Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Yann Disser

Prüfung
Datenstrukturen und Algorithmen
D-INFK

9. August 2012

Name, Vorname: _____

Stud.-Nummer: _____

Ich bestätige mit meiner Unterschrift, dass ich diese Prüfung unter regulären Bedingungen ablegen konnte und dass ich die untenstehenden Hinweise gelesen und verstanden habe.

Unterschrift: _____

Hinweise:

- Ausser einem Wörterbuch und Schreibzeug dürfen Sie keine Hilfsmittel verwenden.
- Bitte schreiben Sie Ihre Studierenden-Nummer auf **jedes** Blatt.
- Melden Sie sich bitte **sofort**, wenn Sie sich während der Prüfung in irgendeiner Weise bei der Arbeit gestört fühlen.
- Bitte verwenden Sie für jede Aufgabe ein neues Blatt. Pro Aufgabe kann nur eine Lösung angegeben werden. Ungültige Lösungsversuche müssen klar durchgestrichen werden.
- Bitte schreiben Sie **lesbar** mit blauer oder schwarzer Tinte. Wir werden nur bewerten, was wir lesen können.
- Sie dürfen Algorithmen und Datenstrukturen aus der Vorlesung verwenden, ohne sie noch einmal zu beschreiben. Wenn Sie sie modifizieren, reicht es die Modifikationen zu beschreiben.
- Die Prüfung dauert 120 Minuten.

Viel Erfolg!

Stud.-Nummer: _____

Aufgabe	1	2	3	4	5	Σ
Mögl. Punkte	9	7	9	9	9	43
Σ Punkte						

Aufgabe 1

Hinweise:

1. In dieser Aufgabe sollen Sie **nur die Ergebnisse** angeben. Diese können Sie direkt bei den Aufgaben notieren.
2. Falls Sie jedoch andere Methoden als die der Vorlesung benutzen, müssen Sie diese **kurz** soweit erklären, dass Ihre Ergebnisse verständlich und nachvollziehbar sind.
3. Als Ordnung verwenden wir für Buchstaben die alphabetische Reihenfolge, für Zahlen die aufsteigende Anordnung gemäss ihrer Grösse.

1 P a) Kreuzen Sie bei jeder der folgenden Datenstrukturen an, ob der in ihr verwendete Baum immer logarithmische Höhe hat (balanciert) oder nicht (unbalanciert):

Natürlicher Suchbaum	☐ BALANCIERT	/ ☐ UNBALANCIERT
AVL-Baum	☐ BALANCIERT	/ ☐ UNBALANCIERT
Optimaler Suchbaum	☐ BALANCIERT	/ ☐ UNBALANCIERT
Heap	☐ BALANCIERT	/ ☐ UNBALANCIERT
Intervallbaum	☐ BALANCIERT	/ ☐ UNBALANCIERT
Segmentbaum	☐ BALANCIERT	/ ☐ UNBALANCIERT
Range Tree	☐ BALANCIERT	/ ☐ UNBALANCIERT

1 P b) Führen Sie auf dem gegebenen Array einen Aufteilungsschritt (in-situ, d.h. ohne Hilfsarray) des Sortieralgorithmus Quicksort durch. Benutzen Sie als Pivot das am linken Ende stehende Element im Array.

32	13	19	8	69	34	77	52	17	49	37	26	5
1	2	3	4	5	6	7	8	9	10	11	12	13

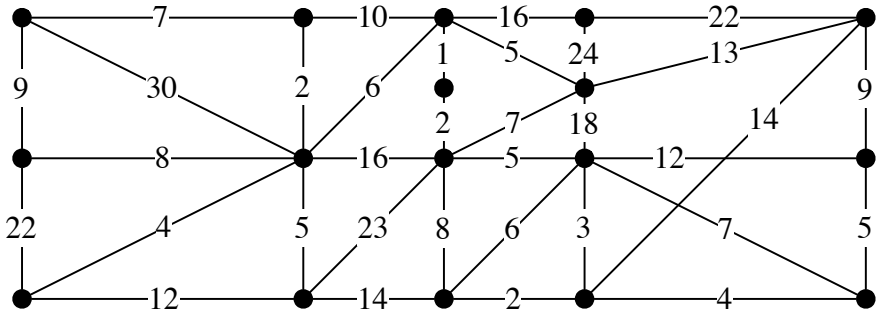
1 P c) Das untenstehende Array enthält die Elemente eines Min-Heaps in der üblichen Form gespeichert. Wie sieht das Array aus, nachdem das Minimum entfernt wurde und die Heap-Bedingung wieder hergestellt wurde?

