Graphs and Complexity

3) Graph search

Irena.Rusu@univ-nantes.fr

LS2N, bât. 34, bureau 303

tél. 02.51.12.58.16

Plan

- Generalities
- Breadth-first search (BFS)
- Depth-first search (DFS)

Graph Search

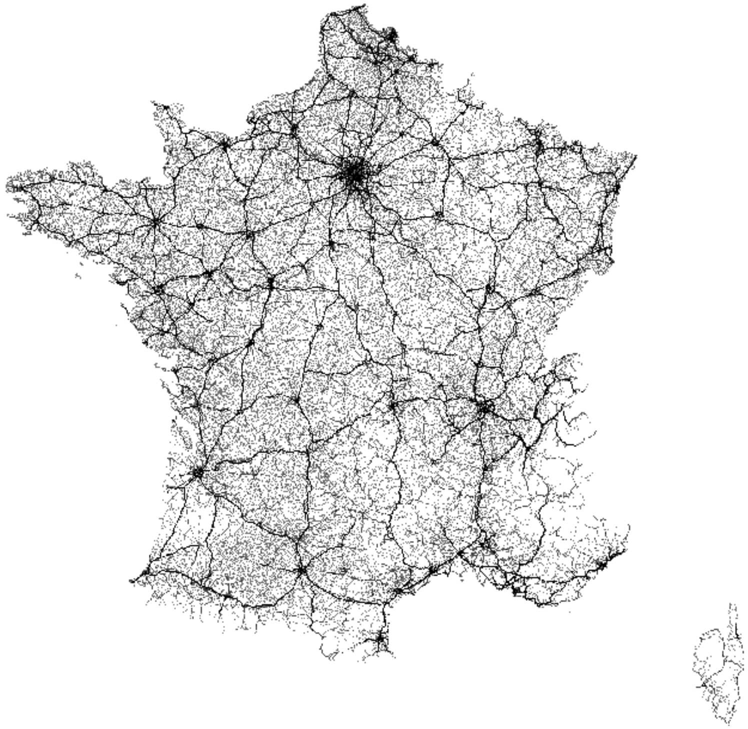
Digraph $G=(S,A)$ (works similarly for undirected)

$|S|=n$, $|A|=m$

Graph search intuitively:

- Start with an unvisited vertex
- Use one or several of its edges to visit other vertices (not yet visited)
- Do the same with the recently visited vertices
- And so on, as long as possible
- If there are unvisited vertices, start again using one of them.

Why?



© Anders Elias, www.data.gouv.fr, m.a.j. 25 mai 2023

- Allows to look for paths and cycles in the graph, possibly focusing on special vertices or edges/arcs
- Allows to find particular vertices (hubs …)
- Allows to access the data stored in the vertices, with supplementary information about the order of the vertices
- Allows to test structural properties of the graph (connectivity, bipartition etc.)

An example: the 15-puzzle

- Start with a 4x4 board with 15 tiles numbered from 1 to 15, and an open position.
- Tiles neighboring (top, down, left, right) the open position can be slided towards the open position (the former place of the tile is the new open position).
- Goal: make moves so that to obtain the ordered board.

