# Tentamen Algoritmen en Datastructuren in C

vrijdag 12 april 2013, 9 - 12 uur

Dit tentamen bevat 4 opgaven waarvoor maximaal 90 punten zijn te behalen.
Het tentamencijfer is (# punten)/10 + 1.

1. (25 punt) Deze opgave gaat over gelinkte lijsten.

   (a) Geef een `typedef` in C voor het type `List` van gelinkte lijsten met per knoop een getal van type `int`.

   (b) Definieer een functie van type `List addItem(List lst, int n)` om getal `n` vooraan lijst `lst` toe te voegen.

   (c) Definieer een functie van type `List removeFirstItem(List lst, int *np)`. Na uitvoering van de functie is de eerste knoop van lijst `lst` verwijderd en wijst `np` naar het getal dat in die knoop stond. Houd rekening met de mogelijkheid dat `lst` een lege lijst is. Zorg ervoor dat vrijkomend geheugen correct wordt vrijgegeven.

   (d) Geef een functie van type `List addInOrder(List lst, int n)` die getal `n` toevoegt aan de zwak stijgend geordende lijst `lst`, zodanig dat de lijst geordend blijft.

2. (20 punt) Onderstaande C-code definieert functies voor de implementatie van een priority queue mbv. een heap. Echter, er zitten 5 fouten in de code waardoor functies niet naar behoren werken. Vind deze fouten, geef aan wat er fout aan is en verbeter ze.

```
1   int heapArray[1000];
2   int front = 1;
3
4   void swap(int *x, int *y) {
5     int h = *x;
6     *x = *y;
7     *y = h;
8   }
9
10  void upheap(int n) {
11    if ( n>1 && heapArray[n] > heapArray[n/2] ) {
12      swap(heapArray[n],heapArray[n/2]);
13    }
14  }
15
16  void downheap(int n) {
17    if ( 2*n + 1 < front) {
18      if (heapArray[n] < heapArray[2*n+1]) {
19        swap(&heapArray[n],&heapArray[2*n+1]);
20        downheap(2*n+1);
21      } else if ( heapArray[n] < heapArray[2*n]) {
22        swap(&heapArray[n],&heapArray[2*n]);
23        downheap(2*n);
24      }
25    } else if ( front == 2*n+1 && heapArray[n] < heapArray[2*n] ) {
26      swap(&heapArray[n],&heapArray[2*n]);
27      downheap (2*n);
28    }
29  }
30
31  void enqueue (int n) {
32    assert((1 <= front) && (front < 1000));
33    heapArray[front] = n;
34    upheap(front);
35  }
36
37  int removeMax() {
38    assert((1 < front) && (front <= 1000));
39    int n = heapArray[1];
40    front--;
41    heapArray[1] = heapArray[front];
42    downheap(1);
43    return n;
44  }
```

3. (25 punt) Deze opgave gaat over tries.

   (a) Geef een definitie van het begrip *suffix trie* van een string $T$.

   (b) Geef de suffix trie van de string `ATGAGCGAGC$`. Je mag ook de gecomprimeerde of de compacte suffix trie geven.

   (c) Gegeven is de suffix trie van string $T$. Geef een algoritme in pseudocode dat nagaat of een patroon $P$ voorkomt in $T$. Geef ook de tijdscomplexiteit van het algoritme, en beargumenteer deze.

4. (20 punt) Gegeven is het volgende algoritme voor breadth-first search:

```
algoritme BFS(G,v)
    invoer ongerichte samenhangende graaf G met knoop v
    resultaat labeling van de kanten van G met NIEUW en OUD;
        de kanten met label NIEUW vormen een opspannende boom,
        en alle knopen worden bezocht (en gelabeld met BEZOCHT)
geef v het label BEZOCHT en alle knopen ≠ v het label LEEG
geef alle kanten het label LEEG
maak een lege queue Q
while Q niet leeg do
    u ← dequeue()
    forall e incident met u do
        if e heeft label LEEG then
            w ← de andere knoop incident met e
            if w heeft label LEEG then
                geef e het label NIEUW
                geef w het label BEZOCHT
            else
                geef e het label OUD
```

Naar aanleiding van dit algoritme enkele vragen.

   (a) Wanneer is een graaf samenhangend (*connected*)?

   (b) Wat is een opspannende boom (*spanning tree*)?

   (c) Het gegeven algoritme is onvolledig. Geef aan wat er niet aan klopt en verbeter het.

   (d) B is de opspannende boom van kanten met label NIEUW. Beargumenteer dat na uitvoering van het verbeterde algoritme het volgende geldt:

   > elk pad P in B dat loopt van v naar een andere knoop w is *minimaal* in het aantal kanten, dwz. er is geen pad in G van v naar w met minder kanten dan P.