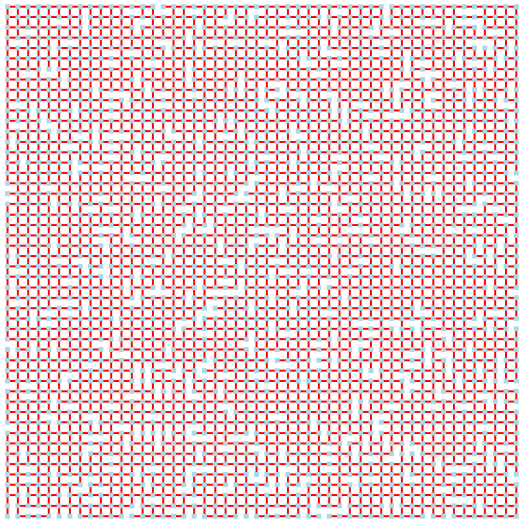
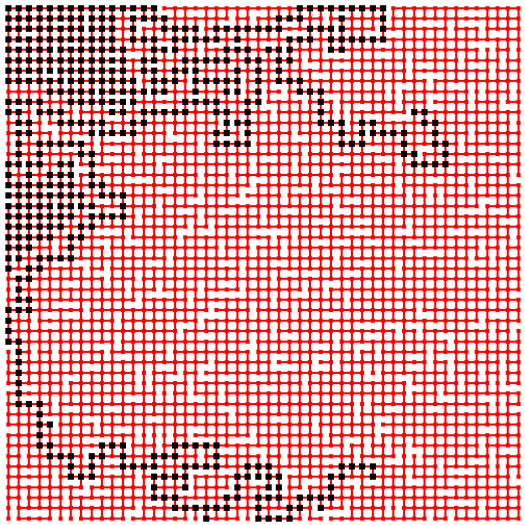
Graph: Fundamental Algorithms

