Graphs

Graphs

- Graphs are a generalization of sequences and trees.
- In a sequence, every node except the first has exactly one preceding node, and every node except the last has exactly one succeeding node.
- In a tree, every node except the root has exactly one preceding node, but a node can have any number of succeeding nodes.
- No restrictions in a graph.

Graphs

- Finite number of nodes (also called vertices) assumed to be numbered from *0* to *n-1*.
- If nodes are other objects, a map maintained from the set of nodes to *{0, 1, …,n-1}*.
- Algorithms on graphs assume this.
- Graph defined by a subset of ordered pairs *(i,j), 0* <= *i,j* < *n*, called edges.
- Same as a binary relation on the set of nodes.

Representation of graphs

- Adjacency matrix – n by n matrix A with $A[i][j] = 1$ if and only if (i,j) is an edge.
  - Takes $\Omega(n^2)$ space and time for just reading.
  - Not useful for sparse graphs with $O(n)$ edges, many 0 entries.
- Adjacency lists – a vector of lists with $A[i]$ containing the list of nodes j such that (i,j) is an edge.
  - Takes $O(n+m)$ space and time to read, where m is the number of edges.
  - Most commonly used for general graphs.
  - Order of nodes in list is not important, unlike in a rooted tree.

Different kinds of graphs

- The graphs as defined are *directed* graphs, an edge *(i,j)* is directed from *i* to *j*.
- Can also have self-loops, that is edges *(i,i)*.
- Undirected graphs have no self-loops and edges are unordered pairs of nodes.
  - Equivalent to symmetric directed graphs without self-loops, *(i,j)* is an edge iff $i \neq j$ and *(j,i)* is an edge.
- Multigraphs can have many edges from one node to another.
  - Equivalent to a graph with weights assigned to edges.

Terminology

- Node j is a successor of node *i*, and *i* is a predecessor of *j*, if *(i,j)* is an edge.
  - Also say *j* is adjacent to *i*.
- Outdegree of a node is the number of successors, indegree the number of predecessors, just degree for undirected graphs.
- Edge *(i,j)* is said to be incident from *i* and to *j*.

Paths and Walks

- A walk in a graph is a sequence of nodes $v_0, v_1, \ldots, v_l$ such that (v_i, v_{i+1}) is an edge for $0 <= i < l$.
- The walk is said to be of length l and from v_0 *to* v_l.
- A walk with distinct nodes is called a path.
- A walk is closed if $l >= 1$ and $v_0 = v_l$.
- A cycle is a closed walk in which all pairs of vertices are distinct, except for the first and the last.

Reachability

- The most basic problem on graphs – given two nodes i and j in the graph, does there exist a path from i to j?
- Many (all?) problems can be reduced to this for a suitable graph.
- Nodes represent the state of a system.
- An edge represents a possible transition.
- A path is a way of converting the system from one state to another.

Example

- Nodes are possible configurations of the Rubik’s cube (a very large number.)
- An edge represents a rotation of the cube from one configuration to another.
- Solving the cube – find a path to the final state.
- Known that there exists a path of length at most 20, if there is one at all, and this is best possible.

Depth-first search

- A standard technique for finding paths in a graph.
- Generalizes preorder traversal of trees.
- Imposes a structure on the graph that can be used to solve many problems.
- The first step for many graph algorithms.
- Assumes the graph is given completely by adjacency lists.

Depth-first search

```
Dfs( node u) {
    visited[u] = true;
     for each node v in A[u]
     if (!visited[v]) {
         parent[v] = u; Dfs(v);
     }
}
```

Depth-first search

- Main idea – when searching from a node, traverse the list of successors and as soon as an unvisited successor is found, start searching from it.
- Nodes need to be marked as visited before searching from successors, otherwise an infinite loop can occur.
- Apart from visited array, same as preorder.

Depth-first search

- *Dfs(u)* will cause exactly the vertices reachable from *u* to be marked visited.
- Actual path can be found by keeping a parent array, *parent[v] = u* if *DFS(v)* is called when *v* is found unvisited in *A[u]*.
- This defines a rooted tree with the starting vertex of *Dfs* as the root, called a *Dfs* tree.

Depth-first ordering

- A depth-first search on the whole graph imposes an ordering on the vertices.
- *for (int i = 0; i < n; i++) if (!visted[i]) Dfs(i);*
- Number nodes in order visited- Dfs numbering.
  - Preorder traversal of rooted trees generated.
- Also number in order in which Dfs is completed - Dfs_finish_number.
  - Postorder traversal of Dfs trees.

Classifying edges

- Tree edges – edge in the DFS tree.
- Forward edges – Edge from a node to any proper descendant in the Dfs tree that is not a child.
- Back edges – Edge from a node to any ancestor in the Dfs tree (including itself).
- Cross edges – All other edges.
- Only back edges directed from a node with lower Dfs_finish_number to a node with higher.

Detecting cycles

- A directed graph contains a cycle iff there is a back edge.
- If the graph has no cycle, every edge is directed from a node with higher Dfs_finish_number to a node with lower.
  - Called Topological sorting of an acyclic graph.
- Solve many problems for directed acylic graphs.
  - Longest path, similar to height in trees.

Breadth first search

- Dfs gives one path from a node to another.
- May be very long even if there is a direct edge.
- Breadth first search is an alternative that gives minimum length paths from a vertex u to all vertices reachable from it.
- Generates a subtree in the graph level-wise.

Breadth first search

```
Bfs(node u) {
      visited[u] = true;
       queue<node> q;  q.push(u);
       while (!q.empty()) {
             node v = q.front(); q.pop();
             for each node w in A[v]
             if (!visited[w]) {
                    q.push(w); visited[w] = true; parent[w] = v;
             } } }
```

Strongly connected components

- Two nodes *u*,*v* in a directed graph are said to be connected if there exists a path from *u* to *v* and from *v* to *u*.
- Defines an equivalence relation on nodes.
- The equivalence classes are called the strongly connected components.
- Just components for an undirected graph.

Strongly connected graphs

- A directed graph is strongly connected if it has only one strongly connected component.
- Easy to determine if a directed graph is strongly connected.
- *Dfs(0)* should visit all nodes in the graph.
- *Dfs(0)* should also visit all nodes in the reverse of the graph, obtained by replacing edge *(i,j)* by *(j,i)*.

Strongly connected components

- First do a complete Dfs on the graph and assign Dfs_finish_numbers to the nodes.
- Relabel node with Dfs_finish_number *i* as *n-1-i*.
- Do a complete Dfs on the reverse of the graph with the new node numbers – starting a new Dfs with the unvisited node having highest Dfs_finish_number in the first Dfs.
- The trees formed in the second Dfs are the strongly connected components of the graph (and its reverse).