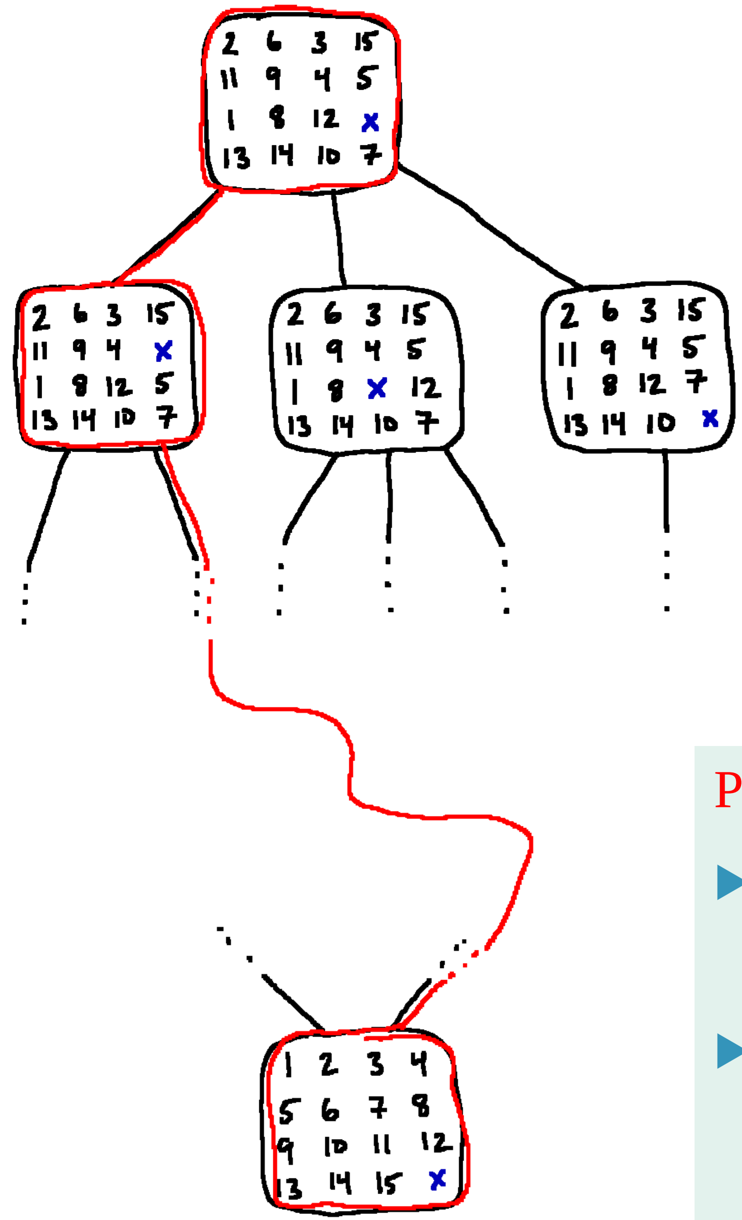
Start (Scrambled)

2	6	3	15
11	9	4	5
1	8	12	
13	14	10	7

Goal (Ordered)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

An approach



- Graph search
- Potentially a very large graph
- Total number of possible boards ?

  16 ! (=20.922.789.888.000)
- Solution with 41 moves for this example

Problems:

- Given a starting board, is there a solution ?

  Belongs to P: polynomial algorithms exist
- Find the solution with minimum number of moves (that is, the shortest path in the graph).

  NP-complete: probably no polynomial algorithm exists.

Search trees

Graph search intuitively:

- Start with an unvisited vertex
- Use one or several of its edges to visit other vertices (not yet visited)
- Do the same with the recently visited vertices
- And so on, as long as possible
- If there are unvisited vertices, start again using one of them.

Builds one or several vertex-disjoint search trees:

- One tree $T(s)$ for each starting vertex s
- s is the root of $T(s)$
- $T(s)$ contains the vertices v such that s is the first starting vertex for which a path exists in G from s to v
- For each v, there is a unique path from s to v in $T(s)$: the first path found during the search.

Main graph searches

- Two main types of graph searches:
  - Breadth-first search (or BFS)
  - Depth-first search (or DFS)

- They may be modified, enriched etc. as needed for any precise application.

- One may define other searches: using DFS-like or BFS-like progression according to a specific need.

Plan

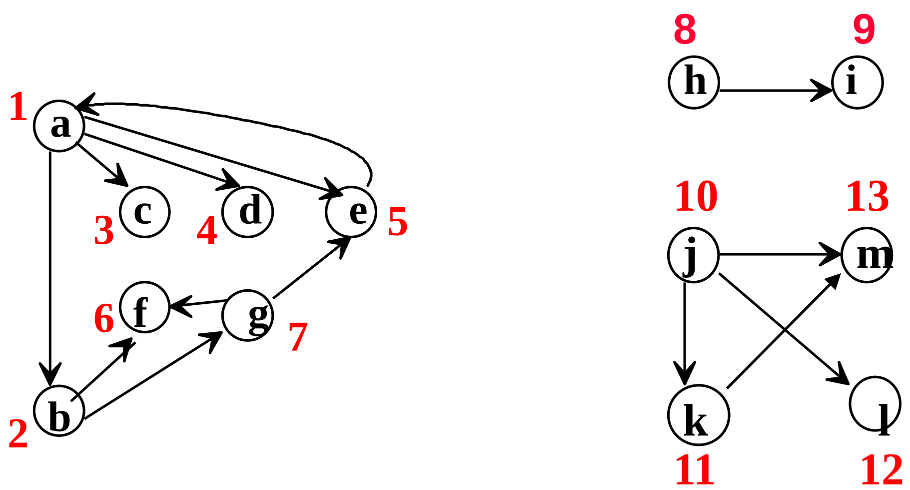
- Generalities
- Breadth-first search (BFS)
- Depth-first search (DFS)

