Data Structure for Graphs

1

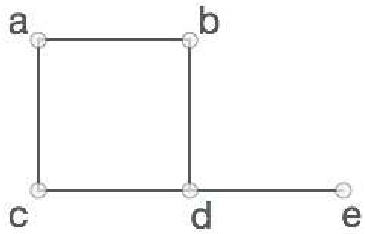
Introduction

- A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links.
- The interconnected objects are represented by points termed as **vertices**
- The links that connect the vertices are called **edges**.

2

Introduction…

- Formally, a graph is a pair of sets **(V, E)**, where
  - **V** is the set of vertices
  - **E** is the set of edges, connecting the pairs of vertices.



  - V={a,b,c,d,e}
  - E={ab,ac,cd,bd,de}

3

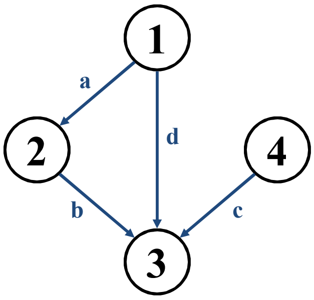
Graph Data Structure

- Edge List
- Adjacency matrix
- Adjacency list

4

Graphs: Adjacency Matrix

- Example:

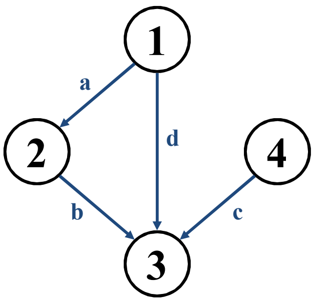


A	1	2	3	4
1				
2				
3				
4				

5

Graphs: Adjacency Matrix

- Example:



A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

6

Graphs: Adjacency Matrix

- *How much maximum storage does the adjacency matrix require?*
- *What is the minimum amount of storage needed by an adjacency matrix representation of an undirected graph with 4 vertices?*
  - Hint: Undirected graph → matrix is symmetric, No self-loops → don't need diagonal

7

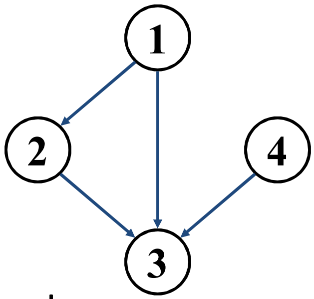
Graphs: Adjacency Matrix

- The adjacency matrix is a dense representation
  - Usually too much storage for large graphs
  - But can be very efficient for small graphs
- Most large interesting graphs are sparse
  - For sparse graphs, $|E| \approx |V|$
  - For this reason the *adjacency list* is often a more appropriate representation

8

Graphs: Adjacency List

- Adjacency list: for each vertex $v \in V$, store a list of vertices adjacent to v
- Example:
  - Adj[1] = {2,3}
  - Adj[2] = {3}
  - Adj[3] = {}
  - Adj[4] = {3}
- Variation: can also keep a list of edges coming *into* vertex



9

Graphs: Adjacency List

- The *degree* of a vertex v = # incident edges
    - Directed graphs have in-degree, out-degree
  - For directed graphs, # of items in adjacency lists is
    Σ out-degree(v) = |E|
  - For undirected graphs, # items in adj lists is
    Σ degree(v) = 2 |E|

10

Graph Searching

- Given: a graph G = (V, E), directed or undirected
- Goal: methodically explore every vertex
- Ultimately: build a tree on the graph
  - Pick a vertex as the root
  - Choose certain edges to produce a tree
  - Note: might also build a *forest* if graph is not connected

11

Breadth-First Search

- "Explore" a graph, turning it into a tree
  - One vertex at a time
  - Expand frontier of explored vertices across the *breadth* of the frontier
- Builds a tree over the graph
  - Pick a *source vertex* to be the root
  - Find ("discover") its children, then their children, etc.

