

Midterm Exam
Algorithms and Data Structures WS03/04

Name:

Matr.-Nr.:

--	--	--	--	--	--	--

Time: 120 min

Aufgabe	1	2	3	4	5
Maximalpunktzahl	8	8	8	18	18
erreichte Punktzahl					

Σ : von 60

Problem 1: (Symmetric binary relation, 8 points)

Assume you are given a set M of pairs of integers in the range $1, \dots, |M|$. M defines a binary relation R_M ¹. Outline an algorithm that checks in expected time $\mathcal{O}(|M|)$ whether R_M is symmetric². Hint: there are at least two basic approaches to a solution. One yields a deterministic algorithm the other is randomized but would also work without the restriction on the size of the numbers.

We give two solutions.

- (a) We store all the pairs $(a, b) \in M$ in the hash table H with (a, b) as a key. Then we go through all pairs again, checking for each (a, b) whether (b, a) is in the hash table. If the test is passed then relation R_M is symmetric. Each hash table operation has expected cost $\mathcal{O}(1)$. Therefore the expected running time is $\mathcal{O}(|M|)$.

```
for each  $(a, b) \in M$  do  $H.insert((a, b))$  //  $\mathcal{O}(|M|)$  expected time  
for each  $(a, b) \in M$  do if  $H.find((b, a)) = \emptyset$  then return false //  $\mathcal{O}(|M|)$  exp. time  
return true
```

- (b) The running time of the previous algorithm depends on the running times of the hash table operations, which are in fact *expected* running times. Therefore the algorithm is a *randomized* one. To achieve *deterministic* run time guarantees, one can use a different approach.

First, insert the pairs $(a, b) \in M$ into array A with $a \leq b$, if $a > b$ insert (b, a) instead. Sort array A lexicographically using LSD radix sort. If each pair occurs exactly twice (the copies must come together in the sorted array A) then the relation R_M is symmetric. Sorting takes time $\mathcal{O}(|M|)$. Checking is just one scan through A .

```
for each  $(a, b) \in M$  do  $A.push\_back((\min(a, b), \max(a, b)))$   
 $sort(A)$   
for  $i := 0$  to  $|M|/2 - 1$   
  if  $A[2i] \neq A[2i + 1]$  then return false  
return true
```

¹Binary relation R on a set X is a subset of the Cartesian product $X \times X$.

²A relation R is symmetric if and only if $\forall (a, b) \in R : (b, a) \in R$.

Aufgabe 1: (Symmetrische binäre Relation, 8 Punkte)

Gegeben sei eine Menge M von Paaren ganzer Zahlen im Bereich $1, \dots, |M|$. M definiert eine binäre Relation R_M ³. Skizzieren Sie einen Algorithmus, der in erwarteter Zeit $\mathcal{O}(|M|)$ überprüft ob R_M symmetrisch⁴ ist. Hinweis: Es gibt mindestens zwei Lösungsansätze. Einer ergibt einen deterministischen Algorithmus. Der andere ist randomisiert und würde auch mit Zahlen unbeschränkter Größe funktionieren.

³Eine binäre Relation R zwischen Elementen einer Menge X ist eine Teilmenge des kartesischen Produkts $X \times X$.

⁴Eine Relation R ist symmetrisch genau dann wenn $\forall (a, b) \in R : (b, a) \in R$.

Problem 2: (Maximum flow with node capacities, 8 points)

Explain how to solve a maximum flow problem with *node capacities*, i.e., besides edge capacities there is also a bound $\text{cap}(v)$ on the flow through a node v . Do not forget to give a (short) argument why your algorithm actually computes a feasible *and* optimal solution.

Suppose $G = (V, E)$ is the original graph. We define a new graph $G' = (V', E')$. For each vertex v in V , there are two vertices v_{in} and v_{out} in V' . There is an edge $(v_{\text{in}}, v_{\text{out}}) \in E'$ with capacity $\text{cap}(v)$. Furthermore, an edge $(u, v) \in E$ is translated into an edge $(u_{\text{out}}, v_{\text{in}})$ in E' .

