Algorithmus von Rabin und Karp

Wähle eine Hashfunktion, so daß alle

$$h(u[i \dots i+m-1])$$

in $O(m+|u|)$ Schritten berechnet werden können.

Zum Beispiel:

$$h(a_1 \dots a_n) = \left(\sum_{i=1}^{n} q^i a_i \right) \bmod p,$$

wobei p und q große Primzahlen sind.

Es gilt

$$h(a_2 \dots a_{n+1}) = \left(h(a_1 \dots a_n)/q - a_1 + q^n a_{n+1} \right) \bmod p.$$

Übersicht

Editdistanz

Gegeben seien zwei Zeichenketten u und v.

Wir dürfen u auf drei Arten ändern:

1. Ersetzen: Ersetze ein Symbol in u durch ein anderes
2. Löschen: Entferne ein Symbol aus u (Länge wird kleiner)
3. Einfügen: Füge ein Symbol an beliebiger Stelle in u ein (Länge wird größer)

Frage: Wieviele solche Operationen sind mindestens nötig, um u zu v zu verwandeln.

Anwendung: Wie ähnlich sind die beiden Zeichenketten (z.B. DNA-Strings)?

Editdistanz

Lösung durch dynamisches Programmieren.

GACGTCAGCTTACGTACGATCATTTGACTACG

GACGTCAGCTACAGTAGATCATTTGACTACG

Übersicht

Übersicht

Divide-and-Conquer

- Teilen der Eingabe in möglichst gleich grosse Teile
- Rekursives Lösen (Beherrschen) der Teilprobleme
- Zusammenfügen der Teillösungen zur Gesamtlösung

Übersicht

Dynamisches Programmieren

- Optimale Lösungen werden aus optimalen Teillösungen zusammengesetzt.
- Eine Tabelle von Teillösungen wird bottom-up aufgebaut.
- Sonderfall Memoization

0/1-Knapsack

Gegeben sei ein Rucksack mit einer beschränkten Kapazität und verschiedene Gegenstände. Jeder Gegenstand hat eine gegebene Größe und einen gegebenen Wert. Es gibt einen Zielwert. Frage: Können wir Gegenstände auswählen, die in den Rucksack hineinpassen und deren zusammengezählte Werte den Zielwert überschreiten?

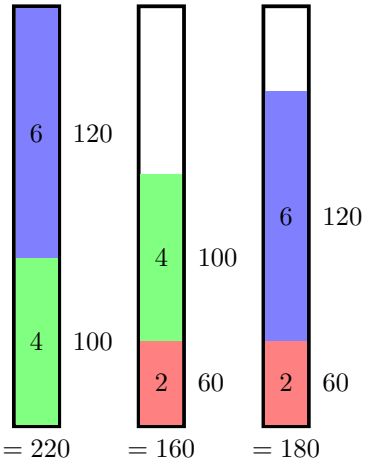
Formaler:

Definition

Eingabe: Positive Integer $(c_1, \ldots, c_n)$, $(v_1, \ldots, v_n)$, und C
Ausgabe: Der Maximalwert v, so daß es eine Menge $I \subseteq \{1, \ldots, n\}$ gibt mit

- $\sum_{i \in I} c_i \leq C$
- $\sum_{i \in I} v_i = v$

Knapsack



0/1-Knapsack

0/1-Knapsack kann durch *brute force* gelöst werden, indem alle Untermengen $I \subseteq \{1, \ldots, n\}$ aufgezählt werden und beide Bedingungen überprüft werden.

Es gibt allerdings 2^n solche Untermengen. Zu langsam!

Können wir dieses Problem durch dynamisches Programmieren lösen?

Ja, indem alle Unterprobleme mit unterer Kapazität zuerst gelöst werden.

0/1-Knapsack

Betrachte den folgenden Algorithmus:

Algorithmus

procedure Knapsack :
for i $= 1, ..., n$ **do**
  for k $= 0, ..., C$ **do**
    best$[0, k] := 0$;
    best$[i, k] :=$ best$[i - 1, k]$;
    if k $\geq$ c_i **and** best$[i, k] <$ v_i $+$ best$[i - 1, k -$ c_i$]$
    then best$[i, k] :=$ v_i $+$ best$[i - 1, k -$ c_i$]$ **fi**
  od
od

$best[i, k]$ ist der höchste Wert, den man mit Gegenständen aus der Menge $\{1, \ldots, i\}$ erhalten kann, die zusammen Kapazität k haben.

Laufzeit

Die Laufzeit ist $O(Cn)$.

Wir nennen einen solchen Algorythmus pseudo-polynomiell.

Die Laufzeit ist nicht polynomiell in der Eingabelänge, aber polynomiell in der Eingabelänge und der Größe der Zahlen in der Eingabe.

Falls C klein ist, dann ist dieser Ansatz viel schneller als alle Untermengen durchzuprobieren.