Table of Contents

TP 7 - Programmation pour la préparation à l'agrégation maths option info

TP 5 : Algorithmes gloutons et files de priorité.

- En OCaml.

```
In [1]: let print = Printf.printf;;
        Sys.command "ocaml -version";;

Out[1]: val print : ('a, out_channel, unit) format → 'a = <fun>

        The OCaml toplevel, version 4.04.2

Out[1]: - : int = 0
```

```
In [2]: print_endline

Out[2]: - : string → unit = <fun>
```

Algorithme de Dijkstra - avec des files non mutables

Déjà vu, on le retraite ici.

Files de priorité min

In [3]:
```
(* file de priorité version non-mutable *)
type 'a priopqueue = (int * 'a) list;;
```

Out[3]:
```
type 'a priopqueue = (int * 'a) list
```

In [4]:
```
(* file vide *)
let vide : 'a priopqueue = [ ];;
```

Out[4]:
```
val vide : 'a priopqueue = []
```

In [5]:
```
(* [inserer x clef q] insere l'element [x] dans la file [q]
   avec le clef [x], et renvoie la nouvelle file ainsi créée.
   Termine avec une exception si la file contient déjà [x]  *)
let inserer (x:'a) (clef:int) (q:'a priopqueue) : 'a priopqueue =
   if List.exists (fun (_, v) -> x = v) q
   then failwith "l'element est déjà dans la file"
   else (clef,x) :: q
;;
```

Out[5]:
```
val inserer : 'a → int → 'a priopqueue → 'a priopqueue = <fun>
```

In [6]:
```
(* [est_vide q] teste si la file [q] est vide *)
let est_vide (q:'a priopqueue) : bool = (q = [ ]);;
```

Out[6]:
```
val est_vide : 'a priopqueue → bool = <fun>
```

In [7]:
```
(* [trouve_min_aux min_val min_clef q] renvoie un couple de clef minimale
    dans (min_val,min_clef)::q *)
let rec trouve_min_aux (min_val:'a) (min_clef:int) (q:'a priopqueue) : int * 'a =
   match q with
   | [ ] -> (min_clef, min_val)
   | (clef, _) :: q when clef > min_clef -> trouve_min_aux min_val min_clef q
   | (clef, v) :: q -> trouve_min_aux v clef q
;;
```

Out[7]:
```
val trouve_min_aux : 'a → int → 'a priopqueue → int * 'a = <fun>
```

In [8]:
```
(* [trouve_min q] renvoie un élément de clef minimale la file [q].
   Lance une exception si la liste est vide *)
let trouve_min (q:'a priopqueue) : 'a =
   match q with
   | [ ] -> failwith "trouve_min: la file est vide"
   | (clef, v) :: q -> snd (trouve_min_aux v clef q)
;;
```

Out[8]:
```
val trouve_min : 'a priopqueue → 'a = <fun>
```

In [9]:
```
let _ = trouve_min (inserer '1' 1 (inserer '2' 2 (inserer '3' 3 vide)));;
let _ = trouve_min (inserer '1' 4 (inserer '2' 2 (inserer '3' 3 vide)));;
```

Out[9]:
```
- : char = '1'
```

Out[9]:
```
- : char = '2'
```

In [10]:
```
(* [supprime v q] renvoie une file contenant les éléments de [q], sauf [x].
   [x] doit apparaitre une et une seule fois dans la file. *)
let rec supprime (x:'a) (q:'a priopqueue) : 'a priopqueue =
    match q with
    | [ ] -> [ ]
    | (_, v) :: q when v=x -> q
    | (clef, v) :: q -> (clef, v) :: (supprime x q)
;;
```

Out[10]:
```
val supprime : 'a → 'a priopqueue → 'a priopqueue = <fun>
```

In [11]:
```
(* [extraire_min q] renvoie un élément de q, de clef minimal,
   ainsi que la nouvelle file obtenue en supprimant cet
   élément; termine avec une exception si la file est vide  *)
let extraire_min (q:'a priopqueue) : 'a * 'a priopqueue =
    if q = [ ] then
        failwith "extraire_min: file vide"
    else
        let min = trouve_min q in
        (min, supprime min q)
;;
```

Out[11]:
```
val extraire_min : 'a priopqueue → 'a * 'a priopqueue = <fun>
```

In [12]:
```
let _ = extraire_min (inserer '1' 1 (inserer '2' 2 (inserer '3' 3 vide)));;
let _ = extraire_min (inserer '1' 4 (inserer '2' 2 (inserer '3' 3 vide)));;
```

Out[12]:
```
- : char * char priopqueue = ('1', [(2, '2'); (3, '3')])
```

Out[12]:
```
- : char * char priopqueue = ('2', [(4, '1'); (3, '3')])
```

In [13]:
```
(* [diminuer_clef q clef x] modifie la clef de l'élément [x]
   dans la file q en lui associant la nouvelle clef [clef], qui
   doit être inferieur à la clef actuelle de [x].
   Termine avec une exception si la file ne contient pas [x]  *)
let rec diminuer_clef (x:'a) (clef:int) (q:'a priopqueue) : 'a priopqueue =
    match q with
    | [ ] -> failwith "diminuer_clef : l'élément n'est pas présent"
    | (_, v) :: q when v=x -> (clef, x) :: q
    | (c, v) :: q -> (c, v) :: diminuer_clef x clef q
;;
```

Out[13]:
```
val diminuer_clef : 'a → int → 'a priopqueue → 'a priopqueue = <fun>
```

In [14]:
```
let f =  inserer '1' 1 (inserer '2' 2 (inserer '3' 3 vide));;
let _ = diminuer_clef '3' 0 f;;
let _ = diminuer_clef '2' 0 f;;
```

Out[14]:
```
val f : char priopqueue = [(1, '1'); (2, '2'); (3, '3')]
```

Out[14]:
```
- : char priopqueue = [(1, '1'); (2, '2'); (0, '3')]
```

Out[14]:
```
- : char priopqueue = [(1, '1'); (0, '2'); (3, '3')]
```

Graphe par tableau de listes d'adjacence

In [15]:
```
type sommet = int;;
type graph = {
    taille: int; (* les sommets sont des entiers entre 0 et taille-1 *)
    adj: (int * sommet) list array;
    entree: sommet
};;
```

Out[15]:
```
type sommet = int
```

Out[15]:
```
type graph = {
  taille : int;
  adj : (int * sommet) list array;
  entree : sommet;
}
```

Ce qui suit est purement optionnel, ce n'était pas demandé, ne vous embêtez pas à chercher à tout comprendre, c'est simplement pour visualiser les graphes et les afficher ensuite.