12

Breadth-First Search

- We will associate vertex "colors" to guide the algorithm
  - White vertices have not been discovered
    - All vertices start out white
  - Grey vertices are discovered but not fully explored
    - They may be adjacent to white vertices
  - Black vertices are discovered and fully explored
    - They are adjacent only to black and grey vertices
- Explore vertices by scanning adjacency list of grey vertices

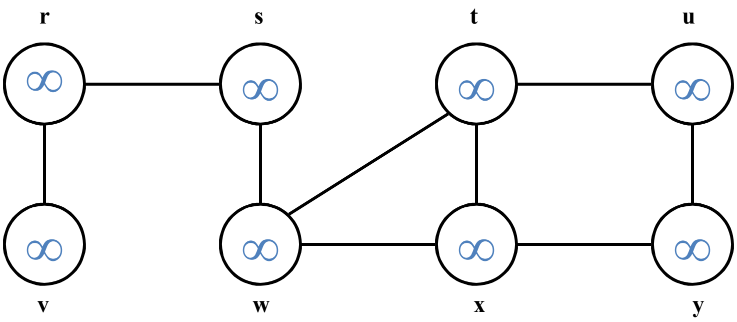
13

Breadth-First Search

```
BFS(G, s) {
   for each v ∈ V-{s}
        v->color ← white
        v->d ← infinity
        v->p ← NIL
S->color ← grey
S->d ← 0
S->p ← NIL
initialize vertices;
    Q = {s};              // Q is a queue; initialize to s
    while (Q not empty) {
        u = deQueue(Q);
        for each v ∈ u->adj {
            if (v->color = WHITE)
                v->color ← GREY;
                v->d ← u->d + 1;
                v->p ← u;
                enQueue(Q, v);
        }
        u->color = BLACK;
    }
}
```

