tentamen datastructuren en algoritmen 27 oktober 2008

Opgave 1.

(a) Implementeer een stack (met operaties push en pop) met twee queues.

(5 punten)

(b) We implementeren een priority queue met een ongeordend rijtje. Wat is dan de complexiteit van priority-queue-sort, in termen van grote-O? Beargumenteer je antwoord.

(5 punten)

(c) Maak een hash-tabel ter grootte 9. Gebruik de hash-functie $h(k) = k \bmod 5$. Voeg aan de (initieel lege) hash-tabel de volgende getallen toe (in deze volgorde):

$$15, 28, 29, 25, 0, 33, 12, 17$$

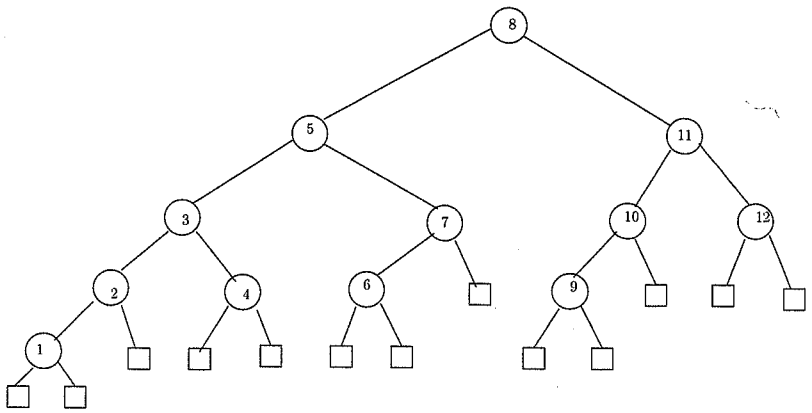
en wel op twee manieren:

- door collision met separate chaining op te lossen,
- door collision met linear probing op te lossen.

(6 punten)

Opgave 2.

(a) Verwijder uit de volgende AVL-boom de knoop met label 4:



(5 punten)

(b) Bewering: een preorder traversal van een heap geeft de elementen in niet-dalende volgorde.

Toon met een voorbeeld aan dat deze bewering onwaar is.

(5 punten)

(c) Gegeven is een array ter lengte n met daarin de gesorteerde lijst van alle getallen van 0 tot en met n op één na. Een voorbeeld van zo'n array met $n = 5$ is $[0, 1, 3, 4, 5]$ waarbij 2 het ontbrekende getal is.

Geef een algoritme in $O(\log n)$ dat als input zo'n array neemt en als output het ontbrekende getal teruggeeft.

(6 punten)

Opgave 3.

(a) Geef de merge-sort-boom voor het sorteren van de rij $8, 5, 2, 7, 1, 3, 4, 6, 9$.

(5 punten)

(b) De recurrente betrekking die bij merge-sort hoort is:

$$T(n) = \begin{cases} 1 & \text{als } n = 1 \\ 2T(\frac{n}{2}) + n & \text{als } n > 1 \end{cases}$$

Los deze recurrente betrekking op, en geef aan wat dus de tijdscomplexiteit van merge-sort is in termen van grote-O.

(6 punten)

Opgave 4.

(a) Geef een voorbeeld waaruit blijkt dat steeds een item met een grootste waarde kiezen niet altijd leidt tot een optimale oplossing van het fractional knapsack probleem.

(5 punten)

(b) Gegeven zijn 5 items met benefit-gewicht waarden als volgt:

$$(50, 1) \quad (12, 4) \quad (30, 5) \quad (1, 1) \quad (10, 2)$$

en gegeven is een maximum totaalgewicht $W = 10$.

Pas het algoritme voor fractional knapsack toe; geef je volledige berekening.

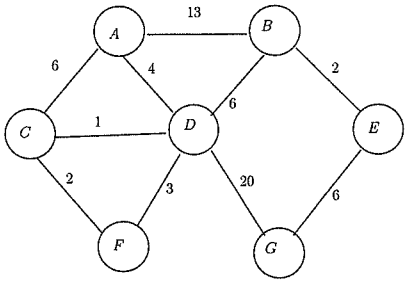
(5 punten)

(c) Leg uit waarom het algoritme voor fractional knapsack in $O(n \log n)$ is, met n het aantal elementen in de verzameling S waaruit we (delen van) items kiezen.

(5 punten)

Opgave 5.

(a) Pas Dijkstra's kortste pad algoritme toe op de volgende graaf, met A als startknoop. Geef stap voor stap aan wat er gebeurt, door in elke stap de graaf met de relevante data te tekenen.



(9 punten)

(b) Geef (beargumenteerd) de complexiteit van Dijkstra's korste pad algoritme in termen van grote-O.

(6 punten)

Opgave 6. Gegeven zijn de tekst T en het patroon P als volgt:

$$\begin{aligned} T &= \textit{abababdcababababcabbadab} \\ P &= \textit{ababcab} \end{aligned}$$

(a) Beschrijf stap voor stap het toepassen van het brute-force pattern matching algoritme; nummer de stappen zodat je kunt zien hoeveel stappen gedaan worden.

(6 punten)

(b) Geef de failure functie van het Knuth–Morris–Pratt pattern matching algoritme voor het patroon P.

(5 punten)

(c) Beschrijf stap voor stap het toepassen van het Knuth–Morris–Pratt pattern matching algoritme; nummer de stappen zodat je kunt zien hoeveel stappen gedaan worden.

