Inf-2101 - Algoritmer

Introduction

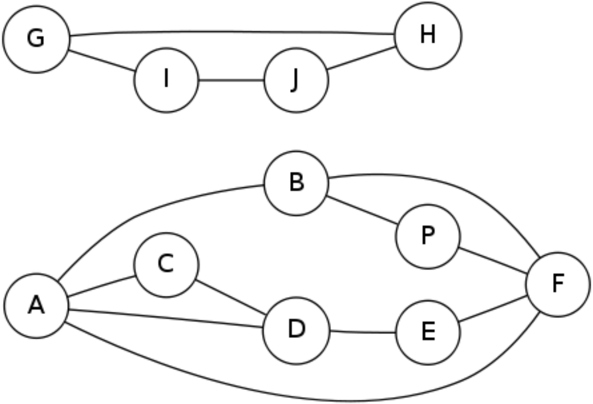
John Markus Bjørndalen

2010-08-16

Some foils are adapted from the book and the book's homepage.

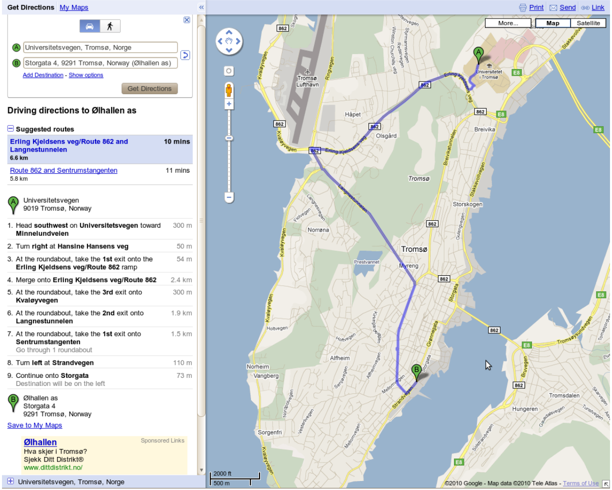
What is a graph?

Set of objects (nodes/vertices) with pairwise connections (edges, arcs, links).



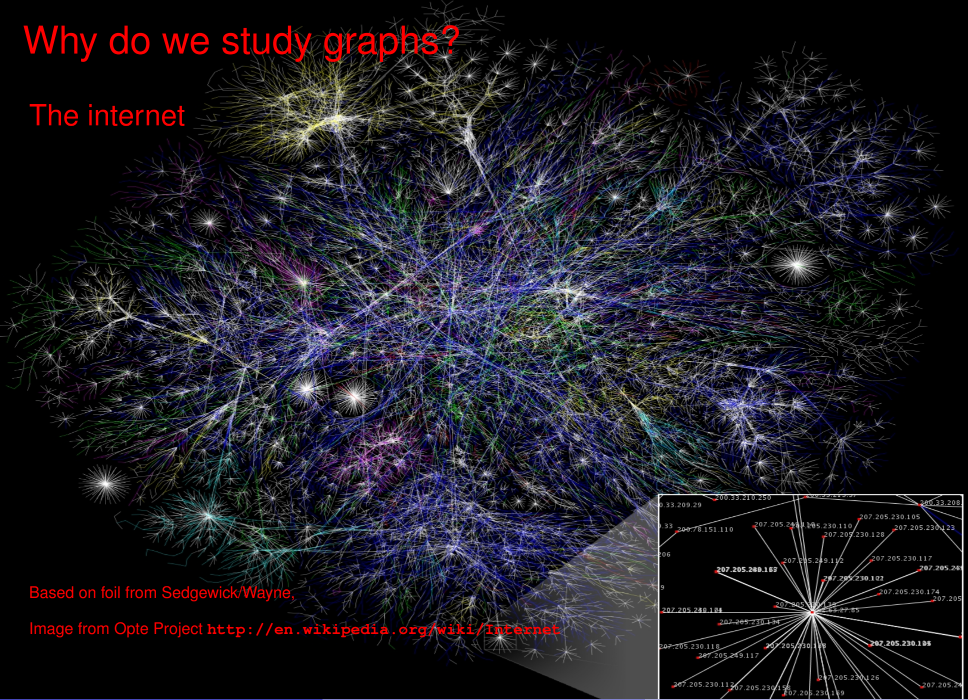
Why do we study graphs?

