herkansing datastructuren en algoritmen 8 januari 2009

Opgave 1.

(a) Maak een hash-tabel ter grootte 10. Gebruik de hash-functie $h(k) = k \bmod 7$. Voeg aan de (initieel lege) hash-tabel de volgende getallen toe (in deze volgorde):

$$12, 44, 13, 88, 23, 94, 11, 39, 20$$

en wel op twee manieren:

(i) door collision met separate chaining op te lossen,

(ii) door collision met linear probing op te lossen.

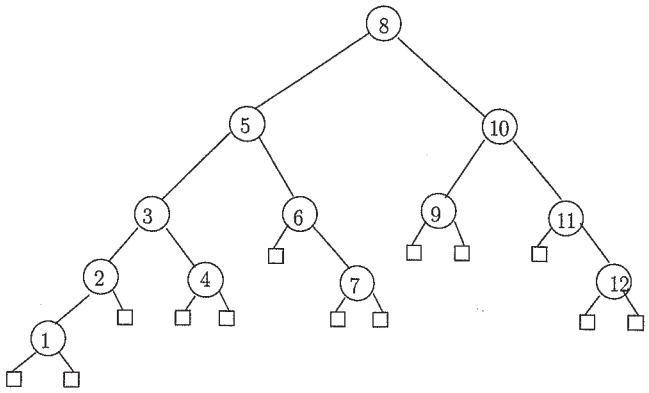
(6 punten)

(b) Geef een algoritme dat als input een knoop v in een (echte) binaire boom T neemt, en als output levert de knoop die na v bezocht wordt in een preorder traversal van T.

(6 punten)

Opgave 2.

(a) Verwijder stap voor stap de knoop met label 9 uit de volgende AVL-boom:



(6 punten)

(b) In de bovenstaande AVL-boom is het verschil in diepte tussen twee willekeurige externe knopen maximaal 2.

Is dat altijd het geval? Zo ja, leg uit waarom. Zo nee, geef een tegenvoorbeeld.

(5 punten)

(c) De recurrente betrekking die bij binary search hoort is

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

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

(6 punten)

Opgave 3.

(a) Geef de quick-sort boom voor het uitvoeren van quick-sort op de volgende input (de pivot is steeds het laatste element): $8, 5, 2, 7, 1, 3, 4, 6, 9$.

(6 punten)

(b) Geef een algoritme in $O(n \log n)$ dat als input een rij A van n gehele getallen neemt, en als output levert een rij waarin elk getal uit A precies één keer voorkomt (dus de dubbele worden uit A weggehaald).

NB: je mag bestaande algoritmes gebruiken.

(6 punten)

Opgave 4.

(a) Een variant op het task-scheduling probleem is het machine-scheduling probleem: gegeven een verzameling taken met start-tijd en eind-tijd zoeken we een maximale (d.w.z. grootst mogelijke) verzameling taken die op één machine uitgevoerd kan worden.

Geef een greedy algoritme voor het machine-scheduling probleem.

(6 punten)

(b) Geef een voorbeeld waaruit blijkt dat het algoritme voor fractional knapsack niet altijd een optimale oplossing oplevert voor het 01knapsack-probleem.

(5 punten)

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

$$(3, 2) \quad (5, 4) \quad (8, 5) \quad (4, 3) \quad (10, 9)$$

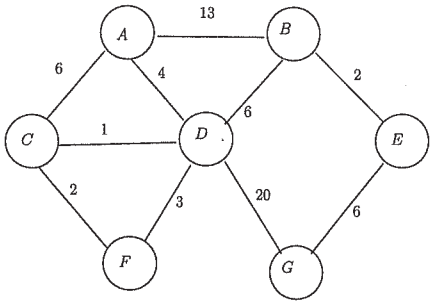
en gegeven is een maximum totaalgewicht $W = 20$.

Pas het algoritme voor 01knapsack toe; geef je volledige berekening.

(6 punten)

Opgave 5.

