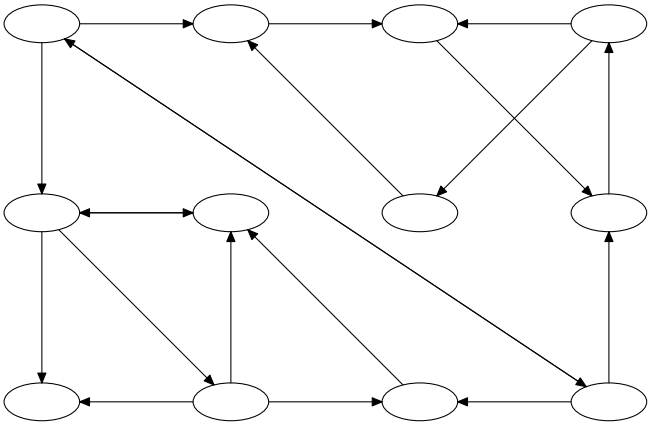
RWTH Aachen **SoSe 2011**
Lehrgebiet Theoretische Informatik
Emmes—Langer—Reidl—Rossmanith 2. August 2011

Klausur zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe 1 (10 Punkte)

a) Führen Sie auf dem folgenden Graphen eine Tiefensuche durch und tragen Sie die *discovery*- und *finish*-Zeiten ein. Geben Sie außerdem den Typ jeder Kante an, indem sie alle Kanten mit B, V, R oder Q markieren. Die Buchstaben stehen für Baumkante, Vorwärtskante, Rückwärtskante und Querkante. Beachten Sie, daß es Doppelkanten gibt, die in Wirklichkeit aus zwei Kanten bestehen. Beide müssen beschriftet werden.



b) Wie kann man nach einer durchgeführten Tiefensuche einfach feststellen, ob der entsprechende gerichtete Graph kreisfrei ist?

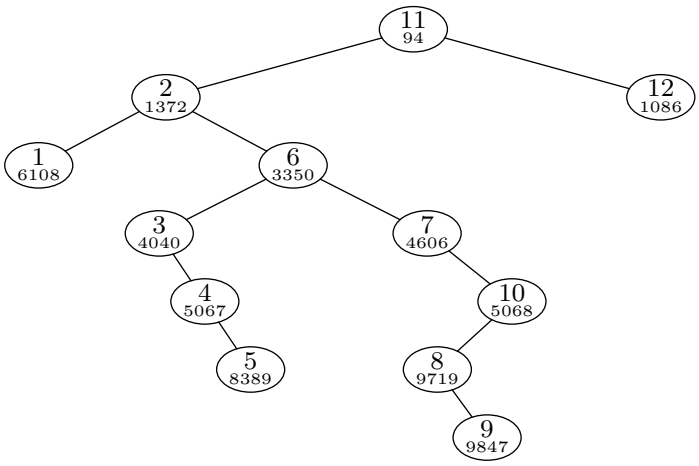
c) Falls sich herausstellt, daß der Graph tatsächlich kreisfrei ist, können wir die Knoten topologisch sortieren. Erklären Sie genau, wie man dies in $O(n)$ Schritten bewerkstelligen kann, falls der Graph n Knoten besitzt und wir die in der Tiefensuche gefundene Information verwenden können.

Aufgabe 2 (10 Punkte)

a) Beweisen Sie, daß ein Treap eindeutig ist, wenn alle Schlüssel und alle Prioritäten verschieden sind.

b) Geben Sie ein Gegenbeispiel für den Fall an, daß zwar alle Schlüssel verschieden sind, aber gleiche Prioritäten erlaubt sind. Genauer: Geben sie zwei verschiedene Treaps an, welche die gleichen Schlüssel enthalten.

c) Erklären Sie, wie ein Element aus einem Treap gelöscht wird. Wie sieht nebenstehender Treap aus, nachdem die 6 gelöscht wurde?



Aufgabe 3 (10 Punkte)

Die Produktionsaufträge $J_1, J_2, \ldots, J_n$ eines Unternehmens benötigen verschiedene Maschinen $M_1, M_2, \ldots, M_m$, wobei ein Auftrag auch mehrere Maschinen belegen kann. Ein Auftrag bringt natürlich einen gewissen Geldbetrag ein. Die Maschinen können entweder gekauft werden—diese Kosten entstehen dann nur einmal und jede weitere Nutzung ist kostenfrei—oder gemietet werden. Letzteres kostet einen vom Auftrag abhängigen Betrag (dieser Betrag variiert also von Auftrag zu Auftrag!).

Für solch ein Szenario mit gegebenen Aufträgen, Maschinen und Kosten soll eine gewinnmaximierende Menge von Aufträgen mit entsprechender Zuteilung von Maschinen berechnet werden. Beschreiben Sie ein allgemeines Verfahren, um dieses Problem möglichst effizient zu lösen. Begründen Sie, warum Ihr Verfahren korrekt funktioniert und geben Sie eine Schranke für die Laufzeit an, die nicht schlechter als $O(n^2m^2(n+m))$ sein sollte. Beantworten Sie die Frage, ob dies asymptotisch schneller ist, als stur alle 2^{n+m} Möglichkeiten zu probieren, welche Maschinen zu kaufen und welche Aufträge anzunehmen sind?

Benutzen Sie Ihr Verfahren, um eine optimale Lösung für das folgende Szenario zu berechnen. Die dritte Tabelle enthält die Mietkosten der Maschinen für die jeweiligen Aufträge. Eine leere Zelle bedeutet, daß diese Maschine für diesen Auftrag nicht benötigt wird. Zum Beispiel kostet Auftrag J_1 auf der Maschine M_1 30 Geldeinheiten (vorausgesetzt, M_1 wurde nicht gekauft).

Auftrag	Zahlung
J_1	80
J_2	80
J_3	120

Maschine	Kaufpreis
M_1	60
M_2	80
M_3	100
M_4	30

	M_1	M_2	M_3	M_4
J_1	30		50	
J_2		60	22	
J_3	30	30	30	30

Aufgabe 4 (10 Punkte)

Nebenstehendes rekursives Unterprogramm, das die Elemente $a[l], \ldots, a[r]$ in einem Array a in die richtige Reihenfolge sortieren soll, kann nicht direkt verwendet werden, um das Array $a[0], \ldots, a[N]$ mittels des Aufrufs *quicksort*$(0, N)$ zu sortieren.

Das Problem träte beispielsweise auf, wenn wir ein Array $a[0], \ldots, a[4]$ mit dem Inhalt $4, 2, 3, 2, 1$ nehmen und das Programm mittels *quicksort*$(0, 4)$ aufrufen.

```
procedure quicksort(L, R) :
  if R ≤ L then return fi;
  p := a[L]; l := L; r := R + 1;
  do
    do l := l + 1 while a[l] < p;
    do r := r − 1 while p < a[r];
    vertausche a[l] und a[r];
  while l < r;
  a[L] := a[l]; a[l] := a[r]; a[r] := p;
  quicksort(L, r − 1); quicksort(r + 1, R)
```

a) Erklären Sie, welches Problem bei dieser Eingabe auftritt.

b) Schreiben Sie ein Programm *sort*(N) in Pseudocode, Java oder C++, welches mithilfe obigen Unterprogramms das Array $a[0], \ldots, a[N]$ stets korrekt sortiert.

* * * Viel Erfolg! * * *