In [16]:
```
let print = Printf.fprintf;;

let dot outname (g:graph) (bold:(int*int) list) : unit =
   let f = open_out (outname ^ ".dot") in
   print f "digraph G {\n";
   for i=0 to g.taille-1 do
      print f "  som%d [label=\"%d\"];\n" i i
   done;
   for i=0 to g.taille-1 do
      List.iter (fun (c,j) ->
         let option = if List.mem (i,j) bold then ",style=bold" else "" in
         print f "  som%d -> som%d [label=\"%d\"%s];\n" i j c option
      ) g.adj.(i);
   done;
   print f "}\n";
   close_out f
;;

let dot2svg outname =
   Sys.command (Printf.sprintf "dot -Tsvg %s.dot > %s.svg" outname outname);
```

Out[16]:
```
val print : out_channel → ('a, out_channel, unit) format → 'a = <fun>
```

Out[16]:
```
val dot : string → graph → (int * int) list → unit = <fun>
```

Out[16]:
```
val dot2svg : string → int = <fun>
```

Exemple de visualisation de graphe

In [17]:
```
let s = 0
and a = 1
and b = 2
and c = 3
and d = 4;;
let g1 = {
   taille = 5;
   entree = s;
   adj = [|
      [(2,a); (4,b); (2,c)]; (* adj(s) *)
      [(1,d)]; (* adj(A) *)
      [(4,d)]; (* adj(B) *)
      [(1,b)]; (* adj(C) *)
      [ ]; (* adj(D) *)
   |]
};;
```

Out[17]:
```
val s : int = 0
val a : int = 1
val b : int = 2
val c : int = 3
val d : int = 4
```

Out[17]:
```
val g1 : graph =
  {taille = 5;
   adj = [|[(2, 1); (4, 2); (2, 3)]; [(1, 4)]; [(4, 4)]; [(1, 2)]; []|];
   entree = 0}
```

In [18]:
```
let _ = dot "TP7__g1" g1 [ ];;
dot2svg "TP7__g1";;
```

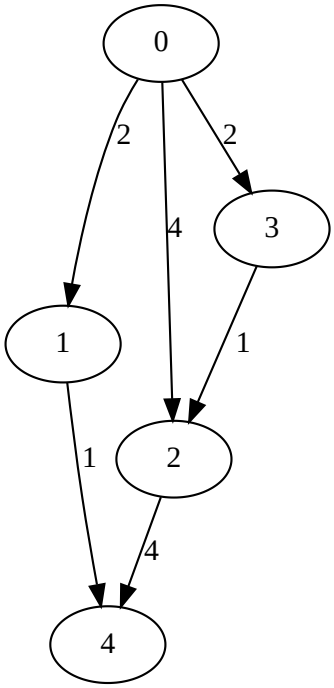
Out[18]:
```
- : unit = ()
```

Out[18]:
```
- : int = 0
```

In [19]:
```
Sys.command "cat TP7__g1.dot";;
```

```
digraph G {
  som0 [label="0"];
  som1 [label="1"];
  som2 [label="2"];
  som3 [label="3"];
  som4 [label="4"];
  som0 → som1 [label="2"];
  som0 → som2 [label="4"];
  som0 → som3 [label="2"];
  som1 → som4 [label="1"];
  som2 → som4 [label="4"];
  som3 → som2 [label="1"];
}
```

Out[19]:
```
- : int = 0
```



Le second argument permet d'afficher un certain chemin :

In [20]:
```
let _ = dot "TP7__g2" g1 [(0,3);(3,2);(2,4)];;
```

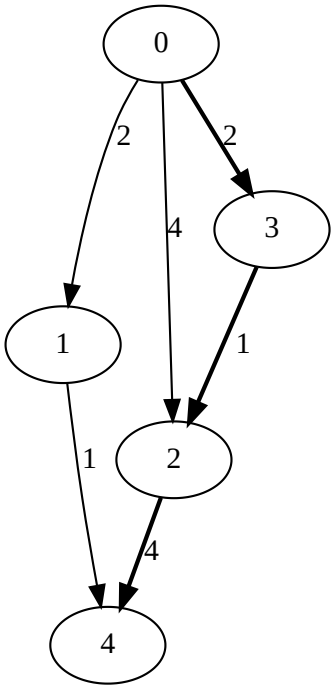
Out[20]:
```
- : unit = ()
```

In [21]:
```
Sys.command "cat TP7__g2.dot";;
dot2svg "TP7__g2";;
```

```
digraph G {
  som0 [label="0"];
  som1 [label="1"];
  som2 [label="2"];
  som3 [label="3"];
  som4 [label="4"];
  som0 → som1 [label="2"];
  som0 → som2 [label="4"];
  som0 → som3 [label="2",style=bold];
  som1 → som4 [label="1"];
  som2 → som4 [label="4",style=bold];
  som3 → som2 [label="1",style=bold];
}
```

Out[21]:
```
- : int = 0
```

Out[21]:
```
- : int = 0
```



Dijkstra

Une fois qu'on dispose de tout ça, écrire l'algorithme de Dijkstra est relativement rapide.

- Voir ce site (https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html) pour de belles visualisations de l'algorithme de Dijkstra.
- Et cette page (https://jilljenn.github.io/tryalgo/_modules/tryalgo/dijkstra.html#dijkstra) pour une implémentation propre en Python (lien direct vers le code (https://jilljenn.github.io/tryalgo/_modules/tryalgo/dijkstra.html#dijkstra)).

In [22]:
```
let dijkstra g =
    let q = ref vide in
    let dist = Array.init g.taille (fun i ->
        if i=g.entree then 0 else max_int
    ) in
    for i=0 to g.taille - 1 do (* initialisation de la file *)
        q := inserer i dist.(i) !q
    done;
    while not (est_vide !q) do
        let (x, q') = extraire_min !q in
        q := q';  (* ne pas oublier de mettre à jour la file *)
        (* on regarde les adjacents de x *)
        List.iter (fun (c,y) ->
            if dist.(y) > dist.(x) + c
            then begin
                dist.(y) <- dist.(x) + c;
                q := diminuer_clef y dist.(y) !q
            end
        ) g.adj.(x)
    done;
    dist
;;
```

Out[22]:
```
val dijkstra : graph → int array = <fun>
```

In [23]:
```
let _ = dijkstra g1
```

Out[23]:
```
- : int array = [|0; 2; 3; 2; 3|]
```

Contre-exemples

Trouver des cas simples faisant échouer l'algorithme si une des hypothèses n'est pas satisfaite : par exemple un graphe non connexe, ou un graphe avec une arête de poids négatif.

Un graphe non connexe

In [32]:
```
let s = 0
and a = 1
and b = 2
and c = 3
and d = 4
and e = 5
and f = 6;;
let g2 = {
   taille = 7;
   entree = s;
   adj = [|
      [(2,a); (4,b); (2,c)]; (* adj(s) *)
      [(1,d)]; (* adj(A) *)
      [(4,d)]; (* adj(B) *)
      [(1,b)]; (* adj(C) *)
      [ ]; (* adj(D) *)
      [(5,f)]; (* adj(E) *)
      [ ]; (* adj(F) *)
   |]
};;
```

Out[32]:
```
val s : int = 0
val a : int = 1
val b : int = 2
val c : int = 3
val d : int = 4
val e : int = 5
val f : int = 6
```

Out[32]:
```
val g2 : graph =
  {taille = 7;
   adj =
    [|[(2, 1); (4, 2); (2, 3)]; [(1, 4)]; [(4, 4)]; [(1, 2)]; []; [(5, 6)];
      []|];
   entree = 0}
```

In [33]:
```
let _ = dijkstra g2;;
```

Out[33]:
```
- : int array = [|0; 2; 3; 2; 3; 4611686018427387903; -4611686018427387900|]
```

Oups, ça n'a pas l'air correct !

Un graphe avec un poids négatif

In [44]:
```
let s = 0
and a = 1
and b = 2
and c = 3
and d = 4;;
let g3 = {
   taille = 5;
   entree = s;
   adj = [|
      [(2,a); (-4,b); (2,c)]; (* adj(s) *)
      [(1,d)]; (* adj(A) *)
      [(-4,d)]; (* adj(B) *)
      [(1,b)]; (* adj(C) *)
      [(2,b)]; (* adj(D) *)
   |]
};;
```

Out[44]:
```
val s : int = 0
val a : int = 1
val b : int = 2
val c : int = 3
val d : int = 4
```

Out[44]:
```
val g3 : graph =
  {taille = 5;
   adj =
    [|[(2, 1); (-4, 2); (2, 3)]; [(1, 4)]; [(-4, 4)]; [(1, 2)]; [(2, 2)]|];
   entree = 0}
```

In [45]:
```
let _ = dijkstra g3;;
```

```
Exception:
Failure "diminuer_clef : l'\195\169l\195\169ment n'est pas pr\195\169sent".
Raised at file "pervasives.ml", line 32, characters 22-33
Called from file "[13]", line 9, characters 31-53
Called from file "[13]", line 9, characters 31-53
Called from file "[22]", line 17, characters 21-48
Called from file "list.ml", line 77, characters 12-15
Called from file "[22]", line 13, characters 8-220
Called from file "toplevel/toploop.ml", line 180, characters 17-56
```

Oups, ça n'a pas l'air correct non plus !

