

Algorithmen I

Tutorium 1 - 12. Sitzung

Dennis Felsing

`dennis.felsing@student.kit.edu`
`www.stud.uni-karlsruhe.de/~ubcqr/algo`

2011-07-04



Überblick

1 Dynamische Programmierung

- Idee
- Längste gemeinsame Teilfolge

2 Klausuraufgaben

3 Greedy-Algorithmen

- Allgemeines
- Beispiele

Dynamische Programmierung

Idee

- Problem in Teilprobleme aufteilen
- Teilprobleme lösen und speichern
- Auf bereits gespeicherte Lösung zugreifen statt neu zu lösen
- Bottom-Up oder Top-Down mit Memoisation

Dynamische Programmierung

Idee

- Problem in Teilprobleme aufteilen
- Teilprobleme lösen und speichern
- Auf bereits gespeicherte Lösung zugreifen statt neu zu lösen
- Bottom-Up oder Top-Down mit Memoisation

Notwendige Eigenschaften

- 1 **Optimale Unterstruktur:** Optimale Lösung besteht aus optimalen Lösungen von Teilproblemen
- 2 **Überlappende Unterprobleme:** Rekursiver Algorithmus landet mehrfach bei selbem Problem (ansonsten Teile-und-Herrsche)
- 3 **Rekonstruieren einer optimalen Lösung:** Merken, welche Schritte gemacht wurden

Längste gemeinsame Teilfolge

Gegeben

Organismus 1 mit DNA $S_1 =$ ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

Organismus 2 mit DNA $S_2 =$ GTCGTTCCGGAATGCCGTTGCTCTGTAAA

Längste gemeinsame Teilfolge

Gegeben

Organismus 1 mit DNA $S_1 =$ ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

Organismus 2 mit DNA $S_2 =$ GTCGTTCGGAATGCCGTTGCTCTGTAAA

Gesucht

Ähnlichkeit der Organismen \Leftrightarrow Länge der längsten gemeinsamen Teilfolge. Hier: $S_3 =$ GTCGTCGGAAGCCGGCCGAA

Längste gemeinsame Teilfolge

Gegeben

Organismus 1 mit DNA $S_1 =$ ACCGGTCGAGTGC GCGGAAGCCGGCCGAA

Organismus 2 mit DNA $S_2 =$ GTCGTTCGGAATGCCGTTGCTCTGTAAA

Gesucht

Ähnlichkeit der Organismen \Leftrightarrow Länge der längsten gemeinsamen Teilfolge. Hier: $S_3 =$ GTCGTCGGAAGCCGGCCGAA

Längste gemeinsame Teilfolge

Formalisierung

Sequenz $X = \langle x_1, x_2, \dots, x_m \rangle$ statt DNA-Strang.

$X_i = \langle x_1, x_2, \dots, x_i \rangle$

Theorem

Seien $X = \langle x_1, x_2, \dots, x_m \rangle$ und $Y = \langle y_1, y_2, \dots, y_n \rangle$ Sequenzen und sei $Z = \langle z_1, z_2, \dots, z_m \rangle$ die LGT von X und Y .

- 1 Falls $x_m = y_n$: $z_k = x_m = y_n$ und Z_{k-1} ist eine LGT von X_{m-1} und Y_{n-1}
- 2 Falls $x_m \neq y_n$ und $z_k \neq x_m$: Z ist eine LGT von X_{m-1} und Y
- 3 Falls $x_m \neq y_n$ und $z_k \neq y_n$: Z ist eine LGT von X und Y_{n-1}

Längste gemeinsame Teilfolge

Formalisierung

Sequenz $X = \langle x_1, x_2, \dots, x_m \rangle$ statt DNA-Strang.

$X_i = \langle x_1, x_2, \dots, x_i \rangle$

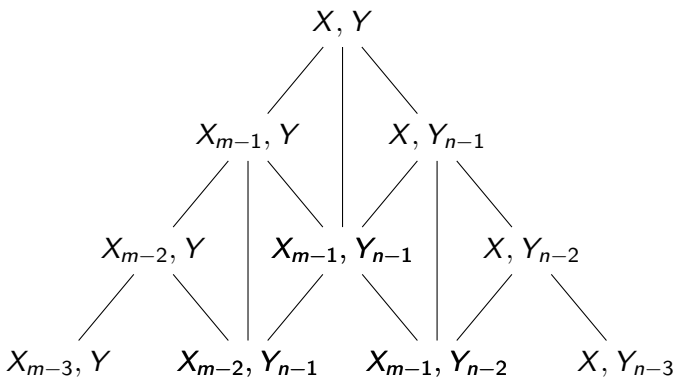
Theorem

Seien $X = \langle x_1, x_2, \dots, x_m \rangle$ und $Y = \langle y_1, y_2, \dots, y_n \rangle$ Sequenzen und sei $Z = \langle z_1, z_2, \dots, z_m \rangle$ die LGT von X und Y .

