Path-based strong component algorithm

From Wikipedia, the free encyclopedia

In graph theory, the strongly connected components of a directed graph may be found using an algorithm that uses depth-first search in combination with two stacks, one to keep track of the vertices in the current component and the second to keep track of the current search path.[1] Versions of this algorithm have been proposed by Purdom (1970), Munro (1971), Dijkstra (1976), Cheriyan & Mehlhorn (1996), and Gabow (2000); of these, Dijkstra's version was the first to achieve linear time.[2]

Contents [hide]

1 Description
2 Related algorithms
3 Notes
4 References

Description [edit]

The algorithm performs a depth-first search of the given graph *G*, maintaining as it does two stacks *S* and *P*. Stack *S* contains all the vertices that have not yet been assigned to a strongly connected component, in the order in which the depth-first search reaches the vertices. Stack *P* contains vertices that have not yet been determined to belong to different strongly connected components from each other. It also uses a counter *C* of the number of vertices reached so far, which it uses to compute the preorder numbers of the vertices.

When the depth-first search reaches a vertex *v*, the algorithm performs the following steps:

1. Set the preorder number of *v* to *C*, and increment *C*.
2. Push *v* onto *S* and also onto *P*.
3. For each edge from *v* to a neighboring vertex *w*:
   - If the preorder number of *w* has not yet been assigned, recursively search *w*;
   - Otherwise, if *w* has not yet been assigned to a strongly connected component:
     - Repeatedly pop vertices from *P* until the top element of *P* has a preorder number less than or equal to the preorder number of *w*.
4. If *v* is the top element of *P*:
   - Pop vertices from *S* until *v* has been popped, and assign the popped vertices to a new component.
   - Pop *v* from *P*.

The overall algorithm consists of a loop through the vertices of the graph, calling this recursive search on each vertex that does not yet have a preorder number assigned to it.

Related algorithms [edit]

Like this algorithm, Tarjan's strongly connected components algorithm also uses depth first search together with a stack to keep track of vertices that have not yet been assigned to a component, and moves these vertices into a new component when it finishes expanding the final vertex of its component. However, in place of the second stack, Tarjan's algorithm uses a vertex-indexed array of preorder numbers, assigned in the order that vertices are first visited in the depth-first search. The preorder array is used to keep track of when to form a new component.

Notes [edit]

1. ^ Sedgewick (2004).
2. ^ History of Path-based DFS for Strong Components, Hal Gabow, accessed 2012-04-24.

References [edit]

- Cheriyan, J.; Mehlhorn, K. (1996), "Algorithms for dense graphs and networks on the random access computer", *Algorithmica* **15**: 521–549, doi:10.1007/BF01940880.

- Dijkstra, Edsger (1976), *A Discipline of Programming*, NJ: Prentice Hall, Ch. 25.
- Gabow, Harold N. (2000), "Path-based depth-first search for strong and biconnected components", *Information Processing Letters* **74** (3-4): 107–114, doi:10.1016/S0020-0190(00)00051-X, MR 1761551.
- Munro, Ian (1971), "Efficient determination of the transitive closure of a directed graph", *Information Processing Letters* **1**: 56–58, doi:10.1016/0020-0190(71)90006-8.
- Purdom, P., Jr. (1970), "A transitive closure algorithm", *BIT* **10**: 76–94, doi:10.1007/bf01940892.
- Sedgewick, R. (2004), "19.8 Strong Components in Digraphs", *Algorithms in Java, Part 5 – Graph Algorithms* (3rd ed.), Cambridge MA: Addison-Wesley, pp. 205–216.

Categories: Graph algorithms | Graph connectivity