Algorithme de Dijkstra - avec des files mutables

Déjà vu, on le retraite ici.

Files de priorité min mutables

In [46]:
```
(* file de priorité version non-mutable *)
type 'a priopqueue = (int * 'a) list ref;;
```

Out[46]:
```
type 'a priopqueue = (int * 'a) list ref
```

In [47]:
```
(* file vide *)
let vide () : 'a priopqueue = ref [ ];;
```

Out[47]:
```
val vide : unit → 'a priopqueue = <fun>
```

In [48]:
```
(* [inserer x clef q] insere l'element [x] dans la file [q]
   avec le clef [x].
   Termine avec une exception si la file contient déjà [x]  *)
let inserer (x:'a) (clef:int) (q:'a priopqueue) : unit =
   if List.exists (fun (_, v) -> x=v) !q
   then failwith "l'element est déjà dans la file"
   else q := (clef,x) :: !q
;;
```

Out[48]:
```
val inserer : 'a → int → 'a priopqueue → unit = <fun>
```

In [49]:
```
(* [est_vide q] teste si la file [q] est vide *)
let est_vide (q:'a priopqueue) : bool = (!q = [ ]);;
```

Out[49]:
```
val est_vide : 'a priopqueue → bool = <fun>
```

In [50]:
```
(* [trouve_min_aux min_val min_clef q] renvoie un couple de clef minimale
    dans (min_val,min_clef)::q *)
let rec trouve_min_aux (min_val:'a) (min_clef:int) (q:(int*'a) list) : int * 'a =
   match q with
   | [ ] -> (min_clef, min_val)
   | (clef, _) :: q when clef > min_clef -> trouve_min_aux min_val min_clef q
   | (clef, v) :: q -> trouve_min_aux v clef q
;;
```

Out[50]:
```
val trouve_min_aux : 'a → int → (int * 'a) list → int * 'a = <fun>
```

In [51]:
```
(* [trouve_min q] renvoie un élément de clef minimale la file [q].
   Lance une exception si la liste est vide *)
let trouve_min (q:(int*'a) list) : 'a =
   match q with
   | [ ] -> failwith "trouve_min: la file est vide"
   | (clef, v) :: q -> snd (trouve_min_aux v clef q)
;;
```

Out[51]:
```
val trouve_min : (int * 'a) list → 'a = <fun>
```

In [52]:
```
(* [supprime v q] renvoie une file contenant les éléments de [q], sauf [x].
   [x] doit apparaitre une et une seule fois dans la file. *)
let rec supprime (x:'a) (q:(int*'a) list) : (int*'a) list =
   match q with
   | [ ] -> [ ]
   | (_, v) :: q when v=x -> q
   | (clef, v) :: q -> (clef,v) :: (supprime x q)
;;
```

Out[52]:
```
val supprime : 'a → (int * 'a) list → (int * 'a) list = <fun>
```

In [53]:
```
(* [extraire_min q] renvoie un élément de q, de clef minimal,
   et met à jour la file; termine avec une exception si la file est vide  *)
let extraire_min (q:'a priopqueue) : 'a =
   if !q = [ ] then failwith "extraire_min: file vide"
   else
      let min = trouve_min !q in
      q :=  supprime min !q;
      min
;;
```

Out[53]:
```
val extraire_min : 'a priopqueue → 'a = <fun>
```

In [54]:
```
(* [diminuer_clef q clef x] modifie la clef de l'élément [x]
   dans la file q en lui associant la nouvelle clef [clef], qui
   doit être inferieur à la clef actuelle de [x].
   Termine avec une exception si la file ne contient pas [x]  *)
let diminuer_clef (x:'a) (clef:int) (q:'a priopqueue) : unit =
    let rec diminuer_aux (l:(int*'a) list) : (int*'a) list =
        match l with
        | [ ] -> failwith "diminuer_clef : l'élément n'est pas présent"
        | (_, v) :: q when v=x -> (clef, x) :: q
        | (c, v) :: q -> (c, v) :: diminuer_aux q in
    q := diminuer_aux !q
;;
```

Out[54]:
```
val diminuer_clef : 'a → int → 'a priopqueue → unit = <fun>
```

Dijkstra, 2ème version

C'est aussi assez direct :

In [55]:
```
let dijkstra g =
    let q = vide () in
    let dist = Array.init g.taille (fun i ->
        if i=g.entree then 0 else max_int
    ) in
    let pere = Array.init g.taille (fun i -> i) in
    for i=0 to g.taille - 1 do (* initialisation de la file *)
        inserer i dist.(i) q;
    done;
    while not (est_vide q) do
        let x = extraire_min q in
        (* on regarde les adjacents de x *)
        List.iter (fun (c,y) ->
            if dist.(y) > dist.(x) + c
            then begin
                pere.(y) <- x;
                dist.(y) <- dist.(x) + c;
                diminuer_clef y dist.(y) q
        end) g.adj.(x)
    done;
    dist, pere
;;
```

Out[55]:
```
val dijkstra : graph → int array * int array = <fun>
```

Exemples

In [56]:
```
let _ = dijkstra g1;;
```

Out[56]:
```
- : int array * int array = ([|0; 2; 3; 2; 3|], [|0; 0; 3; 0; 1|])
```

Et les contre-exemples maintenant :

In [57]:
```
let _ = dijkstra g2;;
```

Out[57]:
```
- : int array * int array =
([|0; 2; 3; 2; 3; 4611686018427387903; -4611686018427387900|],
 [|0; 0; 3; 0; 1; 5; 5|])
```

In [58]:
```
let _ = dijkstra g3;;
```

```
Exception:
Failure "diminuer_clef : l'\195\169l\195\169ment n'est pas pr\195\169sent".
Raised at file "pervasives.ml", line 32, characters 22-33
Called from file "[54]", line 10, characters 35-49
Called from file "[54]", line 10, characters 35-49
Called from file "[54]", line 11, characters 9-24
Called from file "list.ml", line 77, characters 12-15
Called from file "[55]", line 13, characters 8-232
Called from file "toplevel/toploop.ml", line 180, characters 17-56
```

Arbres couvrants de poids minimal

On ne traite que l'algorithme de Prim. L'algorithme de Kruskal n'est pas plus compliqué, il utilise une autre structure de données (Union-Find, déjà traité aussi).