5	7	21	13	45	36	67	22	17	49	87	86	42
1	2	3	4	5	6	7	8	9	10	11	12	13

1 P d) Fügen Sie die Schlüssel 26,37,33,45,49,29,10,21,7 in dieser Reihenfolge mittels Offenem Hashing in die folgende Hashtabelle ein, und benutzen Sie dabei Double Hashing. Die zu verwendende Hash-Funktion ist $h(k) = k \mod 13$, und für das Sondieren soll die Hashfunktion $h'(k) = 1 + (k \mod 11)$ benutzt werden.

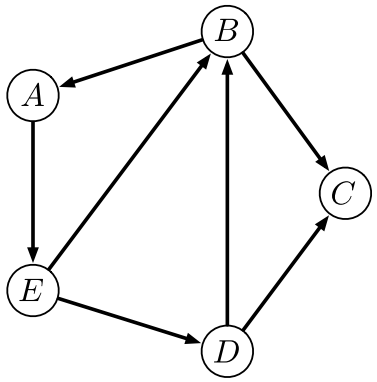
0	1	2	3	4	5	6	7	8	9	10	11	12

1 P e) Markieren Sie die Kanten eines minimalen Spannbaums in folgendem Graphen:

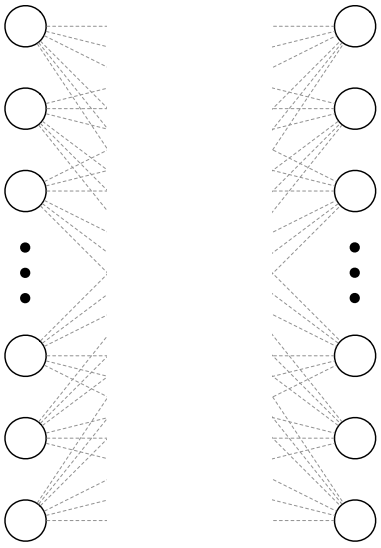


1 P f) Wieviele Kanten kann ein gerichteter Graph mit n Knoten maximal haben, wenn er eine topologische Sortierung besitzen soll?

1 P g) Geben Sie jeweils eine Reihenfolge an, in der die Knoten des folgenden Graphen von einer Breitensuche bzw. Tiefensuche mit Startknoten A besucht werden können.

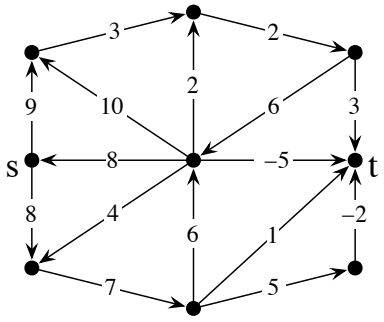


1 P h) Wir betrachten einen beliebigen bipartiten Graphen G (die Knoten links bilden die eine Knotenmenge, die Knoten rechts bilden die andere):



Fügen Sie eine Quelle, eine Senke und zusätzliche Kanten mit Kapazitäten so zu G hinzu, dass ein ganzzahliger, grösstmöglicher Fluss im resultierenden Graphen genau die Kanten eines maximalen Matchings in G verwendet. Für die Flussberechnung dürfen Sie annehmen, dass alle Kanten in G Kapazität 1 haben und von links nach rechts gerichtet sind.

