GRAPH TRAVERSAL

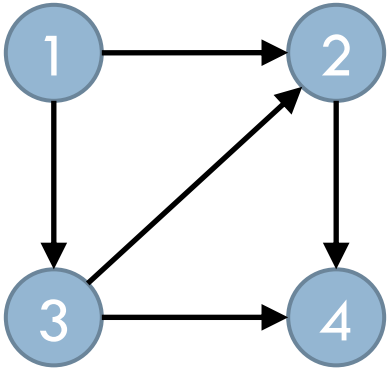
Lecture 18

CS 2110 — Spring 2019

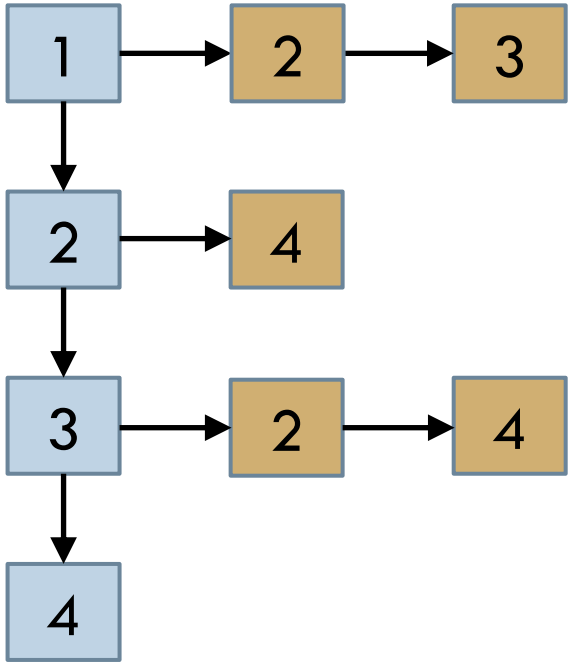
JavaHyperText Topics

"Graphs", topic 8: DFS, BFS

Graph Representations



Adjacency list