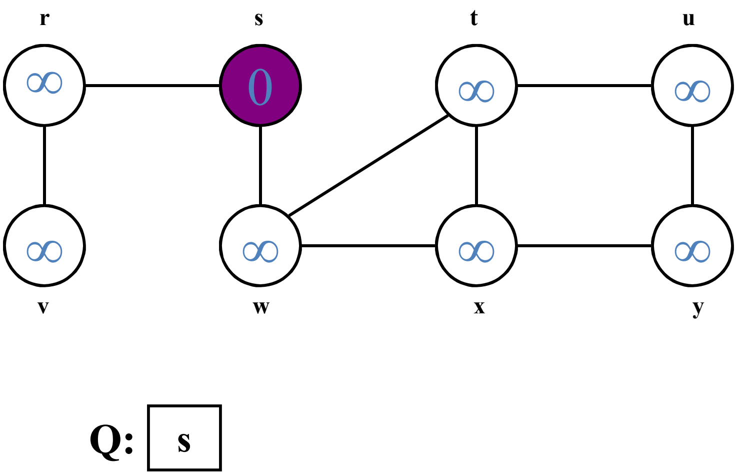
14

Breadth-First Search: Example



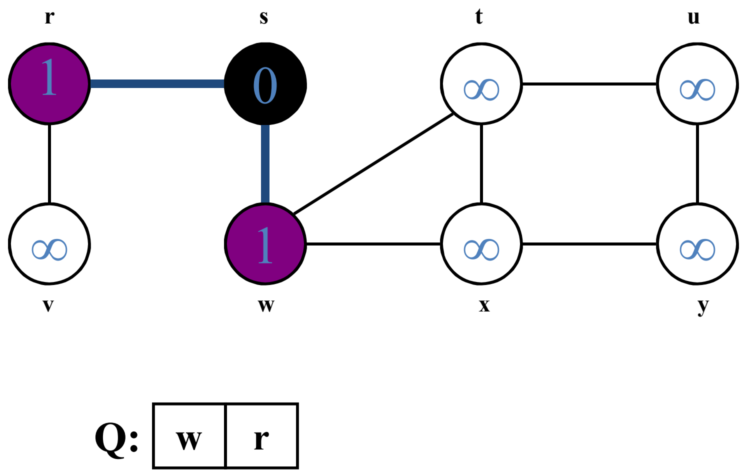
15

Breadth-First Search: Example

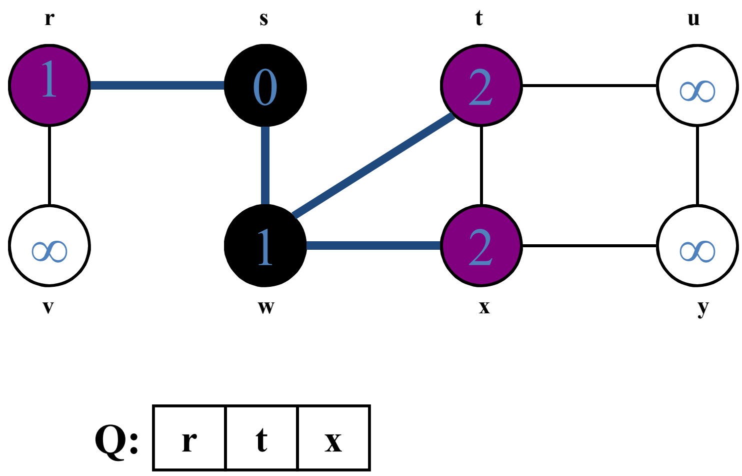


16

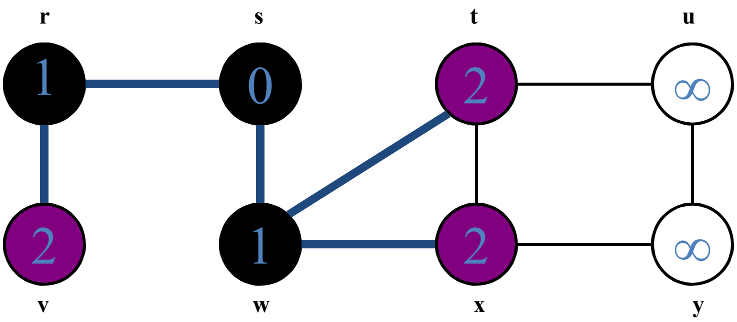
Breadth-First Search: Example



Breadth-First Search: Example



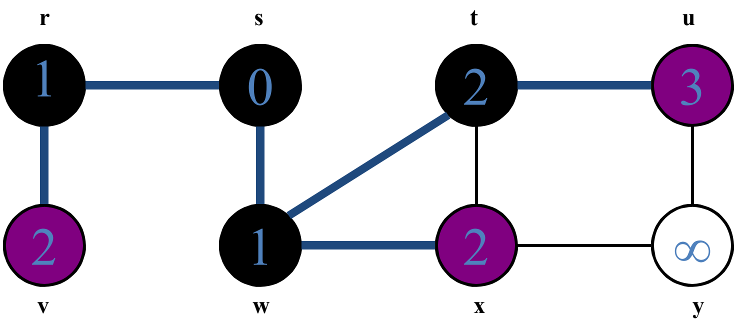
Breadth-First Search: Example



Q:	t	x	v

19

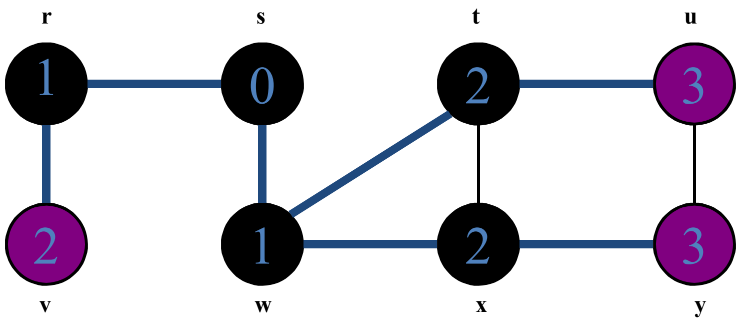
Breadth-First Search: Example



Q:	x	v	u

20

Breadth-First Search: Example

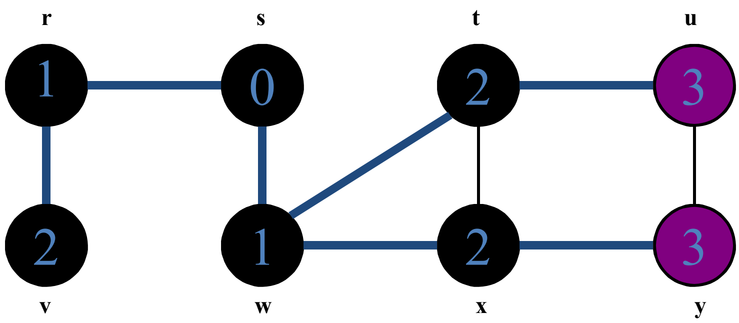


Q:

