Zusammenfassung:

das allgemeine TSP — nicht mal $O(2^n)$-approximierbar

Spezialfall:
das metrische TSP — $\frac{3}{2}$-approximierbar; nicht $\frac{220}{219}$-approximierbar

noch spezieller Fall:
das euklidsche TSP — $(1+\varepsilon)$-approximierbar; Arora's PTAS

d.) Das BIN PACKING Problem

Eingabe: n Objekte mit Gewichten $g_1, g_2 \dots g_n$
$(0 \leq g_i \leq 1)$

Ausgabe: Verteile die Objekte in eine minimale Anzahl von Behälter (Bins) mit Kapazität 1

ähnlich zum SCHEDULING, nur hier ist die Anzahl der Behälter ("Maschinen") zu optimieren, und wir können die Kapazität = 1 der Behälter nicht erweitern.

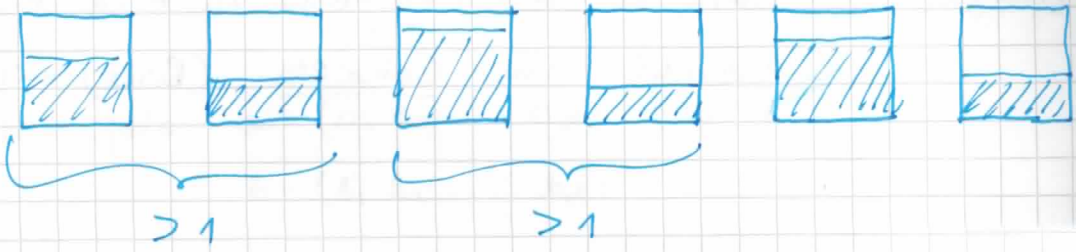
Greedy Algorithmen für BIN PACKING

1. Next-Fit

- setze $B := 1$ (Anzahl der geöffneten Behälter)
- FOR $i = 1$ TO n DO
  falls möglich, füge g_i in Behälter B ein;
  sonst $B := B+1$ und füge g_i in den neu geöffneten Behälter ein;

Behauptung: Next Fit benutzt höchstens $2 \cdot OPT$ Behälter.
(sogar $\leq 2 \cdot OPT - 1$)

Warum?



Je zwei aufeinanderfolgende Behälter enthalten Gesamtgewicht echt größer als 1. Deshalb:

Für B gerade und $G = \sum_{i=1}^{n} g_i$ gilt
(Opt. Anzahl Bins)

(Bin Kapazität) $\rightarrow 1 \cdot OPT \geq G > \frac{B \cdot 1}{2} \Rightarrow OPT \geq \frac{B}{2} + 1$

Für B ungerade gilt

$$OPT \geq G > \frac{B-1}{2} \Rightarrow OPT \geq \frac{B-1}{2} + 1$$

Somit: $B \leq 2 \cdot OPT - 1$ in beiden Fällen

(da die optimale Anzahl der Behälter OPT ganzzahlig ist). □

2. First Fit

$\lfloor 1.7\, OPT \rfloor$ = (approx)

- setze $B = 1$
- FOR $i = 1$ TO n DO

  füge g_i in den ersten Behälter ein wo es reinpasst, wenn es in keinen geöffneten Behälter passt, dann öffne einen neuen Behälter $B := B+1$ und füge g_i ein.

Die ersten beiden waren online Algorithmen:
wir fügen Objekt i in die Behälter, ohne Objekte $i+1, i+2, \ldots, n$ zu kennen.
Erreichen wir bessere Approximation mit offline Strategien — wenn wir die ganze Instanz am Anfang schon kennen, und z.B. sortieren dürfen?

3. First Fit Decreasing (FFD)

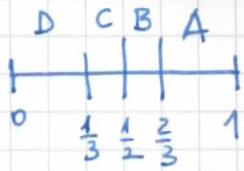
- sortiere die Objekte nach absteigendem Gewicht
- wende First Fit für die sortierte Eingabe an

$$g_1 \geq g_2 \geq g_3 \geq \ldots \geq g_n$$

Theorem: FFD benutzt maximal $\frac{3}{2} OPT + 1$ Behälter.

G 32.

