Writing for

Computer Science & Engineering

Ki-Il Kim

Department of Computer Science and Engineering

CNU

Algorithms

Introduction

- A family of algorithms
  - Core contribution in many computer science papers
  - Yet, in many cases this effort is not reflected in the presentation
  - Not only are the steps of the algorithm often not made clear, but there is no discussion of why the reader should believe that the algorithm is correct, or believe that its behavior is reasonable
- This chapter is about
  - Effective description, analysis, and explantion of algorithms
  - Most challengesing single task in writing of papers in computing
  - Help reader to understand what the algorithm does, what effect it has, what is values is, and what its properties are

Presentation of Algorithms

- You should demonstrate that the algorithm is a worthwhile contribution
  - Show that given appropriate input, it terminates with appropriate outcomes by a combination of proof and experiment that it meets some claimed performance bound
- Algorithm
  - Provide a new or better way to compute a result
  - Can compute the result with fewer resources as demonstrated by mathematical analysis

Presentation of Algorithms

- A reader would expect to find of some or all of the following
  - The steps that make up the algorithm
  - The input and output, and the internal data structures used by the algorithm
  - The scope of application of the algorithms and its limitation
  - The properties that will allow demonstration of correctness, which might bt formally expressed as pre- and post-conditions and loop invariants
  - A demonstration of correctness
  - A formal analysis of cost, for both space and time requirements
  - Experiments confirming that theoretical results
- The presentation of an algorithm, it is usual to give a formal demonstration of correctness and performance, and perhaps an experimental validation

Foramlisms

- Description of an algorithm usually consists of the algorithm itself and the environment it requires
- Common formalisms for presenting algorithms
  - List style : a series of numbered or named steps and loops involving several steps are represented by "go to step X" statement
  - Pseudocode : presented as if written in a block-structured language and each line is numbered
  - Prosecode : described by text which embedded code. Structure is given by numbered lists in which lops are presented as sublists with nested numbering. The assignment symbol "<-" is good choice rather than "= "
  - Literate code : detail of the algorithm is introduced gradually, intermingled with discussion of the underlying ideas and perhaps with the asymptotic analysis and proof of correctness

Foramlisms

- Description of an algorithm usually consists of the algorithm itself and the environment it requires
- Common formalisms for presenting algorithms
  - List style : a series of numbered or named steps and loops involving several steps are represented by "go to step X" statement
  - Pseudocode : presented as if written in a block-structured language and each line is numbered
  - Prosecode : described by text which embedded code. Structure is given by numbered lists in which lops are presented as sublists with nested numbering. The assignment symbol "<-" is good choice rather than "= "
  - Literate code : detail of the algorithm is introduced gradually, intermingled with discussion of the underlying ideas and perhaps with the asymptotic analysis and proof of correctness

Foramlisms

- **List Style**
  - Algorithm broken down into a sequence of steps (numbered or named)
  - See loops as involving ‘goto’ statements

- **Good**
  - Discuss while presenting
  - No restriction to text
- **Bad**
  - Sometimes easy to get lost in the discussion

Foramlisms

❖ Pseudocode

The **WeightedEdit** function computes the edit distance between two strings, assigning a higher penalty for errors closer to the front.

Input: $S1, S2$: strings to be compared.
Output: weighted edit distance
Variables: $L1, L2$: string lengths
$F[L1, L2]$: array of minimum distances
W: current weighting
M: maximum penalty
C: current penalty

WeightedEdit$(S1, S2)$:

1. $L1 = len(S1)$
2. $L2 = len(S2)$
3. $M = 2 \times (L1 + L2)$
4. $F[0,0] = 0$
5. **for** i **from** 1 **to** $L1$
6. $\quad F[i,0] = F[i-1,0] + M - i$
7. **for** j **from** 1 **to** $L2$
8. $\quad F[0,j] = F[0,j-1] + M - j$
9. **for** i **from** 1 **to** $L1$
10. $\quad C = M - i$
11. $\quad$ **for** j **from** 1 **to** $L2$
12. $\quad\quad C = C - 1$
13. $\quad\quad F[i,j] = min(F[i-1,j] + C,$
$\quad\quad\quad\quad\quad\quad F[i,j-1] + C,$
$\quad\quad\quad\quad\quad\quad F[i-1,j-1] + C \times isdiff(S1[i], S2[j]))$
14. **WeightedEdit** $= F[L1, L2]$

Advantage
- Obvious

Disadvantage
- Not easy to include detailed comments

Foramlisms

❖ prosecode

WeightedEdit(s,t) compares two strings s and t, of lengths $k(s)$ and $k(t)$ respectively, to determine their edit distance—the minimum cost in insertions, deletions, and replacements required to convert one into the other. These costs are weighted so that errors near the start of the strings attract a higher penalty than errors near the end.

