Programmering og Problemløsning, 2019

Træstrukturer — Part II

Martin Elsman

Datalogisk Institut
Københavns Universitet
DIKU

31. oktober, 2019

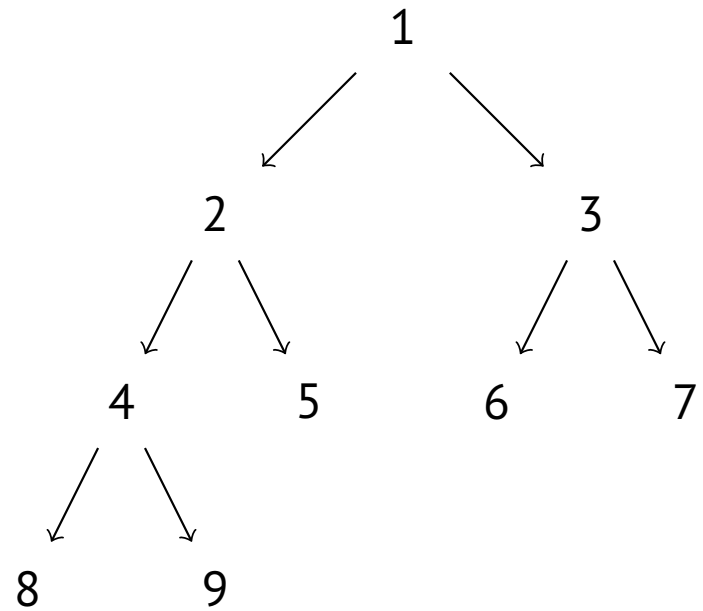
Træer og Gennemløb

Emner for i dag:

1. **Terminologi omkring træer.**
   Knuder, kanter, rødder, binære træer
2. **Trægennemløb.**
   Dybde-først gennemløb og bredde-først gennemløb

Træterminologi

- Et træ består af **knuder** forbundet med **ordnede kanter**.
- En knude har højst en indgående kant (forælder).
- En knude kan have 0 eller flere udgående kanter (børn).
- En knude uden børn kaldes et **blad**, og en knude uden forældre kaldes en **rod**.
- Normalt tegnes træer med forældre ovenover børn.
- Et **binært træ** har præcis en rod og hver knude har højst to børn.



Gennemløb af træer.

Et *gennemløb* (eng. *traversal*) af et træ er et besøg af alle knuderne i træet.

Forskellige slags gennemløb:

- **Dybde-først gennemløb**: besøg alle knuderne i venstre undertræ af en knude før knuderne i højre undertræ. Der er tre undertyper af dybde-først gennemløb:
  1. *Præordens gennemløb*: knude før børn
  2. *Postordens gennemløb*: knude efter børn
  3. *Inordens gennemløb*: knude mellem børn
- **Bredde-først gennemløb**: besøg knuder i rækkefølge efter afstand til roden, og knuder med samme afstand fra venstre mod højre.

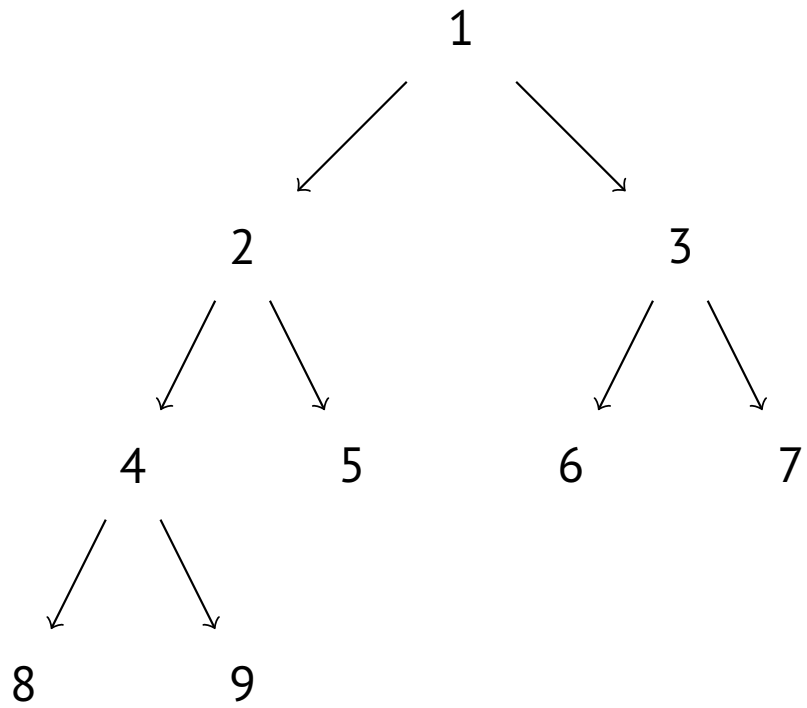
Eksempler på trægennemløb

Præordens gennemløb: 1 2 4 8 9 5 3 6 7
(knude før børn)

Postordens gennemløb: 8 9 4 5 2 6 7 3 1
(knude efter børn)

Inordens gennemløb: 8 4 9 2 5 1 6 3 7
(knude mellem børn)