Beweis:



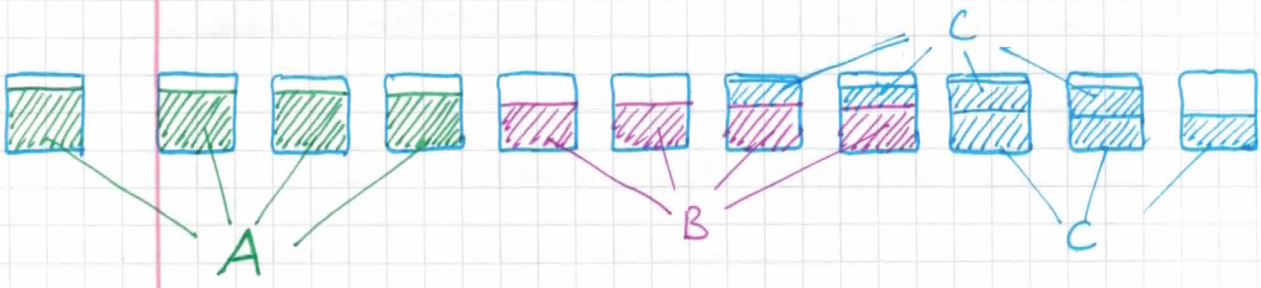
$A = \{ i \mid \frac{2}{3} < g_i \}$ (die Objekte mit Gewicht $> \frac{2}{3}$)

$B = \{ i \mid \frac{1}{2} < g_i \leq \frac{2}{3} \}$

$C = \{ i \mid \frac{1}{3} < g_i \leq \frac{1}{2} \}$

$D = \{ i \mid g_i \leq \frac{1}{3} \}$

Wie geht FFD mit den Objektmengen $A \cup B \cup C$ um?



FFD ist optimal auf den Objekten (nur) in $A \cup B \cup C$

(FFD ist äquivalent mit BFD über $A \cup B \cup C$ →)

Warum?

- Objekte in A und B brauchen in jedem Fall neue Behälter
- manche C-Objekte passen über ein B-Objekt
- C-Objekte die über kein (noch) alleinstehendes B-Objekt passen, passen (genau) zu zweit in einen Behälter

Also: wir wollen so viele C-Objekte über B-Objekte packen wie nur möglich:

FFD ist ~~die bestmögliche~~ eine optimale Methode C-Objekte mit B-Objekte zu matchen: es kann (durch ein Vertausch-Argument) angenommen werden, dass in OPT das größte, zu B-passende C-Objekt in der kleinstmöglichen Lücke ist, und man kann

iterativ (oder mit Induktion) weiter argumentieren. □

– D-Objekte haben Größe $< \frac{1}{3}$

Fall 1: Wenn für D-Objekte keine weitere Behälter geöffnet werden, dann ist die Verteilung der Objekte weiterhin optimal $FFD(I) \leq OPT(I)$

Fall 2: Wenn weitere Behälter für D-Objekte geöffnet wurden, dann sind alle Behälter bis auf den letzten bis über $\frac{2}{3}$ voll. Deshalb

$$OPT(I) \geq \sum_{i=1}^{n} g_i > (FFD(I) - 1)\frac{2}{3}$$

$$\frac{3}{2} OPT(I) + 1 > FFD(I)$$ □

Theorem: FFD benutzt sogar höchstens $\frac{11}{9} OPT + 2$ Behälter.

(ohne Beweis)

4. Best-Fit Decreasing (mindestens so gut wie FFD)

– sortiere die Objekte nach absteigendem Gewicht

$$g_1 \geq g_2 \geq g_3 \geq \ldots \geq g_n$$

– FOR i=1 TO n DO

füge Objekt i in den Behälter ein wo es noch möglich aber am knappsten ist (dessen verbleibende Restkapazität nach dem Einfügen minimal ist)

Nichtapproximierbarkeit des BIN-PACKING Problems

Theorem: BIN-PACKING besitzt keinen (effizienten) c-approximativen Algorithmus für $c < \frac{3}{2}$ (angenommen $P \neq NP$)

(Überlegungen:

Im BIN-PACKING die Anzahl der benutzten Behälter ist zu minimieren.

