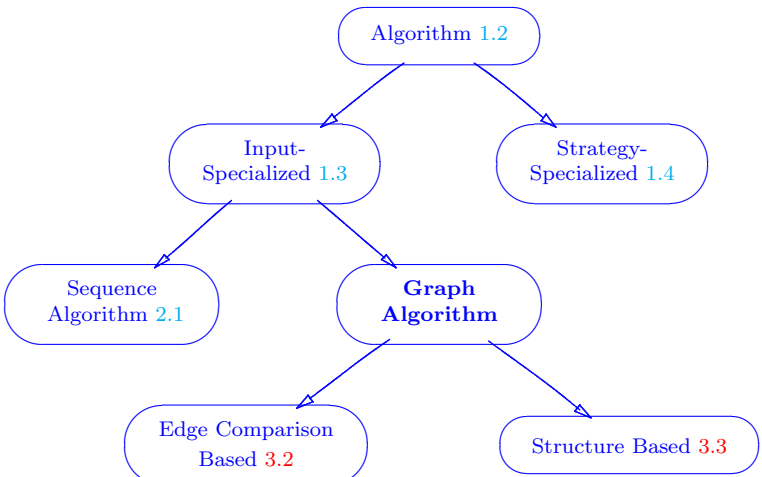
3. Graph Algorithm Concepts

Section authors: David R. Musser and Brian Osman.

3.1. Graph Algorithm

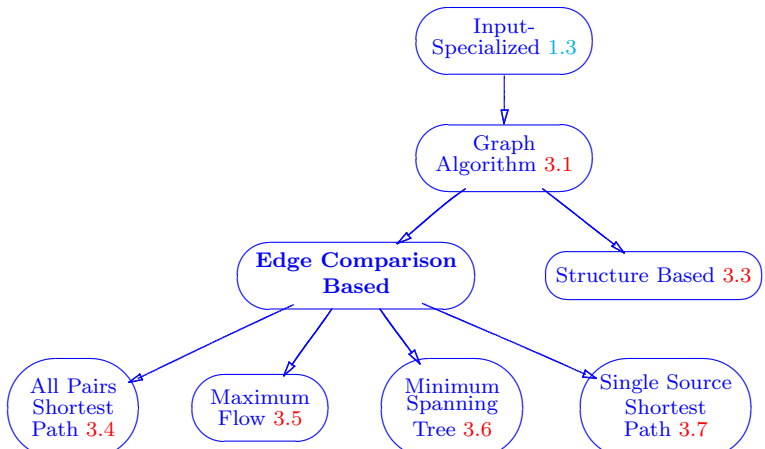


A *graph algorithm* is an algorithm (§1.2) that takes one or more graphs as inputs.

Performance constraints on graph algorithms are generally expressed in terms of the number of vertices ($|V|$) and the number of edges ($|E|$) in the input graph.

Refinement of: Algorithm Specialized by Input (§1.3).

3.2. Edge Comparison Based Graph Algorithm



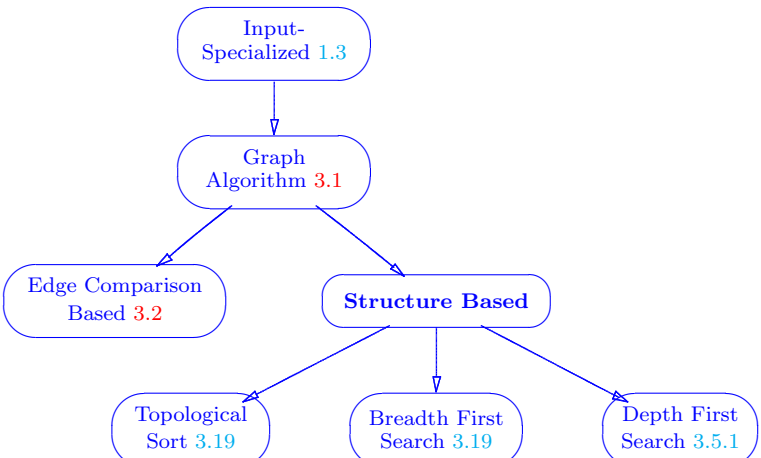
An *edge comparison based graph algorithm* is a graph algorithm (§3.1) whose computation depends on comparisons between pairs of values associated with the edges of the graph. In other words, in addition to an input graph, the algorithm requires at least one edge property map which affects the output of the algorithm.

