CS472 Module 3 Part D - Divide and Conquer - Graph Algorithms

Athens State University

February 18, 2014

Outline

Contents

1 Graphs and divide and conquer

Reviewing some terms: Example

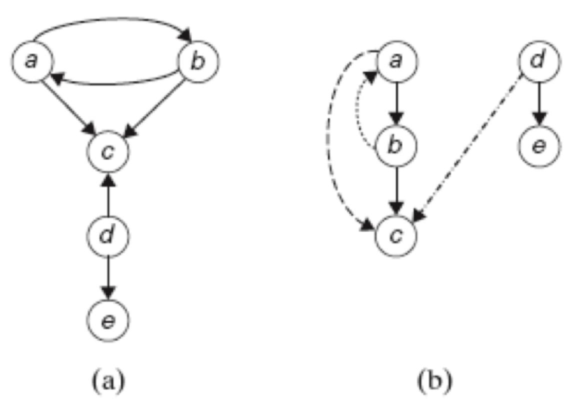


FIGURE 4.5 (a) Digraph. (b) DFS forest of the digraph for the DFS traversal started at a.

Reviewing some terms: Graph Search Trees

- Tree edge
- Back edge: from vertex to ancestor

- Forward edge: vertex to descendant other than children
- Cross edge: None of the other types

Reviewing some terms: types of graphs

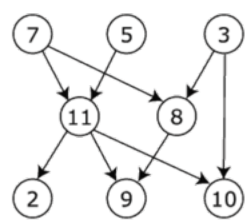
- *Directed graph*: a graph with directions specified for all of its edges.
- *Directed acyclic graph* (dag): Directed graph whose DFS forest has no back edges
- The nature of how we specify graphs means that many of the algorithms that work upon them are divide-and-conquer algorithms

2 Topological sorting

Topological sorting

- *Topological sort*: For a directed graph, a linear ordering of the graphs vertices s.t. for every directed edge uv, u comes before v in the ordering
- How to "rearrange" a directed graph
- Applications
  - Job scheduling in project planning
  - Compiler instruction scheduling
  - Determining in what order to compile and link files in makefiles and IDE project files.

Topological sorting



- 7,5,3,11,8,2,9,10
  - (visual left to right, top to bottom)
- 3,5,7,8,11,2,9,10
  - (smallest-numbered available vertex first)
- 7,5,11,3,10,8,9,2
  - (largest-numbered available vertex first)
- 7,5,11,2,3,8,9,10
- (top-to-bottom, left-to-right, as best as we can)

Topological Sorting: Using DFS

- Perform a DFS traversal of the dag
  - Keep track of the order that vertices are popped off the traversal stack
  - Reverse order solves the topological sorting problem
  - Detection of back edge means that you are not working with a dag!

Topological Sorting: Using DFS: Example

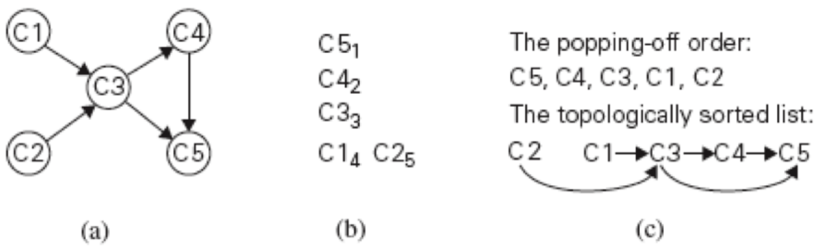
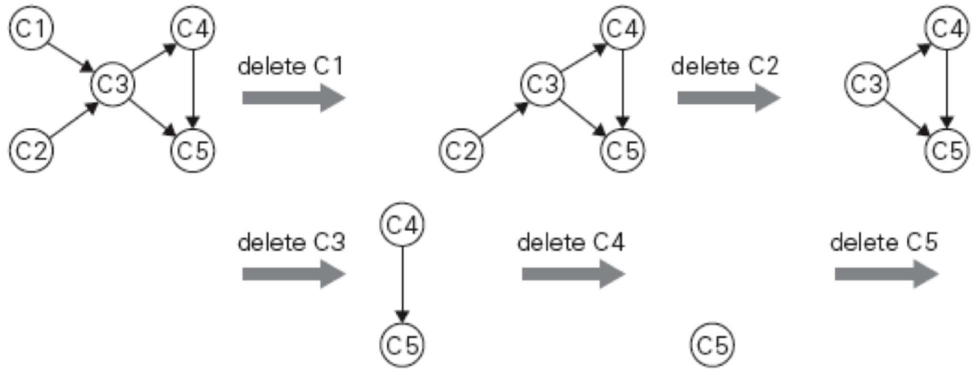