Breadth first search (BFS)

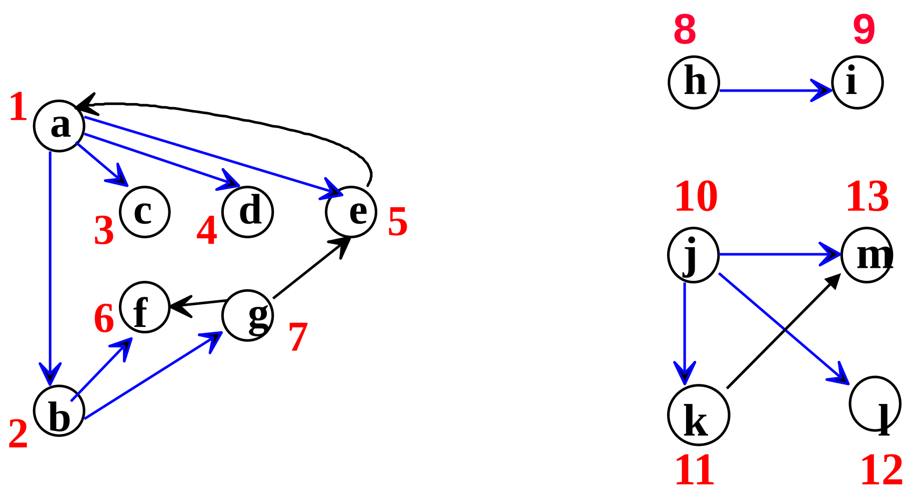
$G=(S,A)$ a graph (directed or not), s a vertex of G

BFS (s):

- Considers all the arcs/edges of G and progressively visits all the vertices which are reachable from s
- Builds a search tree (or BFS-tree) $T(s)$
- For each vertex v that is reachable by a path from s, the path from s to v in $T(s)$ is a shortest path from s to v.
- Visits all the vertices at distance k from s, before visiting the vertices at distance $k+1$ from s.



Ordre du parcours : a b c d e f g h i j k l m



Ordre du parcours : a b c d e f g h i j k l m

BFS Algorithm

Function BFS (*s* vertex of *G*) : void

```
begin
    Q ← Enqueue (Empty-Queue, s) ;
    while not  IsEmpty (Q)  do
        s’ ← First (Q) ; Q ← Dequeue (Q, s') ;
        if not visited [s’] then
                visited [s'] ← true ;
                for  each successor t of s’ do
                        if not visited [ t ] then
                                Q ← Enqueue (Q, t)
                        endif
                endfor
         endif
    endwhile
end
```

Visiting the whole graph *G*

```
    for each vertex s of G  do
        visited[s] ← false
     endfor
    for each vertex s of G  do
        if not visited [s] then
            BFS (s)
        endif
    endfor
```

1) Generic instructions (« for each vertex ») may be used in an algorithm only when the data structure is not specified.
2) Enqueue, Dequeue, IsEmpty, First are classic operations on queues, not Java (or other language) functions.

Running time

c (« for each vertex ») = $O(|S|)$

c (« while not IsEmpty (Q) do ») is significant only when the inner **for** is executed