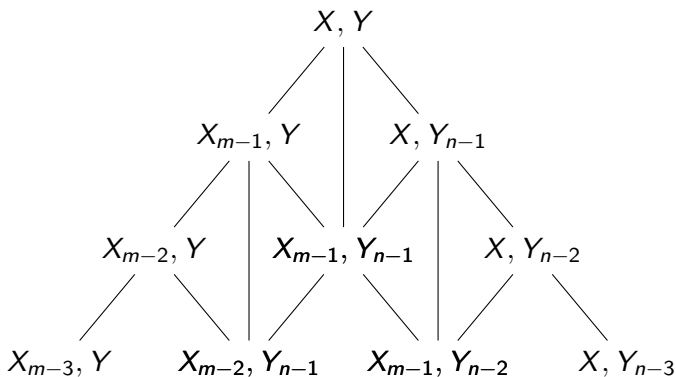
- 1 Falls $x_m = y_n$: $z_k = x_m = y_n$ und Z_{k-1} ist eine LGT von X_{m-1} und Y_{n-1}
- 2 Falls $x_m \neq y_n$ und $z_k \neq x_m$: Z ist eine LGT von X_{m-1} und Y
- 3 Falls $x_m \neq y_n$ und $z_k \neq y_n$: Z ist eine LGT von X und Y_{n-1}

⇒ **Optimale Unterstruktur!**

Längste gemeinsame Teilfolge



Längste gemeinsame Teilfolge



⇒ **Überlappende Unterprobleme!**

Längste gemeinsame Teilfolge

Sei $c[i, j]$ die Länge der LGT der Sequenzen X_i und Y_j :

$$c[i, j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ c[i - 1, j - 1] + 1 & \text{falls } i, j > 0 \text{ und } x_i = y_j \\ \max(c[i, j - 1], c[i - 1, j]) & \text{falls } i, j > 0 \text{ und } x_i \neq y_j \end{cases}$$

LGT-LÄNGE(X, Y)

```
1   $m = X.länge$ 
2   $n = Y.länge$ 
3  Seien  $b[1..m, 1..n]$  und  $c[0..m, 0..n]$  neue Tabellen
4  for  $i = 1$  to  $m$ 
5       $c[i, 0] = 0$ 
6  for  $j = 0$  to  $n$ 
7       $c[0, j] = 0$ 
8  for  $i = 1$  to  $m$ 
9      for  $j = 1$  to  $n$ 
10         if  $x_i == y_j$ 
11              $c[i, j] = c[i - 1, j - 1] + 1$ 
12              $b[i, j] = \nwarrow$ 
13         elseif  $c[i - 1, j] \geq c[i, j - 1]$ 
14              $c[i, j] = c[i - 1, j]$ 
15              $b[i, j] = \uparrow$ 
16         else  $c[i, j] = c[i, j - 1]$ 
17              $b[i, j] = \leftarrow$ 
18  return  $c$  und  $b$ 
```

Längste gemeinsame Teilfolge

PRINT-LGT(b, X, i, j)

1 **if** $i == 0$ oder $j == 0$

2 **return**

3 **if** $b[i, j] == \nwarrow$

4 PRINT-LGT($b, X, i - 1, j - 1$)

5 print x_i

6 **elseif** $b[i, j] == \uparrow$

7 PRINT-LGT($b, X, i - 1, j$)

8 **else** PRINT-LGT($b, X, i, j - 1$)

Definition: Zu einem ungerichteten Graph $G = (V, E)$ ist eine Menge $U \subset V$ eine *unabhängige Menge* genau dann, wenn keine Knoten aus U in G benachbart sind, d.h. $\forall u, v \in U : \{u, v\} \notin E$.

Definition: Zu einem ungerichteten Graph $G = (V, E)$ mit Knotengewichtsfunktion $c : V \rightarrow \mathbb{N}_{>0}$ ist eine Menge $U \subset V$ eine *maximale unabhängige Menge*, wenn U unabhängige Menge ist und unter allen unabhängigen Mengen das größtmögliche Gesamtgewicht hat, d.h. es existiert keine unabhängige Menge $U' \subset V$ mit $\sum_{u \in U'} c(u) > \sum_{u \in U} c(u)$.