FIGURE 4.7 (a) Digraph for which the topological sorting problem needs to be solved. (b) DFS traversal stack with the subscript numbers indicating the popping-off order. (c) Solution to the problem.

Topological Sorting: Source removal

- *Source*: a vertex with no incoming edges
- Repeatedly identify and remove a source and all edges incident to it until
  - either no vertex is left (SOLVED!)
  - no source remains among the remaining vertices (not a dag)

Topological Sorting: Source removal: Example



The solution obtained is C1, C2, C3, C4, C5

FIGURE 4.8 Illustration of the source-removal algorithm for the topological sorting problem. On each iteration, a vertex with no incoming edges is deleted from the digraph.

3 Closed-Pair Problem: A better solution

Closest-Pair Problem

Suppose that we want to find the two closest points in a set of n points in a plane

- Points can be location of physical objects
- Or database records upon which we want to perform cluster analysis

Closed-Pair Problem: Brute Force

- Compute the distance between every pair of distinct points and return the indexes of the points for which the distance is smallest
- This approach is $\Theta(n^2)$
- Can we use divide-and-conquer to do better?

Closed-Pair Problem: Divide-and-Conquer

- Divide the points into subsets P_L and P_R by a vertical line $x = m$ so that half the points lie to the left or on the line and half the points lie to the right or on the line.
- Recursively search for the closet pairs in the left and right subsets
- Set $d = \min(d_L, d_R)$
  - This narrows the points down to the those points in the symmetric vertical strip S of width $2d$ as possible closest pair (with points stored in processed in increasing order of their y coordinate)
- Scan the points in S from lowest up. For every $p(x, y) \in S$, inspect points in the strip that may be closer to p than d
  - There can be no more than 5 such points following p on the strip list!

Closed-Pair Problem: Analysis

We have the recurrence relation

$$T(n) = 2T(n/2) + M(n)$$

where $M(n)$ is a linear time function that accounts for the time required to split the list into two parts.

- If we set $a = 2$, $b = 2$, and $d = 1$, then by the Master Theorem, $T(n) \in O(n * log(n))$

4 Convex-Hull Problem

Convex-Hull Problem: Definition

A set of points (finite or infinite) in the plane is called *convex* if for any two points p and q in the set, the entire line segment with endpoints of p and q is also in the set.

The *convex hull* of a set S of points is the smallest convex set containing S. The "smallest" requirement says that the convex hull of S must be a subset of any convex set containing S

The brute force algorithm for solving this problem is $O(n^3)$

Convex-Hull Problem: Quickhull

- Assume that the points are sorted by their x coordinate
- Identify two *extreme points* P_1 and P_2
  - The leftmost and rightmost points!
- Recursively compute the *upper hull*
  - Find point P_{max} that is farthest away from the line P_1P_2
  - Compute the upper hull of the points to the left of line P_1P_{max}
  - Compute the upper hull of the points to left of line $P_{max}P_2$
- Recursively compute the *lower hull* in the same manner

Convex-Hull Problem: Quickhull Analysis

- Finding the point farthest away form the line P_1P_2 can be done in linear time
- Time efficiency
  - Worst case: $\Theta(n^2)$
  - Best case: $\Theta(n)$
- Points, in needed, can be sorted in $O(n * log(n))$ time
- Note the similarity to Quicksort in both the algorithm and analysis

5 Key Points

Key Points

- Many graph and geometric algorithms can be solved using divide-and-conquer techniques
- Topological sorting
- Better algorithms for closest-pair and convex-hull