Prof. Frédéric Guinand

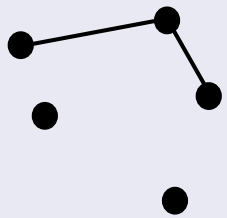




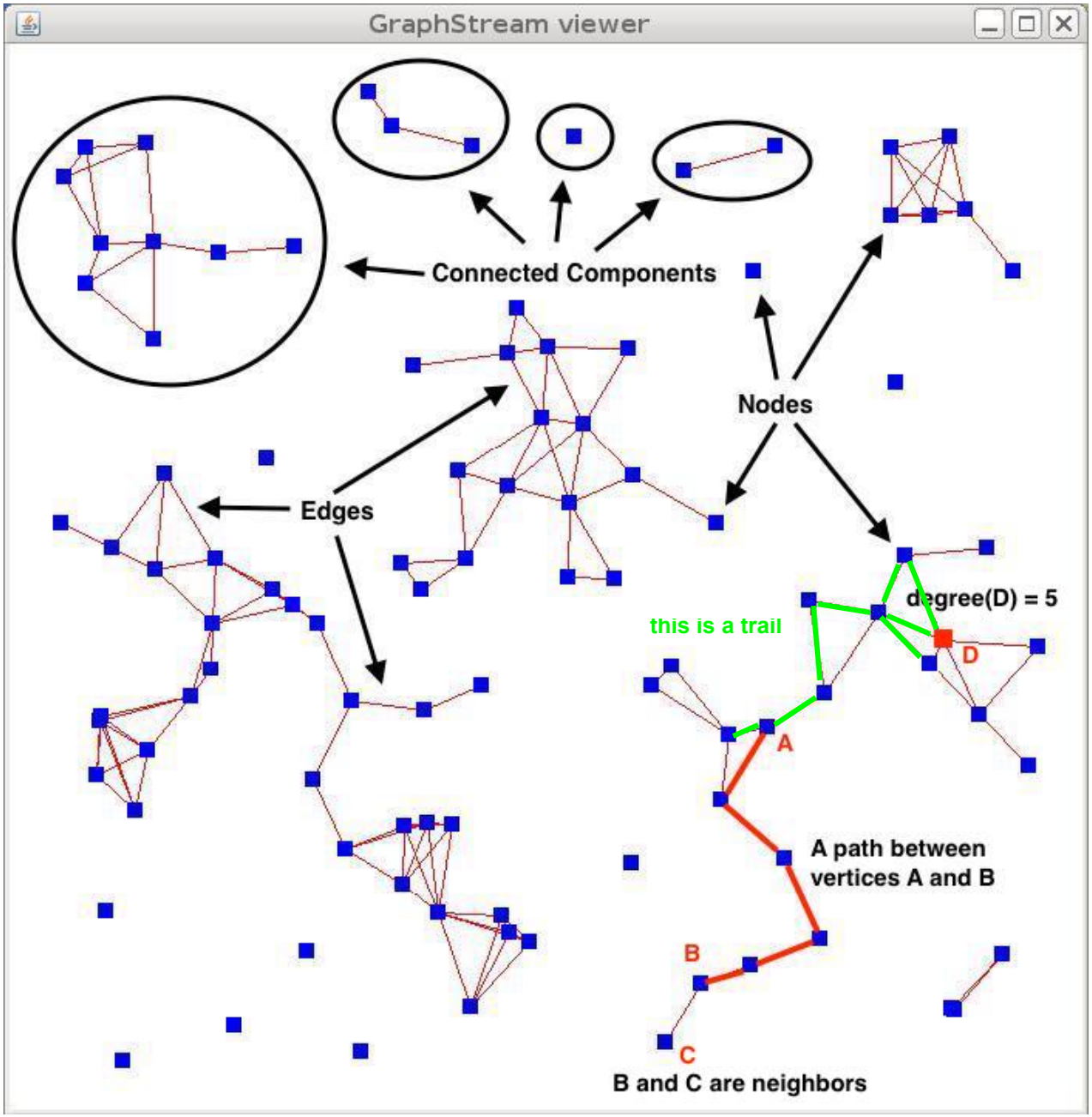
Outline

- Graphs: Basics
- Graph Traversal Algorithms
- The central notion of Distance
- Dijkstra Algorithm

Graphs: basics (again...)

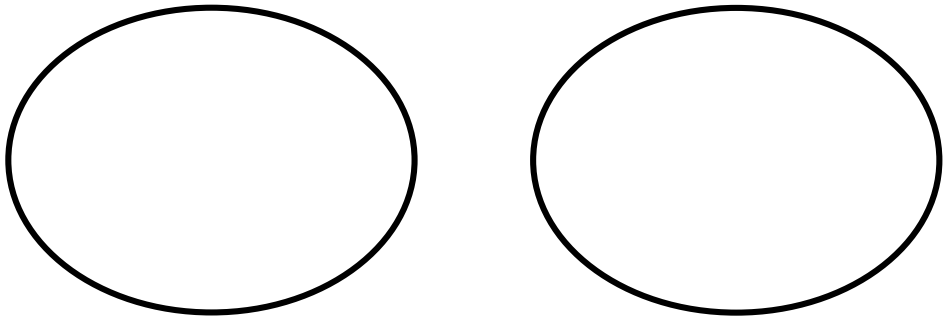


- Given a graph $G = (V, E)$ (a 1-graph)
- given an edge $e = \{v_i, v_j\}$, we say that:
  - v_i is **adjacent** to v_j,
  - v_i is **incident** to e and
  - v_i and v_j are **neighbors**
  - v_i and v_j are the endpoints,extremities of e
- a trail is a sequence of vertices $p = (v_0, v_1, v_2, \ldots, v_k)$ such that $\forall i (v_i, v_{i+1}) \in E$
- a **path** is a trail such that $\forall i$, v_i appears only once in p
- a **cycle** is a closed path (the first and last vertices are the same)
- a **connected component** is a set of vertices $cc = \{v_0, v_1, \ldots, v_n\}$ such that $\forall v_i, v_j \in cc$, $\exists p$ a path containing both vertices
- $G = (V, E)$ is a **connected graph** if $\forall v_i, v_j \in V$, $\exists p$ a path between v_i and v_j


GraphStream viewer
Connected Components
Nodes
Edges
degree(D) = 5
this is a trail
D
A
A path between
vertices A and B
B
C
B and C are neighbors

