Algorithmendesign
Lösungen zu Übungsblatt 2
WS 13/14

Gruppe 2
Max Bannach

Markus Richter (614027)

Aufgabe 2.1 Nichtoptimalität

Teilaufgabe 1)

Die Problemstellung besteht darin, dass einige Leute eine Wanderung entlang des Äppalachian Trail"machen wollen. Gegeben vieler im Vorfeld identifizierter Haltepunkte zum Campen wollen sie aus Angst nur bei Tageslicht wandern. Den Nachteilen dieser Einschränkung wollen sie mit folgendem Algorithmus entgegenwirken, der angeblich in einer minimalen Anzahl an Unterbrechungen resultiert:
Jedes mal wenn sie an einer potentiellen Haltestelle zum Campen angelangt sind überprüfen sie, ob sie es vor Einbruch der Dunkelheit noch zum nächsten Haltepunkt schaffen. Wenn ja, so wandern sie weiter, andernfalls nicht.

Teilaufgabe 2)

Ändert man die Problemstellung dahingehend, dass nach Einbruch der Dunkelheit gewandert werden darf, so berechnet der Algorithmus nicht mehr die optimale Lösung.

Beispiel:
Angenommen innerhalb eines Intervalls mit der Distanz d liegen 3 Haltepunkte x_j, x_{j+1}, x_{j+2} sodass gilt: $d - x_j - x_{j+1} - x_{j+2} = 0$. Da die Distanz zwischen den Haltestellen variieren kann, kann es passieren, dass kurz nach Erreichen von x_{j+1}, aber vor dem Erreichen von x_{j+2} die Nacht hereinbricht. Nach dem Algorithmus wird die Gruppe gezwungen an der Stelle x_{j+1} zu rasten. Ein Algorithmus ohne Beschränkung auf die Tageszeit würde aber erst eine Pause bei x_{j+2} berechnen und somit zu einer geringeren Menge an Zwischenstopps führen.

Aufgabe 2.2 Konferenz-Organisation

Der Lösungs-Algorithmus:

if $i \geq maxval$ **then**
  $i \leftarrow 0$
else
  if $i + k \leq maxval$ **then**
    $i \leftarrow i + k$
  end if
end if

```
Algorithmus Konferenz:
EINGABE: Menge R von Jobs r_j mit Laufzeiten ρ(j), Deadlines δ(j) und
    schlechte Bewertungen s(j)
ordne die Jobs absteigend bzgl. ihrer schlechten Bewertung, d. h.
    s(1) ≥ ...s(n):
t := 0;
FOR i = n TO i = 1 DO
  r := 0;
  FOR j = 1 TO n DO
    IF(δ(r_j) <= i)
      THEN r := j;
      Schleife beenden.
  IF(r != 0)
    THEN schedule r_r fuer den Zeitraum [t, t + τ(r)[;
    t := t + τ(r);
    entferne r_r aus R.
  OD.
WHILE R ≠ ∅ DO
  waehle r_i ∈ R;
  schedule r_i fuer den Zeitraum [t, t + τ(i)[;
  t := t + τ(i);
  entferne r_i aus R;
  OD;
```

Der Algorithmus basiert auf n Zeitslots, die wiederum die Deadlines darstellen, wobei n =Anzahl der Jobs. Slot n entspricht Deadline n. Es liegt eine absteigend nach schlechte Bewertung s geordnete Liste vor. Der Algorithmus sucht inkremental in dieser Liste für jeden Slot einen Job, der mit seiner Deadline ausgeführt werden kann. Es kann also zunächst auch zu Lücken kommen, wenn Jobs mit großem s Jobs mit kleinem s verdrängen. Die verdrängen Jobs können ihre Deadline nicht einhalten und werden daher zum Schluss einfach nacheinander ausgeführt.

Korrektheit
Ich zeige die Korrektheit des Algorithmus indem ich zeige, dass das Schedule S keine Leerzeiten und keine Inversion aufweist.
Leerzeiten
Nach Konstruktion des Algorithmus sind alle Slots ab dem ersten nicht leeren Slot belegt und werden nacheinander ausgeführt. Es kommt also zu keinen Leerzeiten.
Inversion
Nach Konstruktion des Algorithmus wird immer der Jobs r_i mit dem größten $s(i)$ ausgeführt. Inversion ist also nicht möglich
$\Longrightarrow$ Der Algorithmus ist korrekt.

Laufzeit
Der Algorithmus sortiert anfangs eine Menge, was also zunächst zu einer Laufzeit von $\mathcal{O}(n \log n)$ führt. Weiterhin besteht der Algorithmus aus einer äußeren und einer inneren Schleife, wobei beide die Größe n haben, dies führt zur Laufzeit $\mathcal{O}(n^2)$. Die while-Schleife wird linear durchlaufen, also $\mathcal{O}(n)$. Insgesamt beträgt die Laufzeit also $\mathcal{O}(n^2)$

Optimalität
Es ist zu zeigen, dass die Anzahl der schlechten Bewertungen bei Anwendung dieses Algorithmus minimal ist.

Aufgabe 2.3 Intervall-Abdeckung

Der Lösungs-Algorithmus:

```
Algorithmus Intervall-Abdeckung
EINGABE: Array A von Intervalen
Ordne das Array der Intervalle A aufsteigend nach dem linken Ende der
    Intervalle, insbesondere soll gelten: wenn left(a_i)=left(a_j) und right
    (b_j)>right(b_i), dann folgt [left(a_i),right(b_i)] auf [left(a_j),right(b_j)].

n:=lenght(A);
B[1] := S[1];
j := 2;
k := 1;
FOR i = 2 TO n DO
  IF right(A[i])>right(A[k])
    THEN B[j]:=A[i];
    j := j + 1;
    k := i;
RETURN B
```

Korrektheit
Der Algorithmus ist nach Konstruktion korrekt, da er zu jeder Eingabe eine korrekte Lösung liefert.

Laufzeit
Zunächst erfolgt eine Sortierung mit einer Laufzeit von $\mathcal{O}(n \log n)$. Anschließend wird eine Schleife mit der Größe n verarbeitet, d. h. mit linearer Laufzeit $\mathcal{O}(n)$. Daraus resultiert insgesamt eine Laufzeit von $\mathcal{O}(n \log n)$.