Routing (ex: GPS navigation for cars)



Why do we study graphs?

The internet



Based on foil from Sedgewick/Wayne,

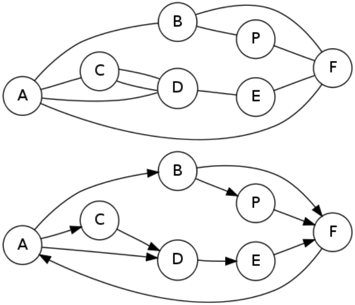
Image from Opte Project **`http://en.wikipedia.org/wiki/Internet`**

Why do we study graphs?

- Routing (ex: GPS navigation for cars)
- Games
- Artificial Intelligence and Knowledge representation
- Scheduling
- Networks and routing
- Study and analyze structure of programs
- Intellectual challenge
- etc.

Some graph terms

- Vertex : v
- Edge : $e = v - w$
- Graph : G
- V vertices, E edges.
- Parallel edge, self loop
- Directed, undirected
- Sparse, Dense
- Path, cycle
- Cyclic path, tour
- Tree, forest
- Subgraph
- Connected, connected component



Some graph processing problems

Path: Is there a path between *s* and *t*?

Shortest path: What is the shortest path between *s* and *t*?

Cycle: Is there a cycle in the path?

Euler tour: Is there a cycle that uses each edge exactly once?

Hamilton tour: Is there a cycle that uses each vertex exactly once?

Connectivity: Is there a way to connect all of the vertices?

MST: What is the best way to connect all of the vertices?

Biconnectivity: Is there a vertex whose removal disconnects a graph?

Planarity: Can you draw the graph in the plane with no crossing edges?

Graph isomorphism: Do two adjacency matrices represent the same graph?

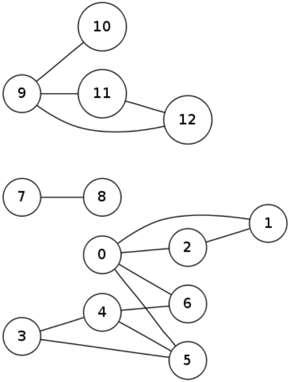
How do we represent a graph in code?

You need to represent vertices and edges.

Two basic options: adjacency matrix and adjacency lists.

Adjacency matrix:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

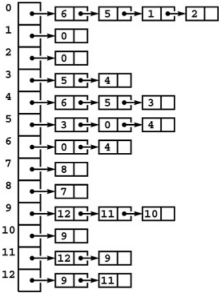


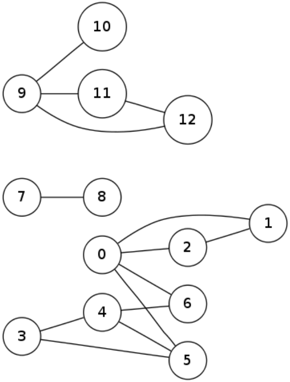
How do we represent a graph in code?

You need to represent vertices and edges.

Two basic options: adjacency matrix and adjacency lists.

Adjacency list:





How do we represent a graph in code?

Storage space?

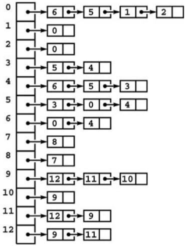
Adjacency Matrix:

Dense graph: efficient
Sparse graph: inefficient (lots of 0s)

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Adjacency list:

Dense graph: inefficient (lots of pointers)
Sparse graph: efficient



How do we represent a graph in code?

Check for existing edges?

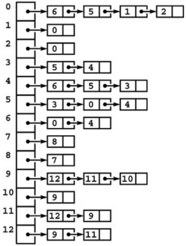
Adjacency Matrix:

matrix lookup

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Adjacency list:

look up vertex, then search list



How do we represent a graph in code?

Add edges?

