Algorithmendesign
Lösungen zu Übungsblatt 5
WS 13/14

Gruppe 1
Stefan Wien 607526

Aufgabe 5.1

1.

Ohne Beschränkung der Allgemeinheit gehen wir von einem topologisch sortierten Graphen mit n Knoten aus. Dies ist möglich, da nur azyklische Graphen betrachtet werden. Die Anzahl von Wegen zwischen zwei Knoten lässt sich maximieren, indem ein Graph konstruiert wird, der alle zulässigen Kanten enthält, also die Kantenmenge $E = \{(v_a, v_b) \mid a < b\}$ ist.
Betrachtet man nun zwei Knoten v_a und v_b mit $a < b$, gibt es einen Pfad von v_a nach v_b für jede Teilmenge von $\{v_{a+1}, ..., v_{b-1}\}$, da man von jedem Knoten einen Knoten mit höherem Index direkt erreichen kann.
Offensichtlich gibt es also die maximale Anzahl Pfade zwischen den Knoten v_1 und v_n, da die zuvor angesprochene Menge $\{v_{a+1}, ..., v_{b-1}\}$ in diesem Fall maximal ist. Sie enthält $n-2$ Elemente, besitzt somit 2^{n-2} Teilmengen. Also gibt es auch 2^{n-2} Wege von v_1 nach v_n.
$\Longrightarrow$ Es kann maximal 2^{n-2} Wege zwischen zwei Knoten geben.

2.

Sei $\Delta_{i,j}^{(k)}$ die Anzahl an Wegen zwischen den Knoten v_i und v_j, die außer am Anfang und am Ende nur Knoten $v_{k'}$ mit $k' \leq k$ verwenden.

Data: gerichteter azyklischer Graph $G = (V, E)$
Result: Anzahl an Wegen zwischen allen Knotenpaaren
for $1 \leq i, j \leq |V|$ **do**
  if $(v_i, v_j) \in E$ **then**
    $\Delta_{i,j}^{(0)} = 1$;
  else
    $\Delta_{i,j}^{(0)} = 0$;
  end
end
for $k = 1$ to $|V|$ **do**
  for $1 \leq i, j \leq |V|$ **do**
    $\Delta_{i,j}^{(k)} = \Delta_{i,j}^{(k-1)} + \Delta_{i,k}^{(k-1)} * \Delta_{k,j}^{(k-1)}$
  end
end

($\Delta_{i,j}^{(|V|)}$ enthält nun die Anzahl an Wegen zwischen v_i und v_j)

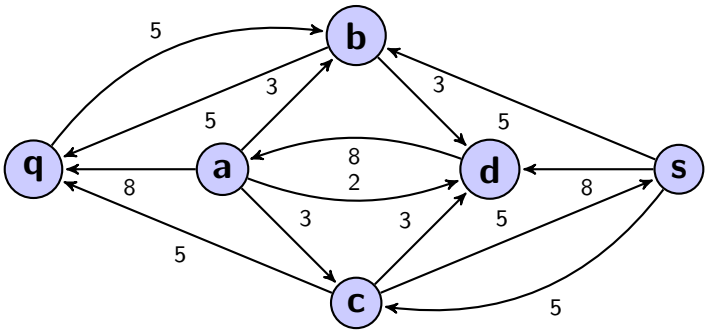
Zuerst wird die Anzahl aller direkten Verbindungen zwischen allen Knotenpaaren bestimmt. Diese kann offensichtlich nur entweder 1 (wenn eine Kante zwischen diesen Knoten existiert) oder 0 sein (wenn keine Kante zwischen diesen Knoten existiert.
Anschließend können die restlichen Wege iterativ mithilfe folgender Regel bestimmt werden:
Möchte man $\Delta_{i,j}^{(k)}$ bestimmen und kennt die Werte von $\Delta_{i,j}^{(k-1)}$ für alle i, j, so enthält $\Delta_{i,j}^{(k)}$ zum einen alle Wege von $\Delta_{i,j}^{(k-1)}$ (alle Wege die bereits ohne den Knoten v_k möglich sind,

und zum anderen $\Delta_{i,k}^{(k-1)} * \Delta_{k,j}^{(k-1)}$ (alle Wege die erst von v_i nach v_k und von dort aus nach v_j führen).
Die Korrektheit des Algorithmus ergibt sich daraus, dass seine Konstruktion genau dieser Strategie entspricht.
Die Initialisierung der Werte für $k = 0$ geschieht in quadratischer Zeit, da für jedes Paar von i, j eine Wertzuweisung stattfindet.
Der Zweite Teil beinhaltet jedoch Wertzuweisungen für alle Tripel von k, i, j, wobei alle einen Wertebereich von 1 bis $|V|$ haben. Somit beträgt die Laufzeit des Algorithmus letztlich $O(|V|^3)$.