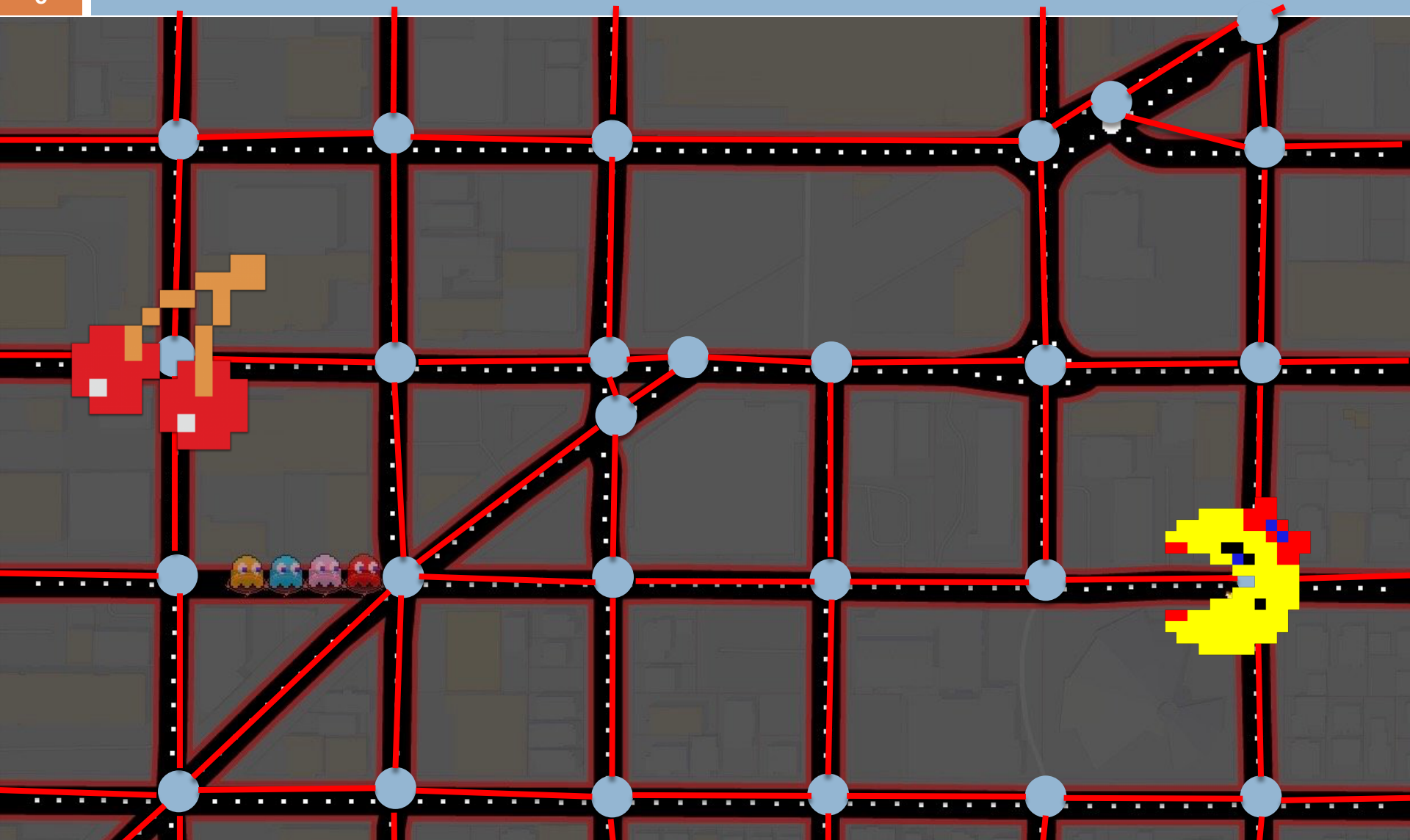
Adjacency matrix

	0	1	2	3	4
0	F	F	F	F	F
1	F	F	**T**	**T**	F
2	F	F	F	F	**T**
3	F	F	**T**	F	**T**
4	F	F	F	F	F

Review: Sparse vs. Dense

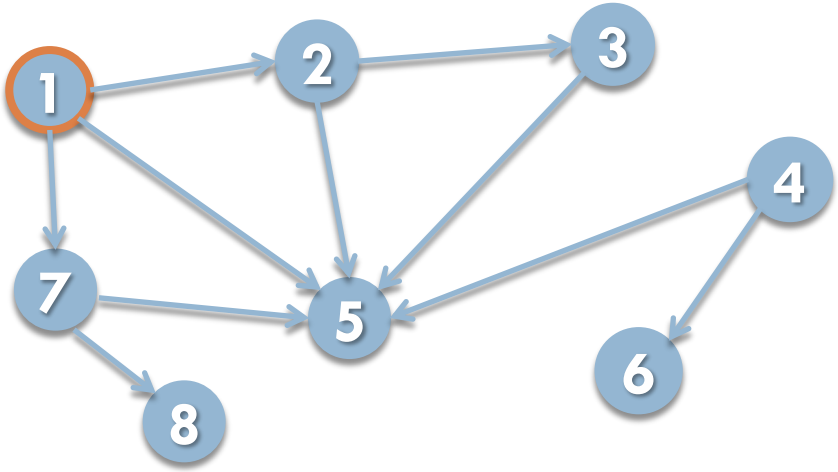
(improved definitions in Lec 17)

