für welche Sequenz von Elementen kennen wir bereits die optimale erwartete Suchzeit (Basisfall)?

(e) Welche Laufzeit hat Ihr dynamisches Programm?