- Ist es für viele, oder für wenige Behälter (in OPT) schwieriger, eine $< \frac{3}{2}$-Approximation zu erreichen?
- Was bedeutet für BIN-PACKING eine besser als $\frac{3}{2}$ Approximation?
  - wenn 200 Behälter ausreichen
    $\Rightarrow$ der Alg findet < 300
  - wenn 6 — " — reichen
    $\Rightarrow$ er findet $< 9 \leq 8$
  - wenn 4 — " — reichen
    $\Rightarrow$ er findet $< 6 \leq 5$
  - wenn 2 reichen
    $\Rightarrow$ er findet $< 3 = 2$ Behälter

Das letzte Problem ist NP-schwer: es ist nämlich "äquivalent" mit dem PARTITION Problem (fast))

PARTITION: für gegebene Zahlen $x_1 x_2 \dots x_n$ entscheide ob sie in zwei Mengen partitioniert werden können, s.d. beide Mengen die Summe

$$\frac{\sum_{i=1}^{n} x_i}{2}$$

haben.

→ wir konstruieren eine Instanz für BIN-PACKING: wir normalisieren die Zahlen (teilen durch $\frac{\sum_{i=1}^{n} x_i}{2}$) so dass die neuen Zahlen die Summe $= 2$ haben: sei $g_i = \frac{2 \cdot x_i}{\sum_{i=1}^{n} x_i}$ $(i = 1, 2, \dots, n)$

→ Falls $x_1 \dots x_n$ eine JA-Instanz für PARTITION ist, dann passen die Gewichte in 2 Behälter, und ein $\left(\frac{3}{2} - \varepsilon\right)$-approximativer Algorithmus für BIN-PACKING sollte sie in weniger als 3, also in 2 Behälter verteilen können.

→ Falls $x_1 \dots x_n$ eine NEIN-Instanz für PARTITION ist, dann gibt der $\left(\frac{3}{2} - \varepsilon\right)$-approximativer Algorithmus auch mindestens 3 Behälter als Lösung für $g_1 g_2 \dots g_n$ aus (weil $OPT \geq 3$).

→ Somit könnte ein effizienter $\left(\frac{3}{2} - \varepsilon\right)$-approximativer Algorithmus für BIN-PACKING beliebige Instanz des PARTITION Problems effizient entscheiden. → geht nicht falls $P \neq NP$ □

Insbesondere gibt es auch kein PTAS für BIN-PACKING

G.36. Ist das alles was wir sagen können?

Beachte:

- Dieses Problem verhält sich anders als z.B. skalierbare Probleme wie SCHEDULING oder min-SPANNBAUM:
  jeder Algorithmus der nicht-optimal ist braucht mindestens OPT+1 Behälter für irgendeine Instanz (und sogar schon mit OPT=2 gret dies)
- die untere Schranke $\frac{3}{2}$ sagt deshalb sehr wenig vom Problem aus (vielmehr nur von Instanzen mit OPT=2)
- Gibt es noch Hoffnung auf einen fast-optimalen Algorithmus? Er kann nicht fast-optimal werden was den Approximationsfaktor – also die multiplikative Güte betrifft. Wie könnte er in einem anderen Sinne fast optimal sein?
  → es könnte noch sein, dass ein effizienter Algorithmus immer eine Lösung mit OPT+1 Bins ausgibt...

Ob ein effizienter Algorithmus mit additiver Güte 1 ($ALG \leq OPT+1$) existiert, ist nicht bekannt. (aber auch nicht ausgeschlossen). Es gibt aber für jedes $\varepsilon > 0$ einen Algorithmus in Zeit poly(n) der $(1+\varepsilon) \cdot OPT + 1$ Behälter braucht. Eine solche Familie von Algorithmen $(A_\varepsilon)_{\varepsilon>0}$ (d.h. mit irgendeiner additiven Konstante) nennt man

APTAS (Asymptotisches PTAS) für ein Minimierungsproblem

weil der Approximationsfaktor gegen $(1+\varepsilon)$ geht

als $OPT \to \infty$

Vorbereitung:

Behauptung: BIN-PACKING ist in polynomieller Zeit lösbar, falls die Anzahl der möglichen verschiedenen Gewichts-Typen eine Konstante G ist, und alle Gewichte größer als eine Konstante $\delta > 0$ sind.
(G ist also die Anzahl der verschiedenen Gewichts-Typen.)

Beispiel: – Nennen wir das speziellere Problem, eingeschränkt auf Instanzen mit höchstens $3 = G$ verschiedene Gewichtstypen, wobei diese Gewichte alle größer als $\frac{1}{10} = \delta$ sind, BIN-PACKING-$(3, \frac{1}{10})$

(Eine Eingabe von BIN-PACKING-$(3, \frac{1}{10})$ könnte z.B. lauter Objekte mit dem Gewicht 0,11 oder $\frac{1}{7}$ oder $\frac{1}{11}$ haben.)

