Algorithms and Data Structures 2010-2011

Lesson 6: *exercises, graphs and trees,* priority queues and heaps

Luciano Bononi

<bononi@cs.unibo.it>

http://www.cs.unibo.it/~bononi/

(slide credits: these slides are a revised version of slides created by Dr. Gabriele D'Angelo)

International Bologna Master in Bioinformatics

University of Bologna

20/05/2011, Bologna

Outline of the lesson

- **Graphs**
  - Exercises
    - Connected components
    - DFS and BFS
- **Trees**
  - Traversals
  - Exercises

Connected components

- **EXERCISE:** write the pseudo-code to find the number and the composition of *connected components* in a given graph G

- **SUGGESTION**: the algorithm could be based on a slightly modified version of the DFS traversal

Connected components: pseudo-code

- **ASSUMING**:
  - **COMP[]** array of integers, size = # of vertex in the graph
  - **SIZE(G)** given a graph G, returns the number of vertices
  - **VERTEX (G, i)** given a graph G and an integer i, returns the vertex identified by i
  - **ORD(G, u)** given a graph G and a vertex u, returns the identifier of u as an integer

Connected components: pseudo-code

```
CONNEXCOMP(G)
   numcomp := 0;
   for i:=1 to SIZE(G) do COMP[i] := 0;
   for i:=1 to SIZE(G) do
      if COMP[i] == 0 then
            numcomp := numcomp + 1;
            DFS-MODIFIED(G, VERTEX(G, i), numcomp);
```


Connected components: pseudo-code

```
DFS-MODIFIED(G, u, i)
   COMP[ORD(G, u)] := i;
   for each v in ADJSET(G, u) do
      if COMP[ORD(G, v)] == 0 then
            DFS-MODIFIED(G,v, i);
```

- Complexity in terms of space and computation?

Graphs: exercises

- Write the Depth-First-Search (DFS) procedure in pseudo-code. Given the graph
  - G=(N, A)
    - N={1, 2, 3, 4,5, 6}
    - A={(1,4), (1,5), (2,5), (3,6), (4,5), (5,2), (5,3), (6,5)}

  Execute the DFS procedure starting from the vertex 2

  Plot the graph and show its representation based on adjacency set implemented using vertex and edges vectors

  Show the visited nodes and edges, assuming that the vectors are in not decreasing order

Graphs traversal: DFS

```
Procedure DFS(G : graph; u : vertex)
    /* visit the vertex u and mark it as visited */
    for each v in AdjSet(G, u)
        /* visit the edge (u, v) */
        if (v is not marked) then  DFS(G, v)
```

Graphs: exercises

- Write the Depth-First-Search (DFS) procedure in pseudo-code. Given the graph
  - G=(N, A)
    - N={1, 2, 3, 4,5, 6}
    - A={[1,4], [1,5], [2,5], [3,6], [4,5], [5,3], [6,5]}

  Execute the DFS procedure starting from the vertex 2

  Plot the graph and show its representation based on adjacency set implemented using vertex and edges vectors

  Show the visited nodes and edges, assuming that the vectors are in not decreasing order

Graphs: exercises

- Write the Breadth-First-Search (BFS) procedure in pseudo-code. Given the graph
  - G=(N, A)
    - N={1, 2, 3, 4,5, 6}
    - A={[1,4], [1,5], [2,5], [3,6], [4,5], [5,2], [5,3], [6,5]}

  Execute the BFS procedure starting from the vertex 2

  Plot the graph and show its representation based on adjacency set implemented using vertex and edges vectors

  Show the visited nodes and edges, assuming that the vectors are in not decreasing order

Graphs traversal: BFS

```
Procedure BFS(G : graph; u : vertex)
   Make(Q); Enqueue(Q, u);
   while not Empty(Q) do
      u := Dequeue(Q);
      /* visit the vertex u and mark it as visited */
      for each v in AdjSet(G, u)
            /* visit the edge (u, v) */
            if (v is not marked) and (v is not in Q) then
                  Enqueue(Q, v)
```