Basic Measures and Fundamental Algorithms

- Graph connectivity?
- Graph Traversal Algorithms
- Dijkstra Algorithm



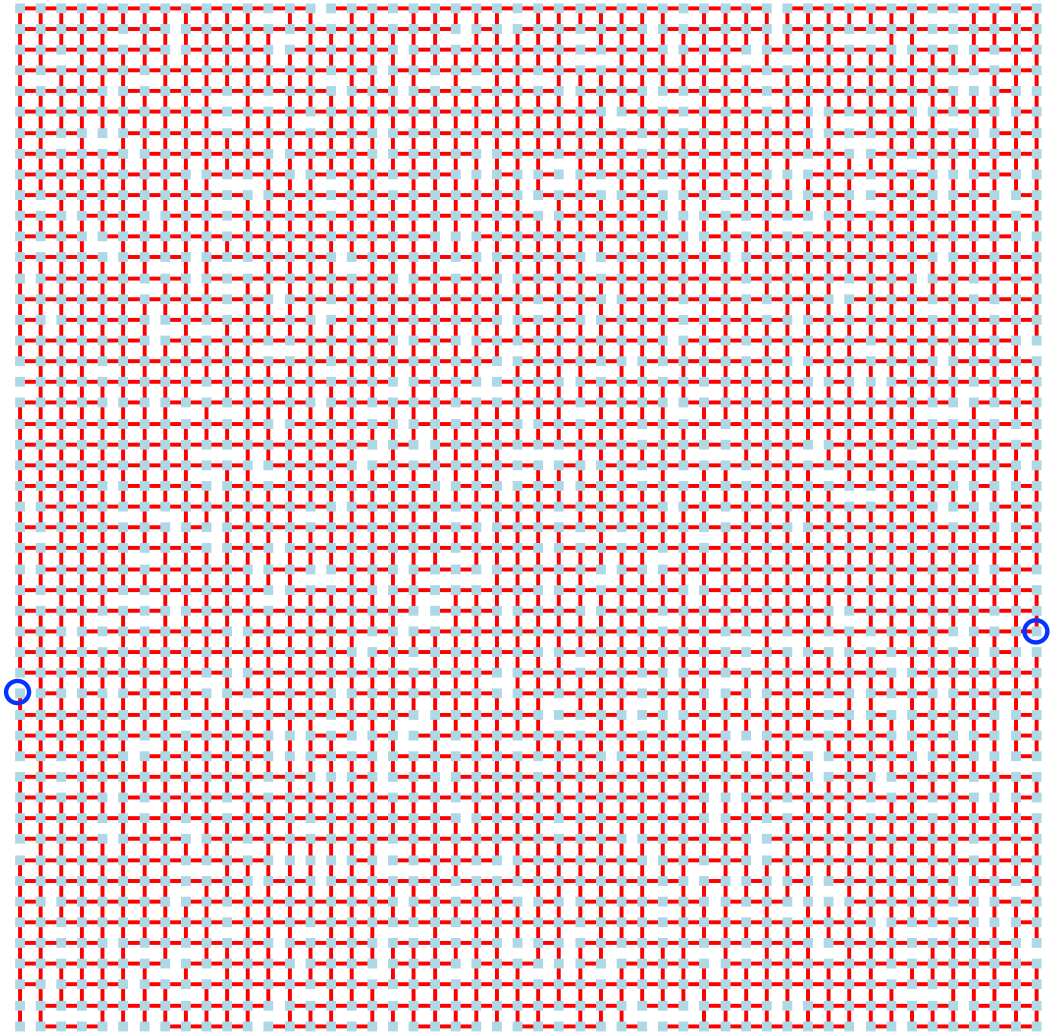
Average Degree

```
sumDegrees ← 0
n ← number of vertices of the graph
for each vertex v of the graph do
    sumDegrees ← sumDegrees + degree(v)
endFor
print "average degree = "+sumDegrees/n
```

- for some measures relying only on local information (like degree) is not sufficient:
  - computing the eccentricity of a node
  - determining if a graph is connected or not

  $\longrightarrow$ visiting neighbors is mandatory

Graph Connectivity



is there a path between these two nodes?

Is this graph connected?