Festlegung: Im Folgenden sei $G := v_1 - v_2 - \dots - v_k$ ein Pfad, d.h. G bestehe ausschließlich aus Kanten $\{v_l, v_{l+1}\}$ für $l \in \{1, \dots, k-1\}$. Für $l \leq k$ sei weiter $G_l := v_1 - \dots - v_l$ der Teilpfad von G , der beim Knoten v_1 beginnt und bis einschließlich Knoten v_l geht. G_0 soll als leerer Pfad interpretiert werden.

a. Geben Sie zu dem abgebildeten Pfad die *maximale unabhängige Menge* U an, indem Sie die Knoten markieren, die zu U gehören. Geben Sie außerdem das Gesamtgewicht von U an. Die Knotengewichte stehen in den Knoten.

b. Es sei $m(l)$ das Gesamtgewicht einer *maximalen unabhängigen Menge* U_l auf G_l . Geben Sie eine Rekurrenz für $m(l)$ an, indem Sie auf die Werte der $m(i)$ für $i < l$ zurückgreifen! Begründen Sie kurz!

$$m(0) = 0$$

$$m(1) = c(v_1)$$

$$m(l) =$$

c. Skizzieren Sie einen Algorithmus, der eine *maximale unabhängige Menge* U auf einem Pfad $G = v_1 - \dots - v_k$ mit Knotengewichtsfunktion c **berechnet** und **ausgibt**. Der Algorithmus darf die Laufzeit $O(k)$ nicht überschreiten.

d. Skizzieren Sie kurz einen Linearzeit-Algorithmus, der eine *maximale unabhängige Menge* auf einem ungerichteten *Baum* mit Knotengewichtsfunktion c **berechnet** und **ausgibt**.

b. Es sei $m(l)$ das Gesamtgewicht einer *maximalen unabhängigen Menge* U_l auf G_l . Geben Sie eine Rekurrenz für $m(l)$ an, indem Sie auf die Werte der $m(i)$ für $i < l$ zurückgreifen! Begründen Sie kurz!

$$m(0) = 0$$

$$m(1) = c(v_1)$$

$$m(l) = \max\{m(l-1), m(l-2) + c(v_l)\}$$

c. Skizzieren Sie einen Algorithmus, der eine *maximale unabhängige Menge* U auf einem Pfad $G = v_1 - \dots - v_k$ mit Knotengewichtsfunktion c **berechnet** und **ausgibt**. Der Algorithmus darf die Laufzeit $O(k)$ nicht überschreiten.

d. Skizzieren Sie kurz einen Linearzeit-Algorithmus, der eine *maximale unabhängige Menge* auf einem ungerichteten *Baum* mit Knotengewichtsfunktion c **berechnet** und **ausgibt**.

Greedy-Algorithmen

Allgemeines

- Idee: Wähle lokal optimale Entscheidung
- Liefert häufig nicht-optimale Lösung
- Bisher gesehen: MST, Dijkstra

Algorithmus von Kruskal

MST-KRUSKAL(G, w)

```
1   $A = \emptyset$ 
2  for each  $v \in G.V$ 
3      MAKE-SET( $v$ )
4  sortiere Kanten aus  $G.E^*$ 
5  for each  $(u, v) \in G.E^*$ 
6      if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )
7           $A = A \cup \{(u, v)\}$ 
8          UNION( $u, v$ )
9  return  $A$ 
```

*: in nichtfallender Reihenfolge nach Gewicht w

Laufzeit: $O(|E| \log |V|)$

Algorithmus von Prim

MST-PRIM(G, w, r)

```
1  for each  $u \in G.V$ 
2       $u.schlüssel = \infty$ 
3       $u.\pi = NIL$ 
4   $r.schlüssel = 0$ 
5   $Q = G.V$ 
6  while  $Q \neq \emptyset$ 
7       $u = \text{EXTRACT-MIN}(Q)$ 
8      for each  $v \in G.Adj[u]$ 
9          if  $v \in Q$  und  $w(u, v) < v.schlüssel$ 
10              $v.\pi = u$ 
11              $v.schlüssel = w(u, v)$ 
```

Laufzeit: $O(|E| \log |V|)$

Dijkstra-Algorithmus

Voraussetzung: nichtnegative Kantengewichten

DIJKSTRA(G, w, s)

```
1 INITIALIZE_SINGLE_SOURCE( $G, s$ )
2  $S = \emptyset$ 
3  $Q = G.V$ 
4 while  $Q \neq \emptyset$ 
5      $u = \text{EXTRACT-MIN}(Q)$ 
6      $S = S \cup \{u\}$ 
7     for each  $v \in G.Adj[u]$ 
8         RELAX( $u, v, w$ )
```

Laufzeit:

- Mit binärem Min-Heap: $O((|V| + |E|) \log |V|)$
- Mit Fibonacci-Heap: $O(|V| \log |V| + |E|)$

Übersicht

1 Dynamische Programmierung

- Idee
- Längste gemeinsame Teilfolge

2 Klausuraufgaben

3 Greedy-Algorithmen

- Allgemeines
- Beispiele