Adjacency matrix

c (« for each successor t of s' ») =

c (« for each vertex t

such that M [s,t] = 1 ») = $\Theta(|S|)$

$\Rightarrow$ Running time in $\Theta(|S|^2)$

Adjacency List(s)

c (« for each successor t of s' ») = $\Theta(|A(s)|)$

$\Rightarrow$ Running time of $\Theta(|S| + |A|)$

Plan

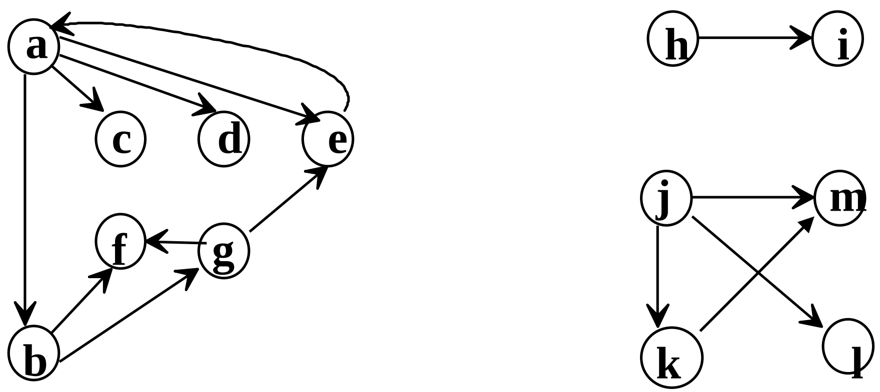
- Generalities
- Breadth-first search (BFS)
- Depth-first search (DFS)

Depth-first search (DFS)

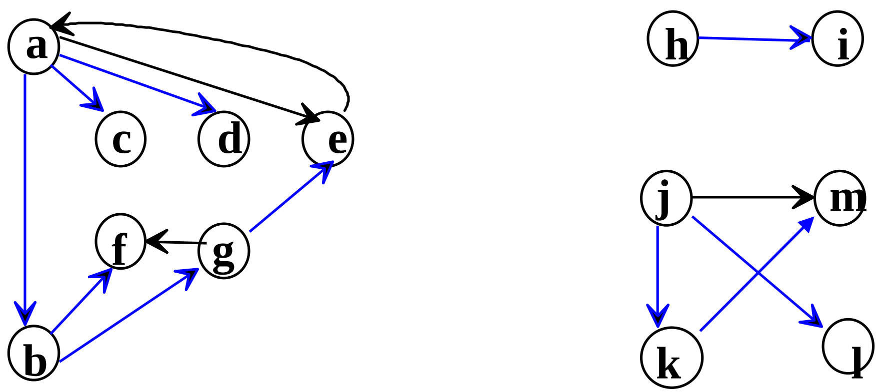
$G=(S,A)$ a graph (directed or not), s a vertex

DFS(s) :

- Considers all the arcs/edges of G and progressively visits all the vertices which are reachable from s
- Builds a search tree (or DFS-tree) $T(s)$
- Visits first the non-visited successors of the deepest visited vertex



Ordre du parcours : a b f g e c d h i j k m l



Ordre du parcours : a b f g e c d h i j k m l

DFS Algorithm

```
Function DFS (s vertex of G) : void
begin
    visited [s] ← true ;
    for  each successor t of s do
        if not visited [ t ] then
            DFS(t)
        endif
    endfor
end
```

Visiting the whole graph *G*

```
    for each vertex s of G  do
        visited[s] ← false
    endfor
    for each vertex s of G  do
        if not visited [s] then
            DFS (s)
        endif
    endfor
```

Running time

c (« for each vertex ») = $O(|S|)$

Adjacency matrix

c (« for each successor t of s ») =

c (« for each vertex t

such that $M[s,t] = 1$ ») $= \Theta(|S|)$

$\Rightarrow$ Running time of $\Theta(|S|^2)$

Adjacency List(s)

c (« for each successor t of s ») = $\Theta(|A(s)|)$

$\Rightarrow$ Running time of $\Theta(|S| + |A|)$

Iterative DFS

```
DFS (s vertex of G) : void //iterative
begin
    T ← Push (Empty-Stack, s) ;
    while not  IsEmpty (T)  do
        s' ← Top (T) ; T ← Pop (T, s') ;
        if not visited [s'] then
                visited [s'] ← true ;
                for  each successor t of s' do
                        if not visited [ t ] then
                                T ← Push (T, t)  endif
                endfor
           endif
    endwhile
end
```

Visiting the whole graph G

```
for each vertex s of G  do
    visited[s] ← false
endfor
for each vertex s of G  do
    if not visited [s] then
        DFS (s)
    endif
endfor
```