Note that there is a one-to-one correspondence between feasible flows from s_{in} to t_{out} in G' and s - t flows that respect node capacities in G' , i.e., flows in G that respect node capacities can be translated into feasible flows in G' and vice versa in the obvious way. Hence, it suffices to find an ordinary maximum flow f' in G' and to translate it into a flow f that respects node capacities in G . This flow is optimal: Assume f were not optimal. Then must be a flow f^* that respects node capacities with higher value in G which corresponds to a flow with higher value than f' in G' . This contradicts the optimality of f' in G' .

Aufgabe 2: (Maximale Flüsse mit Knotenkapazitäten, 8 Punkte)

Erklären Sie wie maximale Flüsse mit *Knotenkapazitäten* berechnet werden können, d.h., neben Kantenkapazitäten gibt es auch eine Schranke $\text{cap}(v)$ für den Fluss durch einen Knoten v . Vergessen Sie nicht, kurz zu begründen, warum Ihr Algorithmus einen zulässigen *und* optimalen Fluss berechnet.

Problem 3: (SSSP in acyclic graphs, 8 points)

Outline an algorithm for single source shortest paths in undirected acyclic graphs with nonnegative edge weights that runs in time $\mathcal{O}(m + n)$.

An undirected acyclic graph is a forest. Hence, all nodes reachable from the source node s are reachable only on a unique simple path. Hence, it suffices to perform BFS from s to find all these paths. Distance values can be computed by adding edge weights rather than by counting edges (as in BFS).

Aufgabe 3: (Kürzeste Wege in azyklischen Graphen, 8 Punkte)

Skizzieren Sie einen Algorithmus für “single source shortest paths” in ungerichteten kreisfreien Graphen mit nichtnegativen Kantengewichten. Der Algorithmus soll in Zeit $\mathcal{O}(m + n)$ arbeiten.

Problem 4: (Short questions, 6×3 points)

Give true/false/choice answers and short explanations. For each correct true/false/choice answer there is 1 point, for a correct explanation we give another 2 points.

- (a) How fast can one sort n integers in the range $0 \dots n^4 - 1$ using algorithms we have seen in this course: $\mathcal{O}(n)$, $\mathcal{O}(n \log \log n)$, $\mathcal{O}(n \log n)$. Give the best possible bound. Shortly explain what algorithm to use and how.

$\mathcal{O}(n)$: use LSD radix sort with radix n .

- (b) Suppose you want to simplify the data structure of suffix trees⁵ by defining edge labels to be just single characters (i.e. an edge with label " $c_1 c_2 \dots c_k$ " is replaced with the chain of k edges with labels " c_1 ", " c_2 ", \dots , " c_k "). What would be the worst space consumption of this data structure for strings with the length n : $\Theta(n)$, $\Theta(n \log n)$, $\Theta(n^2)$ or $\Theta(n^3)$? (You may assume that there is no restriction on the alphabet size.)

$\Theta(n^2)$, e.g., $\langle 1, 2, \dots, n \rangle$

- (c) If graph G is acyclic then for all feasible flows the residual graph G_f is acyclic too.

False. For example, consider the graph $G = (\{s, u, v, t\}, \{(s, u), (s, v), (u, t), (v, t)\})$, assume unit capacities, and consider a flow of value one along the path $\langle s, u, t \rangle$.

- (d) If $|E| \geq |V| - 1$ then the undirected graph $G = (V, E)$ is connected.

No. For example, consider a graph consisting of a three-cycle $\langle s, u, v, s \rangle$ and an isolated vertex x .

- (e) Let G be a directed weighted graph, and let u, v be two vertices. Then a shortest path from u to v remains a shortest path when 1 is added to every edge weight.

False. For example, consider $E = \{(s, v), (v, t), (s, t)\}$ where (s, t) has weight two and the other edges have weight one. After adding one to every edge weight, the shortest path from s to t changes from $\langle s, v, t \rangle$ to $\langle s, t \rangle$.

- (f) For every text length n and pattern length m , give an instance of the string matching problem where the naive algorithm needs time $\Omega(nm)$ although there is no match. (3 points for a correct answer)

text: \mathbf{a}^n , pattern: $\mathbf{a}^{m-1}\mathbf{b}$.

