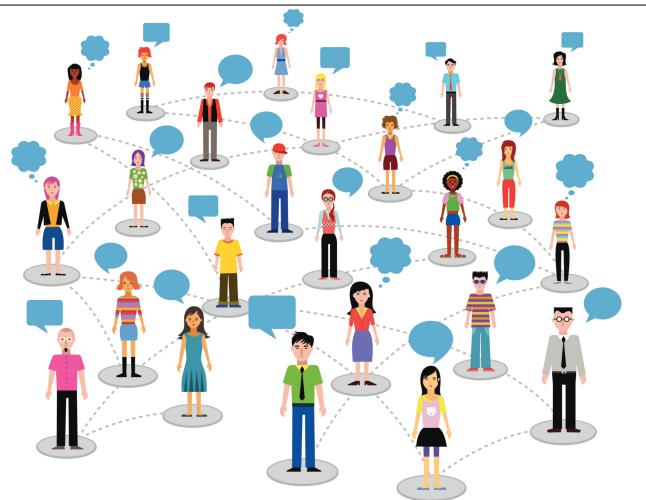




## Estruturas de Dados / Programação 2 Grafos - Parte 1

Márcio Ribeiro  
marcio@ic.ufal.br  
twitter.com/marciomribeiro



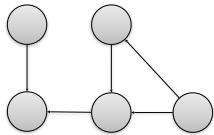
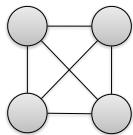
How would you implement one?

Not linear, right?!

Graphs

## Graph

- “A graph is an ordered pair  $G = (V, E)$  comprising a set  $V$  of vertices or nodes together with a set  $E$  of edges or lines”
- An edge is related to two vertices
- We may have undirected and directed graphs

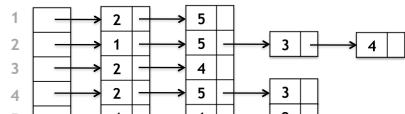
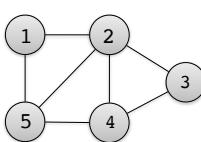


From [http://en.wikipedia.org/wiki/Graph\\_\(mathematics\)](http://en.wikipedia.org/wiki/Graph_(mathematics))

## Representations

### Adjacency lists

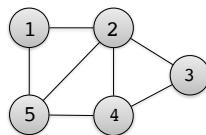
- Consists of an array  $Adj$  of  $|V|$  lists
- For each  $u \in V$ , the adjacency list  $Adj[u]$  contains all vertices  $v$  such that there is an edge  $(u, v) \in E$



### Adjacency matrixes

- Consists of a  $|V| \times |V|$  matrix  $A = (a_{ij})$  such that

$$a_{ij} = \begin{cases} 1, & \text{if } (i, j) \in E \\ 0, & \text{otherwise} \end{cases}$$

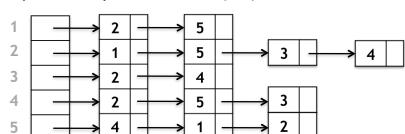


	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



### Adjacency lists versus Adjacency matrixes

- Lists
  - No quicker way to determine  $(u, v) \in E$



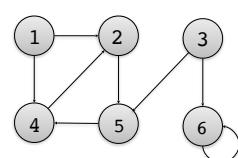
- Matrixes
  - Solve  $(u, v) \in E$  quickly
  - But there is a tradeoff: more memory

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



### Exercise

- Write the adjacency list and the adjacency matrix for the following directed graph



## Abstract Data Type: Graph

### Graph ADT

```
graph* create_graph();
void add_edge(graph *graph, int vertex1, int vertex2);
void dfs(graph *graph, int source);
void bfs(graph *graph, int source);
void print_graph(graph *graph);
```



### Structs

```
struct adj_list {
    int item;
    adj_list *next;
};

struct graph {
    adj_list *vertices[MAX_SIZE];
    short visited[MAX_SIZE];
};
```



### Function to create Graphs...

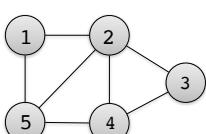
```
graph* create_graph()
{
    graph *graph = (graph*) malloc(sizeof(graph));
    int i;
    for (i = 1; i <= MAX_SIZE - 1; i++) {
        graph->vertices[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}
```