Graphs: exercises

- Write the Breadth-First-Search (BFS) procedure in pseudo-code. Given the directed graph
  - G=(N, A)
    - N={1, 2, 3, 4, 5, 6}
    - A={(1,4), (1,5), (2,1), (2,3), (2,5), (3,6), (4,5), (5,2), (5,3), (6,5)}

  Execute the BFS procedure starting from the vertex 2

  Plot the graph and show its representation based on adjacency set implemented using vertex and edges vectors

  Show the visited nodes and edges, assuming that the vectors are in not decreasing order

Binary trees: pre-order visit

```
function preorder(T, node)
    if (node <> NULL) then {
        visit(T, node);          // visits the node, i.e. prints the data
        preorder(T, GetLeftChild(T, node));
        preorder(T, GetRightChild(T, node));
    }
```


Binary trees: in-order visit

```
function inorder(T, node)
    if (node <> NULL) then {
        inorder(T, GetLeftChild(T, node));
        visit(T, node);          // visits the node, i.e. prints the data
        inorder(T, GetRightChild(T, node));
    }
```

Binary trees: post-order visit

```
function postorder(T, node)
    if (node <> NULL) then {
        postorder(T, GetLeftChild(T, node));
        postorder(T, GetRightChild(T, node));
        visit(T, node);           // visits the node, i.e. prints the data
    }
```


Trees: exercises

- Given 2 trees (I and II)composed of nodes
  - N = {10, 20, 30, 40, 50} and with
    - **PREORDER**(I) = **PREORDER**(II)
    - **POSTORDER**(I) = **POSTORDER**(II)
    - **INORDER**(I) is the inverse of **INORDER**(II)
  - plot both trees

Trees: exercises

- Given a tree (I)composed of nodes
  - N = {1, 2, 7, 8, 9, 14, 27, 31} and with
    - **PREORDER**(I) = 27/31/9/8/7/2/14/1
    - **INORDER(I)** = 9/31/7/2/8/27/14/1
    - **POSTORDER(I)** = 9/2/7/8/31/1/14/27
  - plot the tree

Algorithms and Data Structures 2010 - 2011

Lesson 6: priority queues and heaps

Luciano Bononi
<bononi@cs.unibo.it>
http://www.cs.unibo.it/~bononi/
(slide credits: these slides are a revised version of slides created by Dr. Gabriele D'Angelo)
International Bologna Master in Bioinformatics
University of Bologna

20/05/2011, Bologna

Abstract Data Type: **Priority Queue**

- **PRIORITY QUEUE** is an abstract data type that supports the following operations:
  - **InsertWithPriority(Q, e, p)**
    - add the element (e) to the queue (Q) with an associated priority (p)
  - **GetNext(Q)**
    - remove the element from the queue that has the highest priority

Priority Queue: implementations

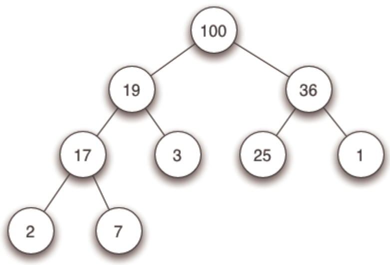
- Some of the data structure that can be used for the implementation:
  - **sorted list implementation**
  - **unsorted list implementation**
- Are these implementations efficient or not?

Heap

- A **Heap** is a **complete** tree that satisfies the **heap property**
- **Heap property**:
  - if B is a child of node A, then key(A) ≥ key(B)
- It the heap property is satisfied then the node with higher value is in the root of the tree (max-heap)

Heap: example

- Heap-max example (from wikipedia)



Heap: implementation

- How is possible to efficiently design these operations?
  - **GetNext()**
  - **InsertWithPriority()**
- What is the cost of such operations?
- How is possible to implement a binary heap using an array?

Algorithms and Data Structures 2010 - 2011

Lesson 6: *exercises, graphs and trees,* priority queues and heaps

Luciano Bononi
<bononi@cs.unibo.it>
http://www.cs.unibo.it/~bononi/
(slide credits: these slides are a revised version of slides created by Dr. Gabriele D'Angelo)
International Bologna Master in Bioinformatics

University of Bologna

20/05/2011, Bologna