Graph Search

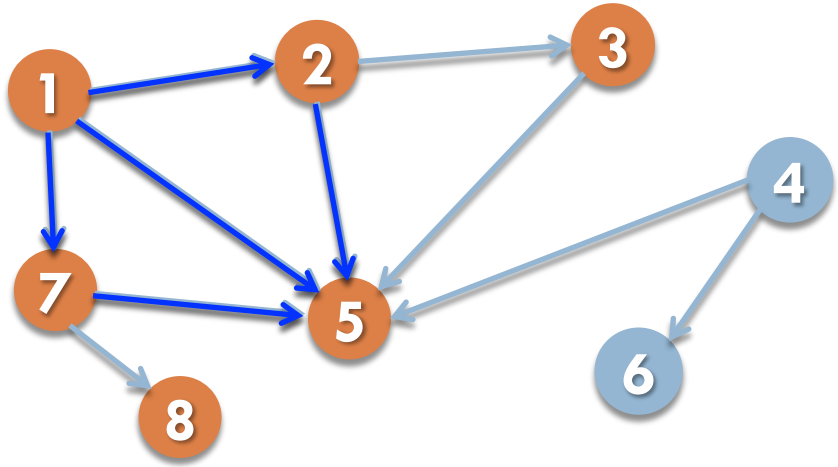


Graph Traversal

Goal: visit each vertex that is reachable from some starting vertex



And: even if there are many paths to a node, visit only once



Two algorithms: DFS, BFS

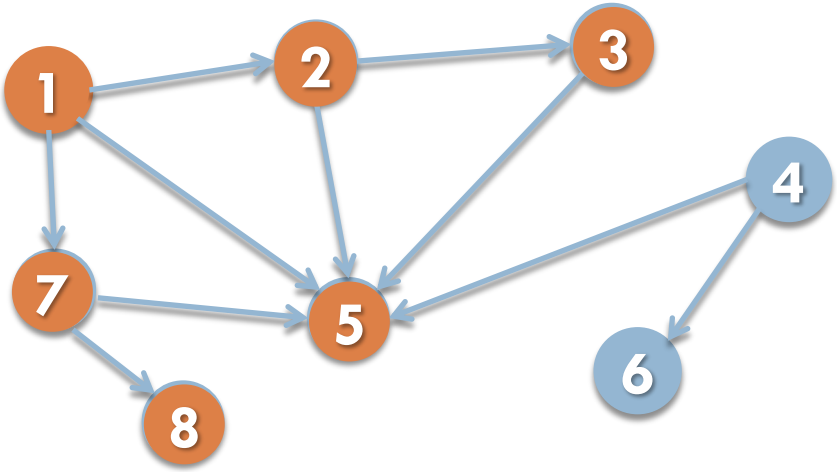
Depth-First Search (DFS)

Intuition: one person exploring a maze

Depth-First Search

Idea: Recursively visit each unvisited neighbor

```
/** Visit every node reachable along a path of
    unvisited nodes from node v.
    Precondition: v is unvisited. */
void dfs(Vertex v) {
    mark v visited;
    for all edges (v,u):
        if u is unmarked:
            dfs(u);
}
```



dfs(1) visits the nodes in this order: 1, 2, 3, 5, 7, 8

Poll #1

Depth-First Search


```
/** Visit every node reachable along a path of
    unvisited nodes from node v.
    Precondition: v is unvisited. */
void dfs(Vertex v) {
    mark v visited;
    for all edges (v,u):
        if u is unmarked:
            dfs(u);
}
```

Demo

DFS Space Efficiency

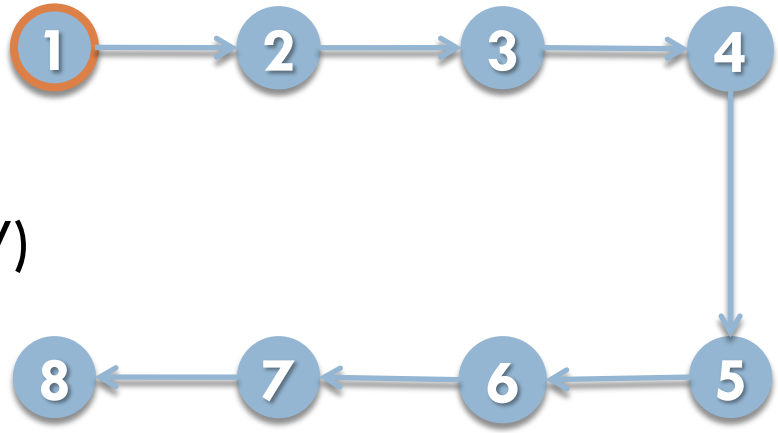

```
void dfs(Vertex v) {
    mark v visited;
    for all edges (v,u):
        if u is unmarked:
            dfs(u);
}
```

Suppose graph has
V vertices and E edges

Space required?

