CS340 Notes, Lecture 20

"Advanced Topics": Graph Algorithms

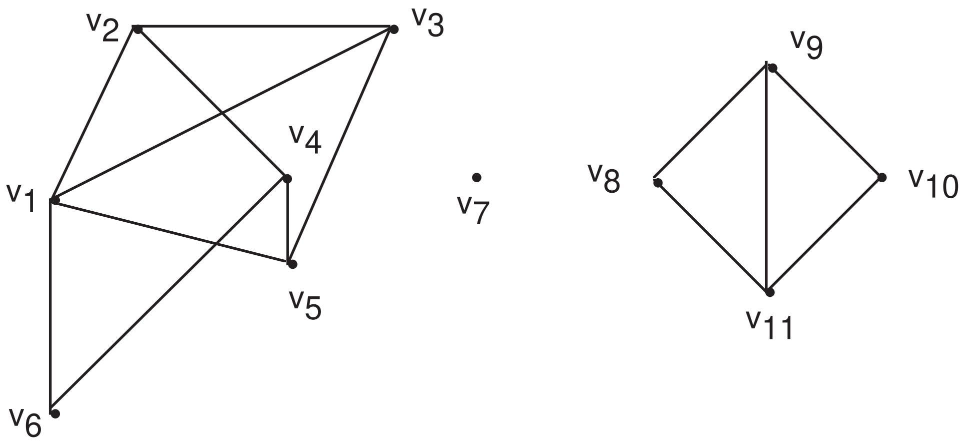
Representation and Manipulation of Graphs (special case: trees)

Undirected Graph $G = (V, E)$, where

- V: finite set of vertices
- E: edges, or unordered pairs of vertices

Directed Graph: (Digraph)

Similar as above, but elements of E are "directed edges" or "ordered pairs".



Example.

$G = (V, E)$

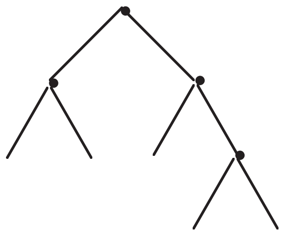
$V = \{v_1 \ldots, v_{11}\}$

$$E = \begin{array}{l}\{(v_2, v_3), (v_1, v_2), (v_1, v_3), (v_1, v_6), (v_1, v_5), \\ \quad (v_2, v_4), (v_4, v_5), (v_4, v_6), (v_3, v_5), (v_8, v_9), \\ \quad\quad (v_8, v_{11}), (v_9, v_{11}), (v_9, v_{10}), (v_{10}, v_{11})\}\end{array}$$

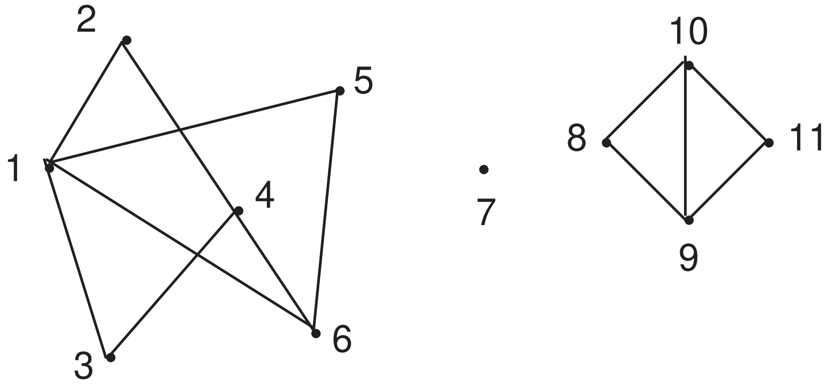
Nodes may have "names".

Edges may have "costs", "weights", ...

A tree is a graph.