– **Wir brauchen:** einen Algorithmus, der für alle solche Eingaben Laufzeit $\leq q(n)$ hat für irgendein fixiertes Polynom $q(n)$. Achtung! 3 und $\frac{1}{10}$ sind für dieses Problem Konstanten, und dürfen in $q(n)$ im Exponenten erscheinen.

– **Idee:** Wir können alle möglichen Bepackungen ausprobieren, und die Bepackung mit den wenigsten Behältern ausgeben.

Sei $D = \frac{1}{\delta} = 10$; die Anzahl der möglichen Bepackungen ist polynomiell in n, aber exponentiell in $D = 10$, und doppelt-exponentiell in $G = 3$.

Schritt 1: Wir geben eine grobe obere Schätzung für die Anzahl (höchstens) aller möglichen Bepackungen <u>eines</u> Behälters.

Wir ordnen hierfür jeder Bepackung einen Vektor von 3 Zahlen zu:

G38.

Da die Gewichte größer als $\frac{1}{10}$ sind, passen von jedem Gewicht (und auch insgesamt) maximal 9 Stücke in einen Bin.

Jede Bepackung eines Behälters kann eindeutig durch einen solchen Vektor repräsentiert werden, wobei die 3 Koordinaten den 3 Gewichts-Typen entsprechen:

Gewicht 1 Gewicht 2 Gewicht 3

$(2 , 6 , 0)$

Die Anzahl verschiedener Vektoren mit 3 ganzzahligen Einträgen zwischen 0 und 9 ist $10 \cdot 10 \cdot 10 = 10^3 = D^G$

Wir haben also allgemein $\leq D^G$ Bepackungstypen für einen Bin.

(viele solche Vektoren repräsentieren ungültige Bepackungen, aber das macht nichts.)

[Eine schärfere Schätzung: Die Anzahl der Vektoren mit 3 natürlichen Zahlen, so dass die Summe dieser Einträge maximum 10 ist (der Einfachheit halber rechnen wir mit 10 statt 9 hier), gleicht der Anzahl der Vektoren mit 4 natürlichen Zahlen, so dass die Summe der Einträge genau 10 ist:

Gew. 1. Gew. 2. Gew. 3 Rest

$(2 , 6 , 0 , 2)$

$\updownarrow$

$(1\,1\,0\,1\,1\,1\,1\,1\,1\,0\,0\,1\,1)$

weiter entspricht ein solcher Vektor eindeutig einem 0-1 String aus 10 1-er und 3 0-en. Solche Strings gibt es $\binom{13}{3}$ und allgemein $\binom{D+G}{G}$, auch exponentiell in G.]

Schritt 2: Sei b die Anzahl verschiedener Bepackungen eines Behälters. (Wir haben gesehen: $b \leq D^G$)

Wir haben maximal n Behälter von jedem Bepackungstyp (und auch insgesamt).

Eine grobe Schätzung ergibt analog:

Typ 1 Typ 2 - - - - - - Typ b

($\leq n$ $\leq n$ $\leq n$)

→ es gibt etwa n^b solche Vektoren, und jede Bepackung in maximal n Bins entspricht einem solchen Vektor.

Es gibt also $\leq n^b \leq n^{(D^G)} = n^{\left(\left(\frac{1}{\delta}\right)^G\right)}$ mögliche Bepackungen auszuprobieren. Eine astronomische Zahl, aber immerhin, polynomiell in n.

[Sorgfältigere Schätzung ergibt $\leq \binom{n+b}{b}$ Bepackungen für $b \leq \binom{\frac{1}{\delta}+G}{G}$. Selbst diese Schätzung ergibt ein Polynom mit viel zu hohem Grad.]

Diese und die folgenden Ergebnisse sind rein theoretisch: wir lernen dabei allgemeine Methoden für den Entwurf eines PTAS kennen. Wir benutzen die obigen Abzählungen um ein APTAS für beliebige Eingaben zu entwerfen.

Definition: Ein asymptotisches polynomielles Approximationsschema (APTAS) für ein bestimmtes Minimierungsproblem ist eine Familie von Polynomialzeit-Algorithmen $(A_\varepsilon)_{\varepsilon>0}$ zusammen mit einer Konstante c, so dass jeder Algorithmus A_ε für jede Instanz I eine Lösung mit Wert höchstens $(1+\varepsilon)\,OPT + c$ berechnet.