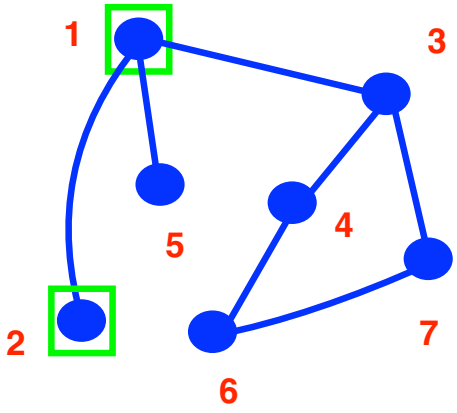
Graph Connectivity

- examining each vertex individually is not enough
- one solution: visiting the graph starting from one vertex and exploring recursively/iteratively the neighborhoods:
  - if all the vertices have been visited: the graph is connected
  - if some vertices were unreachable by a recursive visit of the neighborhoods: the graph is not connected
- requires for each node to have access to the set of its neighbors or to its incident edges.

Graph Visit

Principle: iterative visit of the neighborhoods of vertices

1. $M \leftarrow \emptyset$
2. $M \leftarrow \{$a randomly chosen vertex$\}$
3. **while** $M \neq \emptyset$ **do**
4. $\quad v \leftarrow$ pick one vertex in M
5. $\quad$ mark v as visited
6. $\quad$ **for** each neighbor w of v **do**
7. $\quad\quad$ **if** w has not been visited (AND $w \notin M$) **then**
8. $\quad\quad\quad M \leftarrow M \cup \{w\}$
9. $\quad\quad$ **endIf**
10. $\quad$ **endFor**
11. **endWhile**



