Karlsruher Institut für Technologie

Tutorium 12: Graphalgorithmen und Optimierungsaufgaben

Holger Ebhart | 8. Juli 2015

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS15

$$
\begin{array}{ll}
\text{Minimize} & 28x_{11} + 84x_{12} + 112x_{13} + 112x_{14} + 60x_{21} + 20x_{22} + 50x_{23} + 50x_{24} \\
& +96x_{31} + 60x_{32} + 24x_{33} + 60x_{34} + 64x_{41} + 40x_{42} + 40x_{43} + 16x_{44} \\
& +50y_1 + 50y_2 + 50y_3 + 50y_4 \\
\text{subject to} & x_{11} + x_{12} + x_{13} + x_{14} = 1 \\
& x_{21} + x_{22} + x_{23} + x_{24} = 1 \\
& x_{31} + x_{32} + x_{33} + x_{34} = 1 \\
& x_{41} + x_{42} + x_{43} + x_{44} = 1 \\
& x_{11} + x_{21} + x_{31} + x_{41} \leq 100y_1 \\
& x_{12} + x_{22} + x_{32} + x_{42} \leq 100y_2 \\
& x_{13} + x_{23} + x_{33} + x_{43} \leq 100y_3 \\
& x_{14} + x_{24} + x_{34} + x_{44} \leq 100y_4 \\
& x_{ij}, y_j \in \{0.1\} \quad i = 1, \ldots, 4, \; j = 1, \ldots, 4
\end{array}
$$

Gliederung

10. Übungsblatt

A2) b)

Zeigen Sie: Ein zusammenhängender ungerichteter Graph ist genau dann bipartit, wenn er keinen Kreis ungerader Länge als Teilgraph enthält.

c)

Geben Sie einen Algorithmus an der in linearer Zeit eine Zerlegung V_1, V_2 von V des Graphen $G = (V, E)$ berechnet, so dass durch V_1, V_2 die Bipartitheit von G gegeben ist bzw. abbricht wenn dies nicht möglich ist.

10. Übungsblatt

Karlsruher Institut für Technologie

A2) b)

Zeigen Sie: Ein zusammenhängender ungerichteter Graph ist genau dann bipartit, wenn er keinen Kreis ungerader Länge als Teilgraph enthält.

c)

Geben Sie einen Algorithmus an der in linearer Zeit eine Zerlegung V_1, V_2 von V des Graphen $G = (V, E)$ berechnet, so dass durch V_1, V_2 die Bipartitheit von G gegeben ist bzw. abbricht wenn dies nicht möglich ist.

10. Übungsblatt


```
function bipartit(G = (V[1 ··· n], E[1 ··· m]))
    (parent,d):= bfs(V[1],G)
    V_1, V_2: Queue of Vertices
    f[1 ··· n]: Array of {0,1}
    for i := 1 to n do
        if d[i] mod 2 = 0 then
            V_1.pushBack(i)
            f[i] = 0
        else
            V_2.pushBack(i)
            f[i] = 1
    for j := 1 to m do
        {u, v} := E[i]
        if f[u] = f[v] then
            return "not bipartit"
    return (V_1, V_2)
```

Aufgabe

Karlsruher Institut für Technologie

Betrachte eine Menge von Währungen C mit einem Umtauschkurs von r_{ij} (man erhält r_{ij} Einheiten von Währung j für eine Einheit von Währung i).
Eine Währungs-Arbitrage ist möglich, wenn es eine Folge von elementaren Umtauschoperationen (Transaktionen) gibt, die mit einer Einheit einer Währung beginnt, und mit mehr als einer Einheit derselben Währung endet.
Beschreibe einen Algorithmus, mit dem man für eine gegebenen Umtauschmatrix bestimmen kann, ob Währungs-Arbitrage möglich ist.
Beweise die Korrektheit des Algorithmus.
Hinweis: $\log(xy) = \log x + \log y$, $\log(1) = 0$