For a graph: <u>G</u>

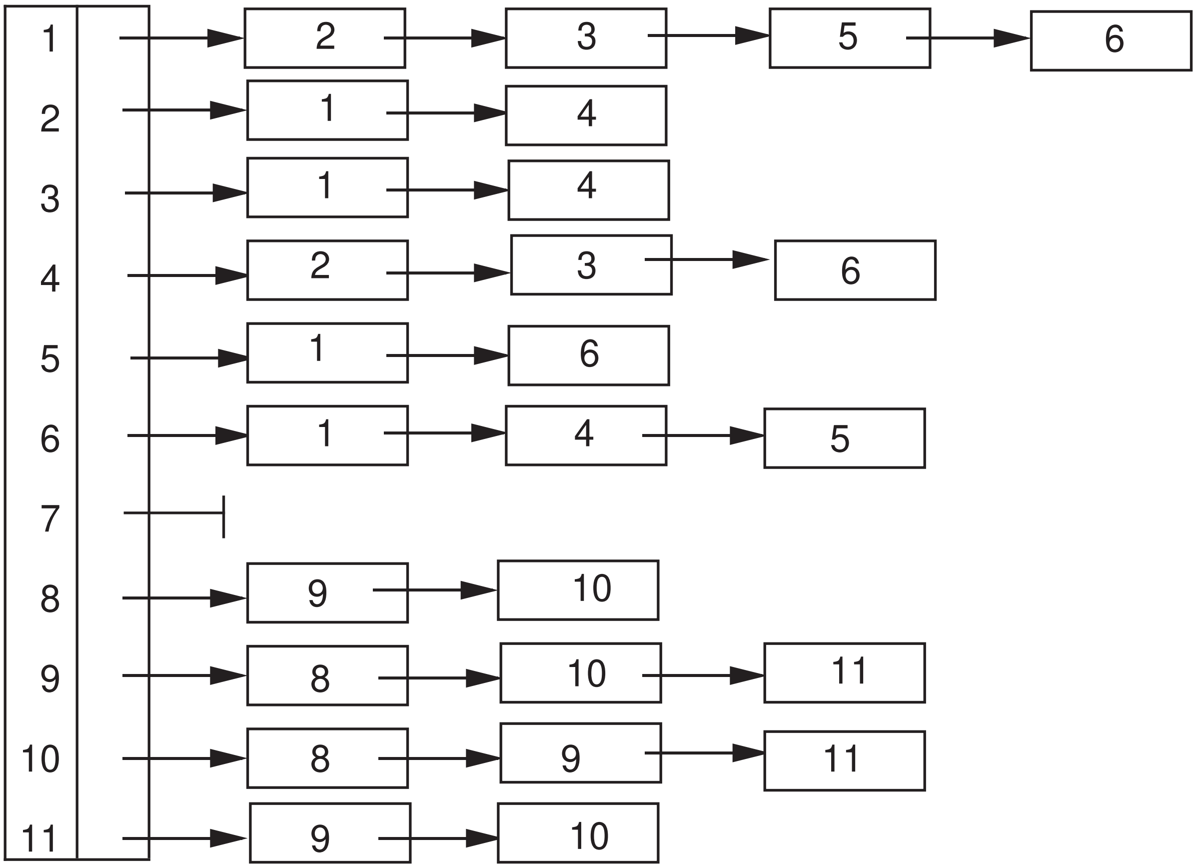


- Representing G by **Adjacency Matrix**.

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

* Adacency Matrix takes n^2 bits. Or $(n^2+n)/2$ for undirected graph since its matrix is symmetric.

• But many interesting graphs have $o(n^2)$ edges. Then we can use **Adjacency List**, as shown.



Such representation takes only $O(|E|)$ space. When $|E| = o(n^2)$, we save space. (And also speed up some algorithms.)

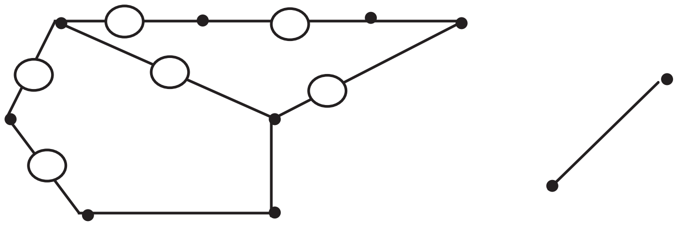
Representation Issues

- finding next edge/an edge exists?
- adding/deleting edges
- directed versus undirected
- weight/labels on edges

Basic Properties:

- Connectivity of the graph
- Shortest path between two nodes.

Finding shortest path from a given node to all others.



Shortest path tree

- Works even if we have (non-negative) weights on edges (and we want minimally weighted paths).

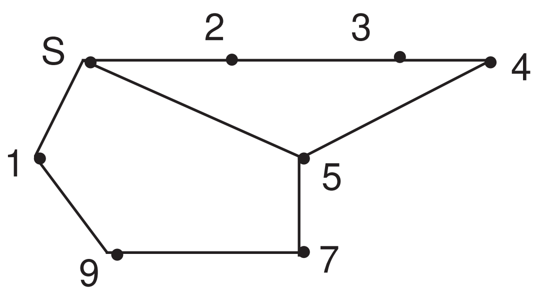
- Works on directed graphs.