M = {2,3,5}
M = {3,5}

odwiedzony wezel

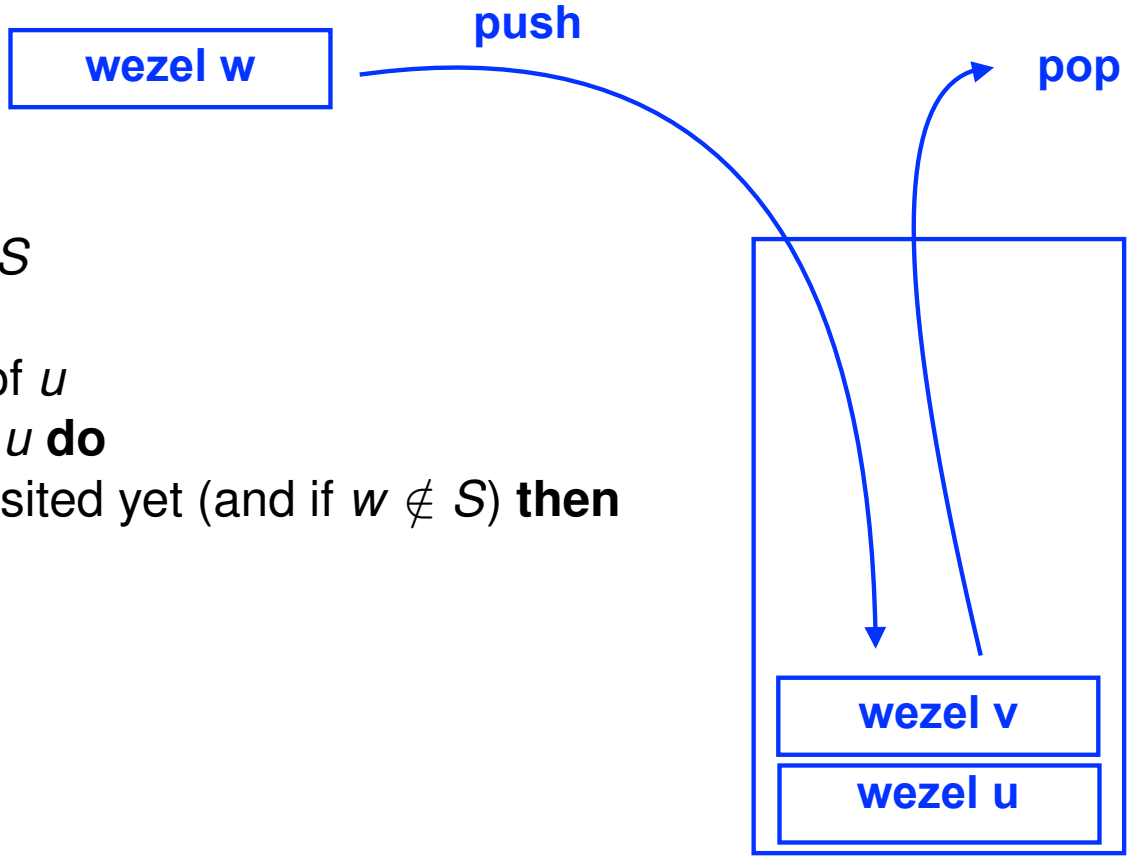
Fundamental Graph Traversal Algorithms

- there exist at least two well-known graph traversal algorithms, they mainly differ in the way the vertices are managed in the memory:
  - Depth-First Search Algorithm (DFS)
  - Breadth-First Search Algorithm (BFS)
- many other strategies can be considered

Depth-First Search

- the memory structure is a Stack (Last In First Out - LIFO data structure)
- push adds an element on the top of the Stack
- pop removes the element located on the top of the Stack

v randomly chosen vertex
push v on the Stack S
while $S \neq \emptyset$ **do**
    $u \leftarrow$ pop element from S
    mark u as visited
    $L_u \leftarrow$ set of neighbors of u
    for each neighbor w of u **do**
        if w has not been visited yet (and if $w \notin S$) **then**
            push w on S
        endIf
    endFor
endWhile

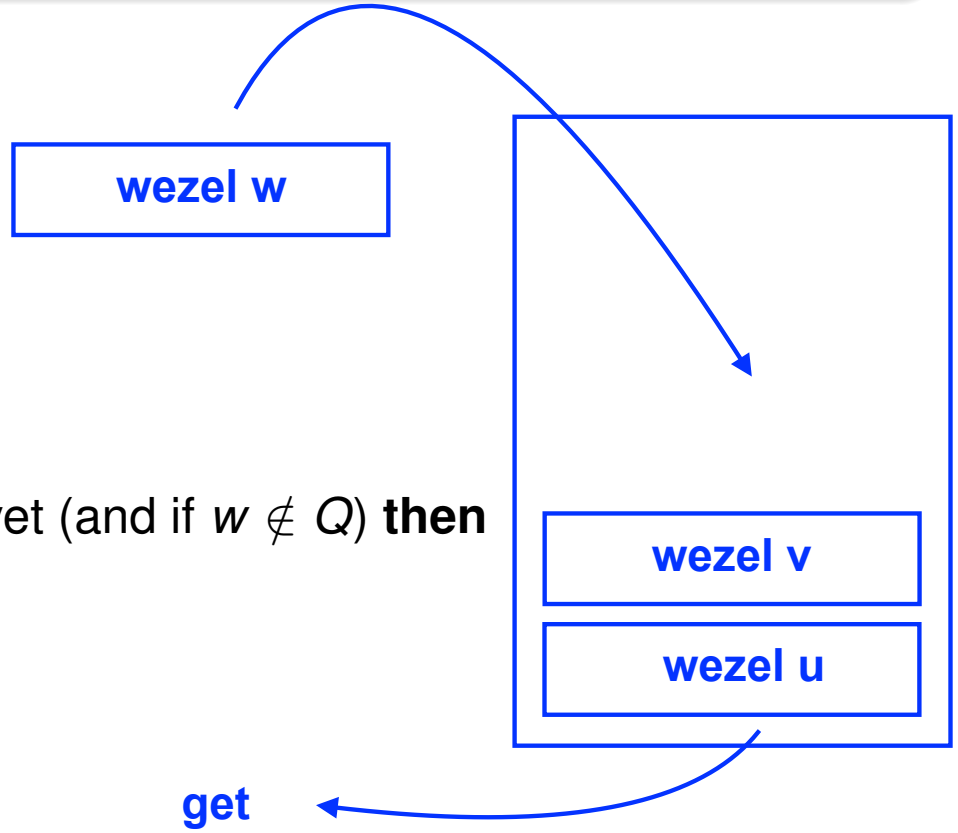


Breadth-First Search

- the memory structure is a Queue (First In First Out - FIFO data structure)
- add puts an element at the end of the Queue
- get removes the element located at the head of the Queue

add

v randomly chosen vertex
add v on the Queue Q
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ get element from Q
    mark u as visited
    $L_u \leftarrow$ set of neighbors of u
    for each neighbor w of u **do**
        if w has not been visited yet (and if $w \notin Q$) **then**
            add w to Q
        endIf
    endFor
endWhile