⁵Recall, that in a suffix trees the edge labels on a path from the root to a leaf represent suffixes and that the labels of edges (v, u) and (v, w) should differ in their first character.

Aufgabe 4: (Kurze Fragen, 6×3 Punkte)

Geben Sie wahr/Falsch/Auswahl-Antworten und kurze Erklärungen. Es gibt einen Punkt für jede korrekte Antwort und zwei Punkte für korrekte Erklärungen.

- (a) Wie schnell kann man n ganze Zahlen im Bereich $0 \dots n^4 - 1$ sortieren (Mit Algorithmen aus der Vorlesung)? $\mathcal{O}(n)$, $\mathcal{O}(n \log \log n)$, $\mathcal{O}(n \log n)$. Geben Sie die bestmögliche Schranke an. Erklären sie kurz welchen Algorithmus Sie wie anwenden.
- (b) Angenommen Sie möchten die Suffix-Baum-Datenstruktur⁶ vereinfachen indem Sie Kantenbeschriftungen als einzelne Buchstaben definieren. (Eine Kantenbeschriftung " $c_1 c_2 \dots c_k$ " wird also durch eine Kette von k Kanten mit Beschriftungen " c_1 ", " c_2 ", \dots ", " c_k " ersetzt.) Was wäre der Speicherplatz einer solchen Datenstruktur für eine Zeichenkette der Länge n im schlechtesten Fall? $\Theta(n)$, $\Theta(n \log n)$, $\Theta(n^2)$ oder $\Theta(n^3)$? (Sie können annehmen, dass es keine Beschränkung bezüglich der Alphabetgröße gibt.)
- (c) Wenn ein Graph G azyklisch ist, dann ist für jeden zulässigen Fluss f in G auch der Residualgraph G_f azyklisch.
- (d) Falls $|E| \geq |V| - 1$ dann ist der ungerichtete Graph $G = (V, E)$ zusammenhängend.
- (e) Sei G ein gerichteter Graph mit Kantengewichten. Seien u, v zwei Knoten. Ein kürzeste Pfad von u nach v bleibt ein kürzester Pfad, wenn 1 zu jedem Kantengewicht hinzuaddiert wird.
- (f) Für jede Textlänge n und jede Suchmusterlänge m geben Sie eine Instanz des string matching Problems bei der der naive Algorithmus $\Omega(nm)$ Zeit braucht obwohl es kein einziges Vorkommen des Musters im Text gibt. (3 Punkte für eine korrekte Antwort.)

⁶Zur Erinnerung: Die Kantenbeschriftungen auf einem Pfad von der Wurzel des Suffix-Baums zu einem Blatt repräsentieren einen Suffix. Außerdem müssen die Beschriftungen zweier Kanten (v, u) und (v, w) sich in ihrem ersten Buchstaben unterscheiden.

Problem 5: (vEB remove, 18 points)

A vEB data structure maintains a set $M \subseteq 0 \dots 2^A - 1$ for some constant A . It supports the following operations in time $\mathcal{O}(\log A)$:

- *insert*(x): $M := M \cup \{x\}$
- *remove*(x): $M := M \setminus \{x\}$
- *locate*(x): returns $\min\{y \in M \mid y \geq x\}$

Recursive description for A -bit vEB tree (A is assumed to be a power of two and special case treatments for the case $A = 1$ are ignored):

- $|M| = 1$, store members:
 - e stores the single element, $M = \{e\}$
 - $size = |M| = 1$
- $|M| > 1$, store members:
 - $size = |M|$
 - $K = A/2$
 - $minM = \min_{x \in M}(x)$
 - $maxM = \max_{x \in M}(x)$
 - t stores $\{x \text{ div } 2^K \mid x \in M\}$
 - r_i stores $\{x \text{ mod } 2^K \mid x \in M \text{ and } x \text{ div } 2^K = i\}$

Example of a vEB tree operation:

Procedure *insert*(x)

```
if size = 1 then // M = {e}
  minM := maxM := e
  create t with the single element e div 2^K
  create r[e div 2^K] with the single element e mod 2^K
if r[x div 2^K] = ∅ then
  create r[x div 2^K] with the single element x mod 2^K
  t.insert(x div 2^K)
else
  r[x div 2^K].insert(x mod 2^K)
minM := min(minM, x)
maxM := max(maxM, x)
size := size + 1
```

Implement the operation *remove*(x) for the case that $x \in M$. Prove that your implementation runs in time $\mathcal{O}(\log A)$. Hint: be careful about the cost of recursive calls. There is also no need to use *locate*.

