Fondamenti di Informatica (Elettronici)

Rappresentazioni dei grafi – Lezione 38

18 gennaio 2021

Rappresentazioni dei grafi[1]

1. Alberi come strutture dati
2. Alberi binari
3. Rappresentazioni dei grafi
4. Rappresentazioni collegate
5. Rappresentazioni Matriciali
6. Algoritmi su grafi

[1] Tratto da: Pathak, Go, Zeng, and Boyd, EE108: Introduction to Matrix Methods, Stanford University, 2016.

Section 1

Alberi come strutture dati

Un tipo IntTree per alberi ordinati

Definiamo un valore del tipo IntTree per ogni nodo dell'albero

```
struct IntTree       # un record per ogni nodo
    node :: Int64        # il codice del nodo
    childs :: Array{IntTree,1}      # un array per i figli del nodo
end

# foglia := nodo senza figli
leaf(x:: Int64) = IntTree(x,[])

# predicato `true` sui nodi foglia
isLeaf(t:: IntTree) = t.childs == []

# predicato `true` quando il nodo `t` ha un solo figlio
oneChild(t:: IntTree) = length(t.childs) == 1

# per "attaccare" i figli in `ts` al nodo `x`
tree(x:: Int64, ts:: Array{IntTree,1}) = IntTree(x,ts)
```

Costruzione di un albero ordinato

```
# Esempi per il tipo IntTree

l1 = leaf(1)
l2 = leaf(64)
l3 = leaf(17)
l4 = leaf(9)

t1 = tree(11,[l1,l2])
t2 = tree(4,[l3,l4])
t3 = tree(6,[l1,l2,l4])
t4 = tree(21,[t1,t3,l3])
```

Prima definiamo le foglie, poi i sottoalberi, partendo dalle foglie, fino a definire la radice.

Funzione `drawTree`

Visita ricorsiva in preordine:

- Prima si elabora il nodo (partendo dalla radice)

Funzione `drawTree`

Visita ricorsiva in preordine:

- Prima si elabora il nodo (partendo dalla radice)
- poi si elaborano i suoi figli

Funzione `drawTree`

Visita ricorsiva in preordine:

- Prima si elabora il nodo (partendo dalla radice)
- poi si elaborano i suoi figli

Funzione drawTree

Visita ricorsiva in preordine:

- Prima si elabora il nodo (partendo dalla radice)
- poi si elaborano i suoi figli

```
function drawTree(r:: IntTree)
    if (isLeaf(r))
        return string(r.node)
    elseif (oneChild(r))
        return string(r.node) * " -- " * drawTree(r.child[1])
    else
        xs = map( drawTree, r.childs )
        return string(r.node) * " -- " * "{ " * join(xs, ", ") * " }"
    end
end
```

‘‘‘

Funzione tree2Latex

Genera dei dati `tikzpicture` per disegnare un albero con il package `tikz` usando LaTeX.

```
function tree2Latex(t:: IntTree, fileName:: String)
    s1 = "\\begin{tikzpicture}[every node/.style={circle, draw,
                minimum size=0.75cm}] \n\n"
    s2 = "\\graph [tree layout, grow=down, fresh nodes,
                level distance=0.5in, sibling distance=0.5in] \n {"
    s3 = " };\n \\end{tikzpicture}"
    s = s1 * s2 * drawTree(t) * s3
    outfile = open(fileName, "w")
    write(outfile, s)
    close(outfile)
end

tree2Latex(t, "t.tex")
```

‘‘‘

Latex

LaTeX è un sistema software per la preparazione dei documenti. Durante la scrittura, lo scrittore utilizza il testo normale invece del testo formattato che si trova in elaboratori di testi "Quello che vedi è quello che ottieni" (WYSIWYG) come Microsoft Word, LibreOffice Writer e Apple Pages.

L'autore utilizza le convenzioni di codifica del linguaggio di markup TeX per definire la struttura generale di un documento (come articolo, libro e lettera), per stilizzare il testo in tutto il documento (come grassetto e corsivo) e per aggiungere citazioni e riferimenti incrociati.

Il sistema LaTeX è un linguaggio di markup che gestisce la composizione e il rendering e può essere esteso arbitrariamente utilizzando il linguaggio macro sottostante per sviluppare macro personalizzate come nuovi ambienti e comandi.

Tali macro sono spesso raccolte in pacchetti, che potrebbero quindi essere resi disponibili per soddisfare alcune specifiche esigenze di composizione come la formattazione di espressioni matematiche complesse o grafici.