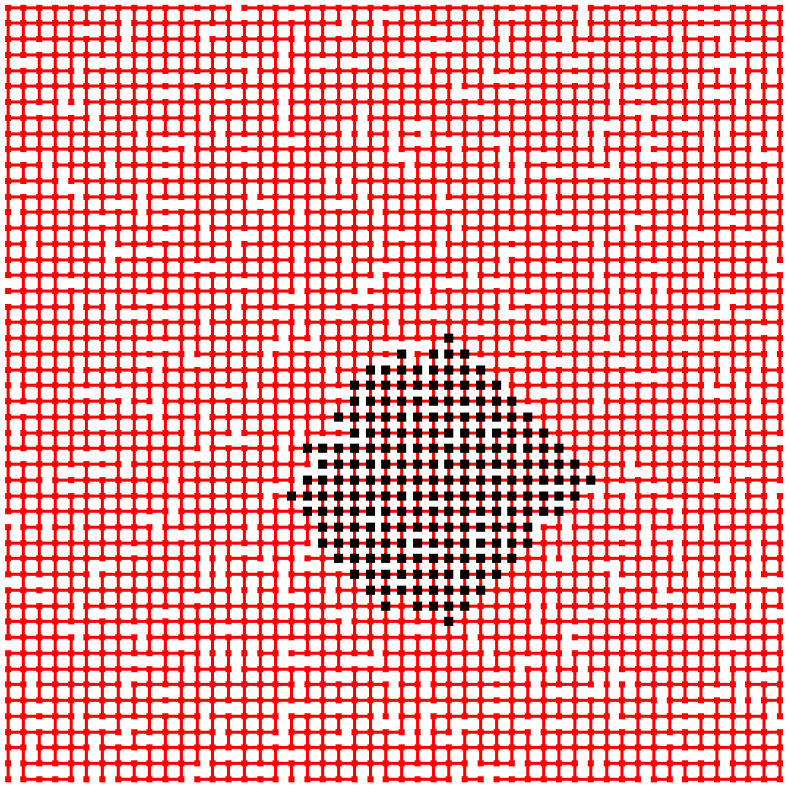


SimilaritiesDifferences

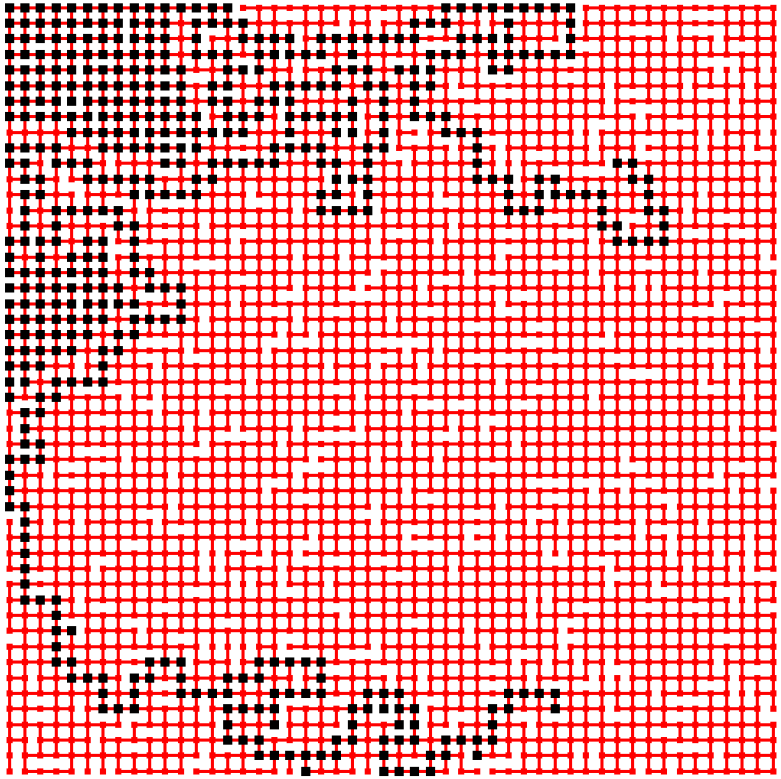
```
v randomly chosen vertex
push v on the Stack S
while S ≠ ∅ do
    u ← pop
    mark u as visited
    L_u ← set of neighbors of u
    for each neighbor w of u do
        if w has not been visited yet then
            push w
        endIf
    endFor
endWhile
```

```
v randomly chosen vertex
add v on the Queue Q
while Q ≠ ∅ do
    u ← get
    mark u as visited
    L_u ← set of neighbors of u
    for each neighbor w of u do
        if w has not been visited yet then
            add w
        endIf
    endFor
endWhile
```