(6 punten)

Het cijfer is (het totaal aantal punten plus 10) gedeeld door 10.

bijlage bij het tentamen datastructuren en algoritmen 2008-2009

Theorem 1.12 For all $n \in \mathbb{N}$ and for all $a \in \mathbb{R}$ with $a > 0$ and $a \neq 1$:

$$\sum_{i=0}^{n} a^i = \frac{1 - a^{n+1}}{1 - a}$$

Theorem 1.13 (= Theorem 1.20) For all $n \in \mathbb{N}$ with $n \geq 1$ we have:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Theorem 2.5 Let T be a tree with n nodes, and let c_v denote the number of children of a node v of T. Then:

$$\sum_{v \in T} c_v = n - 1$$

Theorem 2.8 Let T be a (proper) binary tree with n nodes, and let h denote the height of T. Then T has the following properties:

(a) The number of external nodes in T is at least $h + 1$ and at most 2^h.

(b) The number of internal nodes in T is at least h and at most $2^h - 1$.

(c) The total number of nodes in T is at least $2h + 1$ and at most $2^{h+1} - 1$.

(d) The height of T is at least $\log(n + 1) - 1$ and at most $\frac{n-1}{2}$, that is, $\log(n + 1) - 1 \leq h \leq \frac{n-1}{2}$.

Theorem 2.9 In a (proper) binary tree T, the number of external nodes is 1 more than the number of internal nodes.

Theorem 2.10 A heap T storing n keys has height $h = \lceil \log(n + 1) \rceil$.

Algorithm mergeSort(S):
Input: A sequence S
Output: Sequence S sorted
if S.size() > 1 **then**
let A contain the first $\lceil n/2 \rceil$ elements of S
let B contain the remaining $\lfloor n/2 \rfloor$ elements of S
mergeSort(A)
mergeSort(B)
$S \leftarrow$ merge(A, B)

Algorithm merge(A, B):
Input: Sequences A and B sorted in non-decreasing order
Output: A sorted sequence S containing the elements from A and B
$S \leftarrow$ empty sequence
while (**not** A.isEmpty()) **and** (**not** B.isEmpty()) **do**
if A.first().element() $\leq$ B.first().element() **then**
{move the first element of A at the end of S}
S.insertLast(A.remove(A.first()))
else
{move the first element of B at the end of S}
S.insertLast(B.remove(B.first()))
{move the remaining elements of A at the end of S}
while not A.isEmpty() **do**
S.insertLast(A.remove(A.first()))
{move the remaining elements of B at the end of S}
while not B.isEmpty() **do**
S.insertLast(B.remove(B.first()))
return S

Algorithm FractionalKnapsack(S, W):

Input: Set S of items, such that each item $i \in S$ has a positive benefit b_i and positive weight w_i ; positive maximum total weight W

Output: Amount x_i of each item $i \in S$ that maximizes the total benefit while not exceeding the maximum total weight W

for each item $i \in S$ **do**
  $x_i \leftarrow 0$
  $v_i \leftarrow b_i/w_i$ {***value index*** of item i}
$w \leftarrow 0$ {total weight}
while $w < W$ **do**
  remove from S an item i with highest value index {greedy choice}
  $a \leftarrow \min\{w_i, W - w\}$ {more than $W - w$ causes a weight overflow}
  $x_i \leftarrow a$
  $w \leftarrow w + a$

Algorithm DijkstraShortestPaths(G, v):

Input: A simple undirected weighted graph G with nonnegative edge weights, and a distinguished vertex v of G

Output: A label $D[u]$, for each vertex u of G, such that $D[u]$ is the distance from v to u in G

$D[v] \leftarrow 0$
for each vertex $u \neq v$ of G **do**
  $D[u] \leftarrow +\infty$
Let a priority queue Q contain all the vertices of G using the D labels as keys
while Q is not empty **do**
  {pull a new vertex u into the cloud}
  $u \leftarrow Q$.removeMin()
  for each vertex z adjacent to u such that z is in Q **do**
    {perform the ***relaxation*** operation on $\langle u, z \rangle$}
    if $D[u] + w(\langle u, z \rangle) < D[z]$ **then**
      $D[z] \leftarrow D[u] + w(\langle u, z \rangle)$
      Change to $D[z]$ the key of vertex z in Q.
return the label $D[u]$ of each vertex u

Algorithm BruteForceMatch(T, P):

Input: Strings T (text) with n characters and P (pattern) with m characters

Output: Starting index of the first substring of T matching P,
or an indication that P is not a substring of T

for $i \leftarrow 0, \ldots, n-m$ {for each candidate index in T} **do**
  $j \leftarrow 0$
  while $j < m$ **and** $T[i+j] = P[j]$ **do**
    $j \leftarrow j+1$
  if $j = m$ **then**
    return i
return "There is no substring of T matching P."

Algorithm KnuthMorrisPrattMatch(T, P):

Input: Strings T (text) with n characters and P (pattern) with m characters

Output: Starting index of the first substring of T matching P,
or an indication that P is not a substring of T

$f \leftarrow$ KMPFailureFunction(P) {construct the failure function f for P}
$i \leftarrow 0$
$j \leftarrow 0$
while $i < n$ **do**
  if $P[j] = T[i]$ **then**
    if $j = m-1$ **then**
      return $i-j$ {a match!}
    $i \leftarrow i+1$
    $j \leftarrow j+1$
  else if $j > 0$ {no match, but we have advanced in P} **then**
    $j \leftarrow f(j-1)$ {j indexes just after prefix of P that must match}
  else
    $i \leftarrow i+1$
return "There is no substring of T matching P."