Latex

Università di Udine

Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Introduzione al LaTeX

Gianluca Gorni

10 giugno 2020

Il corso 2

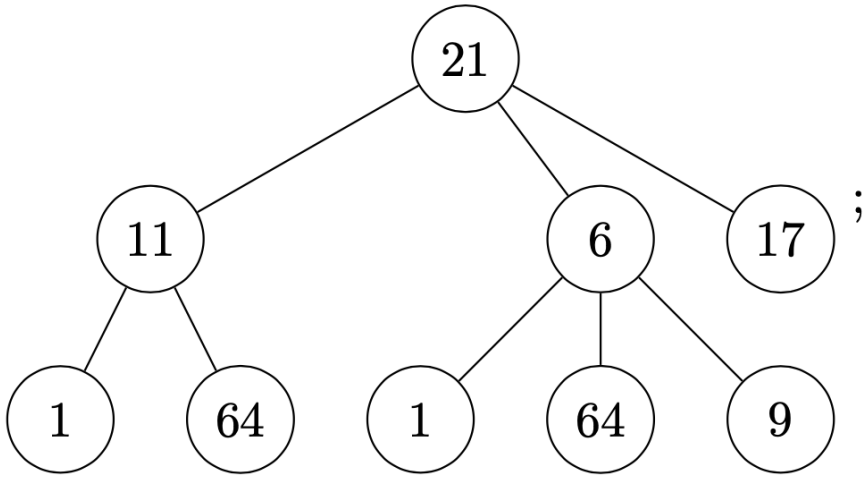
Visualizzazione del grafo

Lo script successivo è salvato nel file sorgente LaTeX di nome `TreeDrawing.tex`:

```
documentclass{article}

\usepackage{tikz}
\usetikzlibrary{graphdrawing}

\usepackage{luacode}

\usetikzlibrary{graphs}
\usegdlibrary{trees}

\begin{document}
    \input{t.tex};
\end{document}
```

Visualizzazione dell'albero

Esecuzione da shell per creare un file oggetto `TreeDrawing.pdf`, dopo essersi posizionati nel direttorio che contiene `TreeDrawing.tex`




```
$ lualatex -interaction=nonstopmode TreeDrawing.tex
```

LuaTeX is an extended version of pdfTeX using Lua as an embedded scripting language.

Una differente struttura dati (notebook treestruct)

```
struct TreeNode
    parent::Int
    children::Vector{Int}
end

struct Tree
    nodes::Vector{TreeNode}
end
Tree() = Tree([TreeNode(0, Vector{Int}())])

function addchild(tree::Tree, id::Int)
    1 <= id <= length(tree.nodes) || throw(BoundsError(tree, id))
    push!(tree.nodes, TreeNode(id, Vector{}()))
    child = length(tree.nodes)
    push!(tree.nodes[id].children, child)
    child
end
children(tree, id) = tree.nodes[id].children
parent(tree,id) = tree.nodes[id].parent
```

Una differente struttura dati

```
TreeNode(22,[])
TreeNode(99,[])
TreeNode(48,[])
TreeNode(100,[22,99])
TreeNode(7,[48])
TreeNode(13,[100,7])

t = Tree([TreeNode(22,[]),
TreeNode(99,[]),
TreeNode(48,[]),
TreeNode(100,[22,99]),
TreeNode(7,[48]),
TreeNode(13,[100,7])
])
```

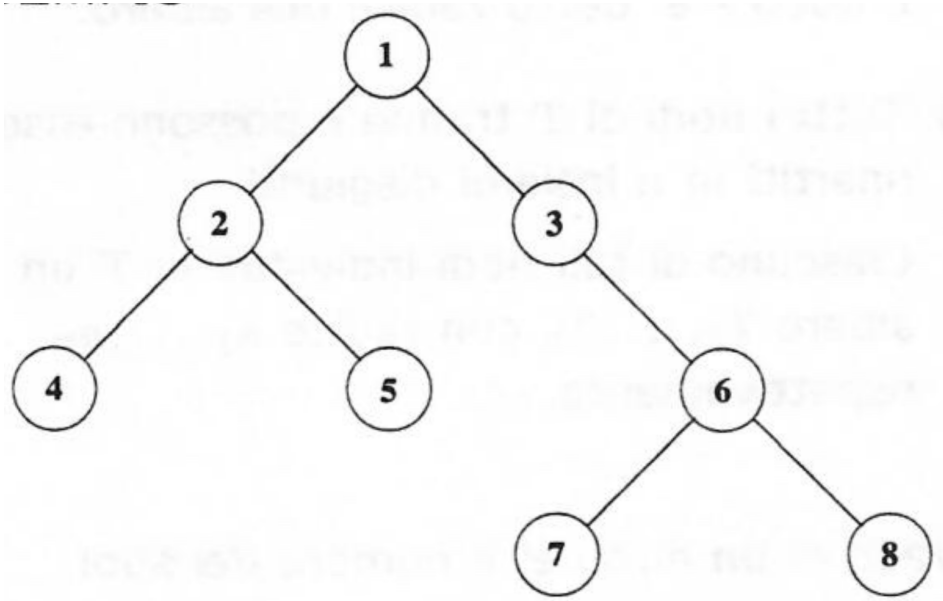
Section 2

Alberi binari

Alberi binari

Albero binario e' un albero ordinato dove ogni nodo ha al massimo due figli, detti figlio sinistro e figlio destro, rispettivamente

