Graph Traversing I

1 Graph Representations

- Humans understand graphs pictorially as a collection of vertices and edges
- For a computer to understand graphs it needs to be stored in a structured way

Two standard data structures to represent graphs:

- A collection of adjacency lists
  - For every vertex, a lined list with all its neighbours
  - These lists are called adjacency lists
- An adjacency matrix
  - The vertices are (arbitrarily) numbered 1,2,3,...,n
  - a square $n \times n$ matrix such that the element (i,j) is:
    - equal to 1 if vertices i and j are adjacent
    - equal to 0 otherwise

For undirected graphs:

- The adjacency matrix is symmetric (the elements (i,j) and (j,i) are equal)

2 Graph traversing

- Both data structures
  - store only ”local” information about the graph (i.e. adjacencies)
  - The ”global” information is provided implicitly
- How can you know if the graph is **connected**?
  - If you start at a specific vertex, can you reach every other vertex?
  - If not, can you list the ”reachable vertices”

Where is the difficulty? - we want to visit all accessible vertices but avoid running into ”cycles”

Algorithm to traverse a labyrinth:

- Whenever you find an unvisited vertex continue to explore it from deeper
- If no more options, use a *ball of string* to return to junctions that you previously saw but did not investigate

3 Depth First Search (DFS)

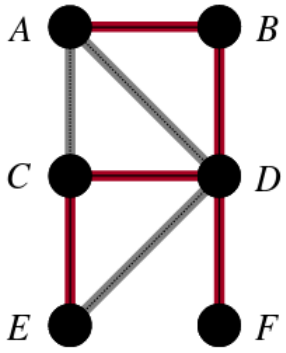
The depth first search algorithm:

```
DFS(G,u)
visited[u]=1
print u
for each vertex v ∈ Adj[u]
    if visited[v]==0 then
        DFS(G,v)
```

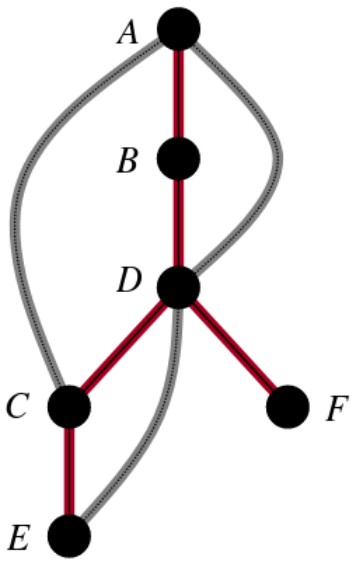
- Initially all vertices are marked as ”unvisited” i.e. visited[u]=0 for all vertices u
- When we visit a new vertex u
  - We mark it as visited (line 1)
  - We call (recursively) the same algorithm (DFS) for all unvisited neighbours v of u (lines 3-5)

4 DFS in action

The graph:



The DFS-traversal schematically:



The algorithm runs in linear time
(one operation for each vertex and edge)

A DFS-visiting order of the vertices: A, B, D, C, E, F

12

5 Graph traversing

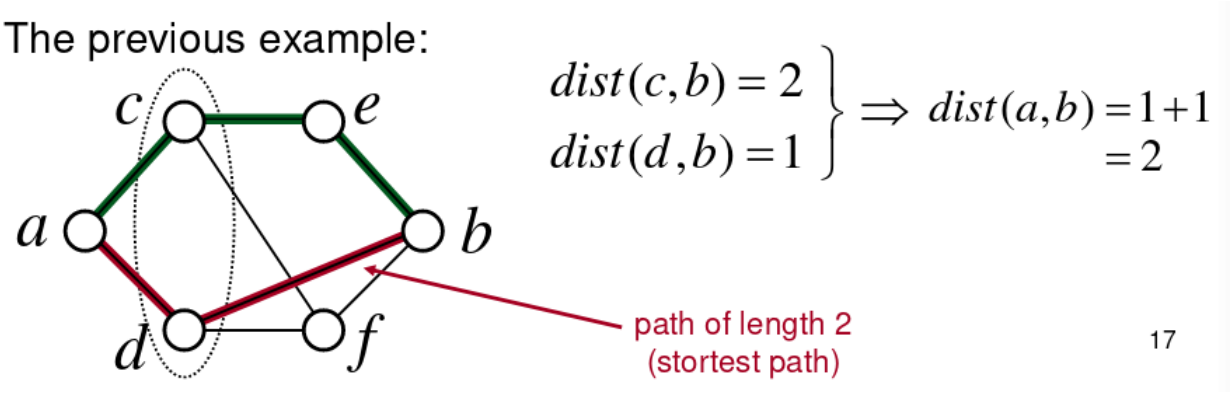
- The Depth First Search (DFS) algorithm can be used to traverse the whole graph
- It can also be used for directed graphs:
  - In this case Adj[u] denotes the set of vertices that are accessible from u with one edge
- Variations of DFS are mainly used for ”connectivity type” problems

6 Shortest Paths

- The length of a path connecting two vertices a,b is the number of edges in the path
- The distance between two vertices a,b in a graph is the smallest length of a path that connects a and b
- The shortest path problem: given two vertices a and b, what is their distance?

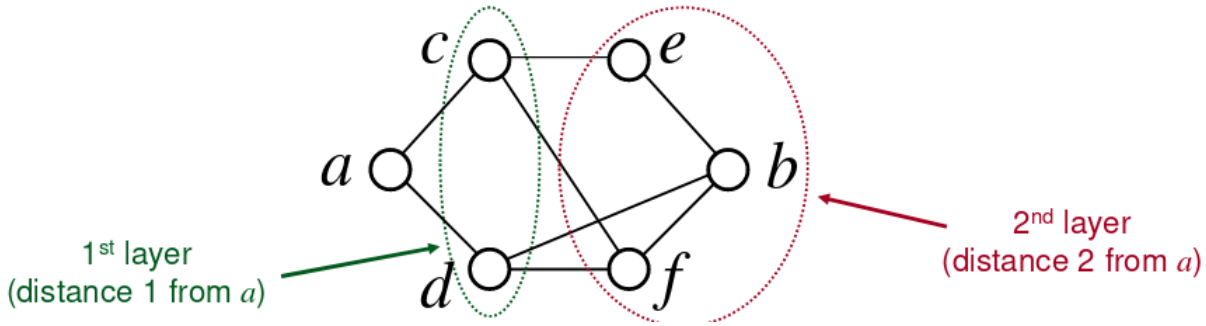
Sketch of a simple algorithm for shortest paths:

- You stand on a vertex a of the graph and need to find your distance to the vertex b
- Ask all your neighbours what is their distance to b and compute the smallest of these distances, say x
- Then your distance to b is equal to $x + 1$



In this algorithm, we proceed layer by layer:

- We expand the "frontier" between visited and unvisited vertices, across the breadth of the frontier



- For every k=1,2,3 the algorithm:
  - First visits all vertices at distance k from a
  - Then all vertices at distance k+1

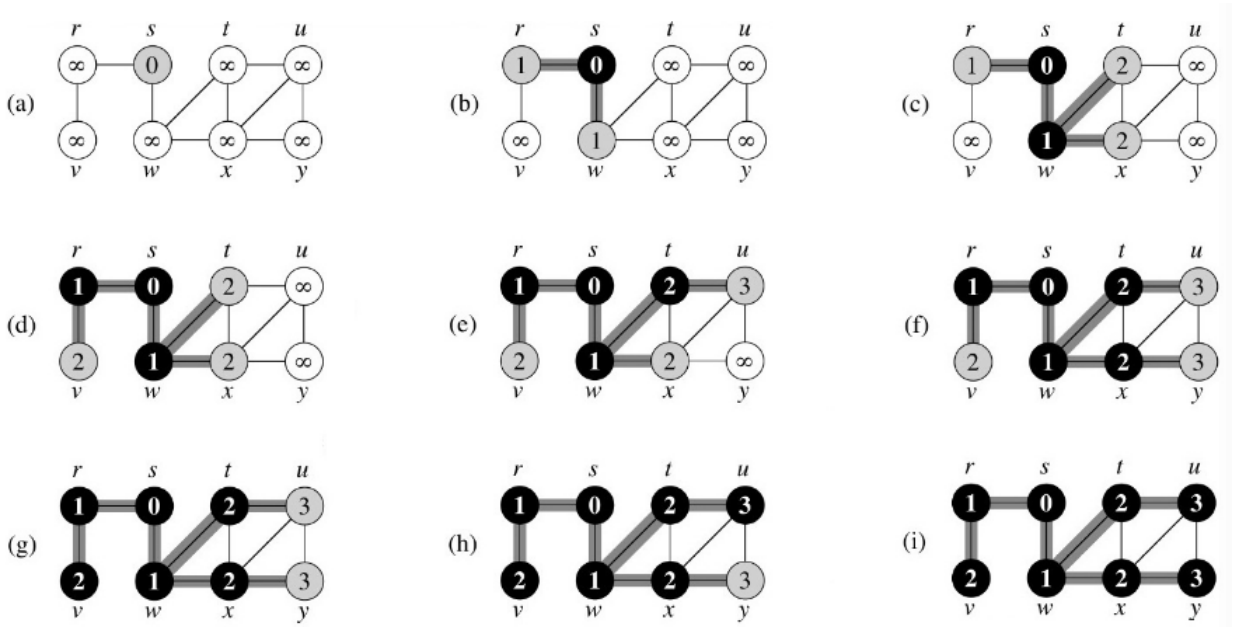
7 Breadth First Search (BFS)

The natural alternative to DFS

```
BFS(G,a,b)
i=0
label=[a]=0
while b is unlabeled
    for each vertex u with label[u]==i
        for each unlabeled vertex v ∈ Adj[u]
            label[v]=i+1
    i=i+1
return label[b]
```

- BFS is an iterative algorithm, i.e. no recursive calls
- The label of a vertex u equals its distance from a
- We could continue iteration until all vertices are labelled
- Initially all vertices are marked a unlabelled i.e. label[u]= -1 or ∞

8 BFS In action



- white vertex: unlabeled
- gray vertex: labelled, but not all its neighbours are labelled
- black vertex: labelled, and all its neighbours are labelled

20