Algorithmics
Final Exam #3 (P3)

Undergraduate 2^{nd} year (s3)
Epita

22 Dec. 2015 - 9:30 (D.S. 308973.68 BW)

Instructions (read it) :

□ You must answer on **the answer sheets provided.**

- No other sheet will be picked up. Keep your rough drafts.
- Answer within the provided space, **answers outside will not be marked**: Use your drafts!
- Do not separate the sheets unless they can be re-stapled before handing in.
- Penciled answers will not be marked.

□ The presentation is negatively marked, which means that you are marked out of 20 points and the presentation points (maximum of 2) are taken off this grade.

□ **Algorithms:**

- All algorithms must be written in the language Algo (no C, Caml or anything else).
- Any Algo code not indented will not be marked.
- All that you need (types, routines) is indicated in the **appendix** (last page)!

□ Duration : 2h

Exercise 1 (Miscillaneous questions... – *3 points*)

1. (a) If in a graph G there exist a chain between x and y, and a chain between y and z; does there exist in G a chain between x and z?
   (b) Justify your answer graphically.

2. (a) If in a graph G there exist two chains between x et y. Do x and y belong to a same cycle of G?
   (b) Justify your answer graphically.

3. Let C and C' be two distinct strongly connected components of a directed graph $G =< S, A >$, Let $x, y \in C$. Let $x', y' \in C'$, and suppose there exist a path $x \rightsquigarrow x'$ in G. Show that there can not also be a path $y' \rightsquigarrow y$ in G.

Exercise 2 (Directed acyclic graph... – *2,5 points*)

1. Concerning the classification of arcs, what is the particularity of a directed acyclic graph?

2. Let $G =< S, A >$ be a directed acyclic graph, let os and op be the tables, containing respectively, the postorder number and the preorder number of all vertices of the graph G obtained during the depth-first search traversal of G. Show that for any pair of distinct vertices $x, y \in S$, if there exist an arc from x to y in G, then $os[y] < os[x]$.

Exercise 3 (Red-black Trees – *4 points*)

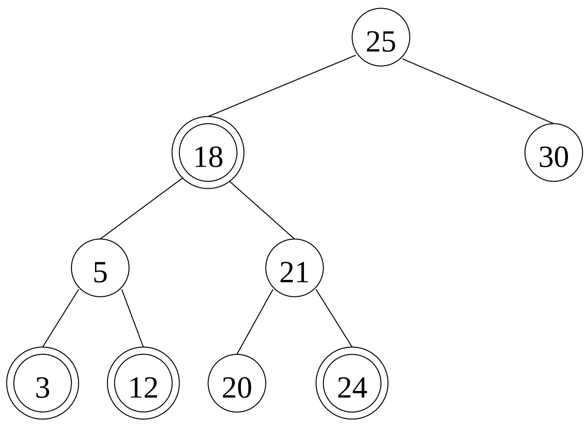

Figure 1: Red-black tree?

Remark: As usual, the red nodes are those with "double circles".

1. Is the tree in figure 1 a red-black tree? If this is not the case, which node (or nodes) has to be removed to make it a red-black tree?

2. Write an algorithm that calculates the size and height of the 2-4 tree represented by a red-black tree.

Exercise 4 (Bipartite graph – *7 points*)

A bipartite graph is a graph (undirected) $G = < S, A >$ where vertices can be partitioned into two sets S_1 et S_2, such that $(u, v) \in A$ implies either $u \in S_1$ and $v \in S_2$, or $u \in S_2$ and $v \in S_1$. That is, no edge connects vertices in the same set.

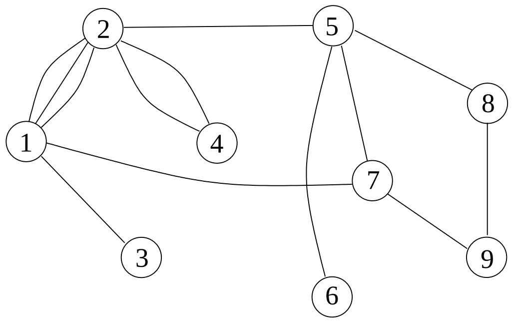

Figure 2: Graph G_1

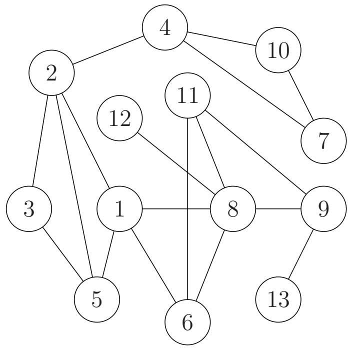

Figure 3: Graph G_2

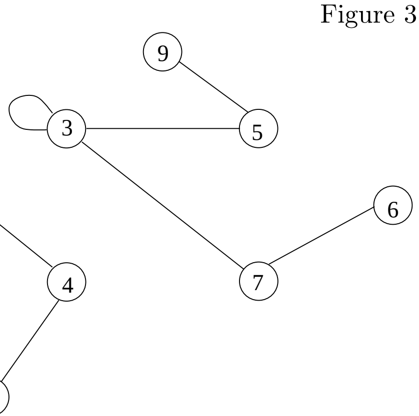

Figure 4: Graph G_3

1. Are the graphs of figures 2, 3 and 4 bipartite? For each bipartite graph give the two sets S_1 and S_2.
2. Write an algorithm that tests, with a **depth-first traversal**, whether a graph is bipartite. The dynamic implementation has to be used.