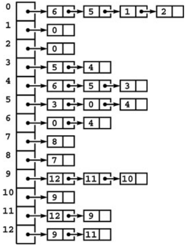
Adjacency Matrix:

write 1 to two locations (undirected graphs)

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Adjacency list:

2x look up vertex, then append to list



How do we represent a graph in code?

Remove edges?

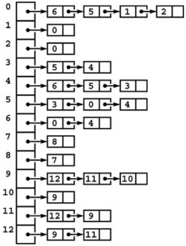
Adjacency Matrix:

write 0 to two locations (undirected graphs)

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Adjacency list:

2x look up vertex, then search list and unlink



Adjacency list implementation in Python

```
class Graph(object):
    def __init__(self, vertices = None, edges = None):
        """Vertices is a list of the vertex ids or numbers.
        Edges is a list of (vertex, vertex) tuples.
        If both are None (or empty lists), the graph is empty.
        """
        self.graph = {}

        if vertices:
            for v in vertices:
                self.graph[v] = []
        if edges:
            for e in edges:
                self.addEdge(e)
```

Adjacency list implementation in Python

```python
def _addEdgeOneDir(self, v0, v1):
    "just add the edge in one direction"
    if v0 in self.graph:
        if v1 not in self.graph[v0]:
            self.graph[v0].append(v1)
    else:
        self.graph[v0] = [v1]

def addEdge(self, edge):
    """Add an edge such that we can easily check for
    both (v0,v1) and (v1,v0).
    Will also add vertices if necessary."""
    v0 = edge[0]
    v1 = edge[1]
    # Undirected graph, just make sure it shows both
    # directions.
    self._addEdgeOneDir(v0, v1)
    self._addEdgeOneDir(v1, v0)
```

Adjacency list implementation in Python

```
def removeEdge(self, edge):
    v0 = edge[0]
    v1 = edge[1]
    # No error checking, just let the default list type handl
    self.graph[v0].remove(v1)
    self.graph[v1].remove(v0)

def hasEdge(self, v0, v1):
    "returns true if there is an edge between v0 and v1"
    return v1 in self.graph[v0]
```

Adjacency list implementation in Python

```
def show(self):
    # Later, we will use graphviz to visualize graphs.

    print "Graph"
    print " Vertices", sorted(self.graph.keys())
    print " Edges"
    for v0 in sorted(self.graph.keys()):
        print "   ", v0, "->", \
              ", ".join([repr(x) for x in \
                          sorted(self.graph[v0])])
```

Adjacency list implementation in Python

```
if __name__ == "__main__":
    g = Graph([1,2,3], [(1,2), (2,1), (2,3), (3,4), (5,1)])
    g.show()
    print "Adding edge (5,2)"
    g.addEdge((5,2))
    g.show()
    print "Removing edge (2,1)"
    g.removeEdge((2,1))
    g.show()
    print "(1,2)? -> ", g.hasEdge(1,2)
    print "(5,2)? -> ", g.hasEdge(5,2)
```

Adjacency list implementation in Python

```
Graph
 Vertices [1, 2, 3, 4, 5]
 Edges
    1 -> 2, 5
    2 -> 1, 3
    3 -> 2, 4
    4 -> 3
    5 -> 1
Adding edge (5,2)
Graph
 Vertices [1, 2, 3, 4, 5]
 Edges
    1 -> 2, 5
    2 -> 1, 3, 5
    3 -> 2, 4
    4 -> 3
    5 -> 1, 2
```

```
Removing edge (2,1)
Graph
 Vertices [1, 2, 3, 4, 5]
 Edges
    1 -> 5
    2 -> 3, 5
    3 -> 2, 4
    4 -> 3
    5 -> 1, 2
(1,2)? ->  False
(5,2)? ->  True
```

Performance of our Python implementation?

As adjacency lists, but with one modification: dictionary of vertices instead of list of vertices!

Performance issues

Data/graph representation will often influence both performance (ex: matrix vs. list), and results (searches).

Note that worst-case analysis may not be representative for the graphs that you give your algorithms.

Knowing graph constraints and properties can lead to a more efficient representation and algorithm than worst-case analysis would do.

Classes of graph problems

- Easy
- Tractable
- Intractable
- Unknown