Many of the well-known graph algorithms are models of this concept. The definition above does not place any restriction on the vertex properties, but often a property map which serves to label the vertices is needed if

the output of the algorithm is to be meaningful.

Refinement of: Graph Algorithm (§3.1)

3.3. Structure Based Graph Algorithm



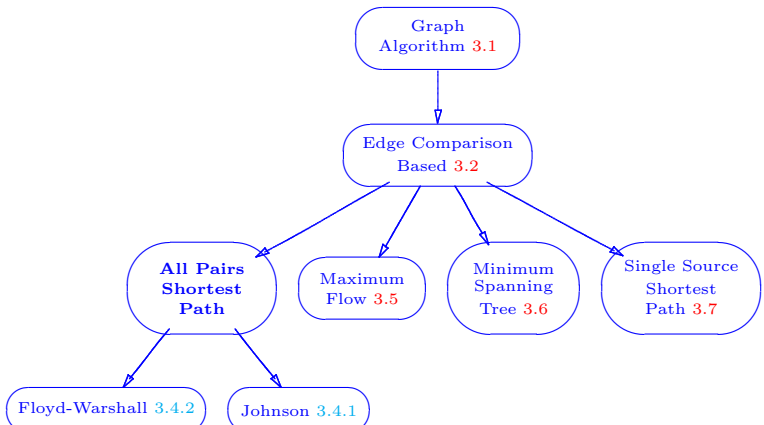
A *structure based graph algorithm* is a graph algorithm (§3.1) which operates strictly on the structural components of the graph. No edge or vertex property maps are required, just the sets of vertices and edges themselves.

There is one important point to be made. The restriction that no property maps are needed only applies to the input. Algorithms for computing a topological sorting (§3.19) do not need any semantic information about a graph, but do need to create a vertex property map to specify their

output.

Refinement of: Graph Algorithm (§3.1)

3.4. All Pairs Shortest Path Algorithm



An *all pairs shortest path algorithm* is any algorithm which calculates the *shortest path* between all pairs of vertices in a given graph. A shortest path is defined in terms of some edge property map, whose range has a well defined ordering and summation operation. For example, if the edges are given weights with an edge property map whose range is the integers, then a shortest path is simply the path between two vertices where the sum of weights for the edges used is smallest.

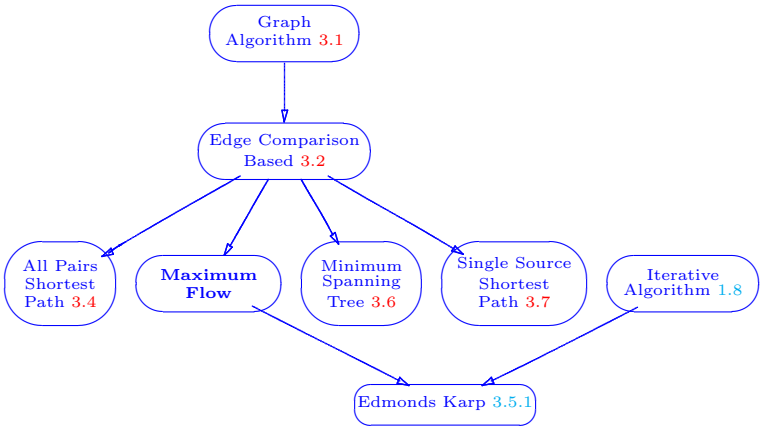
Calculating all such paths and their weights requires both an input graph, and an edge property map. Therefore, such algorithms are a refinement

of edge comparison based algorithms (§3.2).

The problem of finding the shortest path from *one* vertex to all other vertices is similar, but usually solved differently, and is referred to here as single source shortest path (§3.7).

Refinement of: Edge Comparison Based (§3.2)

3.5. Maximum Flow Algorithm



A *maximum flow algorithm* is any algorithm which calculates the greatest quantity of *flow* a graph will allow from some *source* vertex to another *sink* vertex. This is one aspect of a larger field which studies *flow networks*. A flow network is a directed graph (G) and an associated edge property map, which denotes the *capacity* of each edge.