- Mark for each vertex: O(V)
- Frame for each recursive call: O(V)

Worst case: O(V)



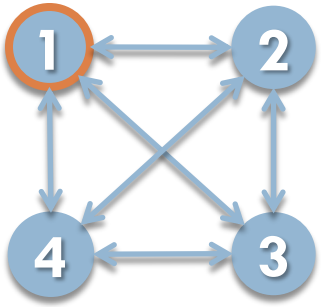
DFS Time Efficiency


```
void dfs(Vertex v) {
    mark v visited;
    for all edges (v,u):
        if u is unmarked:
            dfs(u);
}
```

Suppose graph has
V vertices and E edges

Time required?

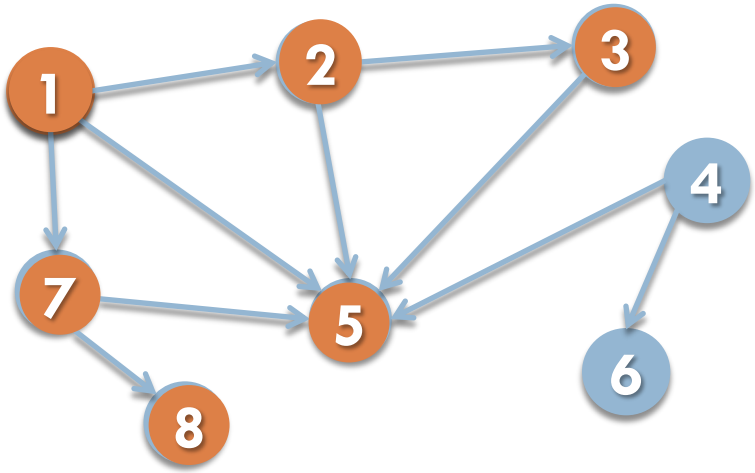
- Mark each vertex: O(V)
- Recursive call for on each unvisited vertex: O(V)
- Find each edge
  - in adj list: O(E): Worst case: O(V+E)
  - In adj matrix: $O(V^2)$: Worst case: $O(V^2)$



Variant: Iterative DFS

Same algorithm; non-recursive implementation

```
void dfs(Vertex u) {
    Stack s= new Stack();
    s.push(u);
    while (s is not empty) {
        u= s.pop();
        if (u not visited) {
            visit u;
            for each edge (u, v):
                s.push(v);
        }
    }
}
```



u: 5

Stack: 8 5 5 5

Demo

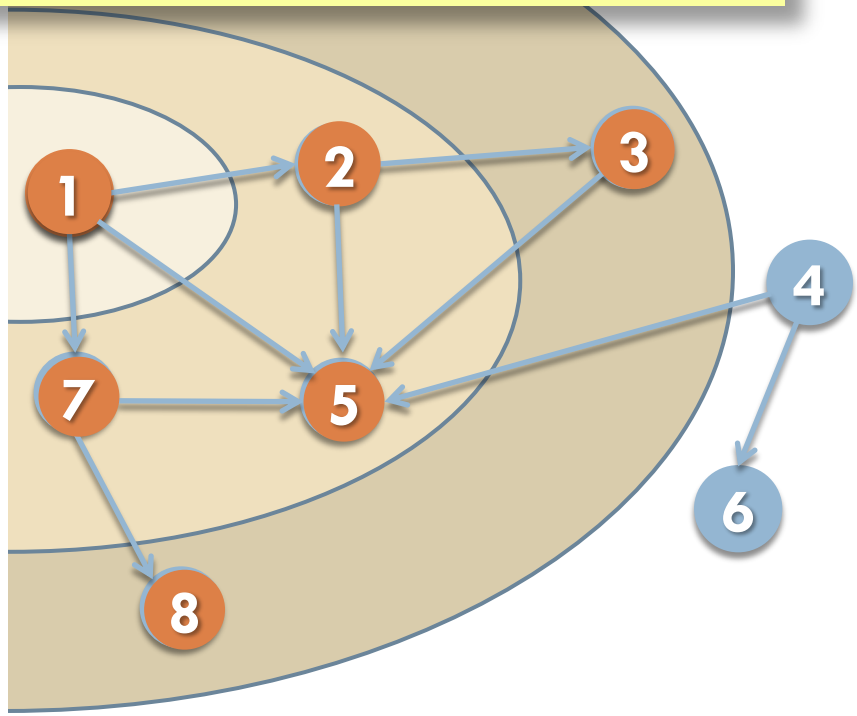
Visit order was 1, 7, 8, 5, 2, 3: differs from before because of order edges processed