Algorithme de Prim

In [37]:
```
let prim g =
   let q = vide () in
   let poids = Array.init g.taille (fun i ->
      if i=g.entree then 0 else max_int
   ) in
   let pere = Array.init g.taille (fun i -> i) in
   for i=0 to g.taille-1 do (* initialisation de la file *)
      inserer i poids.(i) q;
   done;
   while not (est_vide q) do
      let x = extraire_min q in
      (* on regarde les adjacents de x *)
      List.iter (fun (c,y) ->
         if poids.(y) > c
         then begin
            pere.(y) <- x;
            poids.(y) <- c;
            diminuer_clef y poids.(y) q
      end)
      g.adj.(x)
   done;
   Array.iteri (fun i p ->
      if i != p then Printf.printf " (%d, %d)\n" i p
   ) pere;
   poids, pere;
;;
```

Out[37]:
```
val prim : graph → int array * int array = <fun>
```

Exemple

In [38]:
```
let _ = prim g1
```

Out[38]:
```
- : int array * int array = ([|0; 2; 1; 2; 1|], [|0; 0; 3; 0; 1|])
```

Tas binaire min

In [39]:
```
(** {2 Leftist heaps, by Jean-Christophe Filliâtre} *)

(**************************************************************************)
(*                                                                        *)
(*  Copyright (C) Jean-Christophe Filliâtre                               *)
(*                                                                        *)
(*  This software is free software; you can redistribute it and/or        *)
(*  modify it under the terms of the GNU Library General Public           *)
(*  License version 2.1, with the special exception on linking            *)
(*  described in file LICENSE.                                            *)
(*                                                                        *)
(*  This software is distributed in the hope that it will be useful,      *)
(*  but WITHOUT ANY WARRANTY; without even the implied warranty of        *)
(*  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.                  *)
(*                                                                        *)
(**************************************************************************)

(* Leftist heaps.

   See for instance Chris Okasaki's "Purely Functional Data Structures" *)

module type Ordered = sig
  type t
  val le: t -> t -> bool
end

exception Empty

module Make(X : Ordered) :
sig
  type t
  val empty : t
  val is_empty : t -> bool
  val insert : X.t -> t -> t
  val min : t -> X.t
  val extract_min : t -> X.t * t
  val merge : t -> t -> t
  val length : t -> int
end
=
struct

  type t = E | T of int * X.t * t * t

  let rank = function E -> 0 | T (r,_,_,_) -> r

  let rec length = function E -> 0 | T (_,_,t1,t2) -> 1 + (length t1) + (length t2)

  let make x a b =
    let ra = rank a and rb = rank b in
    if ra >= rb then T (rb + 1, x, a, b) else T (ra + 1, x, b, a)

  let empty = E

  let is_empty = function E -> true | T _ -> false

  let rec merge h1 h2 = match h1,h2 with
    | E, h | h, E ->
        h
    | T (_,x,a1,b1), T (_,y,a2,b2) ->
        if X.le x y then make x a1 (merge b1 h2) else make y a2 (merge h1 b2)

  let insert x h = merge (T (1, x, E, E)) h

  let min = function E -> raise Empty | T (_,x,_,_) -> x

  let extract_min = function
    | E -> raise Empty
    | T (_,x,a,b) -> x, merge a b

end
```

```
Out[39]: module type Ordered = sig type t val le : t → t → bool end

Out[39]: exception Empty

Out[39]: module Make :
           functor (X : Ordered) →
             sig
               type t
               val empty : t
               val is_empty : t → bool
               val insert : X.t → t → t
               val min : t → X.t
               val extract_min : t → X.t * t
               val merge : t → t → t
               val length : t → int
             end
```

Codage de Huffman

C'est un grand classique pour les leçons "Programmation dynamique" et "Algorithmique du texte". Le présenter en développement sans l'avoir implémenter est difficilement pardonnable.

TODO : utiliser une file de priorité mutable plutôt qu'un tas min ? Je n'ai pas encore eu le temps de terminer cette correction.

Fréquences de lettres dans un texte

In [40]:
```
(** Calcule l'alphabet du texte [text] ainsi que son tableau de fréquence.
   Linéaire en temps et espace dans la taille du texte. *)
let frequencies text =
   let n = String.length text in
   let f = Hashtbl.create 128 in
   for i = 0 to n-1 do
      if Hashtbl.mem f text.[i] then
         Hashtbl.replace f text.[i] (1 + Hashtbl.find f text.[i])
      else Hashtbl.add f text.[i] 1;
   done;
   f
;;
```

```
Out[40]: val frequencies : string → (char, int) Hashtbl.t = <fun>
```

In [41]:
```
(** Renvoie juste l'alphabet qui compose le texte [text]. *)
let alphabet text =
   let f = frequencies text in
   let alp = ref [ ] in
   Hashtbl.iter (fun carct _ -> alp := carct :: !alp) f;
   !alp
;;
```

```
Out[41]: val alphabet : string → char list = <fun>
```

In [51]:
```
(** Un exemple. https://www.un.org/fr/universal-declaration-human-rights/index.html *)
let text1 = "https://www.un.org/fr/universal-declaration-human-rights/index.html : Article premier\n\nTous les etres humains naissent libres et egaux en dignite et en droits. Ils sont doues de raison et de conscience et doivent agir les uns envers les autres dans un esprit de fraternite.\nArticle 2\n\n1. Chacun peut se prevaloir de tous les droits et de toutes les libertes proclames dans la presente Declaration, sans distinction aucune, notamment de race, de couleur, de sexe, de langue, de religion, d'opinion politique ou de toute autre opinion, d'origine nationale ou sociale, de fortune, de naissance ou de toute autre situation.\n2. De plus, il ne sera fait aucune distinction fondee sur le statut politique, juridique ou international du pays ou du territoire dont une personne est ressortissante, que ce pays ou territoire soit independant, sous tutelle, non autonome ou soumis a une limitation quelconque de souverainete.";;
let _ = alphabet text1;;
```

Out[51]:
```
val text1 : string =
  "https://www.un.org/fr/universal-declaration-human-rights/index.html : Article premier\n\nTous les etres humains naissent libres et egaux en dignite et en droits. Ils sont doues de raison et de conscience et doivent agir les uns envers les autres dans un esprit de fraternite.\nArticle 2\n\n1. Chacun peut se prevaloir de tous les droits et de toutes les libertes proclames dans la presente Declaration, sans distinction aucune, notamment de race, de couleur, de sexe, de langue, de religion, d'opinion politique ou de toute autre opinion, d'origine nationale ou sociale, de fortune, de naissance ou de toute autre situation.\n2. De plus, il ne sera fait aucune distinction fondee sur le statut politique, juridique ou international du pays ou du territoire dont une personne est ressortissante, que ce pays ou territoire soit independant, sous tutelle, non autonome ou soumis a une limitation quelconque de souverainete."
```

Out[51]:
```
- : char list =
['l'; '2'; 'h'; '1'; 'g'; 'f'; 'j'; 'o'; 'y'; 'e'; '-'; 'v'; 'm'; 'D'; 'd';
 'i'; 'b'; ' '; 'C'; 'u'; 'n'; 't'; 'q'; ':'; 'T'; 'p'; 'A'; '/'; 'I'; ',';
 'x'; 'a'; 'r'; 'c'; 'w'; 's'; '\n'; '.'; '\'']
```

In [52]:
```
(** Pour affiche facilement. *)
let print = Format.printf;;

(** Pour visualiser un tableau des fréquences ainsi calculée. *)
let print_frequencies f =
  flush_all();
  print "\n\nTable des fréquences f :\n";
  flush_all();
  Hashtbl.iter (fun carct freq -> print "%C: %i, " carct freq) f;
  flush_all();
;;
```

Out[52]:
```
val print : ('a, Format.formatter, unit) format → 'a = <fun>
```

Out[52]:
```
val print_frequencies : (char, int) Hashtbl.t → unit = <fun>
```

In [73]:
```
print_frequencies (frequencies text1);;
```

```
'\'': 2, '.': 9, '\n': 6, 's': 60, 'w': 3, 'c': 19, 'r': 47, 'a': 46, 'x': 3, ',': 15, 'I': 1, '/': 5, 'A': 2, 'p': 16, 'T': 1, ':': 2, 'q': 6, 't': 70, 'n': 67, 'u': 53, 'C': 1, ' ': 134, 'b': 2, 'i': 64, 'd': 35, 'D': 2, 'm': 10, 'v': 5, '-': 3, 'e': 112, 'y': 2, 'o': 56, 'j': 1, 'f': 5, 'g': 8, '1': 1, 'h': 6, '2': 2, 'l': 32,


Table des fréquences f :
```