Lösungsskizze

- baue Matrix $R = (r_{ij})_{i,j \in \{1, \cdots, |C|\}}$
- logarithmiere und negiere Einträge $d_{ij} = -\log r_{ij}$ (beliebige Basis $b > 1$)
- führe den Algorithmus von Bellman und Ford für jede Zusammenhangskomponente aus (mit bel. Startknoten)
- existiert ein negativer Kreis, so lässt sich eine Währungs-Arbitrage durchführen

Beweis

Genau dann wenn eine Währungs-Arbitrage möglich ist, haben wir eine Folge von Transaktionen von Währung c_0 nach Währung c_0, also einen Kreis $c_0, c_1 \dots c_k, c_0$, für den gilt:

$$\left(\prod_{i=1}^{k} r_{c_{i-1}c_i}\right) r_{c_k c_0} > 1 \iff \left(\sum_{i=1}^{k} \log r_{c_{i-1}c_i}\right) + \log r_{c_k c_0} > 0 \iff$$

$$\left(\sum_{i=1}^{k} -\log r_{c_{i-1}c_i}\right) - \log r_{c_k c_0} < 0 \iff \left(\sum_{i=1}^{k} d_{c_{i-1}c_i}\right) + d_{c_k c_0} < 0$$

Der letzte Term entspricht aber gerade der Länge des Kreises, und dieser soll negativ sein.

■

Greedy-Algorithmen

Definition

Treffe in jedem Schritt die lokal optimale Entscheidung und nimm diese nicht mehr zurück.

Beispiele:

- Dijkstra
- Kruskal
- Jarník-Prim

Greedy-Algorithmen

Definition

Treffe in jedem Schritt die lokal optimale Entscheidung und nimm diese nicht mehr zurück.

Beispiele:

- Dijkstra
- Kruskal
- Jarník-Prim

Geldwechselproblem

Karlsruher Institut für Technologie

Definition

Es sei ein Betrag b und ein Münzsystem M gegeben.
z.B. $M = \{1, 2, 5, 10, 20, 50, 100, 200\}$
Gesucht wird eine Multimenge $L \subset M$ mit

$$\sum_{m \in M} m = b$$

und $|L|$ minimal.

Aufgaben:

- Gibt es einen Greedy-Algorithmus der das Problem für dieses M löst?
- Funktioniert er immer noch wenn man $M' = M \cup \{4\}$ betrachtet?

Geldwechselproblem

Karlsruher Institut für Technologie

Definition

Es sei ein Betrag b und ein Münzsystem M gegeben.
z.B. $M = \{1, 2, 5, 10, 20, 50, 100, 200\}$
Gesucht wird eine Multimenge $L \subset M$ mit

$$\sum_{m \in M} m = b$$

und $|L|$ minimal.

Aufgaben:

- Gibt es einen Greedy-Algorithmus der das Problem für dieses M löst?
- Funktioniert er immer noch wenn man $M' = M \cup \{4\}$ betrachtet?

Optimalitätsprinzip

Voraussetzung für DP: **Optimalitätsprinzip**

- Optimale Lösungen bestehen aus optimalen Teillösungen
- Optimale Lösungen sind austauschbar (d.h. es ist egal welche optimale Lösung genommen wird)

Idee der DP

Konstruiere die optimale Lösung von unten (bottom-up) aus optimalen Teillösungen. Dabei speichert man die Teillösungen meist extra.

Optimalitätsprinzip

Voraussetzung für DP: **Optimalitätsprinzip**

- Optimale Lösungen bestehen aus optimalen Teillösungen
- Optimale Lösungen sind austauschbar (d.h. es ist egal welche optimale Lösung genommen wird)

Idee der DP

Konstruiere die optimale Lösung von unten (bottom-up) aus optimalen Teillösungen. Dabei speichert man die Teillösungen meist extra.

