Oefenzittingen C: Deel 3

Informatica werktuigen

Academiejaar 2014-2015

1 Binaire Zoekbomen

Een binaire zoekboom is een boom waarbij de huidige knoop steeds:

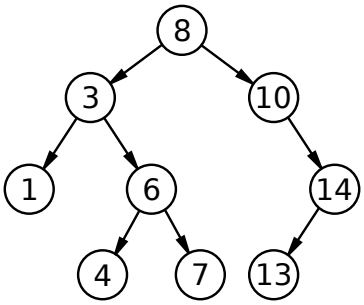
- *groter* is dan eender welk kind in de *linker* deelboom.
- *kleiner* is dan eender welk kind de *rechter* deelboom

Fig. 1 geeft een voorbeeld van een binaire zoekboom. Implementeer je eigen binaire zoekboom door gebruik te maken van de volgende structs:

```
1  struct Tree
2  {
3      struct TreeNode* root;
4  };
5
6  struct TreeNode
7  {
8      int value;
9      struct TreeNode* left_child;
10     struct TreeNode* right_child;
11 };
```

De volgende functies moeten voorzien worden:

- `struct Tree* tree_create()`: Maak een lege tree aan.
- `void tree_insert(struct Tree* tree, int value)`: Voeg value op de juiste plek toe aan de tree. Als je in de tree van Fig. 1 bijvoorbeeld de waarde 5 toevoegt, zal deze het rechter kind van de node met waarde 4 worden.
- `void tree_print(struct Tree* tree)`: Print de waarden in de tree van groot naar klein af. Dit is het eenvoudigste te implementeren met een recursieve functie.



Figuur 1: Een voorbeeld van een binaire zoekboom waarbij (bijvoorbeeld) de elementen 8, 3, 6, 10, 14, 1 en 13 achtereenvolgens werden toegevoegd.

bron: `wikipedia.org`

2 Map

Een map is een datastructuur die "sleutels" afbeeldt op "waarden"; in Python wordt dit een "dictionary" genoemd. Implementeer een map in C die strings afbeeldt op integers. Baseer je hiervoor op een enkelvoudig gelinkte lijst waarin elke node een string en een integer bijhoudt. Je implementatie moet drie functies ondersteunen:

- `struct Map* map_create()`: Maak een lege map aan.
- `void map_insert(struct Map* map, const char* key, int value)`: Voeg een node toe aan de map met de gegeven key en value. Het is de bedoeling dat je een kopie van de gegeven key opslaat (maak hier dus een nieuw stuk geheugen voor aan). Als er al een node bestaat met de gegeven key moet de bestaande waarde vervangen worden door de nieuwe.
- `int map_get(struct Map* map, const char* key, int* value)`: Zoek de node op met de gegeven key als sleutel en sla de corresponderende waarde op in value. Als er geen node bestaat met de gegeven key geeft de functie 0 terug, anders 1.

Ter illustratie: de volgende Python code:

```
a_map = {}
a_map['foo'] = 5
a_map['bar'] = 10

try:
    value = a_map['foo']
    print(value)
except KeyError:
    print('KeyError')
```

Wordt vertaald in de volgende C code:

```
struct Map* a_map = map_create();
map_insert(a_map, "foo", 5);
map_insert(a_map, "bar", 10);

int value;
if (map_get(map, "foo", &value))
    printf("%d\n", value);
else
    puts("KeyError");
```

3 Groter voorbeeld

Een map wordt vaak gebruikt om data aan elkaar te relateren. Je zou de map die je in de vorige oefening hebt gemaakt bijvoorbeeld kunnen gebruiken om de naam van een persoon te associëren met zijn of haar leeftijd. De volgende Python code maakt zo een map aan en definieert een aantal functies die gebruik maken van deze map. Vertaal deze code naar C gebruik makende van de map die je in de vorige oefening hebt gemaakt. Je zult ook een lijst van strings nodig hebben; pas hiervoor het voorbeeld uit de les aan.

```
def get_adults(age_map):
    adults = []

    for name in age_map:
        if age_map[name] >= 18:
            adults.append(name)

    return adults

def birthday(age_map, name):
    try:
        age_map[name] += 1
    except KeyError:
        age_map[name] = 0

if __name__ == '__main__':
    age_map = {
        'Jos':    18,
        'Marie':  25,
        'Tim':     6,
        'Toon':   12,
        'Sofie':  68,
        'Tobias': 35,
        'Jessica': 3
    }

    print(age_map)
    print('All adults:', get_adults(age_map))
    birthday(age_map, 'Marie')
    birthday(age_map, 'Sara')
    print(age_map)
```