v	u	y

21

Breadth-First Search: Example

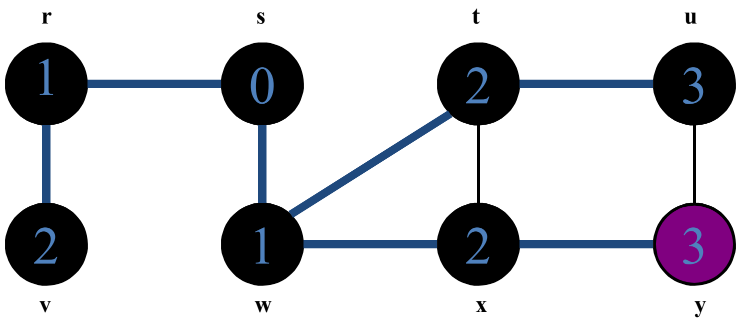


Q:

u	y

22

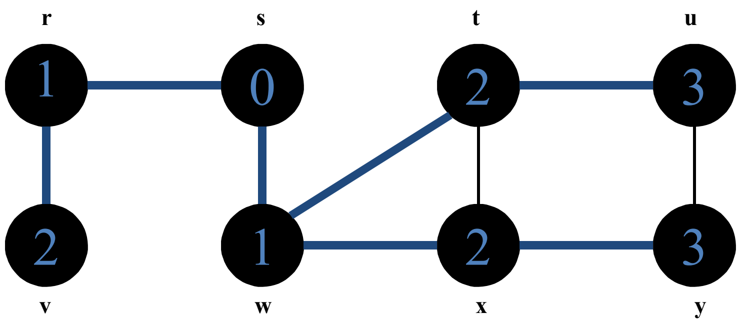
Breadth-First Search: Example



Q: y

23

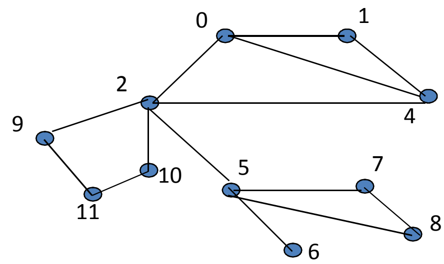
Breadth-First Search: Example



Q: Ø

24

BFS Example2



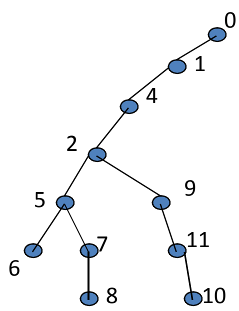
Breadth-First Search: Properties

- BFS calculates the *shortest-path distance* to the source node
  - Shortest-path distance $\delta(s,v)$ = minimum number of edges from s to v, or ∞ if v not reachable from s
- BFS builds *breadth-first tree*, in which paths to root represent shortest paths in G
  - Thus can use BFS to calculate shortest path from one vertex to another

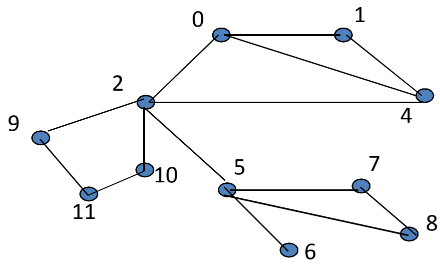
Depth First Search

- DFS follows the following rules:
  1. Select an unvisited node x, visit it, and treat as the current node
  2. Find an unvisited neighbor of the current node, visit it, and make it the new current node;
  3. If the current node has no unvisited neighbors, backtrack to the its parent, and make that parent the new current node;
  4. Repeat steps 3 and 4 until no more nodes can be visited.
  5. If there are still unvisited nodes, repeat from step 1.

Illustration of DFS



DFS Tree



Graph G

Depth First Search

- Can you suggest a data structure for DFS?

Depth-First Search

```
DFS(G, s) {
   Stack s;
   while G has an unvisited node x
   {
       visit(x);
       push(x,S);
       while (S is not empty)
       {
              t← S[TOP]
              if t has unvisited neighbour y
              {
                     visit(y)
                     push(y,S)
              }
              else
                     pop(S)
       }
    }

   }
```

31