Breadth-First Search (BFS)

Intuition: Search party fanning out in all directions

Breadth-First Search

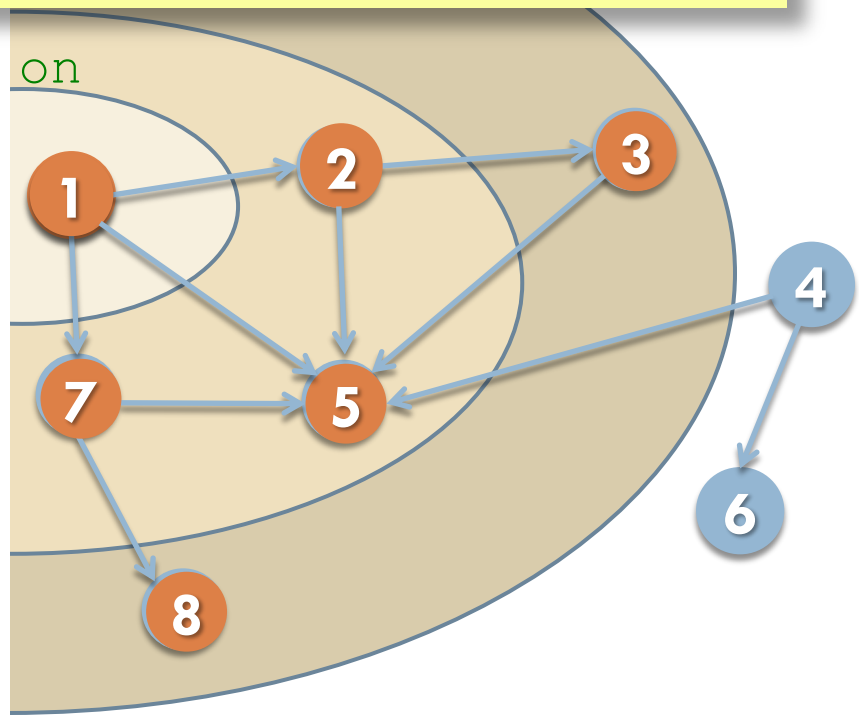
Idea: Iteratively process the graph in "layers" moving further away from the source vertex.



Breadth-First Search

Idea: Iteratively process the graph in "layers" moving further away from the source vertex.

```
/** Visit all vertices reachable on
unvisited paths from u. */
void bfs(int u) {
    Queue q= new Queue()
    q.add(u);
    while ( q is not empty ) {
        u= q.remove();
        if (u not visited) {
            visit u;
            for each (u, v):
                q.add(v);
            }
    }
}
```



Queue: 2 5 7 3 5 8 5

Visit order was 1, 2, 5, 7, 3, 8 Demo

Poll #2

Improved BFS

Idea: Don't put vertex in queue if already encountered

```
/** Visit all vertices reachable on
unvisited paths from u. */
void bfs(int u) {
    Queue q= new Queue
    q.add(u);
    while ( q is not empty ) {
        u= q.remove();
        if (u not visited) {
            visit u;
            for each (u, v):
                if (v not encountered) {
                    mark v as encountered;
                    q.add(v);
} } } }
```

Analyzing BFS


```
/** Visit all vertices reachable on
unvisited paths from u. */
void bfs(int u) {
    Queue q= new Queue
    q.add(u);
    while ( q is not empty ) {
        u= q.remove();
        if (u not visited) {
            visit u;
            for each (u, v):
                if (v not encountered) {
                    mark v as encountered;
                    q.add(v);
} } } }
```

Same as DFS.

Space: O(V)

Time:

- Adj list: O(V+E)
- Adj matrix: $O(V^2)$

Comparing Traversal Algorithms

DFS (recursive)

- Time: $O(V+E)$ or $O(V^2)$
- Space: $O(V)$

DFS & BFS (iterative, improved*)

- Time: $O(V+E)$ or $O(V^2)$
- Space: $O(V)$

*Without improvement, space becomes $O(E)$