### Creating a Graph

```
graph *undirected_graph = create_graph();

add_edge(undirected_graph, 1, 2);
add_edge(undirected_graph, 1, 5);
add_edge(undirected_graph, 2, 5);
add_edge(undirected_graph, 2, 3);
add_edge(undirected_graph, 2, 4);
add_edge(undirected_graph, 3, 4);
add_edge(undirected_graph, 4, 5);
```



### Adding edges...

```
void add_edge(graph *graph, int vertex1, int vertex2)
{
    adj_list *vertex = create_adj_list(vertex2);
    vertex->next = graph->vertices[vertex1];
    graph->vertices[vertex1] = vertex;

    //Undirected graph. Edge to the other direction as well.
    vertex = create_adj_list(vertex1);
    vertex->next = graph->vertices[vertex2];
    graph->vertices[vertex2] = vertex;
}

adj_list* create_adj_list(int item)
{
    adj_list *new_adj_list = (adj_list*) malloc(sizeof(adj_list));
    new_adj_list->item = item;
    new_adj_list->next = NULL;
    return new_adj_list;
}
```



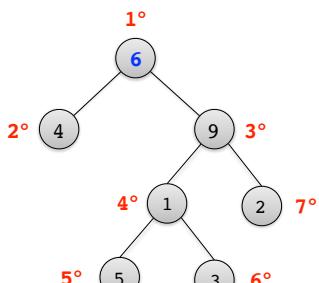
# Graph Traversals

## Depth-first Search (DFS)

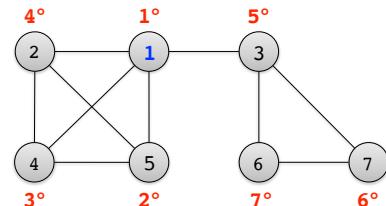
- We mark a vertex as soon as we visit it
- Then we try to move to an unmarked adjacent vertex
- If there are no unmarked vertices at our current position, we backtrack along the vertices we already visited until we come to a vertex that is adjacent to one we haven't visited, visit that one, and continue the process



### DFS Example



### Another DFS Example



```
void dfs(graph *graph, int source)
{
    graph->visited[source] = 1;
    printf("%d\n", source);

    adj_list *adj_list = graph->vertices[source];

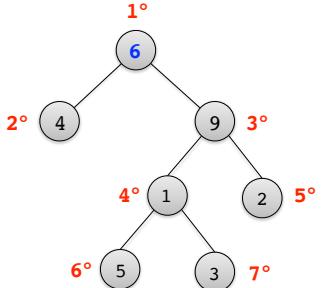
    while (adj_list != NULL) {
        if (!graph->visited[adj_list->item]) {
            dfs(graph, adj_list->item);
        }
        adj_list = adj_list->next;
    }
}
```

## Breadth-first Search (BFS)

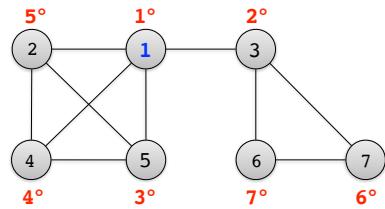
- From a source, it explores every vertex reachable from this source
- This way, it explores the neighbor nodes first, before moving to the next level neighbors



### BFS Example



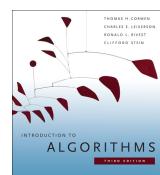
### Another BFS Example



```
void bfs(graph *graph, int source)
{
    queue *queue = create_queue();
    int dequeued;
    adj_list *adj_list = graph->vertices[source];
    graph->visited[source] = 1;
    enqueue(queue, source);

    while (!is_empty(queue)) {
        dequeued = dequeue(queue);
        adj_list = graph->vertices[dequeued];
        while (adj_list != NULL) {
            if (!graph->visited[adj_list->item]) {
                printf("%d\n", adj_list->item);
                graph->visited[adj_list->item] = 1;
                enqueue(queue, adj_list->item);
            }
            adj_list = adj_list->next;
        }
    }
}
```

### References



Chapters 22, 23, and 24



Chapter 8

