Algorithmen und Datenstrukturen 1

ALGO1 · SoSe-2023 · tcs.uni-frankfurt.de/algo1/ · 2023-07-06 · *ed6968a*

Dynamische Programmierung (Woche 13)

Eigenständige Vorbereitung Lies 📖 E Kapitel 3 ohne 3.6 und 3.9 und schau dir das 📺 Video der Woche an.

Zeichenlegende:

- ✏️ Schriftliche Aufgabe, die du fristgerecht in Moodle abgibst. In der Klausur wirst du alle Aufgaben schriftlich bearbeiten, daher ist das Feedback der Tutoren wichtig, damit du deine Schreibfähigkeiten verbessern kannst.
- 👍 Diese Art von Aufgabe musst du sicher können, um die Klausur zu bestehen.
- 🔑 Diese Art von Aufgabe musst du weitgehend können, um die Klausur zu bestehen.
- 🔭 Diese Art von Aufgabe musst du können, um eine gute Note zu erhalten.
- 🌈 Diese Aufgabe ist als Knobelspaß gedacht, der das algorithmische Verständnis vertieft.

Aufgabe 13.1 (Editieren).

a) 👍 Welche Editiersequenz ist hier visualisiert und wie viele Editieroperationen wurden benutzt?

```
P   L U   T   O
P L A   N E T
```

b) 👍 Fülle alle 42 Einträge der Tabelle `Edit[i,j]` mit A = `PLUTO` und B = `PLANET` aus. Gib die optimale Editiersequenz an.

c) 🔑 Die Tabelle `Edit[i,j]` hat mn Einträge und braucht daher $O(mn)$ Platz. Wie kann das dynamische Programm für die Editiersequenz *platzsparend* gemacht werden, sodass zu jedem Zeitpunkt nur noch $O(\min\{m, n\})$ Platz gebraucht wird? Die Laufzeit soll dabei asymptotisch gleich bleiben.

Hinweise zur Abgabe und zum Vorgehen bei Aufgaben des Typs „Dynamische Programmierung“. Im Buch [Erickson, Abschnitt 3.4] ist in kleinen Schritten beschrieben, wie man bei dynamischer Programmierung vorgeht. Folge diesen Schritten! Fang gar nicht erst an, über for-Schleifen nachzudenken, bevor du eine vollständige rekursive Lösung hast! Das beinhaltet eine klar verständliche deutsche oder englische Spezifikation der rekursiven Teilprobleme, die du löst. In der Algorithmenentwicklung, in der Programmierung, und in den meisten Aufgaben des Lebens gilt immer: **Mach es zuerst korrekt, dann effizient.**

Aufgabe 13.2 (Pascalsches Dreieck ✏️).
Für die Binomialkoeffizienten gilt folgende Rekursionsformel:

$$\binom{n}{0} = \binom{n}{n} = 1 \qquad \text{für alle } n \geq 0, \text{ und}$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \qquad \text{für alle } n, k \text{ mit } 1 \leq k \leq n-1.$$

Wir möchten eine Funktion implementieren, die $\binom{n}{k}$ ausrechnet.

a) 👍 Entwirf zunächst eine rekursive Funktion RecBinom(n, k), die schlicht die Rekursionsformel rekursiv benutzt. Wie viele rekursive Aufrufe braucht RecBinom(n, k)? Gib die Antwort in asymptotischer Notation in Abhängigkeit von n und k an.

b) 👍 Entwirf nun mit Hilfe von dynamischer Programmierung eine iterative Funktion IterBinom(n, k), die $O(nk)$ arithemtische Operationen ausführt und dabei $O(nk)$ Zahlen im Speicher hält.

c) 🔑 Entwirf nun mit Hilfe von platzsparender dynamischer Programmierung eine iterative Funktion IterBinom2(n, k), die ebenfalls $O(nk)$ arithmetische Operationen braucht, dabei aber nur $O(k)$ Zahlen im Speicher halten muss.

Aufgabe 13.3 (Betriebsfeier 🔑). Du organisierst die jährliche Betriebsfeier bei der Algorithmen-Bank. Die Mitarbeiter:innen der Bank sind in einer strikten Hierarchie angeordnet: ein Baum, in dem die Eigentümerin der Bank die Wurzel darstellt. Die allwissende Personalabteilung hat für jede:n Angestellte:n eine reelle Zahl berechnet, die angibt, wie „lustig“ diese Person ist. Damit die Feier entspannt bleibt, gibt es folgende Einschränkung für die Gästeliste: ein:e Mitarbeiter:in darf nicht an der Feier teilnehmen, wenn seine:ihre direkte Vorgesetze teilnimmt. Andererseits *muss* die Eigentümerin an der Feier teilnehmen, obwohl sie nicht besonders lustig ist; es ist ja schließlich auch ihre Firma. Beschreibe und analysiere einen Algorithmus, der diejenige Gästeliste für die Betriebsfeier berechnet, die die Summe aller „lustig“ Bewertungen der Gäste maximiert.