1 P i) Nennen Sie den Namen eines möglichst effizienten Algorithmus, der geeignet ist einen kürzesten Pfad von s zu t im unten gezeichneten Graphen zu bestimmen. Welche Laufzeit hat dieser Algorithmus im Allgemeinen für Graphen mit n Knoten und m Kanten?



Aufgabe 2

1 P a) Geben Sie für die untenstehenden Funktionen eine **Reihenfolge** an, so dass folgendes gilt: Wenn Funktion f links von Funktion g steht, so gilt $f \in \mathcal{O}(g)$.

Beispiel: Die drei Funktionen n^3, n^7, n^9 sind bereits in der entsprechenden Reihenfolge, da $n^3 \in \mathcal{O}(n^7)$ und $n^7 \in \mathcal{O}(n^9)$ gilt.

- n^{1000}
- $\log(n^2)$
- $(\log n)^2$
- $n!$
- $1000n$
- $5^{\sqrt{n}}$
- $\sqrt{n}$
- $n^2 + 1000$
- n^n

3 P b) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 2 + 3T(\frac{n}{7}) & n > 1 \\ 2 & n = 1 \end{cases}$$

Geben Sie eine geschlossene (d.h. nicht-rekursive) und möglichst einfache Formel für $T(n)$ an und beweisen Sie diese mit vollständiger Induktion.

Hinweise:
(1) Sie können annehmen, dass n eine Potenz von 7 ist.
(2) Für $q \neq 1$ gilt: $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

1 P

c) Geben Sie die asymptotische Laufzeit des folgenden Codefragmentes abhängig von $n \in \mathbb{N}$ (möglichst knapp) in Θ-Notation an. Sie müssen Ihre Antwort nicht begründen.

```
for(int i = 0; i < n; ++i)
  for(int j = 0; j < i/2; ++j)
    ;
```

1 P

d) Geben Sie die asymptotische Laufzeit des folgenden Codefragmentes abhängig von $n \in \mathbb{N}$ (möglichst knapp) in Θ-Notation an. Sie müssen Ihre Antwort nicht begründen.

```
for(int i = 0; i < n*n; ++i)
  for(int j = 1; j <= i; j *= 3)
    ;
```

1 P

e) Geben Sie die asymptotische Laufzeit der folgenden Funktion abhängig von $n \in \mathbb{N}$ (möglichst knapp) in Θ-Notation an. Sie müssen Ihre Antwort nicht begründen.

```
int f(int n)
{
  if(n <= 0)
    return 1;
  else
    return f(n-1) + f(n-1);
}
```

Aufgabe 3

Wir betrachten ein Spiel, bei dem Münzen in einer Reihe von links nach rechts ausliegen. Jede Münze hat einen ganzzahligen Wert und die anfängliche Anzahl Münzen n ist durch 3 teilbar. In jedem Zug darf der Spieler entweder die Münze am linken oder die am rechten Ende der Reihe nehmen (und behalten), aber muss dafür zwei Münzen von dem anderen Ende weglegen. Die weggelegten Münzen sind für den Spieler verloren. Nimmt der Spieler beispielsweise die Münze am linken Ende, muss er die zwei Münzen am rechten Ende weglegen. Das Ziel des Spielers ist es, den Wert der genommenen Münzen zu maximieren.

Beispiel: Im folgenden Beispiel mit $n = 21$ kann der Spieler einen Wert von maximal 803 nehmen. Dazu muss er zuerst einmal vom rechten Ende nehmen, dann einmal vom linken, dann zweimal vom rechten und schliesslich dreimal vom linken. Jede andere Strategie hat (natürlich) die gleiche Anzahl Züge ($n/3 = 7$), erreicht aber einen kleineren Wert.

20	20	500	20	20	20	20	100	100	100	5	5	5	5	5	5	1	1	1	1	1

5 P a) Entwerfen Sie einen möglichst effizienten Algorithmus nach dem Prinzip der dynamischen Programmierung, der für eine gegebene Münzreihe den maximal für den Spieler erreichbaren Wert berechnet. Beschreiben Sie Ihr dynamisches Programm und geben sie die Laufzeit des Algorithmus an.