Out[73]:
```
- : unit = ()
```

Structure de tas binaire min

In [55]:
```
(* #use "Heap.ml";; *)
(* open Heap;; *)

type codage =
   | F of char * int | N of (int * codage * codage);;

let freq = function F (_, i) -> i | N (i,_ , _) -> i;;

module CodageFreq = struct
   type t = codage
   let le c1 c2 = (freq c1) <= (freq c2)
end;;

module MinHeap = Make(CodageFreq);;

let rec check_freq = function
   | F(_,i) -> assert( (i >= 0) )
   | N(i, c1, c2) -> assert( (i >= 0) && (i = (freq c1) + (freq c2)) ); check_freq c1; check_freq c2;
;;
```

Out[55]:
```
type codage = F of char * int | N of (int * codage * codage)
```

Out[55]:
```
val freq : codage → int = <fun>
```

Out[55]:
```
module CodageFreq : sig type t = codage val le : codage → codage → bool end
```

Out[55]:
```
module MinHeap :
  sig
    type t = Make(CodageFreq).t
    val empty : t
    val is_empty : t → bool
    val insert : CodageFreq.t → t → t
    val min : t → CodageFreq.t
    val extract_min : t → CodageFreq.t * t
    val merge : t → t → t
    val length : t → int
  end
```

Out[55]:
```
val check_freq : codage → unit = <fun>
```

Exemple d'un codage préfixe (Figure 16.4 p378 du Cormen)

In [56]:
```
let sigma_fromlist l =
   let h = Hashtbl.create (List.length l) in
   List.iter (fun (c, f) -> Hashtbl.add h c f) l;
   h
;;

let sigma1 = sigma_fromlist [('f', 5); ('e', 9); ('c', 12); ('b', 13); ('d', 16); ('a', 45) ];;

let codage1 = N(100,
   F('a', 45),
   N(55,
   N(25,
      F('c',12),
      F('b',13)
   ),
   N(30,
      N(14,
      F('f',5),
      F('e',9)
      ),
      F('d',16)
   )
)
);;
```

Out[56]:
```
val sigma_fromlist : ('a * 'b) list → ('a, 'b) Hashtbl.t = <fun>
```

Out[56]:
```
val sigma1 : (char, int) Hashtbl.t = <abstr>
```

Out[56]:
```
val codage1 : codage =
  N
   (100, F ('a', 45),
    N
     (55, N (25, F ('c', 12), F ('b', 13)),
      N (30, N (14, F ('f', 5), F ('e', 9)), F ('d', 16))))
```

In [57]:
```
let _ = check_freq codage1;;
```

Out[57]:
```
- : unit = ()
```

In [58]:
```
(** Calcul du nombre de bits nécessaires pour stocker le fichier. *)
let rec coutx prof = function
  | F(c, f) -> begin
    print "\nFeuille %C de profondeur %i et de fréquence %i." c prof f;
    (f * prof);
  end
  | N(_, c1, c2) -> (coutx (prof+1) c1) + (coutx (prof+1) c2)
;;

(** Il faudrait rajouter la taille du codage lui-même. *)
let cout = coutx 1;;

let _ = cout codage1;;
```

Out[58]:
```
val coutx : int → codage → int = <fun>
```

Out[58]:
```
val cout : codage → int = <fun>
```

Out[58]:
```
- : int = 324
```

Algorithme glouton de codage préfixe optimal de Huffman

On commence avec une fonction auxiliaire :

In [59]:
```
let huffmanx sigma =
   let n = Hashtbl.length sigma in
   let q = ref (MinHeap.empty) in
   Hashtbl.iter (fun c f -> q := (MinHeap.insert (F(c,f)) !q) ) sigma;
   for i = 1 to n-1 do
      flush_all();
      print "\n\nHuffmanx : %i-ième étape. La file q est de taille %i." i (MinHeap.length !q);
      let x, q2 = MinHeap.extract_min (!q) in
      flush_all();
      print "\nOn retire à q le noeud x de fréquence minimale (= %i)." (freq x);
      let y, q3 = MinHeap.extract_min q2 in
      flush_all();
      print "\nOn retire à q second noeud y de fréquence minimale (= %i)." (freq y);
      q := q3;
      let z = N( (freq x) + (freq y), x, y) in
      flush_all();
      print "\nOn les fusionne en z un nouveau noeud de fréquence %i, de fils gauche = x et droit = y." ( freq z );
      q := MinHeap.insert z !q;
      flush_all();
      print "\nOn ajoute ce noeud z a la file de priorité min q.";
   done;
   MinHeap.min !q
;;
```

Out[59]:
```
val huffmanx : (char, int) Hashtbl.t → CodageFreq.t = <fun>
```

In [62]:
```
(* Vérification sur l'exemple. *)
assert(codage1 = (huffmanx sigma1));;
```

```
On les fusionne en z un nouveau noeud de fréquence 54, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 1-ième étape. La file q est de taille 6.
On retire à q le noeud x de fréquence minimale (= 5).
On retire à q second noeud y de fréquence minimale (= 9).
On les fusionne en z un nouveau noeud de fréquence 14, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 2-ième étape. La file q est de taille 5.
On retire à q le noeud x de fréquence minimale (= 12).
On retire à q second noeud y de fréquence minimale (= 13).
On les fusionne en z un nouveau noeud de fréquence 25, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 3-ième étape. La file q est de taille 4.
On retire à q le noeud x de fréquence minimale (= 14).
On retire à q second noeud y de fréquence minimale (= 16).
On les fusionne en z un nouveau noeud de fréquence 30, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 4-ième étape. La file q est de taille 3.
On retire à q le noeud x de fréquence minimale (= 25).
On retire à q second noeud y de fréquence minimale (= 30).
On les fusionne en z un nouveau noeud de fréquence 55, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 5-ième étape. La file q est de taille 2.
On retire à q le noeud x de fréquence minimale (= 45).
On retire à q second noeud y de fréquence minimale (= 55).
```

Out[62]:
```
- : unit = ()
```

In [61]:
```
(** Pour un texte entier, on calcule directement le codage. *)
let huffman text =
   let freq = frequencies text in
   huffmanx freq
;;

let sigma2 = sigma_fromlist [ ('a',1); ('b',1); ('c',2); ('d',3); ('e',5); ('f',8); ('g',13); ('h',21) ];;
let _ = huffmanx sigma2;;
```

Out[61]:
```
val huffman : string → CodageFreq.t = <fun>
```

Out[61]:
```
val sigma2 : (char, int) Hashtbl.t = <abstr>
```

```
On les fusionne en z un nouveau noeud de fréquence 100, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 1-ième étape. La file q est de taille 8.
On retire à q le noeud x de fréquence minimale (= 1).
On retire à q second noeud y de fréquence minimale (= 1).
On les fusionne en z un nouveau noeud de fréquence 2, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 2-ième étape. La file q est de taille 7.
On retire à q le noeud x de fréquence minimale (= 2).
On retire à q second noeud y de fréquence minimale (= 2).
On les fusionne en z un nouveau noeud de fréquence 4, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 3-ième étape. La file q est de taille 6.
On retire à q le noeud x de fréquence minimale (= 3).
On retire à q second noeud y de fréquence minimale (= 4).
On les fusionne en z un nouveau noeud de fréquence 7, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 4-ième étape. La file q est de taille 5.
On retire à q le noeud x de fréquence minimale (= 5).
On retire à q second noeud y de fréquence minimale (= 7).
On les fusionne en z un nouveau noeud de fréquence 12, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 5-ième étape. La file q est de taille 4.
On retire à q le noeud x de fréquence minimale (= 8).
On retire à q second noeud y de fréquence minimale (= 12).
On les fusionne en z un nouveau noeud de fréquence 20, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 6-ième étape. La file q est de taille 3.
On retire à q le noeud x de fréquence minimale (= 13).
On retire à q second noeud y de fréquence minimale (= 20).
On les fusionne en z un nouveau noeud de fréquence 33, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 7-ième étape. La file q est de taille 2.
On retire à q le noeud x de fréquence minimale (= 21).
On retire à q second noeud y de fréquence minimale (= 33).
```