Bredde-først gennemløb: 1 2 3 4 5 6 7 8 9



Implementation af præordens gennemløb

Vi arbejder med følgende træstruktur:

```
type 'a t = L | T of 'a t * 'a * 'a t
let E e = T(L,e,L)
```

Den simple version — bruger @

```
let rec preorder (t: 'a t) : 'a list =
  match t with      // visit node before children
    | L -> []
    | T(l,e,r) -> [e] @ preorder l @ preorder r
```

Effektiv version — uden brug af @

```
let rec preorder_acc (acc:'a list) (t: 'a t) : 'a list =
  match t with      // node before children
    | L -> acc
    | T(l,e,r) -> e :: preorder_acc (preorder_acc acc r) l
```

Implementation af postordens gennemløb

Vi arbejder med følgende træstruktur:

```
type 'a t = L | T of 'a t * 'a * 'a t
let E e = T(L,e,L)
```

Den simple version — bruger @

```
let rec postorder (t: 'a t) : 'a list =
  match t with      // visit node after children
    | L -> []
    | T(l,e,r) -> postorder l @ postorder r @ [e]
```

Effektiv version — uden brug af @

```
let rec postorder_acc (acc:'a list) (t: 'a t) : 'a list =
  match t with      // node after children
    | L -> acc
    | T(l,e,r) -> postorder_acc (postorder_acc (e::acc) r) l
```

Implementation af inordens gennemløb

Vi arbejder med følgende træstruktur:

```
type 'a t = L | T of 'a t * 'a * 'a t
let E e = T(L,e,L)
```

Den simple version — bruger @

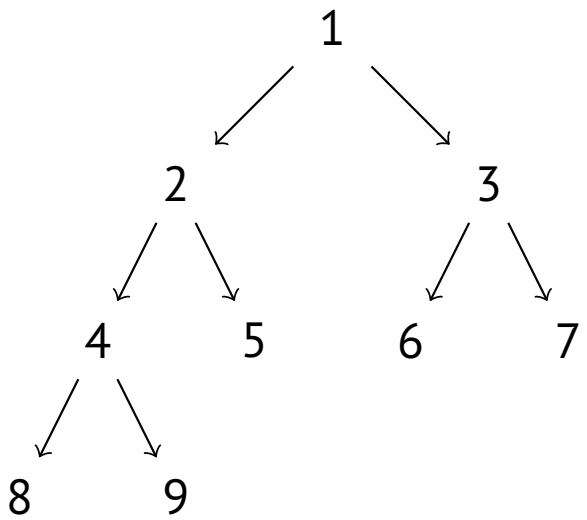
```
let rec inorder (t: 'a t) : 'a list =
  match t with      // visit node between children
    | L -> []
    | T(l,e,r) -> inorder l @ [e] @ inorder r
```

Effektiv version — uden brug af @

```
let rec inorder_acc (acc:'a list) (t: 'a t) : 'a list =
  match t with      // visit node between children
    | L -> acc
    | T(l,e,r) -> inorder_acc (e :: inorder_acc acc r) l
```

Simpel bredde-først implementation — bruger @

```
let breathfirst t =
  let rec bF ts =
    match ts with
      | [] -> []
      | L :: ts -> bF ts
      | T (l,a,r) :: ts ->
         a :: bF (ts @ [l; r])
  bF [t]
```



Bemærk

- Hjælpefunktionen bF laver et bredde-først gennemløb af en liste af træer.
- Listen fungerer som en kø: Vi tager ud fra fronten og sætter ind bagest.

Spørgsmål

- Nogle gode ideer til hvordan vi kan undgå brug af @?

Løsningen er naturligvis at benytte vores effektive kø-modul!

```
module Queue // content of queue.fsi

type 'a queue                                  // FIFO
val empty : unit -> 'a queue
val insert  : 'a queue -> 'a -> 'a queue
val remove  : 'a queue -> ('a * 'a queue) option
```

Den nye effektive implementation:

```
let breathfirst_good (t:'a t) : 'a list =
  let rec bF (q:'a t Queue.queue) : 'a list =
    match Queue.remove q with
      | None -> []
      | Some(L,q) -> bF q
      | Some(T (l,a,r), q) ->
          a :: bF (Queue.insert (Queue.insert q l) r)
  bF (Queue.insert (Queue.empty()) t)
```

Bredde-først gennemløb af generelle træer

Gennemløb kan generaliseres (pånær inorder-gennemløb) til ikke-binære træer.

Eksempel på generel bredde-først gennemløb af et generelt træ

```
type 'a tg = Lg | Tg of 'a * 'a tg list
let breathfirst_gen (t:'a tg) : 'a list =
  let rec bF (q:'a tg list Queue.queue) : 'a list =
    match Queue.remove q with
      | None -> []
      | Some(gts,q) -> bFs q gts
  and bFs (q:'a tg list Queue.queue) tgs : 'a list =
    match tgs with
      | [] -> bF q
      | Lg::rest -> bFs q rest
      | Tg (a,tgs)::rest -> a :: bFs (Queue.insert q tgs) rest
  bF (Queue.insert (Queue.empty()) [t])
```

Bemærk

- Hjælpefunktionen `ins` benyttes til at indsætte en liste af elementer i køen.