Numbering

Function DFSNum (s vertex of G):void

begin

$nb \leftarrow nb + 1$; $d[s] \leftarrow nb$;

for each successor t of s do

if $d[t] = 0$ then

DFSNum (t **)** ;

endif

endfor

$nb \leftarrow nb+1$; $f[s] \leftarrow nb$

end

Visiting the whole graph G

for each vertex s of G do

$d[s] \leftarrow 0$; $f[s] \leftarrow 0$;

endfor

$nb \leftarrow 0$;

for each vertex s of G do

if d $[s] = 0$ then

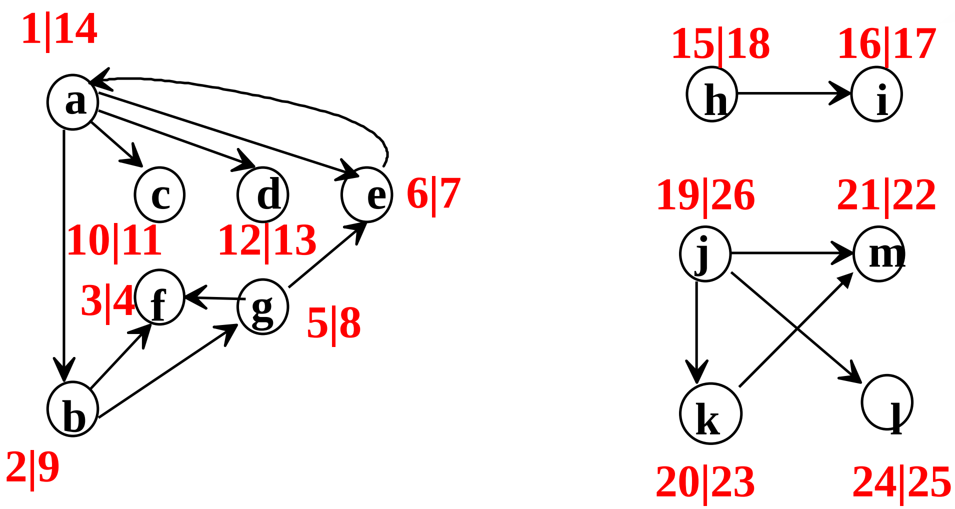
DFSNum (s **)**

endif

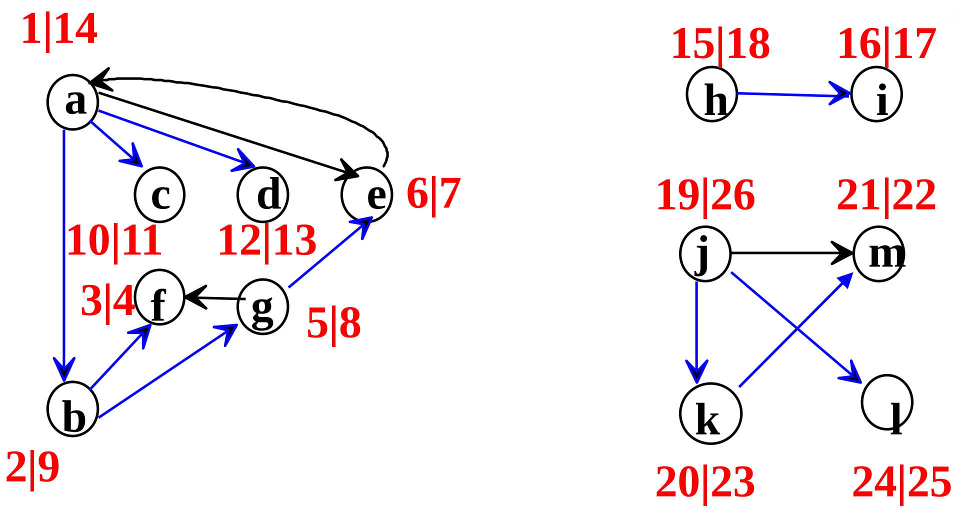
endfor

[d(s)..f(s)] exploration of *s*

Running time: the same as for DFS

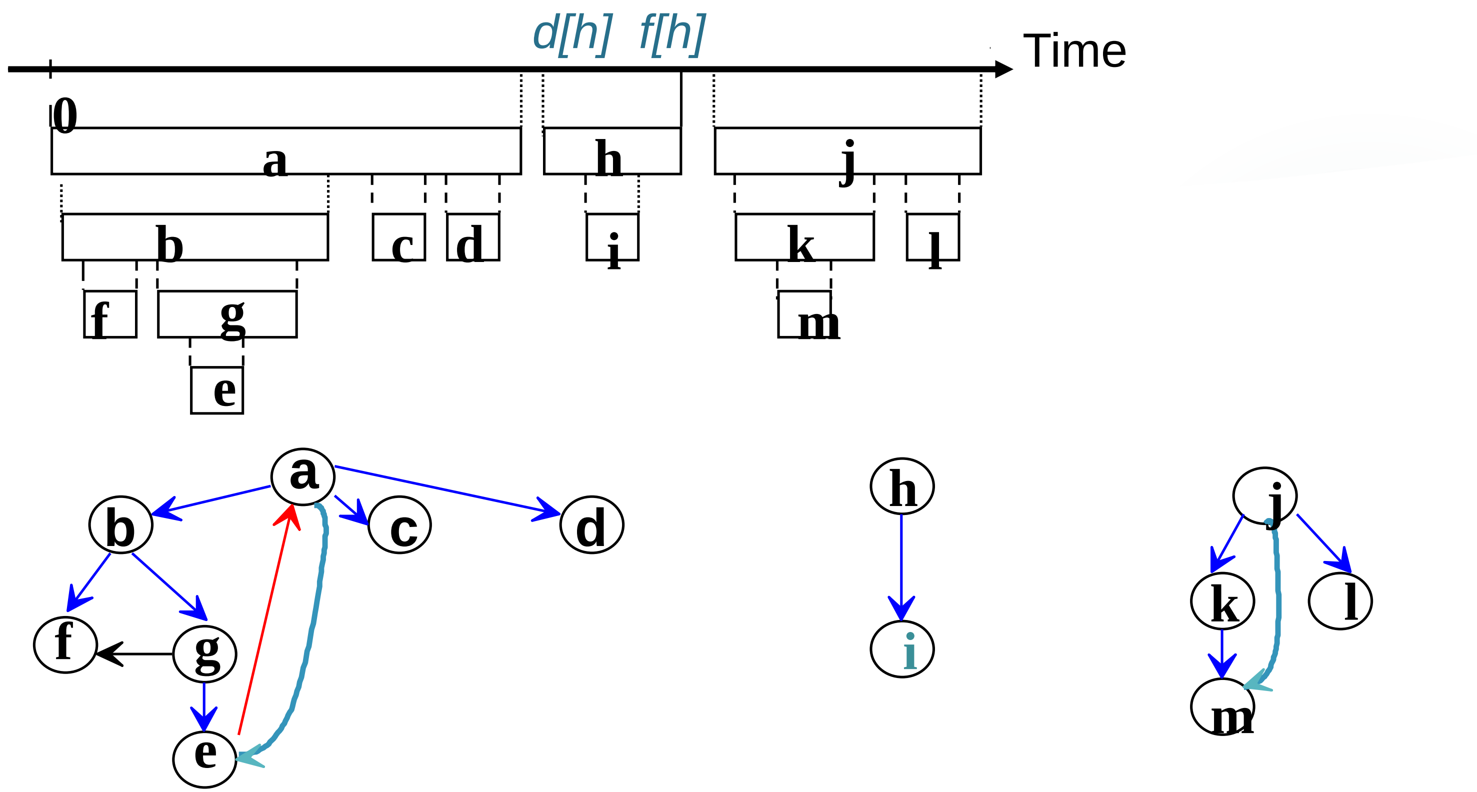


Ordre du parcours : a b f g e c d h i j k m l



Ordre du parcours : a b f g e c d h i j k m l

Embedded or disjoint explorations



« tree » arc

« forward » arc

« back » arc

« cross » arcc

Recognize and use these arcs

(s,t) arc of G is a(n):

- tree or forward arc iff $d[s] < d[t] < f[t] < f[s]$
- back arc iff $d[t] < d[s] < f[s] < f[t]$
- cross arc iff $f[t] < d[s]$

Applications:

- Finding paths (arcs of the tree) or cycles (back arcs)
- Looking for properties involving them (connectivity, bipartition, existence of vertex orderings … see the next course)
- Usually, we make use of these arcs to understand the effects of an algorithm and to prove it … but the algorithms use them only implicitely.