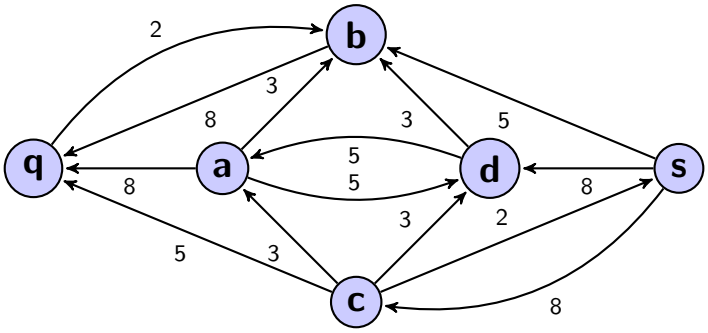
Aufgabe 5.2

1.

Wir konstruieren den Restgraphen und finden den zunehmenden Weg q, b, d, a, c, s mit einer Kapazität von 3.



Der Restgraph ändert sich wie folgt:



Nun lässt sich kein Weg mehr von q nach s finden, wodurch der maximale Fluss erreicht wurde. Er beträgt 21.

2.

Mithilfe des Theorems 5.1 aus der Vorlesung, lässt sich bei bekanntem Restgraphen mit maximalem Fluss ein Mincut auf folgende Weise konstruieren:
Wir wählen einen $q|s$-Schnitt so, dass Q alle Knoten enthält, die im Restgraphen von q aus erreichbar sind und S alle übrigen Knoten.
In diesem Fall ergibt sich $Q = q, b$ und $S = a, c, d, s$. Die Kapazität über diesen Schnitt beträgt 21, ist also gleich dem in 1. bestimmten maximalen Fluss. Folglich ist dieser Schnitt ein Mincut.

3.

Um zusätzlich einen Mincut zu erhalten, muss der Algorithmus, wie bereits in 2. erwähnt, alle von q erreichbaren Knoten in eine Menge Q und alle restlichen Knoten in eine Menge S einsortieren. Dazu werden im zuvor erstellten (finalen) Restgraphen rekursiv alle erreichbaren Knoten von q ermittelt und diese zusammen mit q der Menge Q, und alle anderen entsprechend der Menge S zugewiesen. der $q|s$-Schnitt bestehend aus Q und S stellt nun einen Mincut dar.

Aufgabe 5.3

1.

Nein, wählen wir zum Beispiel die Folge $X = 2, 1$, so gibt es zwei maximale monoton steigende Teilfolgen (nachfolgend mmsT genannt) der Länge 1: Zum einen $Y_1 = 2$ und zum anderen $Y_2 = 1$.

2.

Nachfolgend bezeichnet Y_{max} die mmsT von X.
$|Y_{max}| = 0$ für alle X mit $|X| = 0$
Enthält X keine Elemente, so muss auch Y_{max} leer sein.

$|Y_{max}| = 1$ für alle X mit $|X| = 1$
Eine Folge mit lediglich einem Element ist immer monoton steigend, wodurch in diesem Fall $X = Y_{max}$ ist.

Seien $X = x_1, ..., x_n$ und $X_k = x_1, ..., x_{n-k}$ mit $k \in \{1, ..., n\}$
Sei nun die Länge einer mmsT für alle X_k bekannt.
Dann lässt sich die Länge einer mmsT Y_{max} für X folgendermaßen bestimmen:

1. Fall (x_n kommt nicht in Y_{max} vor):
In diesem Fall entspricht $|Y_{max}|$ der Länge der mmsT von X_1.

2. Fall (x_n kommt in Y_{max} vor):
In diesem Fall muss muss die Länge einer mmsT von X_k gefunden werden, sodass ein Anfügen von x_n die Monotonie der Folge weiterhin gewährleistet. Das bedeutet, wir müssen das kleinste k finden, für das $x_n \geq x_{n-k}$ gilt und die Länge der entsprechenden mmsT um 1 erhöhen.
Formal: $|Y_{max}|$ = Länge der mmsT von $X_k + 1$ für das entsprechend bestimmte k.

Es werden die Werte beider Fälle bestimmt. Der größere Wert entspricht nun der Länge der mmsT von X.

Der nachfolgende Algorithmus ist genau nach diesem Vorgehen konstruiert und ist somit korrekt.

```
Data: Folge X
Result: Länge der maximalen monoton steigenden Teilfolge von X
L(0) = 0;
L(1) = 1;
for i = 2 to n do
    j = i − 1;
    while x_j > x_i do
        j − −;
    end
    L(i) = max{L(i − 1), L(j) + 1};
end
return L(|X|);
```

Die Initialisierung hat einen konstanten Aufwand. Die for-Schleife wird $n-1$ mal durchlaufen. Im worst-Case muss in jedem Durchlauf dieser Schleife nun die maximale Anzahl $(2, 3, 4, ..., n)$ an vergleichen durchgeführt werden. Der Algorithmus besitzt somit eine quadratische Laufzeit.