1 P b) Beschreiben Sie kurz, wie Sie Ihren Algorithmus anpassen können, wenn er nicht nur den maximalen Wert, sondern auch die Zugfolge einer bestmöglichen Strategie bestimmen soll. (Im obigen Beispiel also so etwas wie „RLRRLLL“.)

3 P c) Wir betrachten eine erweiterte Version des Spiels mit einem Spieler A und einem Gegenspieler B. Die Spieler sind abwechselnd am Zug, wobei A das Spiel beginnt. Immer wenn Spieler A am Zug ist, nimmt er *eine* Münze vom linken oder vom rechten Ende. Spieler B nimmt in jedem Zug *zwei* Münzen vom linken oder *zwei* Münzen vom rechten Ende.

Wir interessieren uns für eine gute Strategie für Spieler A, unabhängig von B's Strategie. Wir wollen also bestimmen, welchen maximalen Wert Spieler A **garantiert** erreichen kann, unabhängig von B's Spielweise.

Beschreiben Sie einen möglichst effizienten Algorithmus, der für eine gegebene Münzreihe den maximalen, garantiert erreichbaren Wert für Spieler A berechnet. Geben sie die Laufzeit Ihres Algorithmus an.

Hinweis: Es ist kein Fehler, davon auszugehen, dass B immer einen für A schlimmstmöglichen Zug macht. Die Züge von B versuchen den Wert zu *minimieren*, während A's Züge den Wert zu *maximieren* versuchen. Im obigen Beispiel kann Spieler A maximal einen Wert von 632 garantieren. Insbesondere kann Spieler B nicht verhindern, dass A die 500 bekommt (dazu nimmt A einfach solange vom rechten Ende, bis er die 500 direkt nehmen kann).

Aufgabe 4

Sie sollen einen Algorithmus zur Optimierung von mehrtägigen Fahrradtouren entwerfen, bei denen über Nacht gezeltet wird. Eine Tour besteht aus n Streckenabschnitten und $n+1$ Orten, wobei der i-te Streckenabschnitt die Orte i und $i+1$ verbindet. Gegeben ist eine solche Tour, ein Zeitraum T (in Tagen), den die Tour dauern soll, und eine Wettervorhersage für alle Bereiche der Tour. Die Wettervorhersage beschreibt die Niederschlagsmenge (Regen) auf jedem Streckenabschnitt für jeden Tag und die Niederschlagsmenge für jede Nacht an jedem Ort. Eine Instanz mit $n = 3$ und $T = 6$ sieht also beispielsweise so aus:

	Abschnitt 1	Abschnitt 2	Abschnitt 3
Tag 1	2mm	3mm	1mm
Tag 2	1mm	4mm	0mm
Tag 3	3mm	0mm	8mm
Tag 4	3mm	7mm	5mm
Tag 5	10mm	1mm	2mm
Tag 6	5mm	5mm	5mm

	Ort 1	Ort 2	Ort 3	Ort 4
Nacht $1 \to 2$	0mm	2mm	0mm	0mm
Nacht $2 \to 3$	1mm	0mm	2mm	1mm
Nacht $3 \to 4$	0mm	1mm	15mm	0mm
Nacht $4 \to 5$	2mm	0mm	15mm	4mm
Nacht $5 \to 6$	0mm	4mm	0mm	3mm

Gesucht ist ein Plan, an welchen Tagen die Streckenabschnitte gefahren werden sollten, um die **Summe** der Niederschläge für jeden gefahrenen Abschnitt und jede Übernachtung (gemäss Vorhersage) klein zu halten. Die Streckenabschnitte sollen in der gegebenen Reihenfolge befahren werden (also Abschnitt i vor Abschnitt $i+1$), aber es können *beliebig viele Abschnitte pro Tag* eingeplant werden. Ausserdem muss *jede* der Nächte an einem Ort verbracht werden, es können aber mehrere Nächte am gleichen Ort verbracht werden. Wenn der erste Abschnitt also für Tag j geplant ist, muss vorher erst $j-1$ mal an Ort 1 übernachtet werden. Genauso muss am Ende $T-j$ mal an Ort $n+1$ übernachtet werden, wenn der letzte Abschnitt für Tag j geplant ist.