Exercise 5 (What is this? – *5,5 points*)

```
algorithm procedure build_graph
    local parameters
        t_graph_stat  G
        integer       s, n
    global parameters
        t_graph_stat  NG

    variables
        t_int_vect   map, dist
        t_queue      q
        integer      i, j
begin
    for i ← 1 to G.order do
        map[i] ← 0
        dist[i] ← -1
        for j ← 1 to G.order do
            NG.adj[i,j] ← 0
        end for
    end for
    q ← new_queue()
    enqueue(s, q)
    dist[s] ← 0
    NG.order ← 1
    map[s] ← 1
    do
        s ← dequeue(q)
        for i ← 1 to G.order do
            if G.adj[s,i] <> 0 then
                if (dist[i] = -1) and (dist[s] < n) then
                    dist[i] ← dist[s] + 1
                    NG.order ← NG.order + 1
                    map[i] ← NG.order
                    enqueue(i, q)
                end if
                if dist[i] <> -1 then
                    NG.adj[map[s],map[i]] ← G.adj[s,i]
                    NG.adj[map[i],map[s]] ← G.adj[i,s]
                end if
            end if
        end for
    while not is_empty(q)
end algorithm procedure build_graph
```

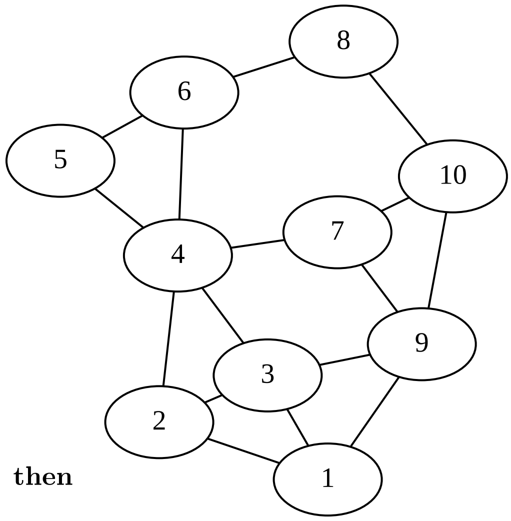

Figure 5 – Graph G_4

1. This algorithm is called with `build_graph(`G_4`, 5, 2, `NG`)` (G_4 the graph in figure 5).
   (a) Fill the array `dist`.
   (b) Fill the array `map`.
   (c) Draw the built graph (NG).
2. `build_graph(`G`, `s`, `n`, `NG`)` is called with G any non-empty graph, s a vertex of G, and n a positive integer.
   (a) During the execution, what does the array `dist` represent?
   (b) During the execution, what is the array `map` used for?
   (c) After the execution, what does the graph NG represent?

Appendix

Implementation of RB-Trees

```
types
 /* t_elmt type declaration */
t_rbt = ↑ s_rbt

s_rbt = record
  t_elmt    key
  boolean   red
  t_rbt     left, right
end record s_rbt
```

Graph implementations

The graphs we use have no cost. Thus we have removed them from the implementation.

Static:

```
constants
  Max = 100

types
  t_edge_mat  = Max × Max integer

  t_graph_stat = record
      boolean          directed
      integer          order
      t_edge_mat       edges
  end record t_graph_stat
```

Dynamic:

```
types
    t_listsom = ↑ s_som
    t_listadj = ↑ s_ladj

    s_som      = record
       integer      som
       t_listadj   succ
       t_listadj   pred
       t_listsom   next
    end record s_som

    s_ladj      = record
       t_listsom   vsom
       integer      nb
       t_listadj   next
    end record s_ladj

    t_graph_dyn  = record
       integer     order
       boolean    directed
       t_listsom  lsom
    end record t_graph_dyn
```

Vectors:

```
types        /* Max > order (G) */
  t_int_vect  = Max integer
  t_bool_vect  = Max boolean
```

Authorized routines

All operations on queues and stacks can be used as long as you specify the type of elements.

Queues

- new_queue():t_queue
- is_empty(t_queue *q*):boolean
- enqueue(t_queueElt *e*, t_queue *q*)
- dequeue(t_queue *q*):t_queueElt
- empty_queue(t_queue *q*)

Stacks

- new_stack():t_stack
- is_empty(t_stack *p*):boolean
- push(t_stackElt *elt*, t_stack *p*)
- pop(t_stack *p*):t_stackElt
- top(t_stack *p*):t_stackElt

Other

- search(integer *v*, t_graph_dyn *G*):t_listsom
  returns the pointer on the vertex number v in the graph G.