5 Opdracht 4: Geavanceerde structuren en output

Het wordt tijd om te starten aan de geavanceerde datastructuren.

1. Kies een geavanceerde datastructuur (heap, hashmap, 2-3-boom, 2-3-4-boom, rood-zwart-boom). De keuze bepaalt je maximumscore. Ze staan in volgorde van moeilijkheid. Let op met de rood-zwart-boom. Die is bijzonder moeilijk en moet net als alle andere geïmplementeerd worden volgens het algoritme uit de cursus (let dus op met algoritmes die je online vindt, want die zijn meestal niet zoals in de cursus). Bij alle boomimplementaties gaan we bij het verwijderen telkens op zoek naar de **inorder successor (dus nooit de predecessor)**. Binnenkort ga je in groep werken. In dezelfde groep mag een geavanceerde structuur meerdere keren voorkomen. Let wel op met het individuele deel dat je zeker geen code samen schrijft. Zeker binnen een groep zal de plagiaatsoftware dat er direct uit halen (over de groepen heen ook natuurlijk maar het valt des te harder op binnen een groep).

   De basisstructuren (stack, queue, dubbelgelinkte ketting en binaire zoekboom) staan op 1 van de 5 punten van het individuele deel van de toepassingsopdracht staan. Voor de overige 4 punten maak je zelf een keuze:

ADT/Klassenaam	**/4**	extra info
Heap	1.5	Je kiest voor een link-based implementatie, dus geen array implementatie. De eerste parameter van de constructor is een boolean die aangeeft of het een maxheap is (True). Als die False is, dan is het een minHeap. Default staat die op True.
Hashmap	1.5	De hashmap is instelbaar via een parameter zodat die kan switchen tussen een hashmap met linear probing, quadratic probing en een hashmap met separate chaining (zorg ervoor dat de gebruikte implementatie voor seperate chaining makkelijk kan aangepast worden, begin bijvoorbeeld met de dubbel gelinkte ketting, maar zorg dat die makkelijk kan aangepast worden naar een andere implementatie). De hashmap moet niet in de loop van je programma aanpasbaar zijn, enkel bij het opstarten moet je makkelijk kunnen switchen.
TwoThreeTree	2	
TwoThreeFourtree	3	Hier is het mogelijk dat je keuzes hebt. Als er twee siblings een item kunnen uitlenen, kies je voor de linkse. Als je moet mergen en je kan kiezen naar waar je merget, kies dan een left merge (d.w.z. de te mergen node komt samen met zijn meest linkse parent en zijn linkersibling). Denk eraan dat je altijd eerst een redistribute gaat proberen (ook al is dat een rechtse).
RedBlackTree	4	Idem als hierboven

2. Stel een contract op voor die geavanceerde structuur. Gebruik de benamingen uit de cursus (dus bv insertItem voor 2-3-, 2-3-4- en rood-zwart-bomen (en analoog deleteItem, retrieveItem, inorderTraverse met parameters analoog aan die van de bst), heapInsert voor een heap en tableInsert voor een hashmap).

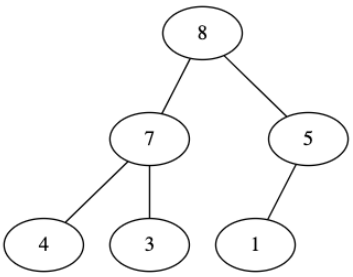
3. Implementeer een save methode die de geavanceerde structuur voorstelt als een dictionary in Python (zie verder voor de correcte output).

4. Implementeer een load methode die een leeg object aanmaakt en vult met een dict. De dict is hetzelfde als de output van een save.

5. Vanaf nu kan je ook beginnen aan de implementatie van je geavanceerde gegevensstructuur (volgens het algoritme uit de cursus!). Het indienen gebeurt via INGinious.

5.1 Een heap

Volgende heap ...



... wordt omgezet naar volgende dict via de methode save. Je toont telkens enkel de zoeksleutel/priority. Het type heap (min of max) mag je bepalen op basis van de relatie tussen de root en zijn kinderen. Als je het niet vindt (bij 1 item bijvoorbeeld dan is het default een maxheap).

```
{
  'root': 8,
  'children':
    [
      {
        'root': 7,
        'children':
          [{'root': 4},{'root': 3}]
      },
      {
        'root': 5,
        'children':
          [{'root': 1},None]
      },
    ]
}
```

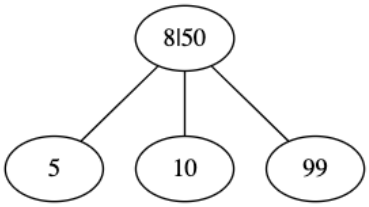
5.2 Een hashmap

Een hashmap wordt voorgesteld door een dictionary en een Python list met None op de lege plekken. De lengte van de hashmap wordt bepaald door de lengte van die list. Bij seperate chaining zijn de elementen van die list opnieuw lijsten (gebruik je eigen ketting daarvoor).

```
{'type': 'lin', 'items':[10,None,2,3,None]}
{'type': 'quad', 'items':[10,None,2,3,None]}
{'type': 'sep', 'items':[[10],None,[2,3],None,None]}
```

5.3 Een 2-3-boom

Volgende 2-3-boom ...

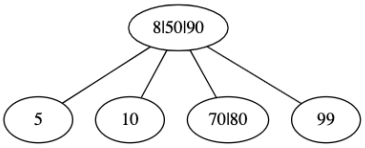


... wordt omgezet naar volgende dict via de methode save. Je toont telkens enkel de zoeksleutel.

```
{
  'root': [8,50],
  'children':
            [{'root': [5]},{'root': [10]},{'root': [99]}]
}
```

5.4 Een 2-3-4-boom

Volgende 2-3-4-boom ...

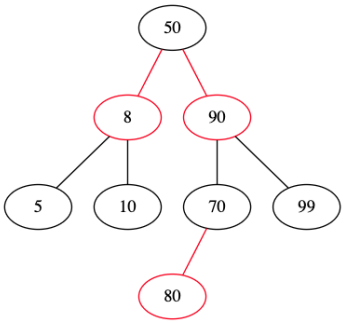


... wordt omgezet naar volgende dict via de methode save. Je toont telkens enkel de zoeksleutel.

```
{
  'root': [8,50,90],
  'children':
            [{'root': [5]},{'root': [10]},{'root': [70,80]},{'root': [99]}]
}
```

5.5 Een rood-zwart-boom

Volgende rood-zwart-boom ...



... wordt omgezet naar volgende dict via de methode save. Je toont telkens enkel de zoeksleutel.

```
{
  'root': 50,
  'color': 'black',
  'children':
    [
      {
        'root': 8,
        'color': 'red',
        'children':
          [{'root': 5,'color': 'black',},{'root':10,'color': 'black',}]
      },
      {
        'root': 90,
        'color': 'red',
        'children':
          [
            {
              'root': 70,
              'color': 'black',
              'children':
                [
                  {'root': 80,'color': 'red',},
                  None
                ]
            },
            {'root': 99,'color': 'black',}
          ]
      }
    ]
}
```

Merk op dat we de nodes kleuren en niet de edges.