Out[61]:
```
- : CodageFreq.t =
N
 (54, F ('h', 21),
  N
   (33, F ('g', 13),
    N
     (20, F ('f', 8),
      N
       (12, F ('e', 5),
        N (7, F ('d', 3), N (4, N (2, F ('b', 1), F ('a', 1)), F ('c', 2)))))))
```

In [74]:
```
let _ = huffman text1;;
```

```
'\'': 2, '.': 9, '\n': 6, 's': 60, 'w': 3, 'c': 19, 'r': 47, 'a': 46, 'x': 3, ',':
15, 'I': 1, '/': 5, 'A': 2, 'p': 16, 'T': 1, ':': 2, 'q': 6, 't': 70, 'n': 67, 'u':
53, 'C': 1, ' ': 134, 'b': 2, 'i': 64, 'd': 35, 'D': 2, 'm': 10, 'v': 5, '-': 3,
'e': 112, 'y': 2, 'o': 56, 'j': 1, 'f': 5, 'g': 8, '1': 1, 'h': 6, '2': 2, 'l': 32,


Huffmanx : 1-ième étape. La file q est de taille 39.
On retire à q le noeud x de fréquence minimale (= 1).
On retire à q second noeud y de fréquence minimale (= 1).
On les fusionne en z un nouveau noeud de fréquence 2, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 2-ième étape. La file q est de taille 38.
On retire à q le noeud x de fréquence minimale (= 1).
On retire à q second noeud y de fréquence minimale (= 1).
On les fusionne en z un nouveau noeud de fréquence 2, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 3-ième étape. La file q est de taille 37.
On retire à q le noeud x de fréquence minimale (= 1).
On retire à q second noeud y de fréquence minimale (= 2).
On les fusionne en z un nouveau noeud de fréquence 3, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 4-ième étape. La file q est de taille 36.
On retire à q le noeud x de fréquence minimale (= 2).
On retire à q second noeud y de fréquence minimale (= 2).
On les fusionne en z un nouveau noeud de fréquence 4, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 5-ième étape. La file q est de taille 35.
On retire à q le noeud x de fréquence minimale (= 2).
On retire à q second noeud y de fréquence minimale (= 2).
On les fusionne en z un nouveau noeud de fréquence 4, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 6-ième étape. La file q est de taille 34.
On retire à q le noeud x de fréquence minimale (= 2).
On retire à q second noeud y de fréquence minimale (= 2).
On les fusionne en z un nouveau noeud de fréquence 4, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 7-ième étape. La file q est de taille 33.
On retire à q le noeud x de fréquence minimale (= 2).
On retire à q second noeud y de fréquence minimale (= 2).
On les fusionne en z un nouveau noeud de fréquence 4, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 8-ième étape. La file q est de taille 32.
On retire à q le noeud x de fréquence minimale (= 3).
On retire à q second noeud y de fréquence minimale (= 3).
On les fusionne en z un nouveau noeud de fréquence 6, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 9-ième étape. La file q est de taille 31.
On retire à q le noeud x de fréquence minimale (= 3).
On retire à q second noeud y de fréquence minimale (= 3).
On les fusionne en z un nouveau noeud de fréquence 6, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 10-ième étape. La file q est de taille 30.
On retire à q le noeud x de fréquence minimale (= 4).
On retire à q second noeud y de fréquence minimale (= 4).
On les fusionne en z un nouveau noeud de fréquence 8, de fils gauche = x et droit =
```

```
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 11-ième étape. La file q est de taille 29.
On retire à q le noeud x de fréquence minimale (= 4).
On retire à q second noeud y de fréquence minimale (= 4).
On les fusionne en z un nouveau noeud de fréquence 8, de fils gauche = x et droit =
y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 12-ième étape. La file q est de taille 28.
On retire à q le noeud x de fréquence minimale (= 5).
On retire à q second noeud y de fréquence minimale (= 5).
On les fusionne en z un nouveau noeud de fréquence 10, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 13-ième étape. La file q est de taille 27.
On retire à q le noeud x de fréquence minimale (= 5).
On retire à q second noeud y de fréquence minimale (= 6).
On les fusionne en z un nouveau noeud de fréquence 11, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 14-ième étape. La file q est de taille 26.
On retire à q le noeud x de fréquence minimale (= 6).
On retire à q second noeud y de fréquence minimale (= 6).
On les fusionne en z un nouveau noeud de fréquence 12, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 15-ième étape. La file q est de taille 25.
On retire à q le noeud x de fréquence minimale (= 6).
On retire à q second noeud y de fréquence minimale (= 6).
On les fusionne en z un nouveau noeud de fréquence 12, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 16-ième étape. La file q est de taille 24.
On retire à q le noeud x de fréquence minimale (= 8).
On retire à q second noeud y de fréquence minimale (= 8).
On les fusionne en z un nouveau noeud de fréquence 16, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 17-ième étape. La file q est de taille 23.
On retire à q le noeud x de fréquence minimale (= 8).
On retire à q second noeud y de fréquence minimale (= 9).
On les fusionne en z un nouveau noeud de fréquence 17, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 18-ième étape. La file q est de taille 22.
On retire à q le noeud x de fréquence minimale (= 10).
On retire à q second noeud y de fréquence minimale (= 10).
On les fusionne en z un nouveau noeud de fréquence 20, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 19-ième étape. La file q est de taille 21.
On retire à q le noeud x de fréquence minimale (= 11).
On retire à q second noeud y de fréquence minimale (= 12).
On les fusionne en z un nouveau noeud de fréquence 23, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 20-ième étape. La file q est de taille 20.
On retire à q le noeud x de fréquence minimale (= 12).
On retire à q second noeud y de fréquence minimale (= 15).
On les fusionne en z un nouveau noeud de fréquence 27, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 21-ième étape. La file q est de taille 19.
```

```
On retire à q le noeud x de fréquence minimale (= 16).
On retire à q second noeud y de fréquence minimale (= 16).
On les fusionne en z un nouveau noeud de fréquence 32, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 22-ième étape. La file q est de taille 18.
On retire à q le noeud x de fréquence minimale (= 17).
On retire à q second noeud y de fréquence minimale (= 19).
On les fusionne en z un nouveau noeud de fréquence 36, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 23-ième étape. La file q est de taille 17.
On retire à q le noeud x de fréquence minimale (= 20).
On retire à q second noeud y de fréquence minimale (= 23).
On les fusionne en z un nouveau noeud de fréquence 43, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 24-ième étape. La file q est de taille 16.
On retire à q le noeud x de fréquence minimale (= 27).
On retire à q second noeud y de fréquence minimale (= 32).
On les fusionne en z un nouveau noeud de fréquence 59, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 25-ième étape. La file q est de taille 15.
On retire à q le noeud x de fréquence minimale (= 32).
On retire à q second noeud y de fréquence minimale (= 35).
On les fusionne en z un nouveau noeud de fréquence 67, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 26-ième étape. La file q est de taille 14.
On retire à q le noeud x de fréquence minimale (= 36).
On retire à q second noeud y de fréquence minimale (= 43).
On les fusionne en z un nouveau noeud de fréquence 79, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 27-ième étape. La file q est de taille 13.
On retire à q le noeud x de fréquence minimale (= 46).
On retire à q second noeud y de fréquence minimale (= 47).
On les fusionne en z un nouveau noeud de fréquence 93, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 28-ième étape. La file q est de taille 12.
On retire à q le noeud x de fréquence minimale (= 53).
On retire à q second noeud y de fréquence minimale (= 56).
On les fusionne en z un nouveau noeud de fréquence 109, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 29-ième étape. La file q est de taille 11.
On retire à q le noeud x de fréquence minimale (= 59).
On retire à q second noeud y de fréquence minimale (= 60).
On les fusionne en z un nouveau noeud de fréquence 119, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 30-ième étape. La file q est de taille 10.
On retire à q le noeud x de fréquence minimale (= 64).
On retire à q second noeud y de fréquence minimale (= 67).
On les fusionne en z un nouveau noeud de fréquence 131, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 31-ième étape. La file q est de taille 9.
On retire à q le noeud x de fréquence minimale (= 67).
On retire à q second noeud y de fréquence minimale (= 70).
On les fusionne en z un nouveau noeud de fréquence 137, de fils gauche = x et droit
= y.
```

```
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 32-ième étape. La file q est de taille 8.
On retire à q le noeud x de fréquence minimale (= 79).
On retire à q second noeud y de fréquence minimale (= 93).
On les fusionne en z un nouveau noeud de fréquence 172, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 33-ième étape. La file q est de taille 7.
On retire à q le noeud x de fréquence minimale (= 109).
On retire à q second noeud y de fréquence minimale (= 112).
On les fusionne en z un nouveau noeud de fréquence 221, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 34-ième étape. La file q est de taille 6.
On retire à q le noeud x de fréquence minimale (= 119).
On retire à q second noeud y de fréquence minimale (= 131).
On les fusionne en z un nouveau noeud de fréquence 250, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 35-ième étape. La file q est de taille 5.
On retire à q le noeud x de fréquence minimale (= 134).
On retire à q second noeud y de fréquence minimale (= 137).
On les fusionne en z un nouveau noeud de fréquence 271, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 36-ième étape. La file q est de taille 4.
On retire à q le noeud x de fréquence minimale (= 172).
On retire à q second noeud y de fréquence minimale (= 221).
On les fusionne en z un nouveau noeud de fréquence 393, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 37-ième étape. La file q est de taille 3.
On retire à q le noeud x de fréquence minimale (= 250).
On retire à q second noeud y de fréquence minimale (= 271).
On les fusionne en z un nouveau noeud de fréquence 521, de fils gauche = x et droit
= y.
On ajoute ce noeud z a la file de priorité min q.

Huffmanx : 38-ième étape. La file q est de taille 2.
On retire à q le noeud x de fréquence minimale (= 393).
On retire à q second noeud y de fréquence minimale (= 521).
```