Aufgabe 13.4 (Genaues Rückgeld). In einem früheren Leben hast du als Kassierer:in in der verlorenen antarktischen Kolonie Nadiria gearbeitet, wo du die meiste Zeit des Tages Rückgeld an deine Kund:innen gegeben hast. Weil Papier ein knappes und wertvolles Gut in der Antarktis ist, gab es das Gesetz, dass Kassierer:innen immer die kleinstmögliche Anzahl an Geldscheinen zurückgeben müssen. Die Währung von Nadiria heißt Traumdollar und war in den folgenden Scheinen verfügbar: \$1, \$4, \$7, \$13, \$28, \$52, \$91, \$365.

a) 👍 Der gierige Wechselalgorithmus nimmt immer den größten Schein, der kleiner ist als das noch zu gebende Rückgeld. Zum Beispiel, um \$122 mit dem gierigen Algorithmus zurückzugeben, nehmen wir zuerst einen \$91 Schein, dann einen \$28 Schein, und am Ende drei \$1 Scheine. Gib ein Beispiel an, in dem der gierige Algorithmus mehr Traumdollarscheine als Wechselgeld gibt als die minimal mögliche Anzahl. *Hinweis: Es könnte einfacher sein, ein kleines Programm zu schreiben, als von Hand ein Beispiel zu konstruieren.*

b) 🔑 Beschreibe und analysiere einen rekursiven Algorithmus, der für eine gegebene Zahl k die kleinste Anzahl an Scheinen berechnet, die benötigt werden, um k Traumdollar Rückgeld zu geben. (Mach dir erstmal keine Gedanken darüber, wie man den Algorithmus schnell macht; stell einfach sicher, dass er korrekt ist.)

c) 🔑 Beschreibe einen Algorithmus, der mithilfe von dynamischer Programmierung dasselbe algorithmische Problem löst. (Dieser Algorithmus soll schnell sein.)

Aufgabe 13.5 (Maximales Teilfeld II 🔑). Erinnere dich zurück an die Aufgabe zu den maximalen Teilfeldern (Aufgabe 6 in Woche 3). Eine Lösung nutzt eine rekursive Funktion MaxTeilfeld(j), die die Summe eines maximalen Teilfelds ausrechnet, das in j endet. Das heißt, die Funktion liefert unter den Teilfeldern $A[0..j]$, $A[1..j]$, …, $A[j..j]$ die maximale Summe. Ohne auf deine Notizen zu schauen, stelle zunächst einen einfachen rekursiven Algorithmus für diese Funktion auf. Wende dann dynamische Programmierung an, um den Algorithmus effizient zu machen. Hier ist noch einmal der Text aus der Aufgabenstellung:

Sei $A \in \mathbb{Z}^n$ als ein Feld $A[0, \ldots, n-1]$ gespeichert. Ein Teilfeld von A ist ein genau dann ein *maximales Teilfeld* $A[i..j]$ mit $0 \leq i \leq j \leq n-1$, wenn die Summe $A[i] + A[i+1] + \cdots + A[j]$ maximal über alle möglichen Teilfelder ist. Beschreibe und analysiere einen Algorithmus, der mithilfe von dynamischer Programmierung die Summe eines maximalen Teilfelds von A in Zeit $O(n)$ berechnet.

Aufgabe 13.6 (RNA). *Ribonucleic acid* (RNA) ist eine lange Kette von Millionen Nukleotiden von vier verschiedenen Typen: Adenin (A), Cytosin (C), Guanin (G), und Uracil (U). Die *Sequenz* eines RNA-Moleküls ist eine Zeichenkette $b[1 \ldots n]$, wobei jedes Zeichen $b[u] \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$ zu einem Nukleotid korrespondiert. Manchmal mutiert ein Virus und die RNA-Sequenz verändert sich leicht. Deine Aufgabe ist es nun, für zwei gegebene RNA-Sequenzen herauszufinden, wie ähnlich diese sich sind; damit kann man Rückschlüsse ziehen darauf, wie ähnlich sich zwei Virusvarianten sind. Aber nicht jede Änderung in der RNA ist gleich wahrscheinlich! Wir haben nun folgende (fiktive) Ähnlichkeitsmatrix $M[a, b]$ gegeben, die für jede Punktmutation (=Substitution) angibt, wie wahrscheinlich diese ist:

	A	G	C	U
A	10	−1	−3	−4
G	−1	7	−5	−3
C	−3	−5	9	0
U	−4	−3	0	8

Je größer die Zahl ist, desto wahrscheinlicher soll die entsprechende Substitution sein. Zum Beispiel hat eine Substitution von C auf G einen Ähnlichkeitwert von -5, aber ein korrektes Alignment von G mit G hat einen Ähnlichkeitswert von 7. Außerdem definieren wir noch einen Wert $d = -5$, der den Ähnlichkeitswert einer Einfügen oder Löschen Operation auf −5 setzt. Der Ähnlichkeitswert einer Editiersequenz ist die Summe der Ähnlichkeitswerte für jede Spalte.

a) 👍 Was ist der Ähnlichkeitswert dieser Editiersequenz?

```
G A U
C   C G
```

b) 🔑 Wie muss das dynamische Programm für die Edit Distance angepasst werden, um eine Editiersequenz zu berechnen, die einen möglichst großen Ähnlichkeitswert erzeugt?