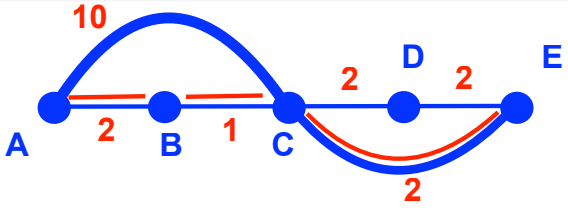
Execution



Breadth-First Search



Depth-First Search



distance(A,E) = 2

graph with values -> valuated graph
distance(A,E) = 5

Distance: a central notion

Distance

- given a non valuated graph $G = (V, E)$ and given two vertices $u, v \in V$

  distance(u, v) is the **shortest path** between these two nodes: a path linking u to v with the minimum number of edges/arcs (if directed graph)
- if the edges are valuated by numbers the distance between two nodes corresponds to the path that minimizes the sum of the values of the edges defining the path

  $\longrightarrow$ **Dijkstra** algorithm

Distance-based metrics

Associated metrics

- the **diameter** of a graph is defined as the maximum value of the distance between any two nodes in the graph:

  $D_G = \max\{\text{distance}(u, v) | u, v \in V\}$

- the **eccentricity** of a vertex v is the maximum distance between v and any other vertex in the graph:

  $e_v = \max\{\text{distance}(v, u) | u \in V\} \Rightarrow D_G = \max\{e_v | v \in V\}$

- the **radius** of a graph is defined as the minimum value of the eccentricity:

  $R_G = \min\{e_v | v \in V\}$

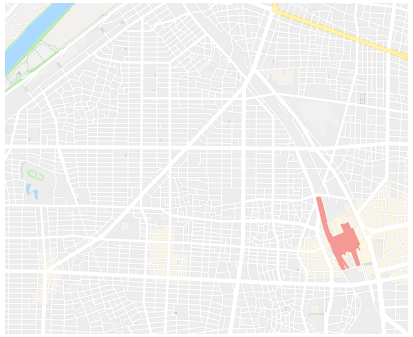
  it corresponds intuitively to the closest node to the other nodes within the graph ($\sim$ "center").

Diameter: an important metrics

- the diameter is an important measure for a graph since it gives an upper bound on the distance/time/cost between any two nodes in the graph

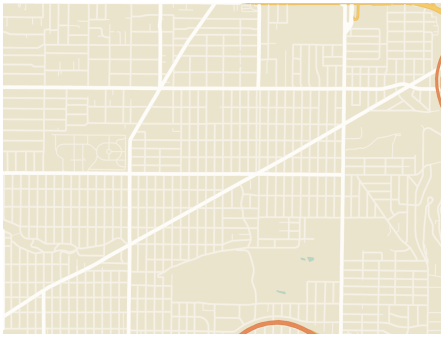
  for instance: reducing/limiting the diameter may be a guide for designing/redefining urban networks

Barcelona (Spain)

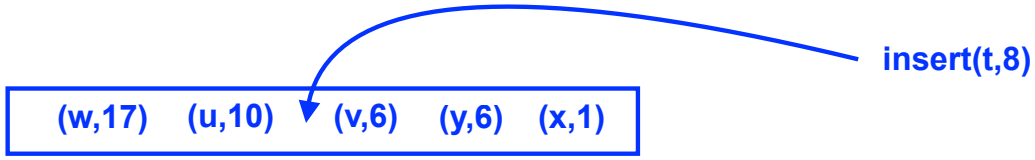


Nagoya (Japan)