Dynamische Programmierung

Stabzerlegungsproblem

Es sei ein Stab der Länge n und eine Preisliste $p_i \in \mathbb{R}(i \in \{1, \cdots, n\})$ gegeben.
Gesucht ist nun eine Zerteilung$(z_1, \cdots, z_m)$ des Stabes, sodass $\sum_{k=1}^{m} z_k = n$ gilt und $\sum_{k=1}^{m} p_{z_k}$ maximal wird.

Wir gehen von folgendem Beispiel aus:

Länge i	1	2	3	4	5	6	7	8	9	10
Preis p_i	1	5	8	9	10	17	17	20	24	30

Fragen

Karlsruher Institut für Technologie

- Was wäre die einfachste Art das Problem zu lösen und wie ist die Laufzeit?
- Rekursiv das Maximum aller möglichen Aufteilungen bestimmen. Laufzeit: 2^{n-1}
- Wieso ist dieser Algorithmus ineffizient?
- Teillösungen werden mehrmals berechnet.
- Schneller geht das mit dynamischer Programmierung.
- Nachteil: Es wird mehr Speicherplatz verbraucht.

Fragen

- Was wäre die einfachste Art das Problem zu lösen und wie ist die Laufzeit?
- Rekursiv das Maximum aller möglichen Aufteilungen bestimmen. Laufzeit: 2^{n-1}
- Wieso ist dieser Algorithmus ineffizient?
- Teillösungen werden mehrmals berechnet.
- Schneller geht das mit dynamischer Programmierung.
- Nachteil: Es wird mehr Speicherplatz verbraucht.

Fragen

- Was wäre die einfachste Art das Problem zu lösen und wie ist die Laufzeit?
- Rekursiv das Maximum aller möglichen Aufteilungen bestimmen. Laufzeit: 2^{n-1}
- Wieso ist dieser Algorithmus ineffizient?
- Teillösungen werden mehrmals berechnet.
- Schneller geht das mit dynamischer Programmierung.
- Nachteil: Es wird mehr Speicherplatz verbraucht.

Stabzerlegungsproblem - DP-Lösung

Eine Lösung mit DP könnte so aussehen:

procedure bottomUpDP(p,n)
    r:Array$[0 \cdots n]$ of $\mathbb{N}$
    r[0] := 0
    for j := 1 **to** n **do**
        q := -∞ :$\mathbb{N}$
        for i := 1 **to** j **do**
            q := **max**(q,p_i + r[j-i])
        r[j] := q
return r[n]

Laufzeit?
Warum ist das kein Greedy-Algorithmus?

Aufgabe 1

Im ZKM finden Sie einen alten Spielautomaten, bei dem Sie auf einem matrixförmigen Spielfeld auf den Zellen Diamanten sammeln oder verlieren. Sie müssen eine Spielfigur vom oberen linken Feld zum unteren rechten Feld bewegen und können dabei nur nach rechts oder nach unten laufen, da die anderen Richtungen des alten Joysticks kaputt sind. Auf Feldern mit positiven Zahlen sammeln Sie entsprechend viele Diamanten ein, auf Feldern mit negativen Zahlen verlieren Sie entsprechend viele an einen Dieb.

Entwickeln Sie einen Algorithmus (Pseudocode), der in $\mathcal{O}(nm)$ für ein gegebenes $m \times n$ Spielfeld eine Zugfolge berechnet, bei der Sie am Ende maximal viele Diamanten gehortet haben.

