Max Wisniewski , Alexander Steen

Tutor: Lena Schlipf

Aufgabe 1 Unabhängige Mengen in Pfaden

Sei $G = (V, E)$ ein *Pfad*, mit den Knoten $v_1, v_2, ..., v_n$.
Dazu gehört eine Funktion $w : V \longrightarrow \mathbb{N}^*$, die einem Knoten ein positives ganzzahliges Gewicht zuordnet, im folgenden mit $w(v_i) = w_i$ bezeichnet.

Zur Notation:
Wir werden im folgenden einen Pfad wie folgt darstellen:
$[w_0, w_1, w_2, w_3, ..., w_{n-1}]$.
Dabei sind zwei Knoten benachbert, wenn diese nebeneinander in der Liste stehen. Die Knoten sind ihrer Nummerierung entsprechend aufsteigend angeordnet und in der Liste steht nur der Wert, die die Funktion w ihm zuordnet.

(a) Geben Sie ein Beispiel, dass der Algorithmus (siehe Aufgabenblatt) keine unabhängige Menge maximalen Gewichts liefert.

Lösung:
Betrachten wir den *Pfad* $[1, 9, 10, 9, 1]$.

Nach dem Algorithmus nehmen wir den Größten Wert, hier Knoten 2 mit dem Wert 10, und entfernen ihn und seine Nachbern (Knoten 1 und 3). Es verbleibt der Folgende Graph: $[1], [1]$. Der restliche Graph ist *unabhängig*, bei weiterer Ausführung werden die beiden 1en genommen und nicht mehr gelöscht als diese beiden in zwei Schritten, da sie keine Nachbern mehr besitzen.

Der Algorithmus liefert uns also die Knoten $0, 2, 4$ mit dem gesammt Gewicht 12.
Wie man an diesem Beispiel aber leicht sieht, wäre die maximale unabängige Menge allerdings $1, 3$ mit dem Gewicht 18. Diese beiden Knoten wurden aber als Nachbern eines wenig kleinere Knotens im 1 Schritt allerdings verworfen.

Da wir ein Gegenbeispiel gefunden haben, kann der Algorithmus nicht korrekt sein.
□

(b) Geben Sie ein Beispiel, bei dem der Algorithmus (vgl. Aufgabenzettel) keine unabhängige Menge maximalen Gewichts bestimmt.

Lösung:
Der Algorithmus zerlegt die Menge in zwei unabängeige Menge, die eine hat gerade Indizes, die andere ungerade.
Wir wollen uns eine Menge konstruieren, in der die maximal unabhängige Menge sowohl gerade, als auch ungerade Indizes enthält.

Wir untersuchen $[1, 9, 1, 1, 9, 1]$.
Die Menge der Werte mit geraden Indizes ist hier $\{9, 1, 1\}$ und die mit ungeraden ist $\{1, 1, 9\}$. Beide ergeben ein Gesammtgewicht von 11.
Auch hier sehen wir schnell, dass Element $1, 4$ ein Gesammtgewicht von 18 hat und somit mehr Gewicht hat, als die beiden Menge, deren Maximum den größten Wert unter den unabhängigen Menge haben soll.

Da dies wie an diesem Beispiel gezeigt nicht immer der Fall ist, bestimmt der Algorithmus lieder nicht die unabhängige Menge mit maximalen Gewicht. □

(c) Geben Sie einen effizienten Algorithmus an, der eine unabhängige Menge maximalen Gewichtes in einem Pfad bestimmt. Analysieren Sie Laufzeit und Platzbedarf.

Lösung:

Teillösungen:

Als Teillösung wählen wir $E[j] :=$ Maximum der *unabhängigen Teilmengen*, wenn wir nur die ersten j Knoten betrachten.

Rekursion:

$$\begin{aligned} E[0] &= w(0) \\ E[1] &= max\{w(0), w(1)\} \\ \forall 2 \leq i \leq n : \quad E[i] &= \max\{E[j-2] + w(j),\ E[j-1]\} \end{aligned}$$

Sollten wir nur 1 Element haben nehmen wir dies einfach. Haben wir 2 Nehmen wir das Maximum von beiden. Nehmen wir das Maximum von erstens, wir nehmen das j - te Element müssen dafür aber das daneben verwerfen, oder zweitens wir nehmen es nicht, aber können das daneben wählen.

Ziel ist es $E[n-1]$ zu bestimmen, weil wir dort alle Knoten betrachtet haben.