Alexandria (Egypt)



Portland (US)



Dijkstra Algorithm

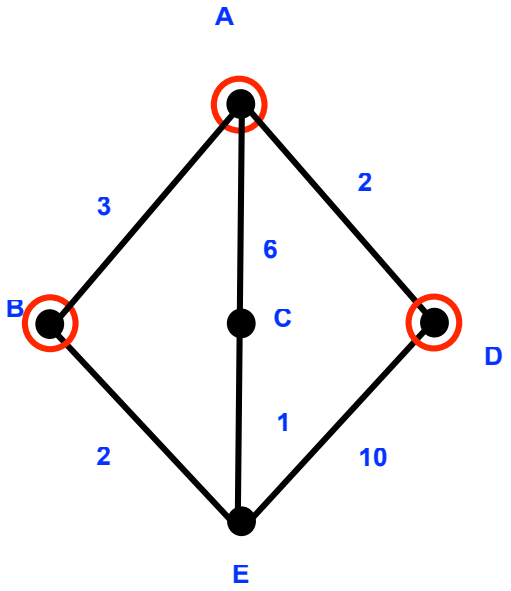
Data Structure

- same idea but the memory structure is a Priority Queue
- insert(v, p_v) inserts the element v in the Priority Queue such that every element u in front of v has a p value greater or equal to p_v, and every element w located after v in P has a p value lower than or equal to p_v
- get removes the element located at the head of the Priority Queue
- remove(v) removes the element v from the Priority Queue, whatever its position

Dijkstra Algorithm

Principle

v randomly chosen vertex
add v to the Priority Queue P
while $P \neq \emptyset$ **do**
    $u \leftarrow$ get element from P
    mark u as visited
    $L_u \leftarrow$ set of neighbors of u
    for each neighbor w of u **do**
        if w has not been visited yet **then**
            if $w \in P$ **then** remove(w)
            compute p_w **and insert**(w, p_w) **in** P
        endIf
    endFor
endWhile



PQ = {(A,0)}
PQ = {(D,2),(B,3)(C,6)}
PQ = {(B,3)(C,6),(E,12)}
PQ = {(E,5),(C,6)}

Dijkstra Algorithm

Distance

- the central notion in Dijkstra algorithm is the distance
- in Dijkstra algorithm...

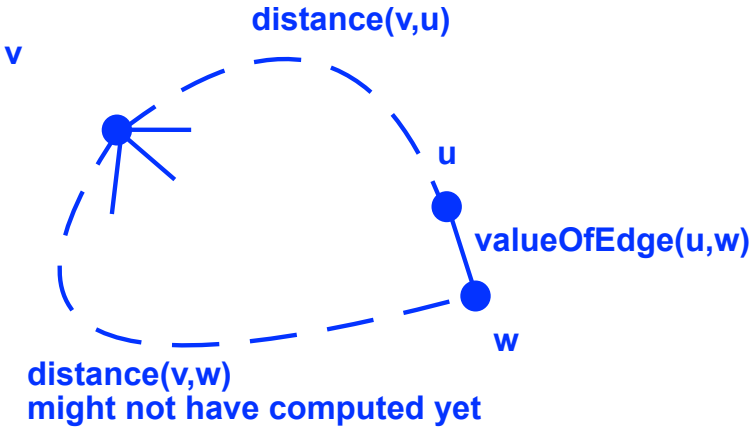
v randomly chosen vertex

...

for each neighbor w of u **do**

...

compute p_w and insert(w, p_w) in P



- p_w is the distance between v and w
- distance(v,w) $\leftarrow$ min(distance(v,w) , distance(v,u) + valueOfEdge(u,w))

Diameter, Eccentricity and Radius

- using Dijkstra, the Eccentricities of the nodes can be computed and...
- the Diameter and Radius of the graph can be deduced

for each $v \in V$ **do**
    $\text{eccentricity}_v \leftarrow \text{dijkstra}(v)$
endFor
$\text{diameter} \leftarrow \max\{\text{eccentricity}_v | v \in V\}$
$\text{radius} \leftarrow \min\{\text{eccentricity}_v | v \in V\}$