(a) Pas het Prim–Jarník algoritme toe op de volgende graaf om een minimale opspannende boom te vinden. Geef stap voor stap aan wat er gebeurt, door in elke stap de graaf met de relevante data te tekenen.



(10 punten)

(b) Geef (beargumenteerd) de complexiteit van het Prim–Jarník algoritme in termen van grote-O.

(6 punten)

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

$$\begin{aligned} T &= ababab dcababaccabbadab \\ P &= abaccab \end{aligned}$$

(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 functie *last* van het Boyer–Moore pattern matching algoritme voor het patroon P.

(4 punten)

(c) Beschrijf stap voor stap het toepassen van het Boyer–Moore 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 de herkansing datastructuren en algoritmen
8 januari 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 quickSort(S):
  Input: A sequence S
  Output: Sequence S sorted
  Select a position p of S for the pivot
  $\langle L, E, G\rangle \leftarrow$ partition(S, p)
  quickSort(L)
  quickSort(G)
  $S \leftarrow L \cdot E \cdot G$ {the concatenation of sequences L, E, G}

Algorithm partition(S, p):
  Input: A sequence S, and a position p for the pivot
  Output: Triple of subsequences L, E, G of the elements of S
    less than, equal to, or greater than the pivot, respectively
  $x \leftarrow S$.remove(p)
  while not S.isEmpty() **do**
    $y \leftarrow S$.remove$(S$.first$())$
    if $y < x$ **then**
      L.insertLast(y)
    else if $y = x$ **then**
      E.insertLast(y)
    else {$y > x$}
      G.insertLast(y)
  return $\langle L, E, G\rangle$

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 01Knapsack(S, W):

Input: Set S of n items, such that item i has a positive benefit b_i and positive integer weight w_i; positive integer maximum total weight W

Output: For $w = 0, \ldots, W$, maximum benefit $B[w]$ of a subset of S with total weight at most w

for $w \leftarrow 0, \ldots, W$ **do**
  $B[w] \leftarrow 0$
for $k \leftarrow 1, \ldots, n$ **do**
  for $w \leftarrow W, \ldots, w_k$ **do**
    if $B[w - w_k] + b_k > B[w]$ **then**
      $B[w] \leftarrow B[w - w_k] + b_k$

Algorithm PrimJarnikMST(G):

Input: A weighted connected graph G with n vertices and m edges

Output: A minimum spanning tree T for G

Pick any vertex v of G

$D[v] \leftarrow 0$

for each vertex $u \neq v$ of G **do**

  $D[u] \leftarrow +\infty$

Initialize $T \leftarrow \varnothing$

Initialize a priority queue Q with an item $\langle \langle u, \text{null} \rangle, D[u] \rangle$ for each vertex u, where $\langle u, \text{null} \rangle$ is the element and $D[u]$ is the key.

while Q is not empty **do**

  $\langle u, e \rangle \leftarrow Q.\text{removeMin}()$

  Add vertex u and edge e to T.

  for each vertex z adjacent to u such that z is in Q **do**

    {perform the relaxation procedure on edge $\langle u, z \rangle$}

    if $w(\langle u, z \rangle) < D[z]$ **then**

      $D[z] \leftarrow w(\langle u, z \rangle)$

      Change to $\langle z, \langle u, z \rangle \rangle$ the element of vertex z in Q.

      Change to $D[z]$ the key of vertex z in Q.

return the tree T

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 BoyerMooreMatch(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

compute function last

$i \leftarrow m - 1$

$j \leftarrow m - 1$

repeat

  if $P[j] = T[i]$ **then**

    if $j = 0$ **then**

      return i {a match!}

    else

      $i \leftarrow i - 1$

      $j \leftarrow j - 1$

  else

    $i \leftarrow i + m - \min(j, 1 + \text{last}(T[i]))$ {jump step}

    $j \leftarrow m - 1$

until $i > n - 1$

return "There is no substring of T matching P."