ESEMPIO



Albero binario di profondità 3 (massimo livello dei nodi) ## Alberi binari

ALBERI COMPLETI E BILANCIATI

Un albero binario e' detto completo se —ogni nodo non foglia ha esattamente due figli

Albero Binario

Notazione parentetica:

- Ogni nodo è un array di tre elementi

```
tree = [1, [2, [4, [7, [],
                        []],
                   []],
               [5, [],
                   []]],
           [3, [6, [8, [],
                       []],
                   [9, [],
                       []]],
               []]]
```

Albero Binario

Notazione parentetica:

- Ogni nodo è un array di tre elementi
- Le foglie contengono due array vuoti

```
tree = [1, [2, [4, [7, [],
                        []],
                   []],
              [5, [],
                   []]],
         [3, [6, [8, [],
                     []],
                [9, [],
                     []]],
              []]]
```

Albero Binario – Operazioni di visita

```
preorder(t, f) = if !isempty(t)
                     f(t[1]); preorder(t[2], f); preorder(t[3], f)
                 end
inorder(t, f)  = if !isempty(t)
                     inorder(t[2], f); f(t[1]); inorder(t[3], f)
                 end
postorder(t, f) = if !isempty(t)
                      postorder(t[2], f); postorder(t[3], f); f(t[1])
                  end
levelorder(t, f) = while !isempty(t)
                       t = mapreduce(x -> isa(x, Number) ?
                                          (f(x); []) :
                                          x, vcat, t)
                   end
```

Albero Binario

```
for f in [preorder, inorder, postorder, levelorder]
    print((lpad("$f: ", 12))); f(tree, x -> print(x, " ")); println()
end
```

```
  preorder: 1 2 4 7 5 3 6 8 9
   inorder: 7 4 2 5 1 8 6 9 3
 postorder: 7 4 5 2 8 9 6 3 1
levelorder: 1 2 3 4 5 6 7 8 9
```

Abstract-BinaryTree notebook

```
using AbstractTrees

mutable struct BinaryNode{T}
    data::T
    parent::BinaryNode{T}
    left::BinaryNode{T}
    right::BinaryNode{T}

    # Root constructor
    BinaryNode{T}(data) where T = new{T}(data)
    # Child node constructor
    BinaryNode{T}(data, parent::BinaryNode{T}) where T = new{T}(data, parent)
end
BinaryNode(data) = BinaryNode{typeof(data)}(data)

function leftchild(data, parent::BinaryNode)
    !isdefined(parent, :left) || error("left child is already assigned")
    node = typeof(parent)(data, parent)
    parent.left = node
end
```

Abstract-BinaryTree notebook

```
function rightchild(data, parent::BinaryNode)
    !isdefined(parent, :right) || error("right child is already assigned")
    node = typeof(parent)(data, parent)
    parent.right = node
end

## Things we need to define
function AbstractTrees.children(node::BinaryNode)
    if isdefined(node, :left)
        if isdefined(node, :right)
            return (node.left, node.right)
        end
        return (node.left,)
    end
    isdefined(node, :right) && return (node.right,)
    return ()
end

## Things that make printing prettier
AbstractTrees.printnode(io::IO, node::BinaryNode) = print(io, node.data)
```

Abstract-BinaryTree notebook

Tree definition

```
## Let's test it. First build a tree.
root = BinaryNode(0)
l = leftchild(1, root)
r = rightchild(2, root)
ll = leftchild(3, l)
lr = rightchild(4, l)
rl = leftchild(5, r)
rr = rightchild(6, r);
```

Tree operations

```
print_tree(root)

collect(PostOrderDFS(root))
@static if isdefined(@__MODULE__, :Test)
    @testset "binarytree_easy.jl" begin
        @test [node.data for node in PostOrderDFS(root)] == [3, 4, 1, 5, 6, 2, 0]
        @test [node.data for node in PreOrderDFS(root)] == [0, 1, 3, 4, 2, 5, 6]
        @test [node.data for node in Leaves(root)] == [3, 2]
    end
end
```

Section 3

Rappresentazioni dei grafi

aaaaaaaa

Section 4

Rappresentazioni collegate

aaaaaaaa

Section 5

Rappresentazioni Matriciali

aaaaaaaa

Section 6

Algoritmi su grafi

Matrice delle distanze minime