A full, formal explanation of maximum flow is beyond the scope of this document. However, a maximum flow can be viewed as another edge property map (f) which defines how "much" each edge is used. No edge can be used beyond its capacity. The total in-flow (sum of f for each

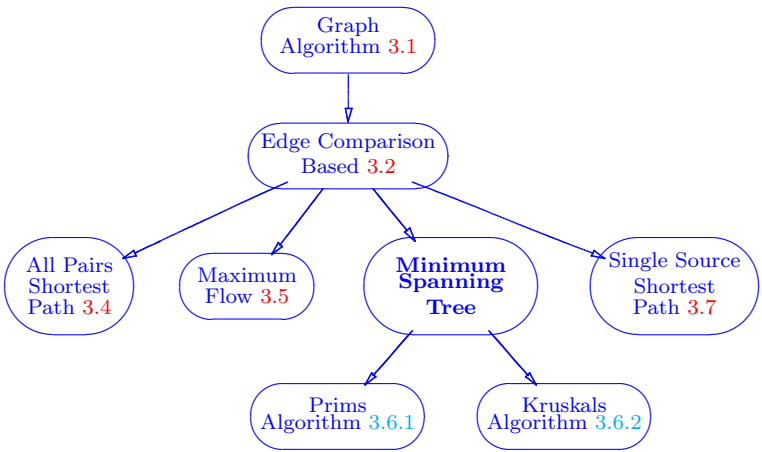
in-edge) at each vertex must be equal to the total out-flow (sum of f for each out-edge) at that vertex.

The sink and source vertices are two distinct nodes in the graph which "produce" and "consume", respectively, the "flow" represented by the f property map. Thus, the out-flow of the souce is equal to the in-flow of the sink. The goal for maximum flow algorithms is to maximize this value.

Calculating such a value requires both a graph and an edge property map. Algorithms to solve the problem are therefore refinements of edge comparison based algorithms (§3.2).

Refinement of: Edge Comparison Based (§3.2)

3.6. Minimum Spanning Tree Algorithm

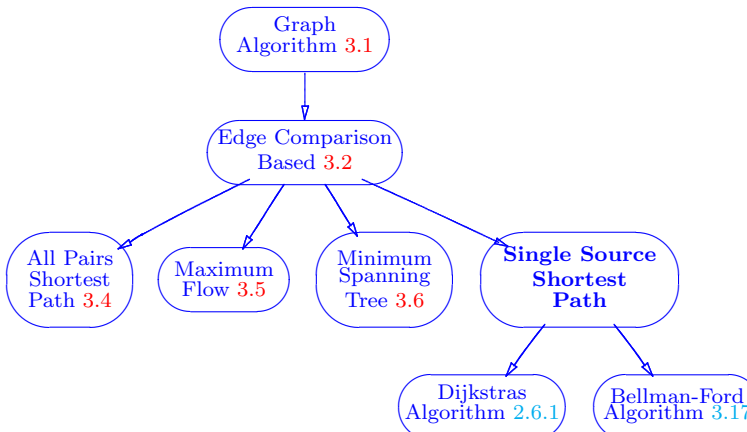


A *minimum spanning tree algorithm* is any algorithm which calculates the least weight *spanning tree* of some graph. A spanning tree is an acyclic, connected subgraph of a given graph which includes all of its vertices. While any graph has many spanning trees, the problem is to find the one with the minimum weight, where the weight is defined by some edge property map whose range has both a well defined ordering and summation operation. For example, if the edges have integer weights, then the minimum spanning tree is the tree which connects all of the vertices, and where the sum of the edge weights is smallest.

Calculating the minimum spanning tree requires both a graph, and an edge property map. Such algorithms are therefore a refinement of edge comparison based algorithms (§3.2).

Refinement of: Edge Comparison Based (§3.2)

3.7. Single Source Shortest Path Algorithm



A *single source shortest path algorithm* is any algorithm which calculates the *shortest path* from a source vertex to all other vertices in a given graph. A shortest path is defined in terms of some edge property map, whose range has a well defined ordering and summation operation. For example, if the edges are given weights with an edge property map whose range is the integers, then a shortest path is simply the path between two vertices where the sum of weights for the edges used is smallest.

Calculating all such paths and their weights requires both an input graph, and an edge property map, so such algorithms are a refinement of edge comparison based algorithms (§3.2).

The problem of finding the shortest path between two specific vertices is a commonly examined subproblem of this one, but which requires the same amount of time (asymptotically). Finding the shortest paths from *every* vertex to every other vertex is another similar problem which can be solved using single source shortest path algorithms, but for which better algorithms do exist. These are categorized as all pairs shortest path algorithms (§3.4).

Refinement of: Edge Comparison Based (§3.2).