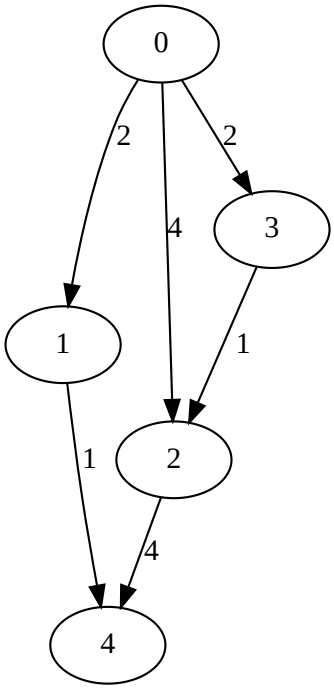
```
Out[74]: - : CodageFreq.t =
         N
          (914,
           N
            (393,
             N
              (172,
               N
                (79, N (36, N (17, F ('g', 8), F ('.', 9)), F ('c', 19)),
                 N
                  (43, N (20, N (10, F ('/', 5), F ('v', 5)), F ('m', 10)),
                   N
                    (23, N (11, F ('f', 5), F ('\n', 6)),
                     N
                      (12, N (6, F ('-', 3), N (3, F ('I', 1), F ('\'', 2))),
                       F ('q', 6))))),
               N (93, F ('a', 46), F ('r', 47))),
             N (221, N (109, F ('u', 53), F ('o', 56)), F ('e', 112))),
           N
            (521,
             N
              (250,
               N
                (119,
                 N
                  (59,
                   N
                    (27, N (12, N (6, F ('x', 3), F ('w', 3)), F ('h', 6)),
                     F (',', 15)),
                   N
                    (32, F ('p', 16),
                     N
                      (16,
                       N
                        (8, N (4, F ('2', 2), F ('b', 2)),
                         N (4, N (2, F ('C', 1), F ('T', 1)), F ('A', 2))),
                       N
                        (8, N (4, F ('y', 2), F ('D', 2)),
                         N (4, F (':', 2), N (2, F ('1', 1), F ('j', 1)))))))),
               F ('s', 60)),
             N (131, F ('i', 64), N (67, F ('l', 32), F ('d', 35)))),
           N (271, F (' ', 134), N (137, F ('n', 67), F ('t', 70)))))
```

Coloration de graphes

Voyons une approche gloutonne.

On va trier les sommets par degrés décroissants, et attribuer à chaque sommet une couleur, soit la plus petite possible parmi celles non utilisées par ces voisins, soit une nouvelle.

On représente un graphe par tableau de listes d'adjacence. Notre exemple sera le graphe suivant, sans considérer les étiquettes des arêtes et en le considérant non orienté :



Représentation des graphes

In [2]:
```
type sommet = int ;;
type graphe = (sommet list) array;;
```

Out[2]:
```
type sommet = int
```

Out[2]:
```
type graphe = sommet list array
```

In [3]:
```
let nb_sommet (g : graphe) = Array.length g;;
```

Out[3]:
```
val nb_sommet : graphe → int = <fun>
```

Et des coloriages

In [4]:
```
type couleur = int;;
type coloriage = couleur array;; (* c.(i) est la couleur du sommet i... *)
```

Out[4]:
```
type couleur = int
```

Out[4]:
```
type coloriage = couleur array
```

In [5]:
```
let verifie_coloriage (g : graphe) (c : coloriage) =
    let res = ref true in
    let n = nb_sommet g in
    for i = 0 to n-1 do
        if c.(i) < 0 || c.(i) >= n then res := false;
        List.iter (fun j ->
            if c.(i) == c.(j) then res := false;
        ) g.(i)
    done;
    !res
;;
```

Out[5]:
```
val verifie_coloriage : graphe → coloriage → bool = <fun>
```

Exemple

In [6]:
```
let g1 : graphe = [|
    [1; 2; 3]; (* voisins de 0 *)
    [0; 4]; (* voisins de 1 *)
    [0; 3; 4]; (* voisins de 2 *)
    [0; 2]; (* voisins de 3 *)
    [1; 2]  (* voisins de 4 *)
|];;
```

Out[6]:
```
val g1 : graphe = [|[1; 2; 3]; [0; 4]; [0; 3; 4]; [0; 2]; [1; 2]|]
```

In [7]:
```
let coloriage1 = [|0; 1; 1; 1; 0|];;
let _ = verifie_coloriage g1 coloriage1;; (* 3 -> 2 mais ont la même couleur *)

let coloriage2 = [|0; 1; 2; 1; 0|];;
let _ = verifie_coloriage g1 coloriage2;;
```

Out[7]:
```
val coloriage1 : int array = [|0; 1; 1; 1; 0|]
```

Out[7]:
```
- : bool = false
```

Out[7]:
```
val coloriage2 : int array = [|0; 1; 2; 1; 0|]
```

Out[7]:
```
- : bool = true
```

On a bien sûr une approche naïve :

In [8]:
```
let coloriage_naif (g : graphe) : coloriage =
    let n = nb_sommet g in
    Array.init n (fun i -> i);
;;
```

Out[8]:
```
val coloriage_naif : graphe → coloriage = <fun>
```

In [9]:
```
let coloriage3 = coloriage_naif g1;;
let _ = verifie_coloriage g1 coloriage3;;
```

Out[9]:
```
val coloriage3 : coloriage = [|0; 1; 2; 3; 4|]
```

Out[9]:
```
- : bool = true
```

C'est une borne supérieure triviale sur le nombre minimal de couleur requis pour colorier un graphe.