Ein APTAS für BIN-PACKING

das für beliebig gegebene Gewichte $g_1, g_2, \ldots, g_n \in [0,1]$ und gegebene $\varepsilon > 0$ eine Bepackung mit $(1+2\varepsilon)OPT(I) + 1$ Behältern ausgibt.

Intuitive Beschreibung:

- Wie werden wir erreichen, dass alle Gewichte g_i eine Mindestgröße δ haben?

→ Unser δ wird das gegebene ε sein. Am Anfang legen wir alle Gewichte der Größe $\leq \varepsilon$ beiseite

Am Ende werden wir sie dann mit First-Fit in die Lücken der Bins einfüllen; falls es keine Lücken mehr gibt die groß genug sind, dann sind alle Behälter bis $1-\varepsilon$ gefüllt, und wir dürfen einen neuen Bin öffnen.

- Wie erreichen wir, dass es konstant-viele verschiedene Gewichte gibt?

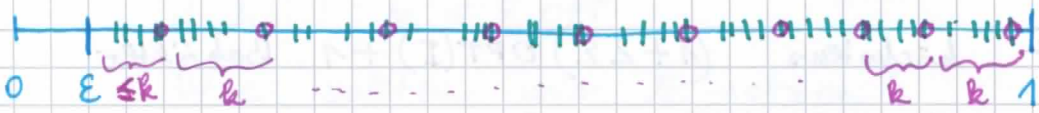
→ Wir werden ähnlich große Gewichte als gleichgroß betrachten: wir bilden konstant-viele Gewichtsklassen

Wir werden die Gewichte g_i aufrunden, so dass wir G Gewichtsklassen erhalten (für ein geeignetes G)

Eine optimale Bepackung der aufgerundeten Gewichte wird auch für die echten Gewichte zumindest eine legale Lösung sein.

(In diesem Fall ist unser Ziel, dass jede Gewichtsklasse gleichviele Gewichte (Objekte) enthält. Beim RUCKSACK Problem werden wir eine andere Methode für Runden sehen.)

Gewicht-Größen in der Eingabe: 1
die gerundeten Gewichte (Gewichtsklassen) o



Wir haben G Gewichtsklassen, und in jeder Gewichtsklasse $k = \lceil \frac{n}{G} \rceil$ Gewichte, also $n \approx G \cdot k$

(In der Abbildung wird angenommen, dass alle Gewichte paarweise unterschiedlich sind. Sonst müssen Gewichte mit Multiplizität gezählt werden (also mehrfach))

<u>Das APTAS für BIN-PACKING</u> ε sei vorgegeben

<u>der Algorithmus A_ε</u>:

- lege Objekte mit Gewicht $\leq \varepsilon$ an die Seite
- zerlege die Menge der restlichen Objekte in G Gewichtsklassen mit je $k = \lceil \frac{n}{G} \rceil$ Objekten (und eine Klasse mit $\leq k$)
- runde jedes Gewicht auf: setze ihn auf das größte Gewicht seiner Klasse
- somit gibt es nur noch G verschiedene Gewichte; löse BIN-PACKING <u>exakt</u> für die <u>gerundeten</u> Gewichte in Polynomieller Laufzeit $O\left(n^{\left(\frac{1}{\varepsilon}\right)^G}\right)$ und teile die echten Objekte genauso zu den Behältern.
- Schließlich füge die kleinen Objekte mit Gewicht $\leq \varepsilon$ mit First-Fit Heuristik in die Behälter (und, wenn nötig, in neue Behälter) ein.

Theorem: Der Algorithmus A_ε für BIN-PACKING benötigt höchstens $(1+2\varepsilon)\,OPT(I)+1$ Behälter bei geeigneter Wahl von G.

Beweis:

Sei $OPT = OPT(I)$ die minimale Anzahl von Behältern für eine beliebig fixierte Eingabe I für BIN-PACKING.