Aufgabe 2

Betrachten Sie folgendes eindimensionales Rucksackproblem: Sie haben eine Liste von n Gegenständen mit Volumina $c_1, \cdots, c_n \in \mathbb{N}$ und Nutzwert $a_1, \cdots, a_n \in \mathbb{N}$. Ihr Rucksack hat eine Kapazität $C \in \mathbb{N}$ und soll so gepackt werden, dass die Summe der Nutzwerte der mitgenommenen Gegenstände maximal ist. Formal ist eine Menge $I \subseteq \{1, \cdots, n\}$ gesucht, die

$$\left\{ \sum_{i \in I} \middle| \sum_{i \in I} c_i \leq C \right\}$$

maximiert.

a) Wiederholen Sie das in der Vorlesung vorgestellte dynamische Programm, dass in $\mathcal{O}(nC)$ eine solche optimale Menge I berechnet.

b) Sei nun Opt der maximale Nutzen eines zulässigen Rucksacks. Entwickeln Sie ein dynamisches Programm, das in $\mathcal{O}(n \cdot Opt)$ eine optimale Lösungsmenge I berechnet.

Aufgabe 2

Betrachten Sie folgendes eindimensionales Rucksackproblem: Sie haben eine Liste von n Gegenständen mit Volumina $c_1, \cdots, c_n \in \mathbb{N}$ und Nutzwert $a_1, \cdots, a_n \in \mathbb{N}$. Ihr Rucksack hat eine Kapazität $C \in \mathbb{N}$ und soll so gepackt werden, dass die Summe der Nutzwerte der mitgenommenen Gegenstände maximal ist. Formal ist eine Menge $I \subseteq \{1, \cdots, n\}$ gesucht, die

$$\left\{ \sum_{i \in I} \middle| \sum_{i \in I} c_i \leq C \right\}$$

maximiert.

a) Wiederholen Sie das in der Vorlesung vorgestellte dynamische Programm, dass in $\mathcal{O}(nC)$ eine solche optimale Menge I berechnet.

b) Sei nun Opt der maximale Nutzen eines zulässigen Rucksacks. Entwickeln Sie ein dynamisches Programm, das in $\mathcal{O}(n \cdot Opt)$ eine optimale Lösungsmenge I berechnet.

Aufgabe 3

Editierdistanz
Die Editierdistanz zweier Wörter ist die minimale Anzahl an Einfüge-, Lösch- und Ersetz-Operationen um das erste Wort in das zweite zu überführen.

Beispiel:
Editierdistanz zwischen Tier und Tor ist 2: Tier $\rightarrow$ Toer $\rightarrow$ Tor

Aufgabe Gib einen Algorithmus an der die Editierdistanz zweier Wörter der Länge m und n in $\mathcal{O}(mn)$ berechnet und dabei einen Speicherbrauch in $\mathcal{O}(mn)$ hat.

Aufgabe 3

Karlsruher Institut für Technologie

Editierdistanz
Die Editierdistanz zweier Wörter ist die minimale Anzahl an Einfüge-, Lösch- und Ersetz-Operationen um das erste Wort in das zweite zu überführen.

Beispiel:
Editierdistanz zwischen Tier und Tor ist 2: Tier $\rightarrow$ Toer $\rightarrow$ Tor

Aufgabe Gib einen Algorithmus an der die Editierdistanz zweier Wörter der Länge m und n in $\mathcal{O}(mn)$ berechnet und dabei einen Speicherbrauch in $\mathcal{O}(mn)$ hat.

Lösung Aufgabe 3

Berechne die Matrix $D \in \mathbb{N}^{m+1 \times n+1}$ rekursiv von oben nach unten. Die Lösung findet sich in $D[m, n]$.

$u : Array[1 \cdots m]$
$v : Array[1 \cdots n]$
$\forall i \in \{1, \cdots, m\} \forall j \in \{1, \cdots, n\} : D[0,0] := 0; D[i,0] := i; D[0,j] = j$

$$D[i,j] = \min \begin{cases} D[i-1, j-1] + 0 & fallsu[i] = v[i] \\ D[i-1, j-1] + 1 & Ersetzung \\ D[i, j-1] + 1 & Einfuegung \\ D[i-1, j] + 1 & Loeschung \end{cases}$$

Laufzeit: $\mathcal{O}(nm)$
Speicherplatzverbrauch: $\mathcal{O}(nm)$

Lösung Aufgabe 3

Berechne die Matrix $D \in \mathbb{N}^{m+1 \times n+1}$ rekursiv von oben nach unten. Die Lösung findet sich in $D[m, n]$.