Algorithme glouton pour le coloriage

Pour plus de détails, voir par exemple cette page là (http://jean-paul.davalan.pagesperso-orange.fr/graphs/col/index.html).

In [10]:
```
let degres (g : graphe) : int array =
  Array.map List.length g
;;
```

Out[10]:
```
val degres : graphe → int array = <fun>
```

In [11]:
```
let _ = degres g1;;
```

Out[11]:
```
- : int array = [|3; 2; 3; 2; 2|]
```

In [12]:
```
type permutation = int array;;

let trie_par_degres (g : graphe) : permutation =
  let n = nb_sommet g in
  let indices = Array.init n (fun i -> i) in
  let d = degres g in
  let cmp_deg i j = Pervasives.compare d.(j) d.(i) in
  Array.stable_sort cmp_deg indices;
  indices
;;
```

Out[12]:
```
type permutation = int array
```

Out[12]:
```
val trie_par_degres : graphe → permutation = <fun>
```

In [13]:
```
let _ = trie_par_degres g1;;
```

Out[13]:
```
- : permutation = [|0; 2; 1; 3; 4|]
```

In [14]:
```
let plus_petite_couleur_libre (n : int) (cs : couleur list) : couleur =
  let rep = ref 0 in
  while List.mem !rep cs do
    incr rep
  done;
  assert (!rep < n);
  !rep
;;
```

Out[14]:
```
val plus_petite_couleur_libre : int → couleur list → couleur = <fun>
```

In [15]:
```
let coloriage_glouton (g : graphe) : coloriage =
  let n = nb_sommet g in
  let c = Array.make n (-1) in
  let perm = trie_par_degres g in
  for i = 0 to n-1 do
    (* on regarde le sommet perm.(i) *)
    let couleurs_voisins = List.map (fun j -> c.(j)) g.(perm.(i)) in
    c.(perm.(i)) <- plus_petite_couleur_libre n couleurs_voisins;
  done;
  c
;;
```

Out[15]:
```
val coloriage_glouton : graphe → coloriage = <fun>
```

In [20]:
```
let coloriage_glouton_pas_trie (g : graphe) : coloriage =
  let n = nb_sommet g in
  let c = Array.make n (-1) in
  for i = 0 to n-1 do
    (* on regarde le sommet i *)
    let couleurs_voisins = List.map (fun j -> c.(j)) g.(i) in
    c.(i) <- plus_petite_couleur_libre n couleurs_voisins;
  done;
  c
;;
```

Out[20]:
```
val coloriage_glouton_pas_trie : graphe → coloriage = <fun>
```

Exemples

In [16]:
```
let coloriage4 = coloriage_glouton g1;;
let _ = verifie_coloriage g1 coloriage4;;
```

```
Out[16]: val coloriage4 : coloriage = [|0; 1; 1; 2; 0|]
Out[16]: - : bool = true
```

In [21]:
```
let coloriage5 = coloriage_glouton_pas_trie g1;;
let _ = verifie_coloriage g1 coloriage5;;
```

```
Out[21]: val coloriage5 : coloriage = [|0; 1; 1; 2; 0|]
Out[21]: - : bool = true
```

On remarque que la procédure gloutonne a ici trouvé un coloriage minimal et optimal mais différent de celui proposé plus haut.

- Comment peut-on vérifier que c'est un coloriage minimal ? Il faudrait essayer chaque changement de couleur (ce n'est pas simple)
- Et comment vérifier l'optimalité ? Il faudrait tester *tous* les coloriages à `c-1` couleurs (si celui là en a `c`) et montrer qu'aucun ne convient (ce n'est pas simple non plus, il y a beaucoup de coloriages possibles).

Note : une première indication est qu'ici on a utilisé `c=3` couleurs avec un graphe de degré maximum $\delta_{\max} = 3$.

Pour un contre exemple :

In [22]:
```
let g2 : graphe = [|
   [1; 2; 3; 4; 5]; (* voisins de 0 *)
   [0; 2; 3; 4; 5]; (* voisins de 1 *)
   [0; 1; 3; 4]; (* voisins de 2 *)
   [0; 1; 2; 5]; (* voisins de 3 *)
   [0; 1; 2];  (* voisins de 4 *)
   [0; 1; 3]  (* voisins de 5 *)
|];;
```

```
Out[22]: val g2 : graphe =
           [|[1; 2; 3; 4; 5]; [0; 2; 3; 4; 5]; [0; 1; 3; 4]; [0; 1; 2; 5]; [0; 1; 2];
             [0; 1; 3]|]
```

In [23]:
```
let coloriage6 = coloriage_glouton g2;;
let _ = verifie_coloriage g2 coloriage6;;
```

```
Out[23]: val coloriage6 : coloriage = [|0; 1; 2; 3; 3; 2|]
Out[23]: - : bool = true
```

In [24]:
```
let coloriage6 = coloriage_glouton_pas_trie g2;;
let _ = verifie_coloriage g2 coloriage6;;
```

```
Out[24]: val coloriage6 : coloriage = [|0; 1; 2; 3; 3; 2|]
Out[24]: - : bool = true
```

Et en changeant l'ordre des sommets :

In [26]:
```
let g3 : graphe = [|
   [5; 4; 2];  (* voisins de 0 *)
   [5; 4; 3];  (* voisins de 1 *)
   [5; 4; 3; 0]; (* voisins de 2 *)
   [5; 4; 2; 1]; (* voisins de 3 *)
   [5; 3; 2; 1; 0]; (* voisins de 4 *)
   [4; 3; 2; 1; 0] (* voisins de 5 *)
|];;
```

```
Out[26]: val g3 : graphe =
           [|[5; 4; 2]; [5; 4; 3]; [5; 4; 3; 0]; [5; 4; 2; 1]; [5; 3; 2; 1; 0];
             [4; 3; 2; 1; 0]|]
```

In [29]:
```
let coloriage6 = coloriage_glouton g3;;
let _ = verifie_coloriage g3 coloriage6;;
```

```
Out[29]: val coloriage6 : coloriage = [|3; 2; 2; 3; 0; 1|]
Out[29]: - : bool = true
```

Là on vérifie que l'ordre des sommets est important, par exemple si on ne trie pas les sommets par degrés décroissants, l'algorithme glouton trouche un coloriage sous-optimal (avec 5 couleurs ici) :

In [28]:
```
let coloriage6 = coloriage_glouton_pas_trie g3;;
let _ = verifie_coloriage g3 coloriage6;;
```

```
Out[28]: val coloriage6 : coloriage = [|0; 0; 1; 2; 3; 4|]
Out[28]: - : bool = true
```

Je n'ai pas trouvé de contre exemple qui donne un coloriage sous-optimal pour la première version (avec tri par degrés décroissants) de l'algorithme. Si vous en avez un, envoyez le moi ! (https://perso.crans.org/besson/contact/)

Conclusion

Fin. À la séance prochaine. Le dernier TP (TP8) traitera de programmation logique (en mai).

> En attendant, essayez de finir ce sujet TP#7 et aussi la partie 2 (avec Python) du TP#6 sur le λ-calcul.