Aufgabe 13.7 (Sekundärstruktur). *For an English version of this exercise, see [Erickson, page 149].* *Ribonucleic acid* (RNA) ist eine lange Kette von Millionen Nukleotiden oder *Basen* von vier verschiedenen Typen: Adenin (A), Cytosin (C), Guanin (G), und Uracil (U). Die *Sequenz* eines RNA-Moleküls ist eine Zeichenkette $b[1 \ldots n]$, wobei jedes Zeichen $b[i] \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$ zu einer Basis korrespondiert. Zusätzlich zu den chemischen Verbindungen zwischen adjazenten Basen in der Sequenz können auch Wasserstoffbrückenbindungen zwischen manchen Basenpaaren entstehen. Die Menge der gebundenen Basenpaare heißt die *Sekundärstruktur* des RNA-Moleküls.

Wir sagen, dass zwei Basenpaare (i, j) und (i', j') mit $i < j$ und $i' < j'$ sich ***überlappen***, wenn $i < i' < j < j'$ oder $i' < i < j' < j$ gilt. In der Praxis überlappen sich die meisten Basenpaare nicht. Überlappende Basenpaaren bilden sogenannte Pseudoknoten in der Sekundärstruktur, die essenziell für bestimmte Funktionen der RNA sind, aber schwierig vorhergesagt werden können.

Wir wollen jetzt die bestmögliche Sekundärstruktur für eine gegebene RNA-Sequenz berechnen. Wir nehmen das folgende vereinfachte Modell der Sekundärstruktur an:

- Jede Basis kann mit höchstens einer weiteren Basis eine Bindung eingehen.
- Nur `A-U` Paare und `C-G` Paare können gebunden sein.
- Paare der Form $(i, i+1)$ und $(i, i+2)$ können sich nicht binden.
- Gebundene Basenpaare können sich nicht überlappen.

Diese letzte (und am wenigsten realistische) Einschränkung erlaubt es uns, die Sekundärstruktur der RNA als eine Art fetten Baum zu visualisieren, siehe Abbildung 1.

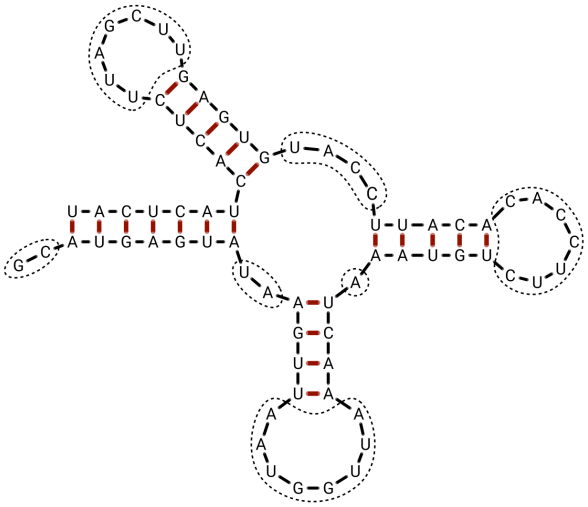

Abbildung 1: Beispiel einer RNA Sekundärstruktur mit 21 gebundenen Basenpaaren, die durch die fetten roten Linien gekennzeichnet sind. Lücken sind durch die gepunkteten Kurven gekennzeichnet. Diese Struktur hat eine Bewertung von $2^2 + 2^2 + 8^2 + 1^2 + 7^2 + 4^2 + 7^2 = 187$.

a) 🔑 Beschreibe und analysiere einen Algorithmus, der die größtmögliche *Anzahl* an gebundenen Basenpaaren in der Sekundärstruktur einer gegebenen RNA-Sequenz berechnet.

b) 🔭 Eine *Lücke* in der Sekundärstruktur ist ein maximaler Teilstring von ungebundenen Basen. Große Lücken sind chemisch instabil, daher sind Sekundärstrukturen mit kleineren Lücken viel wahrscheinlicher. Um diese Präferenz einzubeziehen, definieren wir jetzt die *Bewertung* einer Sekundärstruktur als die Summe der *Quadrate* der Längen aller Lücken; siehe die obige Abbildung. (Diese Bewertung ist fiktional; um die tatsächliche RNA-Struktur vorherzusagen braucht man *deutlich* komplizierte Bewertungsmethoden.)
Beschreibe und analysiere einen Algorithmus, der für eine gegebene RNA-Sequenz die kleinstmögliche Bewertung einer Sekundärstruktur berechnet.