THE PUBLIC IS MORE FAMILIAR WITH BAD DESIGN THAN GOOD DESIGN. IT IS, IN EFFECT, CONDITIONED TO PREFER BAD DESIGN, BECAUSE THAT IS WHAT IT LIVES WITH. THE NEW BECOMES THREATENING, THE OLD REASSURING.

PAUL RAND

A DESIGNER KNOWS THAT HE HAS ACHIEVED PERFECTION NOT WHEN THERE IS NOTHING LEFT TO TAKE

AWAY.

ANTOINE DE SAINT-EXUPÉRY

... THE DESIGNER OF A NEW SYSTEM MUST NOT ONLY BE THE IMPLEMENTOR AND THE FIRST LARGE-SCALE USER; THE DESIGNER SHOULD ALSO WRITE THE FIRST USER MANUAL... IF I HAD NOT PARTICIPATED FULLY IN ALL THESE ACTIVITIES, LITERALLY HUNDREDS OF IMPROVEMENTS WOULD NEVER HAVE BEEN MADE, BECAUSE I WOULD NEVER HAVE THOUGHT OF THEM OR PERCEIVED WHY THEY WERE IMPORTANT.

DONALD E. KNUTH

COLLECTION OF ALGO-RITHMS PSEUDOCODE

Copyright © 2018 Hengfeng Wei PUBLISHED BY ANT TUFTE-LATEX.GOOGLECODE.COM Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/ LICENSE-2.0. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "As is" basis, without warranties or conditions of any kind, either express or implied. See the License for the specific language governing permissions and limitations under

the License.

First printing, April 2018

Contents

Sorting	15		
Dynamic I	Programmi	ing	17
Traversal o	on Trees	19	
Bibliograp	hy 23		

List of Figures

- Calculate $\binom{4}{2}$ recursively. 17 Pascal triangle for binomial coefficients. 17

List of Tables

Dedicated to those who appreciate $atural E_{E}X$ and the work of Edward R. Tufte and Donald E. Knuth.

Introduction

This is a collection of psuedocode for classic algorithms.

Sorting

Algorithm 1 Selection Sort.

```
1: procedure Selection-Sort(A, n)

2: for i \leftarrow 1 to n - 1 do

3: for j \leftarrow i + 1 to n do

4: if A[j] < A[i] then

5: Swap(A[j], A[i])
```

Dynamic Programming

Algorithm 2 Computing $\binom{n}{k}$ recursively.

```
1: procedure BINOM(n,k)
```

 \triangleright Required: $n \ge k \ge 0$

2: **if** $k = 0 \lor n = k$ **then**

3: return 1

4: **return** BINOM(n-1,k) + BINOM(n-1,k-1)

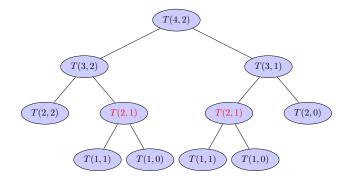


Figure 1: Calculate $\binom{4}{2}$ recursively.

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 2 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$$\begin{pmatrix} 3 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 3 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 3 \\ 2 \end{pmatrix} \quad \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

$$\begin{pmatrix} 4 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 4 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 4 \\ 2 \end{pmatrix} \quad \begin{pmatrix} 4 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

$$\begin{pmatrix} 5 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 5 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 5 \\ 2 \end{pmatrix} \quad \begin{pmatrix} 5 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 5 \\ 4 \end{pmatrix} \quad \begin{pmatrix} 5 \\ 5 \end{pmatrix}$$

Figure 2: Pascal triangle for binomial coefficients.

Algorithm 3 Computing $\binom{n}{k}$ by dynamic programming.

```
ightharpoonup Required: n \ge k \ge 0
 1: procedure BINOM(n,k)
 2:
          \mathbf{for}\ i \leftarrow 0\ \mathbf{to}\ n-k\ \mathbf{do}
               B[i][0] \leftarrow 1
 3:
          for i \leftarrow 1 to k do
 4:
               B[i][i] \leftarrow 1
 5:
          for j \leftarrow 1 to k do
 6:
               for d \leftarrow 1 to n - k do
 7:
                    i \leftarrow j + d
 8:
                    B[i][j] \leftarrow B[i-1][j] + B[i-1][j-1]
 9:
          return B[n][k]
10:
```

$$(n-k+1) + (k) + k(n-k) = nk - k^2 + n + 1$$

Traversal on Trees

DFS on Trees

```
Algorithm 4 Calculate the sum of depths of all nodes of a tree T.

1: procedure Sum-of-Depths()

2: return Sum-of-Depths(T,0)

3: procedure Sum-of-Depths(T, d)

4: if T is a leaf then

5: return d

6: Sum \leftarrow d

6: Sum \leftarrow d

7: for all child vertex T

8: Sum \leftarrow Sum \leftarrow Sum

9: return Sum
```

Algorithm 5 Count the number of nodes in *T* at depth *K*.

```
1: procedure Nodes-at-Depth()
       return Nodes-AT-Depth(T, K)
                                                      \triangleright r: root of a tree
3: procedure Nodes-At-Depth(r, k)
       if k = 0 then
          return 1
       if r is a leaf then
 6:
          return o
 7:
       num \leftarrow 0
 8:
       for all child vertex v of r do
          num \leftarrow num + \text{Nodes-at-Depth}(v, k - 1)
       return num
11:
```

Algorithm 6 Check whether a tree *T* has any leaf at an even depth.

BFS on Trees

Algorithm 7 Calculate the sum of contents of nodes of a tree *T* at each depth.

```
\triangleright r: root of the tree T
 1: procedure SUM-AT-DEPTH(r)
        r.depth \leftarrow 0
 2:
        Q \leftarrow \emptyset
        ENQUEUE(Q,r)
        while Q \neq \emptyset do
 5:
            u \leftarrow \text{Dequeue}(Q)
 6:
            sumAtDepth[u.depth] += u.content
 7:
            for all child vertex v of u do
 8:
                 v.depth \leftarrow u.depth + 1
 9:
                 Enqueue(Q, v)
10:
```

Algorithm 8 Count the number of nodes of a tree *T* at each depth.

```
1: procedure Nodes-at-Depth(r)
                                                               \triangleright r: root of the tree T
        r.depth \leftarrow 0
 2:
        Q \leftarrow \emptyset
 3:
        Enqueue(Q,r)
 4:
        while Q \neq \emptyset do
 5:
             u \leftarrow \mathsf{Dequeue}(Q)
 6:
             nodesAtDepth[u.depth] += 1
 7:
             for all child vertex v of u do
 8:
                 v.depth \leftarrow u.depth + 1
 9:
                 Enqueue(Q, v)
10:
        \textbf{return} \ \text{argmax}_{K} nodes AtDepth[k]
11:
```

Bibliography