Beispiel: Im obigen Beispiel kommt die beste Lösung mit 8mm Niederschlag aus. Dazu wird zunächst an Ort 1 übernachtet (0mm), an Tag 2 wird Streckenabschnitt 1 gefahren (1mm), dann wird dreimal an Ort 2 übernachtet (0+1+0=1mm), an Tag 5 werden die Abschnitte 2 und 3 gefahren (1+2=3mm), und schliesslich wird noch einmal an Ort 4 übernachtet (3mm). Wenn wir stattdessen zum Beispiel alle Abschnitte gleich an Tag 1 fahren (2+3+1=6mm), müssen wir fünf mal an Ort 4 übernachten (0+1+0+4+3=8mm), und haben also insgesamt einen Niederschlag von 14mm.

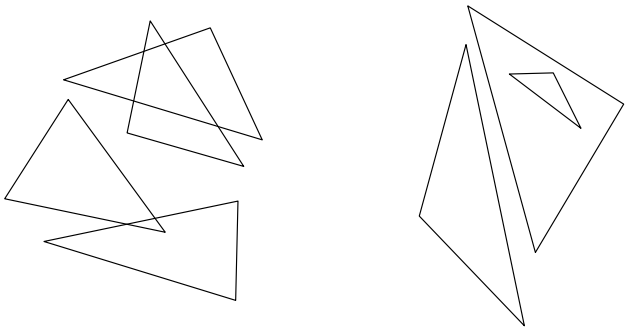
4 P a) Konstruieren Sie einen gerichteten Graphen G, so dass ein kürzester Weg in G einer Fahrradtour mit minimalem Niederschlag entspricht. Führen Sie dazu für jeden Zustand des Systems einen Knoten in G ein und definieren Sie (gerichtete und gewichtete) Kanten, die den Zustandsübergängen entsprechen. Beschreiben Sie Ihre Konstruktion und leiten Sie daraus einen möglichst effizienten Algorithmus zur Planung von Fahrradtouren mit minimalem Niederschlag ab.

2 P b) Welche Laufzeit hat Ihr Algorithmus in Abhängigkeit von n und T?

3 P c) Nehmen Sie nun an, dass nicht mehr als 3 Streckenabschnitte pro Tag gefahren werden können. Wie können Sie die Konstruktion von G anpassen, so dass ein Plan mit minimalem Niederschlag und maximal 3 Streckenabschnitten pro Tag einem kürzesten Weg in G entspricht (und umgekehrt)? Wie verändert sich dadurch die Laufzeit Ihres Algorithmus?

Aufgabe 5

Gegeben ist eine Menge von Dreiecken in der Ebene. Jedes Dreieck ist durch die Koordinaten seiner Eckpunkte gegeben, wobei wir der Einfachheit halber annehmen, dass keine zwei Eckpunkte die gleiche x-Koordinate haben. Wir möchten herausfinden, ob die Menge zwei Dreiecke enthält, die sich überlappen.



3 P a) Wir ermitteln zunächst, ob es Dreiecke gibt, deren Ränder sich schneiden (linkes Bild). Beschreiben Sie einen möglichst effizienten *Scanlinealgorithmus*, der dies bestimmt. Sie dürfen davon ausgehen, dass sich die Ränder keiner zwei Dreiecke berühren ohne sich zu schneiden. Welche Laufzeit hat Ihr Verfahren?

6 P b) Falls sich die Ränder keiner zwei Dreiecke schneiden, müssen wir noch feststellen, ob es ein Dreieck gibt, das in einem anderen enthalten ist (rechtes Bild). Beschreiben Sie einen möglichst effizienten *Scanlinealgorithmus*, der bestimmt, ob dies der Fall ist. Sie dürfen davon ausgehen, dass sich die Ränder keiner zwei Dreiecke *berühren oder schneiden*. Welche Laufzeit hat Ihr Verfahren?