1. Wir nehmen zuerst an, dass es gar keine kleinen Objekte $\leq \varepsilon$ gibt in der Eingabe I.

→ die aufgerundeten Gewichte, mit Ausnahme der höchsten Klasse passen in $OPT(I)$ Behälter.*

→ die k aufgerundeten Gewichte der höchsten Klasse passen in k Behälter

⇒ alle aufgerundeten Gewichte passen in $OPT(I)+k$ Behälter

→ A_ε ist optimal für die gerundeten Gewichte, und verwendet deshalb $\leq OPT(I)+k$ Behälter

→ Wenn $k < \varepsilon \cdot OPT$, dann hat A_ε $\leq (1+\varepsilon)OPT$ Bins geöffnet, falls es keine kleinen Objekte gibt.

→ Wir brauchen also, dass $OPT + k \leq (1+\varepsilon) OPT$,

d.h. $k \leq \varepsilon \cdot OPT$

Wie sollten wir k setzen?

→ Wir brauchen dazu eine Abschätzung von OPT;

Wir wissen nur, dass $OPT \geq n \cdot \varepsilon$, da

$$OPT \cdot 1 \geq \sum_{i=1}^{n} g_i \geq n \cdot \varepsilon$$

weil wir n Objekte der Größe $> \varepsilon$ haben.

Wenn wir k so setzen, dass $k \leq \varepsilon^2 \cdot n$ gilt,

dann $OPT + k \leq OPT + \varepsilon^2 n \leq OPT + \varepsilon \cdot OPT = (1+\varepsilon) OPT$

→ Setzen wir also $k := \lfloor n \cdot \varepsilon^2 \rfloor$

und $G = \left\lfloor \frac{n}{k} \right\rfloor \approx \frac{n}{n \cdot \varepsilon^2} = \frac{1}{\varepsilon^2}$

→ Die Laufzeit wird $O\left(n^{\left(\frac{1}{\varepsilon}^6\right)}\right) = O\left(n^{\left(\frac{1}{\varepsilon}^{\frac{1}{\varepsilon^2}}\right)}\right)$.

(wobei n die Anzahl der Objekte mit Gewicht $> \varepsilon$ ist)

(* Betrachte eine optimale Bepackung der ungerundeten Objekte in OPT(I) Behälter. In dieser Bepackung ersetze die k Objekte | in der höchsten Gewichtsklasse mit den aufgerundeten ○ der zweithöchsten Gewichtsklasse; die k Objekte | in der zweiten Gewichtsklasse mit den k aufgerundeten ○ aus der dritten Gewichtsklasse, u.s.w. - - -

Somit reichen OPT(I) Behälter für alle aufgerundeten Gewichte, die nicht in der höchsten Gewichtsklasse sind, und insgesamt reichen $OPT(I) + k$ Behälter für A_ε (ohne kleine Objekte $\leq \varepsilon$).)

2. Wenn es in der Instanz auch kleine Objekte mit Gewicht $\leq \varepsilon$ gibt

→ der Algorithmus für die großen Objekte ($> \varepsilon$) gibt eine Bepackung in maximal

$$(1+\varepsilon)\,OPT(\text{Große}) \leq (1+\varepsilon)\,OPT(I) \text{ Bins aus}$$

→ Die kleinen Objekte ($\leq \varepsilon$) werden mit First Fit anschließend in die Behälter gefüllt.

→ Wenn keine weiteren Behälter für die kleinen Objekte geöffnet werden, dann gilt $(1+\varepsilon)\,OPT(I)$. ~~dann hat jeder Bin bis auf den letzten höher als $1-\varepsilon$ Gewicht~~

→ Wenn weitere Behälter geöffnet werden, dann hat jeder Bin bis auf den letzten Gesamtgewicht mehr als $1-\varepsilon$. Es gilt:

$$\left(A_\varepsilon(I)-1\right)\cdot(1-\varepsilon) < \sum_{i=1}^{n} g_i \leq OPT(I)$$

$$A_\varepsilon(I) < \frac{1}{1-\varepsilon}\,OPT(I)+1 < (1+2\varepsilon)\,OPT(I)+1$$

falls $\varepsilon < \frac{1}{2}$.

□

DYNAMISCHE PROGRAMMIERUNG

INTERVALL-SCHEDULING II.