$u : Array[1 \cdots m]$
$v : Array[1 \cdots n]$
$\forall i \in \{1, \cdots, m\} \forall j \in \{1, \cdots, n\} : D[0, 0] := 0; D[i, 0] := i; D[0, j] = j$

$$D[i, j] = \min \begin{cases} D[i-1, j-1] + 0 & \mathit{falls u[i] = v[i]} \\ D[i-1, j-1] + 1 & \mathit{Ersetzung} \\ D[i, j-1] + 1 & \mathit{Einfuegung} \\ D[i-1, j] + 1 & \mathit{Loeschung} \end{cases}$$

Laufzeit: $\mathcal{O}(nm)$
Speicherplatzverbrauch: $\mathcal{O}(nm)$

Lösung Aufgabe 3

Können wir an dem Algorithmus noch die Laufzeit oder den Speicherplatzverbrauch verbessern?

Ja, wir können den Speicherplatzverbrauch senken indem wir immer aus der vorhergehenden Zeile die Nächste berechnen. Dazu brauchen wir nur 2 Zeilen.

Speicherplatzverbrauch vorher: $\mathcal{O}(nm)$
Nun: $\mathcal{O}(\min\{m, n\})$

Lösung Aufgabe 3

Können wir an dem Algorithmus noch die Laufzeit oder den Speicherplatzverbrauch verbessern?

Ja, wir können den Speicherplatzverbrauch senken indem wir immer aus der vorhergehenden Zeile die Nächste berechnen. Dazu brauchen wir nur 2 Zeilen.

Speicherplatzverbrauch vorher: $\mathcal{O}(nm)$
Nun: $\mathcal{O}(\min\{m, n\})$

Lösung Aufgabe 3

Können wir an dem Algorithmus noch die Laufzeit oder den Speicherplatzverbrauch verbessern?

Ja, wir können den Speicherplatzverbrauch senken indem wir immer aus der vorhergehenden Zeile die Nächste berechnen. Dazu brauchen wir nur 2 Zeilen.

Speicherplatzverbrauch vorher: $\mathcal{O}(nm)$
Nun: $\mathcal{O}(\min\{m, n\})$

Zu Übungsblatt 11

- Führe auf dem Graphen an der Tafel den Bellman-Ford Algorithmus aus. Der Startknoten sei A.
- Führe auf dem Graphen an der Tafel den Jarník-Prim Algorithmus aus. Der Startknoten sei 1.
- Führe auf dem selben Graphen nun den Algorithmus von Kruskal aus.

Zu Übungsblatt 11

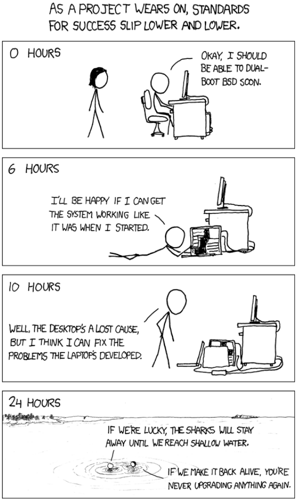
- Führe auf dem Graphen an der Tafel den Bellman-Ford Algorithmus aus. Der Startknoten sei A.
- Führe auf dem Graphen an der Tafel den Jarník-Prim Algorithmus aus. Der Startknoten sei 1.
- Führe auf dem selben Graphen nun den Algorithmus von Kruskal aus.

Thema letztes Tutorium???

Vorschläge zu Themen die wir im letzten Tutorium nochmals wiederholen sollen?

Gerne auch per Mail an *holger.ebhart@ira.uka.de*

Vielen Dank für eure Aufmerksamkeit!
Bis zum nächsten Mal.



stackoverflow.com