Analysis of Graphs Operations

Let n = # vertices, e = # edges ***But typically e is much less than n!!!***

2 graph implementations: **adjacency matrix** or **adjacency list**

	Adj. List	**Adj. Matrix**
Adding or removing an edge	**O(e)**	**O(1)**
Checking whether particular edge exists	**O(e)**	**O(1)**
Processing each edge of particular vertex	**O(e)**	**O(n)**

	Adj. List	**Adj. Matrix**
Depth-First Search (DFS)		
initialize visited[n] to false;		
for each vertex v do	**O(n)**	**O(n)**
if (visted[v] == false)		
DFS(v);		
void DFS(vertex v) {		
visited[v] = true;		
for each neighbor w of v do	**O(e)**	**O(n)**
if (visited[w] == false)		
DFS(w);		
}		
	So O(n * e)	**So O(n^2)**

Breadth-First Search (BFS)

initialize visited[n] to false;		
for each vertex v do		
if (visted[v] == false)		
BFS(v);		
void BFS(vertex v) {		
visited[v] = true;		
enqueue vertex v into rear of queue;		
while (queue is not empty) {	**O(n)**	**O(n)**
dequeue vertex u from front of queue;		
for each neighbor w of u do	**O(e)**	**O(n)**
if (visited[w] == false) {		
visited[w] = true;		
enqueue w into rear of queue;		
}		
}		
}		
}	**So O(n * e)**	**So O(n^2)**

Shortest-Paths Problem

- Assume each edge has a (positive value) weight associated with it
- Represent with adjacency matrix as A[v][w] = weight for edge (v, w), or ∞ if no such edge
- Want to find the "cost" of the **shortest path from a source vertex to every other vertex**

```
// This returns 1D array D where D[ i ] = cost of shortest path from source to vertex i

Dijksta's Algorithm:

for each vertex v do {
   D[ v ] = A[ start ][  v ];
   visited[ v ] = false;
}

D[ start ] = 0; visited[ start ] = true;

for i =1..(n – 1) do    // where n = # vertices
{
    v = unvisited vertex with smallest value in D[ ];
    visited[ v ] = true;

    for each neighbor w of v do
       D[ w ] = min(D[ w ], D[ v ] + A[ v ][ w ]);
}
```

Example:

	0	1	2	3	4
0	∞	**10**	**3**	**20**	∞
1	∞	∞	∞	**5**	∞
2	∞	**2**	∞	∞	**15**
3	∞	∞	∞	∞	**11**
4	∞	∞	∞	∞	∞