*Functies mogen in pseudo-code gegeven worden.*
*Waar expliciet C++ code gevraagd wordt, geeft u deze, maar Pascal mag ook.*

**1.** Beschouw hashen in een zgn. 'open' hashtabel met twee hash-functies $h$ en $p$. Het $i+1$-ste bezochte adres $h(K,i)$ is zoals gewoonlijk $h(K) - i \cdot p(K)$ (modulo $M$).

**a)** Welke twee punten zijn belangrijk bij de keuze van de adresfunctie $h$ ? Waarop moeten we letten bij de keuze van de stapfunctie $p$?

**b)** Welke twee soorten clustering onderscheiden we bij hashen met open adressering? Geef een korte beschrijving. Wanneer (bij welke methoden) treden ze op?

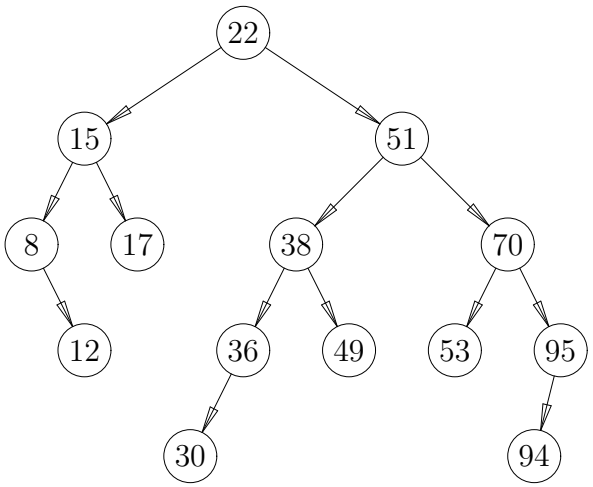
**c)** In een tabel `T[0..10]`, dus $M = 11$, worden achtereenvolgens de sleutels 10, 22, 31, 4, 15, 28, 17, 88, 59 geplaatst, met adresfunctie $h(K) = K \bmod M$ en lineair hashen (stapgrootte 1).
Laat zien welke tabel ontstaat, maar geef op een overzichtelijke manier ook alle plekken waar de sleutels geprobeerd worden.

**d)** Idem, nu met een stapfunctie $p(K) = 1 + (K \bmod (M-1))$.

**2.** De fibonacci getallen worden gegeven door $F_0 = 0$, $F_1 = 1$, en $F_i = F_{i-1} + F_{i-2}$, voor $i \geq 2$. Er geldt $F_i \approx \phi^i$, met $\phi = \frac{1}{2}(1 + \sqrt{5}) \approx 1.62$.

**a)** Hoeveel knopen bevat een AVL-boom van hoogte $h \geq 1$ minimaal, en hoeveel maximaal? (Hierbij heeft de boom met één knoop hoogte $h = 1$.)
Wat kunnen we hieruit afleiden over de hoogte van een AVL-boom met $n$ knopen? (Geef voldoende uitleg.)

**b)** Construeer een AVL-boom door, uitgaande van een lege boom, achtereenvolgens de volgende getallen toe te voegen: 51, 17, 30, 53, 94, 70, 22, 49, 12, 8, 38 en 36. Uiteraard dienen de bomen na iedere toevoeging de AVL-eigenschap te bezitten. Geef via tussenresultaten en een korte toelichting duidelijk aan hoe je aan je antwoord komt.




**c)** **i.** Wat is het resultaat bij verwijdering van knoop 51 uit bovenstaande AVL-boom.
**ii.** En wat bij verwijdering van knoop 17 uit dezelfde (originele) boom.
Geef ook hierbij aan hoe je aan je antwoord komt.

**3.** **a)** Geef C++ code voor een *adjacency list* representatie voor een graaf met gewichten op de takken.

Op het college is het algoritme van Dijksta gepresenteerd in een implementatie waarbij voor elke knoop een *kandidaat-tak* wordt bijgehouden. In pseudo++:

```
kies beginknoop x ;                                                    1
laat de boom uit knoop x bestaan ;                                     2
de kandidaat-tak voor elke andere knoop z wordt de tak (x,z) ;         3
while ( nog niet alle knopen in de boom )                              4
{     kies knoop y buiten boom met optimale kandidaat-tak ;            5
      voeg y toe aan de knopen van de boom ;                           6
      pas de kandidaat-tak aan voor elke knoop z buiten de boom ;      7
}                                                                      8
```

**b)** Beschrijf hoe de kandidaat-takken vervangen worden (regel 7). Geef een efficiënte implementatie in pseudo-code, uitgaande van een representatie mbv. *adjacency lists*.

**c)** Illustreer de werking van het algoritme aan de hand van de graaf met onderstaande *adjacency matrix*. ('x' geeft aan dat er geen tak loopt, '.' op de diagonaal).
Begin in knoop 1.

| | 1 | 2 | 3 | 4 | 5 | naar |
|---|---|---|---|---|---|---|
| van 1 | . | 10 | x | x | 5 | |
| 2 | x | . | 1 | x | 2 | |
| 3 | x | x | . | 4 | x | |
| 4 | 7 | x | 6 | . | x | |
| 5 | x | 3 | 9 | 2 | . | |

Geef de stappen weer in een tabel als hieronder geschetst, 'stap 0' toont de berekeningen in de initialisatiefase (tm. regel 3), de overige stappen steeds de berekeningen in de **while**-lus.
(Bezoekers van het werkcollege mogen de tabel ook spiegelen zoals daar de gewoonte was...)

| | toegevoegde | | kand-waarde bij knoop | | | | |
|---|---|---|---|---|---|---|---|
| | knoop | (afstand) | 1 | 2 | 3 | 4 | 5 |
| stap 0 | 1 | 0 | .. | .. | .. | .. | .. |
| 1 | .. | .. | .. | .. | .. | .. | .. |
| 2 | .. | .. | .. | .. | .. | .. | .. |
| ⋮ | | | | | | | |

**d)** Bij het algoritme wordt een *priority queue* gebruikt. Hoe zien de elementen van deze priority queue eruit, dwz. welke informatie bevatten ze hier?

Geef precies aan welke operaties op de priority queue uitgevoerd worden tijdens het algoritme. Verwijs naar de regelnummers.

**e)** Hoeveel maal wordt (in het slechtste geval) elk van deze operaties uitgevoerd, uitgedrukt in het aantal knopen $n$ en het aantal takken $e$ van de graaf?

Wat betekent dit voor de complexiteit als de priority queue als *heap* geïmplementeerd wordt?

**f)** Wat is het voordeel van de *fibonacci queue* ?

**4.** De methode van *Knuth-Morris-Pratt* wordt gebruikt om een patroon P[1..m] in een tekst T[1..Lengte] te zoeken. Daartoe worden *failure-links* opgesteld.

**a)** Stel de failure link bij positie i wijst naar positie k in P: Flink[i] = k.
Wat geldt dan voor de woorden P[1]...P[i] en P[1]...P[k] ?
In het bijzonder, wanneer geldt Flink[i] = 0 ?

**b)** Bepaal de *failure-links* voor het patroon P = `ABAABABA`.

**c)** We zoeken naar P in de tekst T = `ABAB AABA BBAB AA`.... (hier staan de spaties voor de leesbaarheid). Geef nauwkeurig aan hoe het zoeken volgens de KMP-methode gebeurt. Welke letters worden telkens met elkaar vergeleken?

**d)** Hoe zien de *failure-links* er voor Nederlandse woorden in het algemeen uit?
Wanneer heb je daar voordeel van bij het zoeken, tov. het 'naïve' algoritme?