Parcours de Graphes

Florian Bourse

Listes d'adjacences

```
type graphe = int list array
let succ g u = g.(u)
```

Parcours en Largeur (Breadth First Search)

```
let plus_un = function
  | None -> None
  | Some x -> Some (x + 1)
let bfs g s =
  let n = Array.length g in
  let dist = Array.make n None in
  let parent = Array.make n None in
  let q = Queue.create () in
  Queue.push s q;
  dist.(s) <- Some 0;
  while not (Queue.is_empty q) do
    let u = Queue.pop q in
    let traiter v =
      if dist.(v) = None then
        begin
          dist.(v) <- plus_un dist.(u);
          parent.(v) <- Some u;
          Queue.push v q
        end
    in
    List.iter traiter (succ g u)
  done;
  parent
```

Parcours en Profondeur (Depth First Search)

```
exception Cycle
let dfs g s =
  let t = ref(-1) in
  let time () = incr t; !t in
  let n = Array.length g in
  let debut = Array.make n (-1) in
  let fin = Array.make n (-1) in
  let rec parcours u =
    debut.(u) <- time ();
    List.iter
      (fun v -> if debut.(v) < 0 then parcours v
                else if fin.(v) < 0 then raise Cycle)
      (succ g u);
    fin.(u) <- time ()
  in
  parcours s;
  fin
```