Graphs

Graphs and graph algorithms are pervasive in Computer Science. The graph is such a general data structure that almost all computational problems can be formulated using one of the primary graph processing algorithms. Lists and trees are subsets of graphs.

The major problems in hardware synthesis, operating system scheduling and sharing, compiler optimization, software design and minimization, network communication and synchronization, and requirements and specification modeling are graph problems. Variations of the fundamental graph algorithms of traversal, covering, and coloring model most of the problems in these application areas.

Definition

A ***graph*** is a collection of vertices `V` (also called nodes) and edges `E` (also called arcs or links) which connect the vertices. A single edge can be described by the two vertices which it connects:

```
e = (u,v)
```

Thus, a graph consists of two sets drawn from the carrier set of nodes, `S`. The first set is simply the set of nodes; the second set is a set of unordered pairs of nodes, representing edges.

```
G  =  <S,(S,S)>
```

Note that the single set of unordered pairs is sufficient to characterize the graph, assuming that edges meet at a “virtual node”. In the simplest from, graphs are unordered pairs.

A ***generator*** (ie inductive definition) for fully connected graphs:

let `Sn` be the set of nodes of cardinality `n`,
`Ni` be the `i`th node
`Ei` be the edges that have `Ni` as an endpoint.

Base: `N1`
Generator: `Si+1` = insert `Ni+1` into `Si`
`Ei+1` = insert one pair `(Ni+1 Nk)` into `Ei`
for each `k` ranging over all `n` nodes.

In a complete graph of n nodes, the number of edges is

```
sum[n] = n(n+1)/2
```

Types of Graphs

undirected graph:
The edges can be traversed in either direction. Undirected graphs are either trees or cyclic.

cyclic graph:
The graph contains loops than permit a node to be visited recurrently.

directed graph:
The edges are directional and support only one direction of traversal.

directed acyclic graph (DAG):
No paths permit visiting a node more than once.

weighted graph:
The edges have a numerical value which may represent the cost of traversing that edge. Weights can be both positive and negative.

connected graph:
All of the vertices can be visited when starting from any vertex. The list of vertices visited between a starting vertex and an ending vertex is called a *path*.

sparce graph:
There are many more vertices than edges, that is `E << V^2`.

dense graph:
The number of edges is about the same as the number of vertices squared; `E =~= V^2`

complete graph:
Every pair of vertices has an edge connecting them. The graph is thus maximally connected.

clique:
A subgraph (that is a subset of nodes and edges in a graph) which is complete.

bipartite graph:
A graph with two sets of vertices, such that each edge connects vertices from different sets. No edges connect vertices in the same set.

hypergraph:
A graph with edges that connect more than two vertices. All hypergraphs can be converted into regular bipartite graphs.

graph complement:
The complement of a graph is a graph with the same vertices, but all edges exchanged (ie if an edge is present, then delete it; if an edge is missing, then add it).

Representation of Graphs

The choice of graph data structures depends on the density of the graph connectivity. The complexity of different algorithms depends both on the number of edges and on the number of vertices. Unlike sets, graphs often have a direct representation in memory. When the data field of a memory cell contains an address of another cell, the cell can be interpreted as a node and the shared address can be interpreted as a link.

adjacency lists:
A collection of `V` lists (usually stored as an array and often hashed), each listing the adjacent vertices of a particular vertex.

adjacency matrix:
A square matrix of `VxV` boolean entries, with a `1` recording adjacent vertices, and a `0` recording non-adjacent vertices.

indexed arrays:
For graphs which change during processing, two arrays are used. The first array contains the `V` vertices, and provides an index for each vertex. This array is changed only when the status of the vertex changes, the vertex index is never changed. The second array contains the `E` edges, and provides an index for each edge. The data field of the second array contains the indices of the two vertices connected by that edge. The connectivity of the graph is changed by changing the edge array entries.

Parenthesis Structure

Graphs can be expressed in a linear parenthesis notation which is convenient for computation and analysis.

Example

```
V = {1,2,3,4,5,a,b,c}         E = {12,13,24,25,2b,35,4a,5b,5c}

      G = ( ( b ( a ) ( b c ) ) ( ( b c ) ) )
          1 2   4     5         3 5

          (                                 )
            (                 ) (         )
              b (   ) (     )     (     )
                  a     b c         b c


           ---------------------------------
                     |               |
             -----------------   ---------
              |   |      |           |
              |  ---   -----       -----
              |   |     | |         | |
              b   a     b c         b c
```

Structure sharing in graphs is represented by multiple labels:

```
G = ( ( b ( a )    5  ) (       5     ) )
5 = ( b c )
```

Topological Sort

Sorting lists and trees makes them easier to process, since the vertices have a consistent ordering. Directed acyclic graphs (DAGs) can also be sorted by the partial ordering implicit in the graph connectivity. Cyclic and undirected graphs do not have a consistent sort.
A ***topological sort*** of a DAG is a consistent ordering of the vertices that serves as an ordering for processing vertices. One convenient sorting is by length of shortest paths from the root vertex to the leaf vertices.

Primary Graph Algorithms

searching

The search algorithm ***traverses a graph***, usually identifying the shortest (or longest) path between two vertices. ***Spanning trees*** list paths that connect all vertices. A ***Hamiltonian path*** is a path which visits each vertex while not traversing the same edge more than once. The travelling saleman problem (ie, what is the shortest route available to visit a collection of sites) is an example of computing a minimal spanning tree.

covering

A ***vertex cover*** is a subset of vertices of an undirected graph such that every edge has at least one end in the subset. Minimization of boolean expressions (by Karnaugh maps or by algebraic techniques) is an example of graph covering. Each product term in a sum-of-products (SOP) representation is a subset of vertices of a boolean hypercube. The covering problem is to include each vertex in the SOP using a minimum number of sub-cubes of the hypercube.

coloring

A ***vertex coloring*** is a labeling of the vertices of an undirected graph such that no edge has two endpoints with the same label. An example of graph coloring is scheduling and resource allocation using a *resource conflict graph*. Vertices in a resource conflict graph represent operations to perform; edges represent pairs of operations which are in conflict because they cannot use the same resource.