(gewichtet)

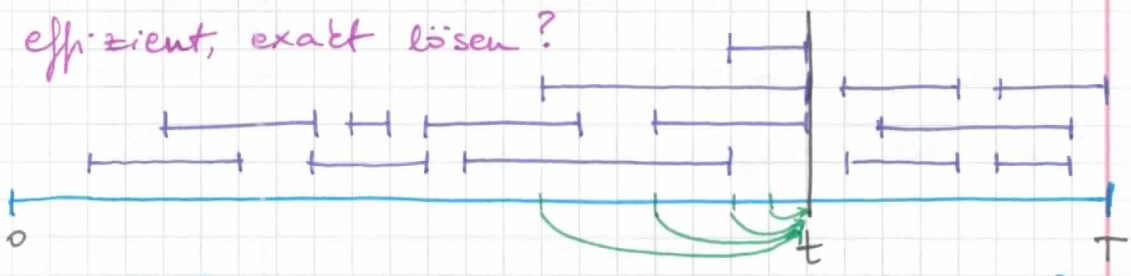
Eingabe: Aufgaben $A_1\ A_2 \ldots A_i \ldots A_n$ mit ganzzahligen (nicht wichtig) Startpunkten $s_1\ s_2 \ldots s_i \ldots s_n$

und Endpunkten $e_1\ e_2 \ldots e_i \ldots e_n$

die Länge von Aufgabe i ist $l_i = e_i - s_i$

Ausgabe: ein Schedule der Aufgaben auf einem Prozessor, ohne Überlappen, mit maximaler Gesamtlänge der ausgeführten Aufgaben

Das Problem mit maximaler Anzahl der ausgeführten Aufgaben haben wir mit einem greedy Algorithmus gelöst. Wie kann man diese Variante des Problems effizient, exakt lösen?



- sei $[0,T]$ das ganze Zeitintervall $(T = e_{max})$
- Bezeichne $L(t)$ die Gesamtlänge in einem optimalen-Schedule-auf-dem-Intervall-$[0,t]$ für jede $t = 1,2,3,\ldots,T$
- $L(0) = 0$ ist trivial
- $L(T)$ ergibt den optimalen Zielwert für das ganze Intervall
- $L(t)$ kann man berechnen, wenn man $L(t-j)$ für alle (oder: bestimmte) $j \geq 1$ kennt:

DP 2.

- setze $L(0) = 0$

- FOR $t = 1$ to T DO

$$L(t) = \max \{ L(t-1), \max \{ l_i + L(s_i) \mid \text{über alle } i \text{ so dass } e_i = t \} \}$$

der beste Zielwert wenn keine Aufgabe in t endet in der Lösung

der beste Zielwert so dass eine Aufgabe k genau in t endet $e_k = t$

1. Wir haben eine Rekursionsgleichung für $L(t)$

2. Wir berechnen alle Werte $L(0), L(1), L(2), \ldots, L(T)$, die Optima für die Teilprobleme von $[0, t]$, und speichern sie als Einträge einer "Tabelle". Wir benutzen diese Werte immer wieder für die Optimierung größerer Teilprobleme.

3. Wir berechnen also das Optimum $L(T)$ 'bottom-up'.

4. Wir können dann eine optimale Lösung (Schedule) 'top-down' konstruieren, mit Hilfe von zusätzlicher Information gespeichert in der bottom-up Phase

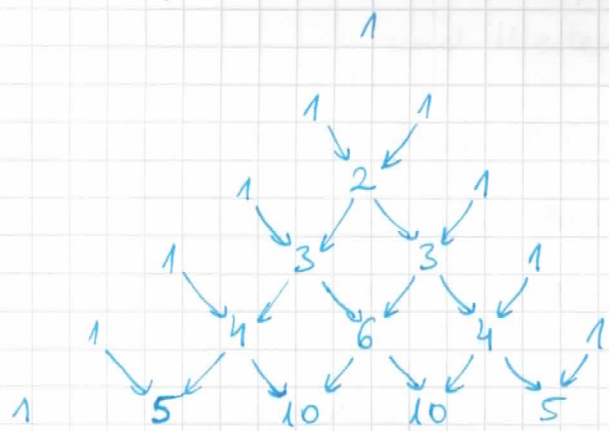
↳ d.h. welche / keine A_i mit $e_i = t$ für $L(t)$ verwendet wird.

Bemerkungen: - Es reicht für jede $t = s_i$ und $t = e_i$ den $L(t)$ zu berechnen (und $L(t-1)$ entsprechend ersetzen in der Gleichung). Dies geht auch wenn e_i und s_i nicht ganzzahlig sind. Die Laufzeit ist somit $O(n)$ (Warum??)