Procedure *remove*(x)

if $size = 1$ **then**

$size := 0$

return

if $size = 2$ **then** // back to the trivial case

if $x = minM$ **then** $e := minM := maxM$ **else** $e := maxM := minM$

$t := \emptyset$ // deallocation

$r[minM \operatorname{div} 2^K] := \emptyset$ // deallocation

$r[maxM \operatorname{div} 2^K] := \emptyset$ // deallocation

$size := 1$

return

if $r[x \operatorname{div} 2^K].size = 1$ **then**

$r[x \operatorname{div} 2^K] := \emptyset$ // deallocation

$t.remove(x \operatorname{div} 2^K)$

else

$r[x \operatorname{div} 2^K].remove(x \operatorname{mod} 2^K)$

$minM := 2^K \cdot t.minM + r[t.minM].minM$

$maxM := 2^K \cdot t.maxM + r[t.maxM].maxM$

$size := size - 1$

return

There is at most one recursive call to an $A/2$ -bit tree. Therefore the running time is bounded as $T(A) = \mathcal{O}(1) + T(A/2)$, which solves to $\mathcal{O}(\log A)$.

Aufgabe 5: (vEB remove, 18 Punkte)

Eine vEB Datenstruktur verwaltet eine Menge $M \subseteq 0 \dots 2^A - 1$. A ist eine Konstante. Die folgenden Operationen mit Laufzeit $\mathcal{O}(\log A)$ werden zur Verfügung gestellt:

- $insert(x)$: $M := M \cup \{x\}$
- $remove(x)$: $M := M \setminus \{x\}$
- $locate(x)$: returns $\min\{y \in M \mid y \geq x\}$

Hier ist eine rekursive Beschreibung für einen A -bit vEB tree (A wird als Zweierpotenz angenommen und Sonderfallbehandlungen für den Fall $A = 1$ werden ignoriert.):

- $|M| = 1$:
 - e speichert das einzige Element, $M = \{e\}$
 - $size = |M| = 1$
- $|M| > 1$ speichere:
 - $size = |M|$
 - $K = A/2$
 - $minM = \min_{x \in M}(x)$
 - $maxM = \max_{x \in M}(x)$
 - t speichert $\{x \text{ div } 2^K \mid x \in M\}$
 - r_i speichert $\{x \text{ mod } 2^K \mid x \in M \text{ and } x \text{ div } 2^K = i\}$

Beispiel für eine vEB-tree-Operation:

Procedure $insert(x)$

```
if  $size = 1$  then //  $M = \{e\}$ 
   $minM := maxM := e$ 
  create  $t$  mit dem einzigen Element  $e \text{ div } 2^K$ 
  create  $r[e \text{ div } 2^K]$  mit dem einzigen Element  $e \text{ mod } 2^K$ 
if  $r[x \text{ div } 2^K] = \emptyset$  then
  create  $r[x \text{ div } 2^K]$  mit dem einzigen Element  $x \text{ mod } 2^K$ 
   $t.insert(x \text{ div } 2^K)$ 
else
   $r[x \text{ div } 2^K].insert(x \text{ mod } 2^K)$ 
 $minM := \min(minM, x)$ 
 $maxM := \max(maxM, x)$ 
 $size := size + 1$ 
```

Realisieren Sie die Operation $remove(x)$ für den Fall, dass $x \in M$. Zeigen Sie, dass Ihre Implementierung in Zeit $\mathcal{O}(\log A)$ arbeitet. Hinweis: Seien Sie vorsichtig bei den Kosten rekursiver Aufrufe. Es besteht auch keine Notwendigkeit, $locate$ zu benutzen.