A General Graph Search Algorithm

```
All nodes initially marked unvisited;
Let S be the initial node;
Searchdirector := {S};
while  searchdirector != empty do
  begin
    Remove ``next node'', c, from searchdirector
    if c is unvisited then
       mark c visited
       process c {for example: printing it}
       for each edge (c,j) put j
         into searchdirector if j is unvisited.
  end
```

Example: Shortest path from start node in an unweighted graph

- Searchdirector is a queue $\Rightarrow$ Breadth first Search!



- Question: What if we had lengths on edges and wanted shrotest path by sum of edge length? (minimum spanning tree)

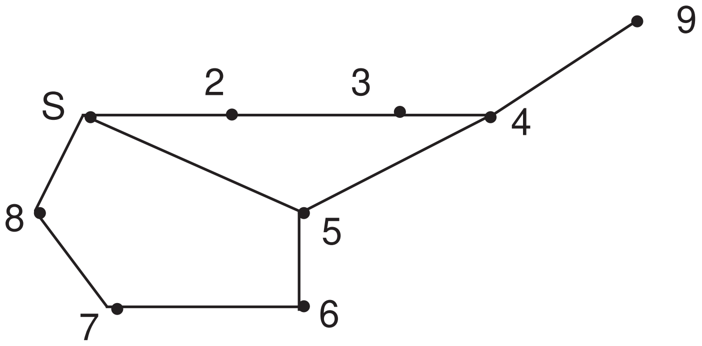
Example. If searchdirector is a stack, then

- We are doing **depth first search**,

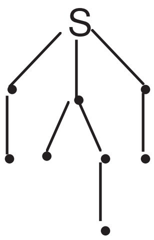
- Often more easily viewed recursively.

```
Depthsearch(S)
  Process S;
  mark S visited;
  for each unvisited neighbour, U, of S
     do depthsearch(U);
```

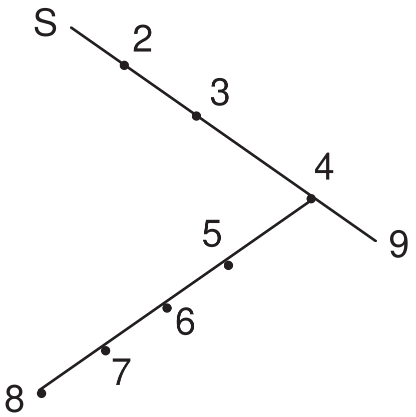
Any such search method will give a spanning tree (forest, if graph not connected).



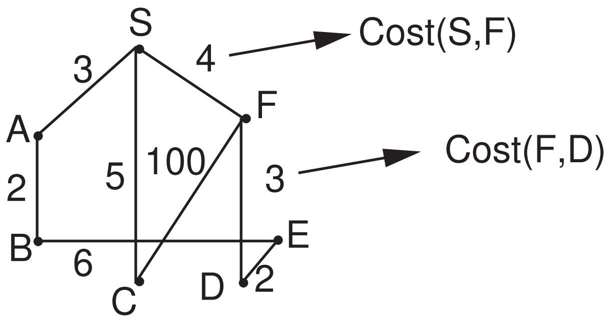
Breadth-first



Depth-first



Example (Prim's Alg). Given a connected weighted graph, find minimum spanning tree. (i.e. a spanning tree of minimum total edge weight)



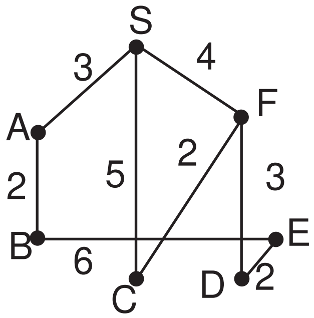
As before apply search using a <u>priority queue</u> as search director.

Priority Queue on edges

- remove min.
- insert $wt(i,j) = cost(i,j)$ when i visited
- $wt(S,S) = 0$.

Greedy algorithm: all the time choose min edge from priority queue, and insert all neighbouring edges.

Example (Dijkstra's Alg). Given weighted graph ($wts \geq 0$) and start node S, find shortest path from S to every other node in the graph.



Apply search using a priority queue as searchdirector.

Priority Queue on edges

- remove min cost edge
- $dist(S, j) = \min_i \{dist(S, i) + cost(i, j)\}$ this is the min distance going from S to i, then to j.
- $dist(S, S) = 0$.

Comments. Nodes are taken in the order of distance from S. This alg. works only when $cost(i, j) \geq 0$. It works for digraphs.

Example: Find longest path from S to T. It appears we have no good handle on this.

Example: Find a path visiting all nodes of minimum total length. Same as above: no good algorithm known.

Observe: We can

- find all paths, and
- output the one with minimum cost.

But: *this take exponential time*, as there can be $n!$ paths for a graph of n nodes.