Korrektheit

Algorithmus

```
if n = 1 then
  Erg := [0];
  return w(0);
Erg1 := [0];
w1 := w(0);
if(w(0) > w(1) then
  Erg2 := [0];
  w2 := w(0);
else
  Erg2 := [1];
  w2 := w(1);

for i := 2 to n-1 do
  if w1 + w(i) > w2 then
    save := w1 +w(i);
    w1 := w2;
    w2 := save;
    ergSave := Erg2;
    Erg2 := Erg1 ++ [i];
    Erg1 := Erg2;
  else
    save := w1;
    w1 := w2;
    w2 := save;
```

```
    ergSave := Erg2;
    Erg2 := Erg1;
    Erg1 := ergSave;
Erg := Erg2;
return w2;
```

Da wir immer nur den letzte Wert (w2) und den vorletzten Wert (w1) für den nächsten Schirtt benötigen, müssen wir auch nur diese beiden speichern. Danach gehen wir wie beim dynamischen Programmieren unserere Parameter von klein nach groß durch.

Laufzeit & Platz

Platzbedarf: $\Theta(1)$, da wir nur 2 Werte und swap speichern müssen. Dies wird durch die Speicherung der liste allerdings relativiert, da wir nun noch maximal $\frac{n}{2}$ Elemente in Erg1, Erg2 mitführen.

Laufzeit: Wir betrachten jeden Knoten von V genau einmal $\Rightarrow \Theta(n)$.

Aufgabe 2 Vorlesungsplanung

In der Vorlesung haben wir einen *Greedy - Algorithmus* gesehen, mit der wir aus einer Menge von Vorlesungen, eine Maximale Menge von Vorlesungen finden können, die alle paarweise *verträglich* sind. Untersuchen Sie, ob der Algorithmus mit einer der folgenden Ordnungen im Sortierschritt immer noch funktionert. Geben Sie Pseudocode an und begründen Sie seine Korrektheit und Laufzeit.

(a) Sortiere aufsteigend nach den Anfangszeiten

Dies ist kein valide Lösung. Nehmen wir das Beispiel
$R = \{I_1, I_2, I_3\}$, wobei $I_1 = (1, 100),\ I_2 = (2, 10),\ I_3 = (12, 20)$.

Wie wir schnell sehen, ist das beste Ergebnis I_2, I_3, diese beiden sind *verträglich*, da sie nach einander liegen. Der Algorithmus würde uns allerdings I_1 liefern, weil es am frühesten anfängt.
Danach würden alle unverträglichen verworfen und damit auch I_2, I_3.

Da wir ein Gegenbeispiel gefunden haben, kann der Algorithmus mit dieser Ordnung nicht korrekt sein.

(b) Sortiere absteigend nach den Anfangszeiten.

(c) Sortiere absteigend nach den Endzeiten.

Dieser Fall verhält sich analog zu *a)*. Nehmen wir die selben Intervalle $R = \{I_1, I_2, I_3\}$, wobei $I_1 = (1, 100),\ I_2 = (2, 10),\ I_3 = (12, 20)$.

Da I_1 auch das letzte Ende hat, wird der Algorithmus dieses Interval zu erst nehme und alle anderen verwerfen, da diese Überlappen. Optimal hier wäre wie gehabt I_2, I_3.

Wir haben wieder ein Gegenbeispiel gefunden, damit kann der Algorithmus nicht korrekt sein.

(d) Sortiere aufsteigend nach der Länge

Diese Ordnung bildet keine valide Lösung.
Nehmen wir als Beispiel $R = \{I_1, I_2, I_3\}$ mit $I_1 = (1, 10)$, $I_2 = (9, 12)$, $I_3 = (11, 20)$.

Der Algorithmus würde uns als kürzestes Interval I_2 liefern. Damit fallen I_1, I_3 aus der Menge, weil sie nicht veträglich sind. Die maximale Anzahl von Intervallen ist aber I_1, I_2, da diese beiden *verträglich* sind.

(e) Sortiere absteigend nach der Länge

Dieser Fall verhält sich analog zu *a), b)*. Nehmen wieder die selben Intervalle $R = \{I_1, I_2, I_3\}$, wobei $I_1 = (1, 100),\ I_2 = (2, 10),\ I_3 = (12, 20)$.

Das längeste Intervall ist I_1. Da dieses I_2, I_3 überdeckt, werden die beiden aus der Menge geschmissen. Damit liefert der Algorithmus I_1. Das maximale Menge von *verträglichen* Intervallen wäre allerdings I_2, I_3.

Wir haben ein Gegenbeispiel gefunden und damit ist der Algorithmus nicht korrekt.

Aufgabe 3 Autobahnfahrt

Gegeben ist eine Menge von Tankstellen, entlang eines Weges (wir setzten an dieser Stelle vorraus, dass Sie nach der Anordnung am Weg sortiert sind). Wir haben ein Auto, dass bei vollem Tank (nach jeder Tankstelle an der Man hält) n Kilometer fahren kann. Wir suchen nun eine Minimale Anzahl von Tankstellen, mit der Wir unser Ziel erreichen können (ohne mit einem leeren Tank liegen zu bleiben).

Lösung: