Data Structures

Lecture 14: Graph Traversals (cont.)

Nopadon Juneam
Department of Computer Science
Kasetsart university

Outlines

- Graph traversals (cont.)
- Breath-first search and its implementation
- Breath-first search properties

Breath-First Search / Breath-First Traversal

- ***Breath-First Search*** (**BFS**): Like DFS, but in BFS we unroll the string in a more conservative manner;

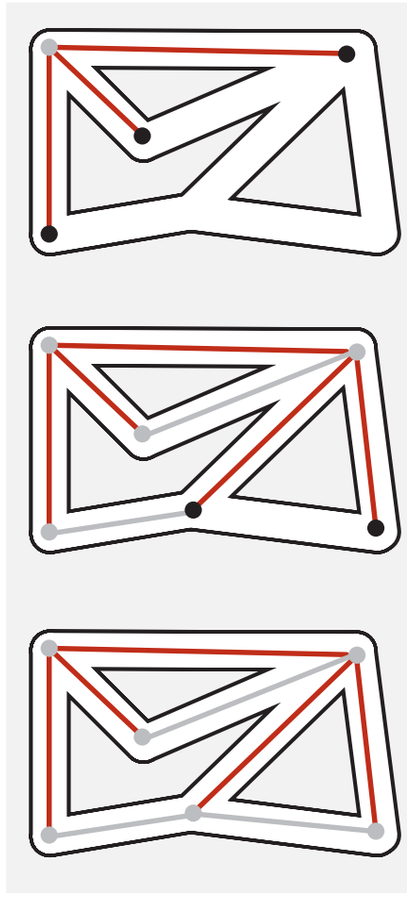
  - Start at vertex *s* which we assign it as level 0, define the anchor for the string.

  - *First round*: let out the string the length of one edge, we visit all the vertices we can reach from *s.* These vertices are assigned as level 1

  - *Second round*: let out the string the length of two edges, we we visit all the vertices we can reach from level 1 vertices and not previously assigned to a level. These vertices are assigned as level 2

  - ….

  - Terminates when every vertex has been visited



BFS in Undirected Graph (1)

- BFS proceeds in rounds and subdivides the vertices into levels, we will need to memorize vertices at each level

```
BFS-visit(s, adjList, visited = {}):
    visited = visited U {s}
    L0 = {s}
    i = 0
    while Li is not empty:
        Li+1 = {}
        for each u in Li:
            for each v in adjList[u]:
                if v is not in visited:
                    visited = visited U {v}
                    Li+1 = Li+1 U {v}
        i = i+1
```

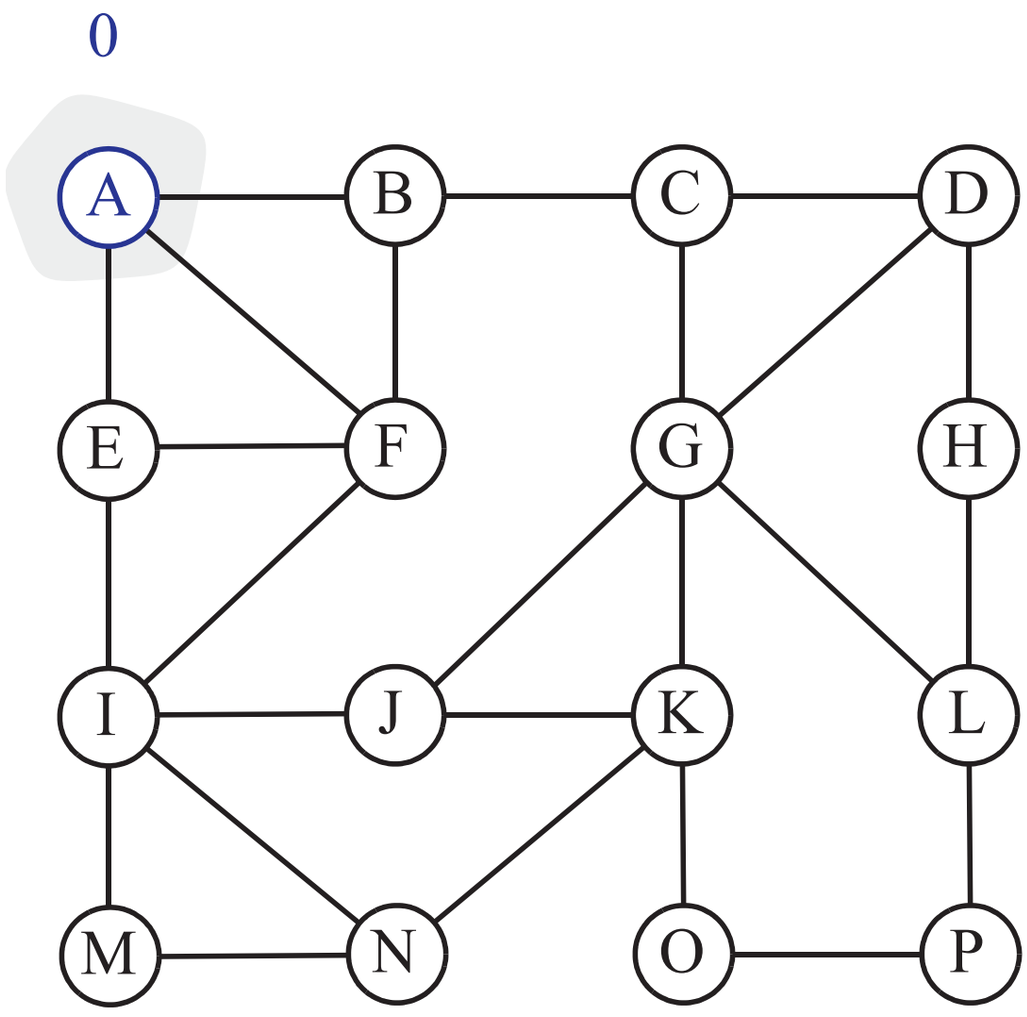
- *Remarks:* `BFS-visit(s, adjList, {})` only sees stuff reachable from vertex `s`; only the connected component that contains `s`; Therefore, it will explore the entire graph if the graph is connected

BFS in Undirected Graph (2)

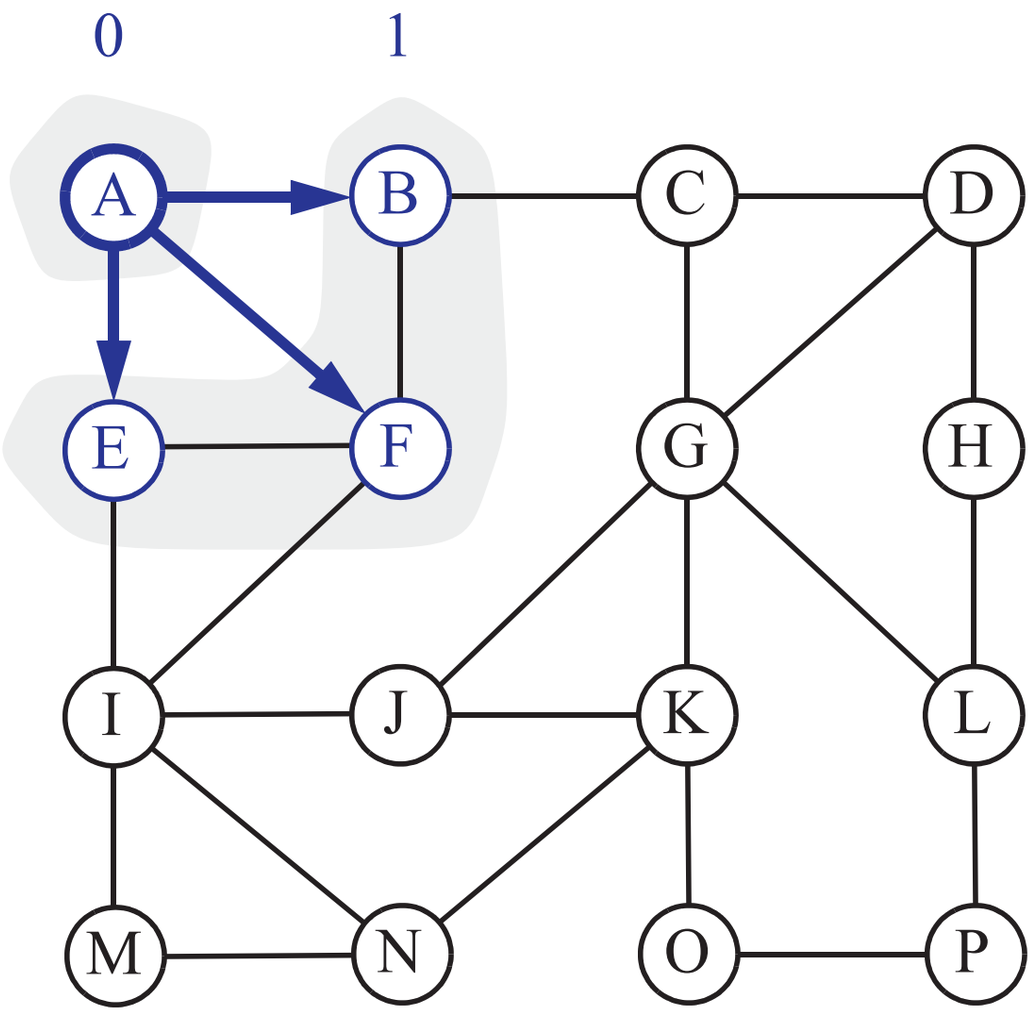
- To really explore the entire graph, we need to apply BFS at each unvisited vertices

```
BFS-explore(V, adjList):
    visited = {}
    for each s in V:
        if s not in visited:
            BFS-visit(s, adjList, visited)
```

BFS Example (1)

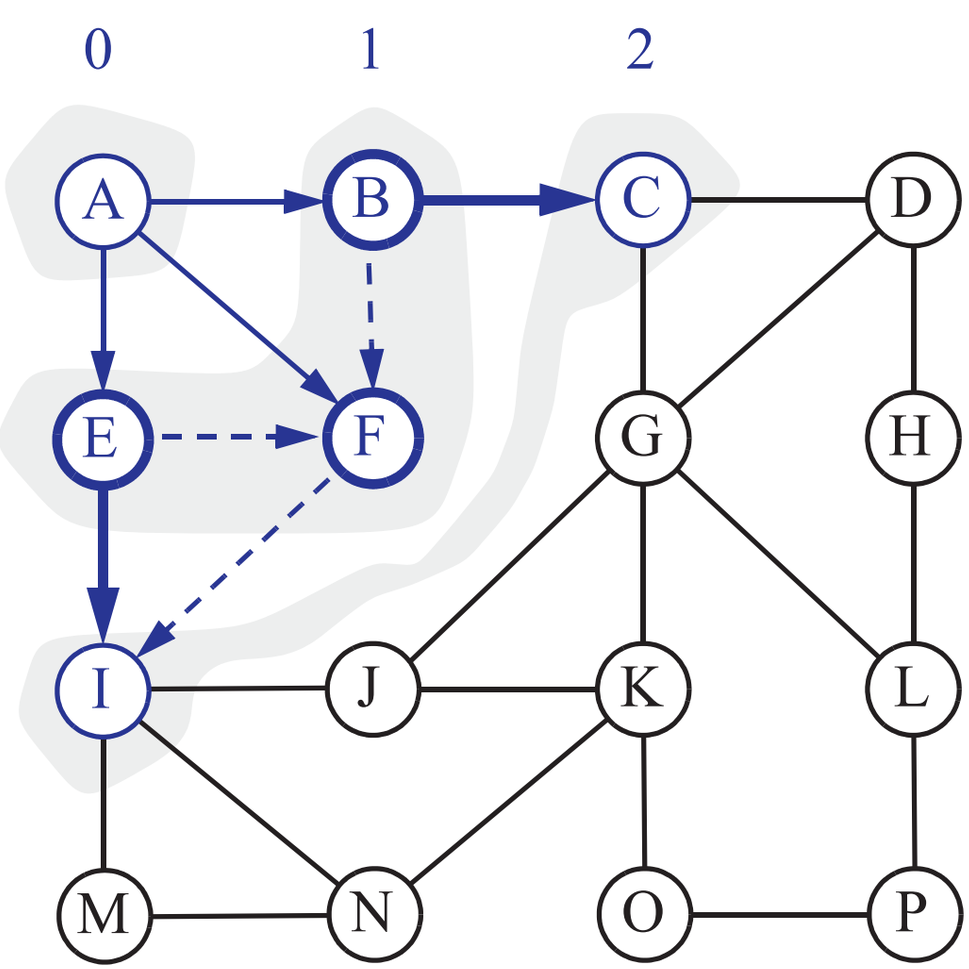


(a)

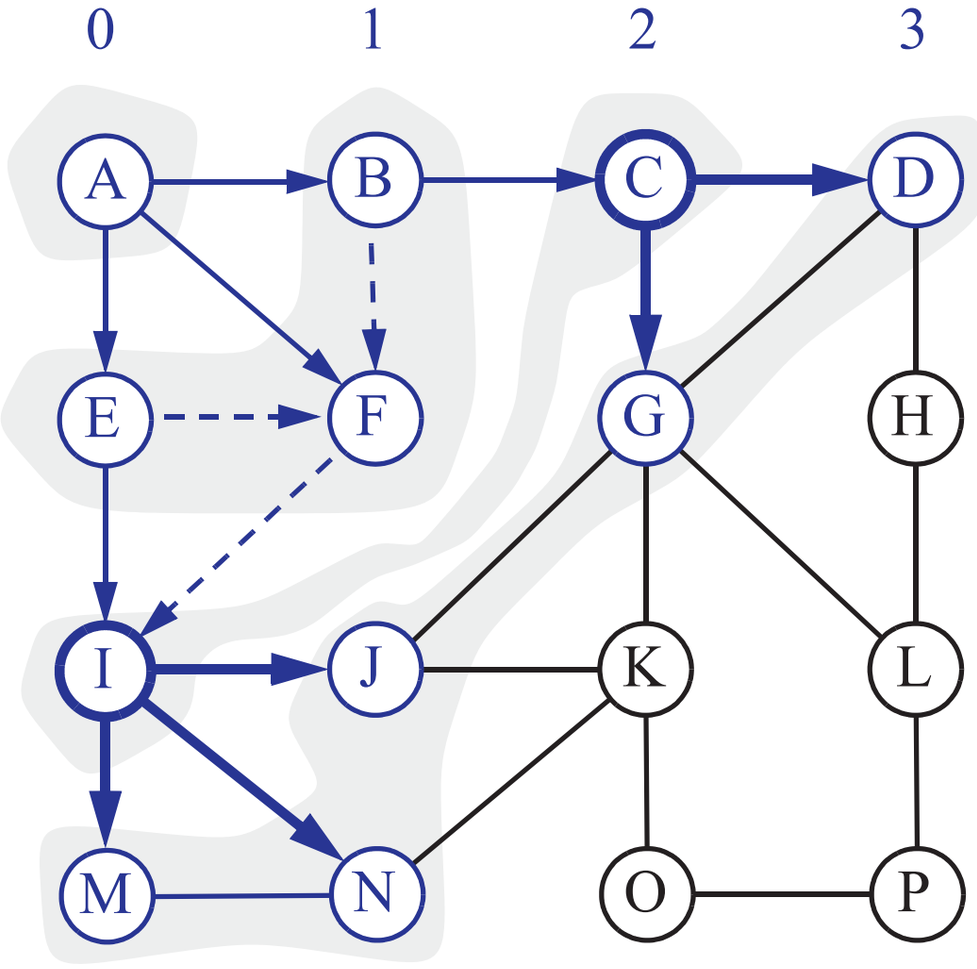


(b)

BFS Example (2)

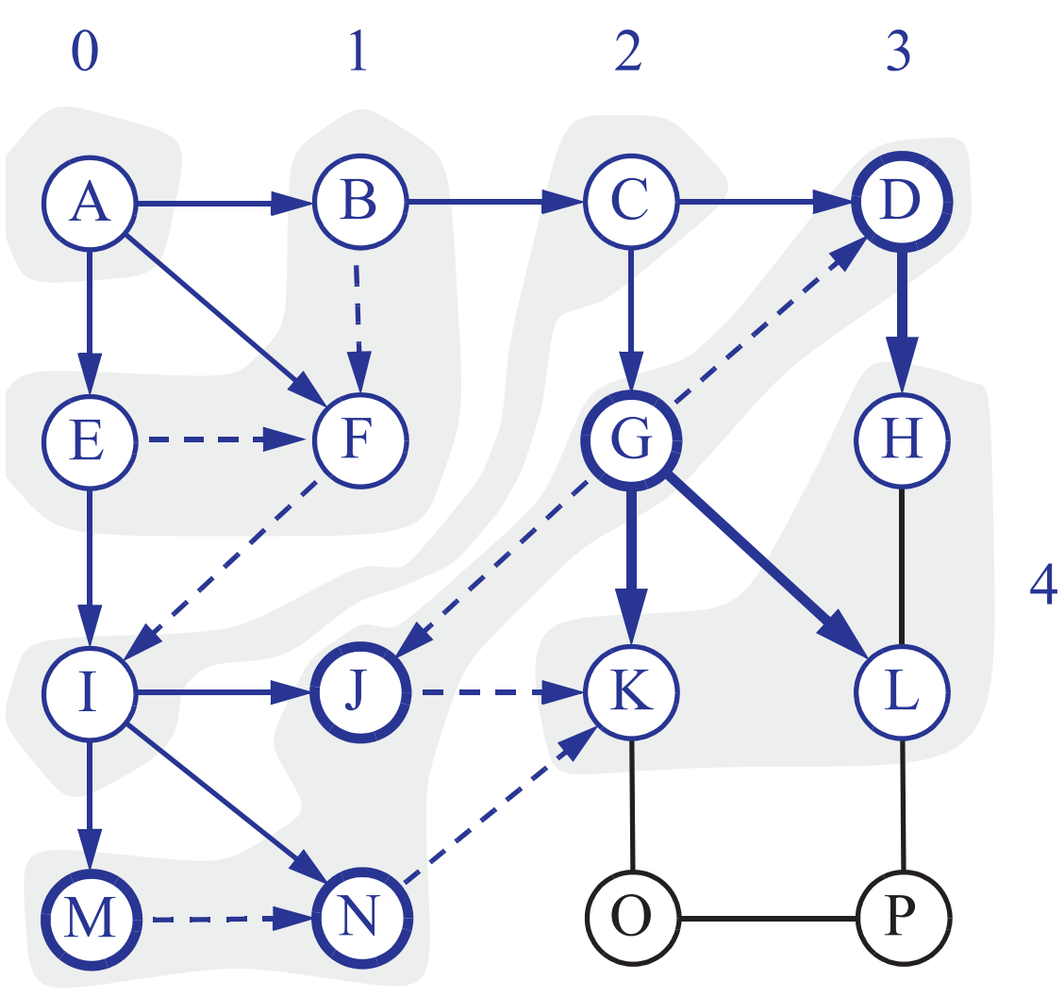


(c)

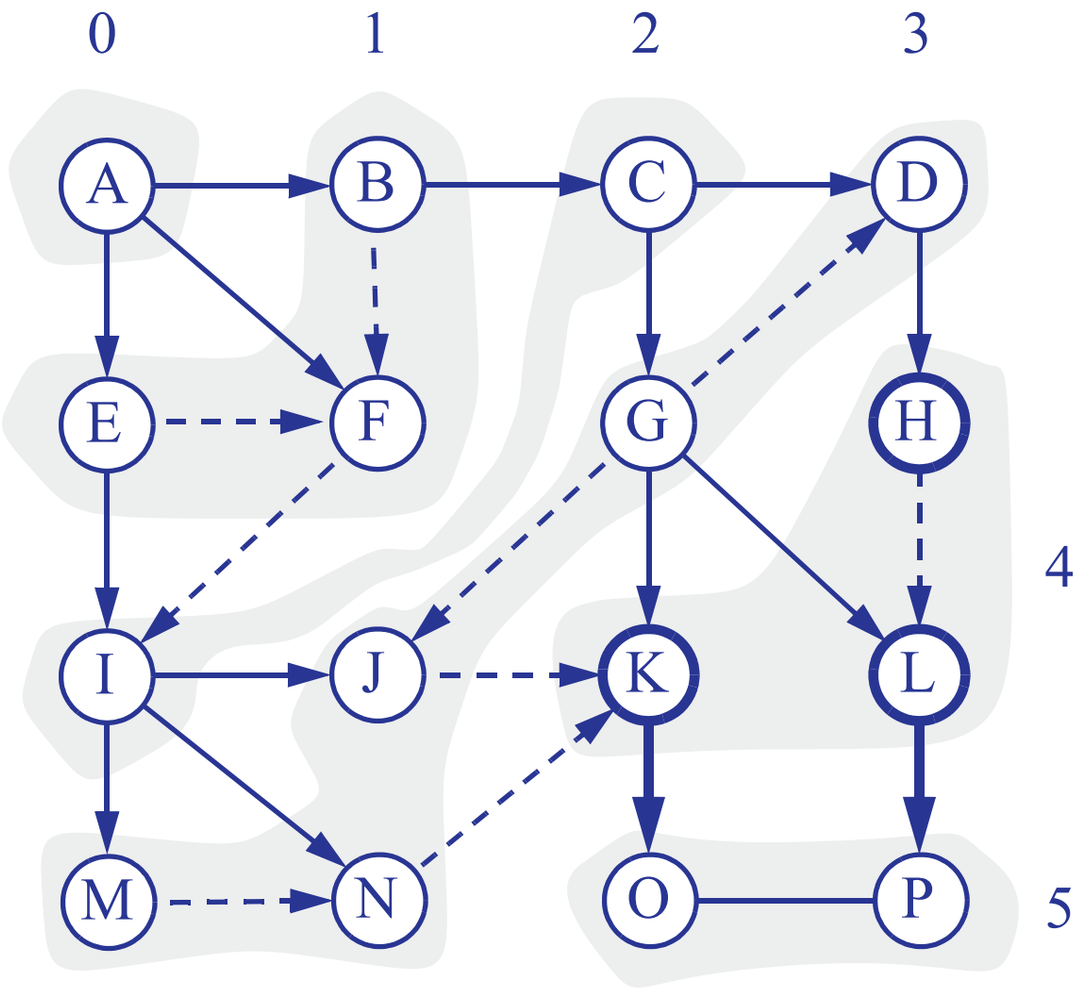


(d)

BFS Example (3)



(e)



(f)

BFS's Complexity

- With adjacency-list representation, like DFS, BFS takes time $O(n+m)$ to traverse a graph with n vertices and m edges

- *Analysis*: In BFS–`visit`, we only need to visit the neighbours of *u,* for each *u* in V

$$\sum_{u \in V} deg(u) = O(m)$$

- The time taken by BFS–`explore` just adds $O(n)$

BFS Implementation in C++ (1)

```
// C++ program to print BFS traversal from a given vertex in a  given graph
#include<iostream>
#include<list>
using namespace std;

// Graph class represents a undirected graph using adjacency list
// representation
class Graph
{
    int V;    // No. of vertices
    list<int> *adj; // Pointer to an array containing adjacency lists
public:
    Graph(int V);   // Constructor
    void addEdge(int v, int w); // Function to add an edge to graph
    void BFSVisit(int s); // BFS traversal of the vertices reachable from s
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list
    adj[w].push_back(v); // Add v to w's list
}
```

BFS Implementation in C++ (2)

```
// BFS traversal of the vertices reachable from s
void Graph::BFSVisit(int s)
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;
    // Create a queue for BFS
    list<int> queue;
    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue.push_back(s);
    // 'i' will be used to get all adjacent vertices of a vertex u
    list<int>::iterator i;
    int u;

    while(!queue.empty()){
        // Dequeue a vertex from queue and print it
        u = queue.front();
        cout << u << endl;
        queue.pop_front();
        // Get all adjacent vertices of the dequeued vertex u. If a adjacent has not been visited,
        // then mark it visited and enqueue it
        for (i = adj[u].begin(); i != adj[u].end(); ++i) {
            if (!visited[*i]) {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}
```

BFS Implementation in C++ (3)

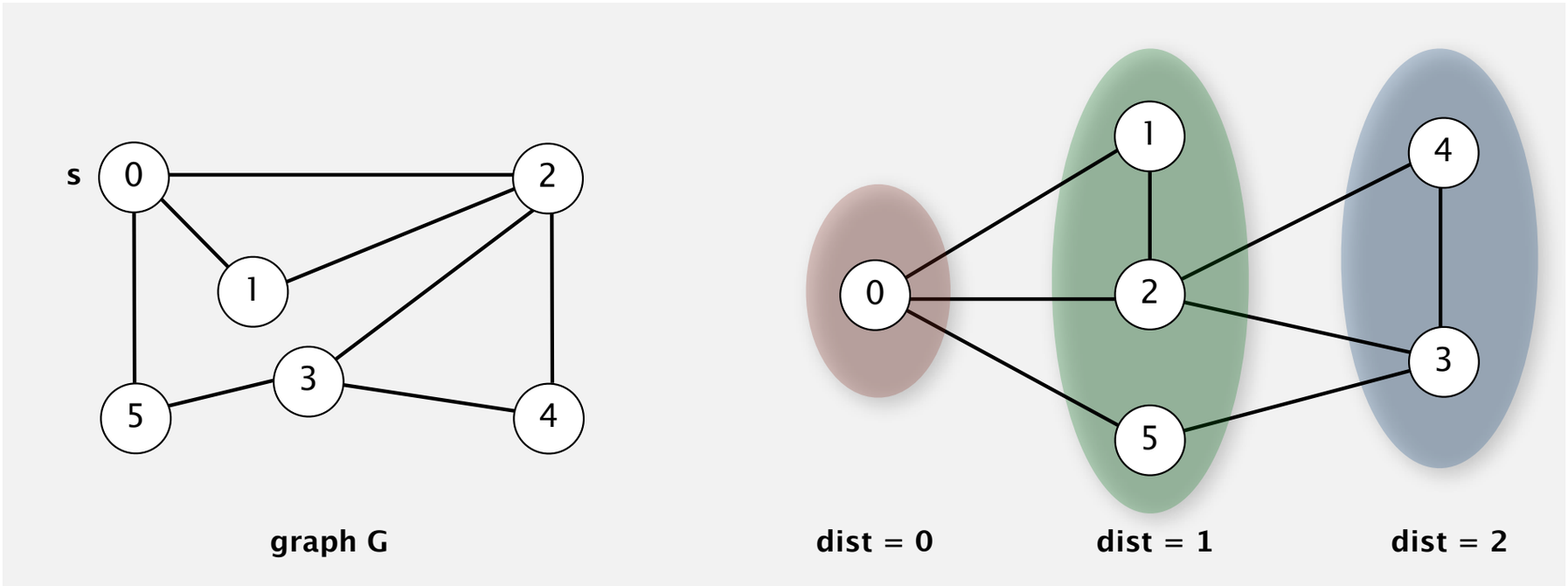
```
int main()
{
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);

    cout << "Following is Breath-First Traversal (starting from vertex 0) \n";
    g.BFSVisit(0);
    return 0;
}
```

Output:
Following is Breath-First Traversal (starting from vertex 0)
0
1
2
3

BFS's Properties

- **Proposition**: If G is a connected graph, BFS computes a *shortest paths* (paths of shortest length) from *s* to all other vertices.



- *Remark*: BFS examines vertices in increasing distance order from *s.* So, if *v* is at distance *k* from *s*, i.e., the length of shortest paths from *u* to *v* is *k,* then *k* will appear as one of the level k vertices during the search

BFS Applications

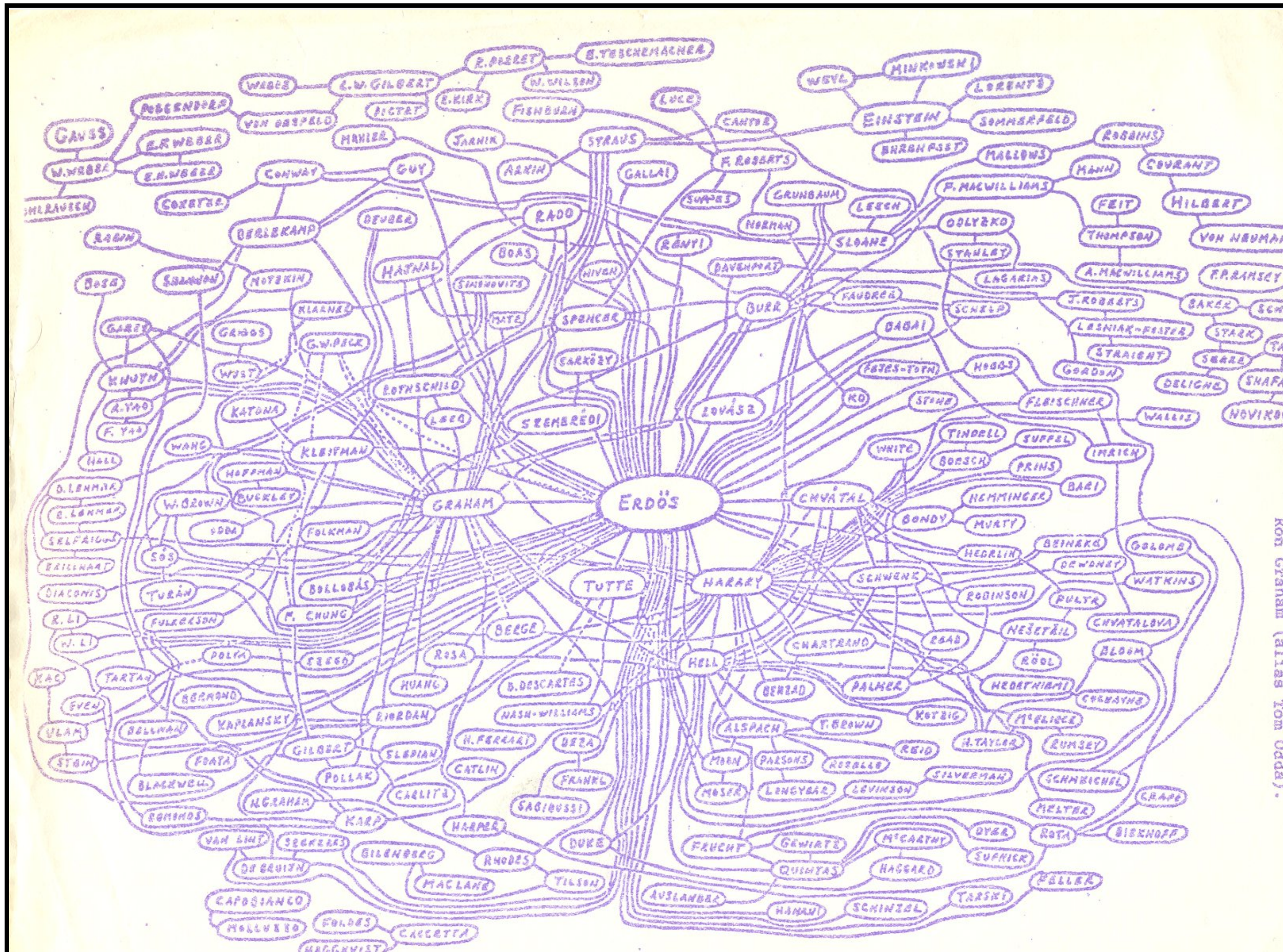
- *Applications*: Like DFS, BFS can be used to testing a number of properties of graphs

  - Test whether there is a path from one vertex to another (why?)

  - Test whether a graph is connected (how?)

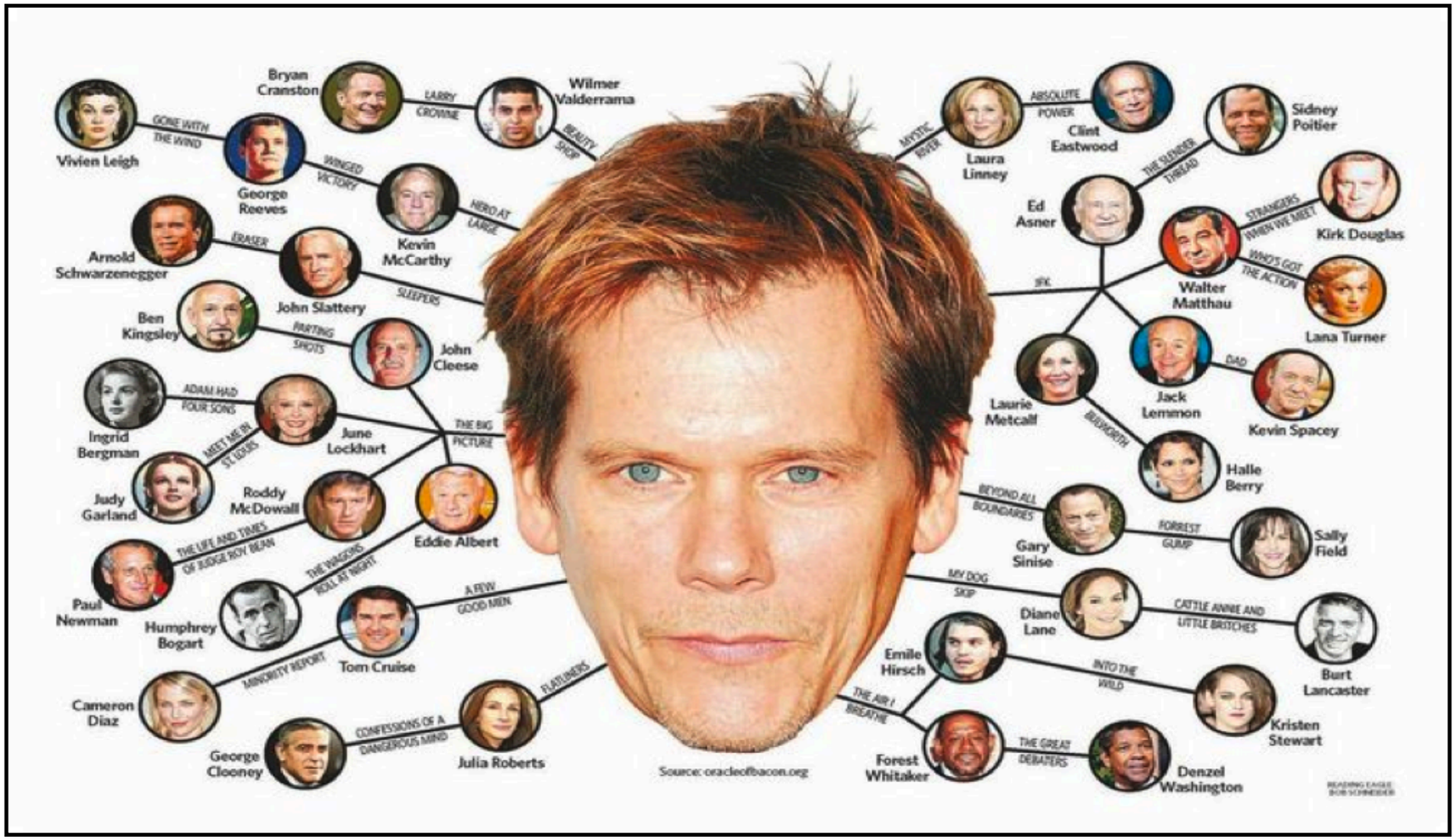
  - Test whether a graph has a cycle (how?)

Example of BFS Applications: Erdös Number



hand-drawing of part of the Erdös graph by Ron Graham

Example of BFS Applications: Oracle of Kevin Bacon



Example of BFS Applications: Path Finder