- der selbe Algorithmus löst das Problem mit beliebigen Gewichten w_i (statt l_i) für die Aufgaben A_i

Schulbeispiel für Dynamische Programmierung:

Das Pascalsche Dreieck



$\binom{0}{0}$

$\binom{1}{0}$ $\binom{1}{1}$

$\binom{2}{0}$ $\binom{2}{1}$ $\binom{2}{2}$

$\binom{3}{0}$ $\binom{3}{1}$ $\binom{3}{2}$ $\binom{3}{3}$

$\binom{4}{0}$ $\binom{4}{1}$ $\binom{4}{2}$ $\binom{4}{3}$ $\binom{4}{4}$

$\binom{5}{0}$ $\binom{5}{1}$ $\binom{5}{2}$ $\binom{5}{3}$ $\binom{5}{4}$ $\binom{5}{5}$

- man könnte die (dumme) Idee haben den Binomialkoeffizienten $\frac{n!}{k!(n-k)!} = \binom{n}{k}$ mit Hilfe der Rekursionsgleichung:

$$\binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

zu berechnen. Wenn er schon so rechnen möchte,

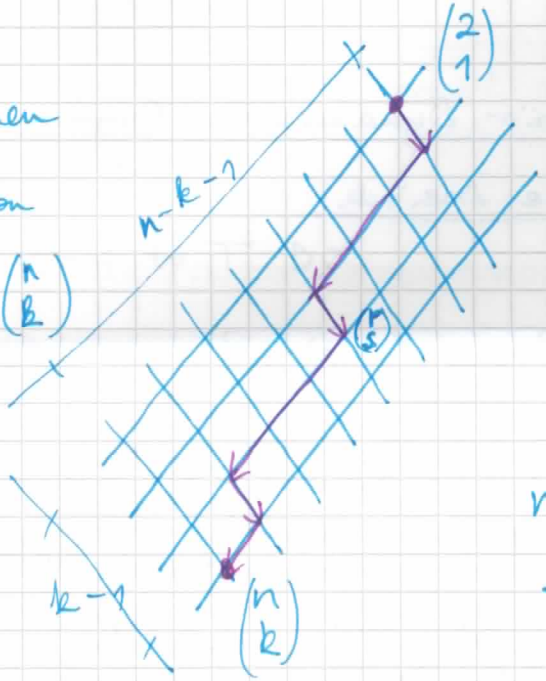
sollte er einen rekursiven Algorithmus, oder einen Algorithmus mit Dynamischer Programmierung verwenden?

- Falls der Algorithmus rekursiv implementiert wird, werden die Werte der früheren Teilprobleme $\binom{n-i}{k-j}$ nicht gespeichert, und jedes mal wenn gebraucht, neu berechnet.

Beispiel:
Ungefähr wie oft wird $\binom{2}{1} = \binom{1}{0} + \binom{1}{1}$ berechnet, während der einzige Berechnung von $\binom{n}{k}$ mit der rekursiven Methode?

DP4.

für jeden solchen Weg im Gitter von $\binom{2}{1}$ nach unten zum $\binom{n}{k}$ (bzw. von $\binom{n}{k}$ durch rekursive Rufe nach $\binom{2}{1}$))



und solche Wege gibt es $\binom{n-2}{k-1} \sim \binom{n}{k}$

- ein Algorithmus mit dynamischer Programmierung würde stattdessen mit den einfachsten Teilproblemen anfangen, und die berechneten Werte für spätere Nutzung in der Tabelle (d.h. im Pascalschen-Dreieck) speichern. ("bottom-up" entspricht in unserer Darstellung von oben nach unten). Er würde die $\sim (n-k) \cdot k$ nötigen Tabellenwerte $\binom{r}{s}$ (siehe oben) jeweils einmal mit der Rekursionsgleichung $\binom{r}{s} = \binom{r-1}{s-1} + \binom{r-1}{s}$ berechnen.

(Allgemein gilt:

'top' = die ganze (ursprüngliche) Problem-Instanz

'bottom' = die elementaren Teilprobleme)

DAS RUCKSACK Problem

Eingabe: eine Gewichtsschranke G, n Objekte mit Gewichten $g_1, g_2, \ldots, g_n$ $(g_i \leq G)$ und Werten $w_1, w_2, \ldots, w_n$

Ausgabe: finde eine Auswahl von Objekten mit maximalem Gesamtwert, so dass die Gewichtsschranke nicht überschritten wird.