We denote the ith character of string s by s_i. The principal internal data structure is a 2-dimensional array F in which the dimensions have ranges 0 to $k(s)$ and 0 to $k(t)$, respectively. When the array is filled, $F_{i,j}$ is the minimum edit distance between the strings $s_1 \cdots s_i$ and $t_1 \cdots t_j$; and $F_{k(s),k(t)}$ is the minimum edit distance between s and t.

The value p is the maximum penalty, set to $2(k(s)+k(t))$, and the penalty for a discrepancy between positions i and j of s and t, respectively, is $p-i-j$, so that the minimum penalty is $p-k(s)-k(t) = p/2$ and the next-smallest penalty is $p/2+1$. Two errors, wherever they occur, will outweigh one.

1. (Set penalty.) Set $p \leftarrow 2 \times (k(s)+k(t))$.
2. (Initialize data structure.) The boundaries of array F are initialized with the penalty for deletions at start of string; for example, $F_{i,0}$ is the penalty for deleting i characters from the start of s.
   (a) Set $F_{0,0} \leftarrow 0$.
   (b) For each position i in s, set $F_{i,0} \leftarrow F_{i-1,0} + p - i$.
   (c) For each position j in t, set $F_{0,j} \leftarrow F_{0,j-1} + p - j$.
3. (Compute edit distance.) For each position i in s and position j in t:
   (a) The penalty is $C = p - i - j$.
   (b) The cost of inserting a character into t (equivalently, deleting from s) is $I = F_{i-1,j} + C$.
   (c) The cost of deleting a character from t is $D = F_{i,j-1} + C$.
   (d) If s_i is identical to t_j, the replacement cost is $R = F_{i-1,j-1}$. Otherwise, the replacement cost is $R = F_{i-1,j-1} + C$.
   (e) Set $F_{i,j} \leftarrow \min(I, D, R)$.
4. (Return.) Return $F_{k(s),k(t)}$.

Advantage
- The specification of the algorithm is direct and clear

Disadvantage
- Only effective when the concepts underlying the algorithm have been discussed before the algorithm is given

Foramlisms

❖ Literate code

WeightedEdit(s,t) compares two strings s and t, of lengths $k(s)$ and $k(t)$ respectively, to determine their edit distance—the minimum cost in insertions, deletions, and replacements required to convert one into the other. These costs are weighted so that errors near the start of the strings attract a higher penalty than errors near the end.

The major steps of the algorithm are as follows.

1. Set the penalty.
2. Initialize the data structure.
3. Compute the edit distance.

We now examine these steps in detail.

1. Set the penalty.

   The main property that we require of the penalty scheme is that costs reduce smoothly from start to end of string. As we will see, the algorithm proceeds by comparing each position i in s to each position j in t. Thus a diminishing penalty can be computed with the expression $p-i-j$, where p is the maximum penalty. By setting the penalty thus

   (a) Set $p \leftarrow 2 \times (k(s)+k(t))$

   the minimum penalty is $p-k(s)-k(t) = p/2$ and the next-smallest penalty is $p/2+1$. This means that two errors—regardless of position in the strings—will outweigh one.

2. Initialize data structures ...

Formalism

- Regardless of the presentation form chosen, you need to consider the extent to which the algorithms, or its components, can be presented as mathematical abstractions
- To understand pseudocode, a reader must reinterpret the sequence of statements as a higher-level abstraction; the algorithm should be presented at such a level

Level of Detail

- Algorithm should be specified in sufficient detail to allow them to be implemented without undue inventiveness
- Don't provide too much detail

3. (Summation.) $sum \leftarrow 0$. For each j, where $1 \leq j \leq n$,
   (a) $c \leftarrow 1$; the variable c is a temporary accumulator.
   (b) For each k, where $1 \leq k \leq m$, set $c \leftarrow c \times A_{jk}$.
   (c) $sum \leftarrow sum + c$.

- Use text rather than mathematics if the former is sufficiently clear

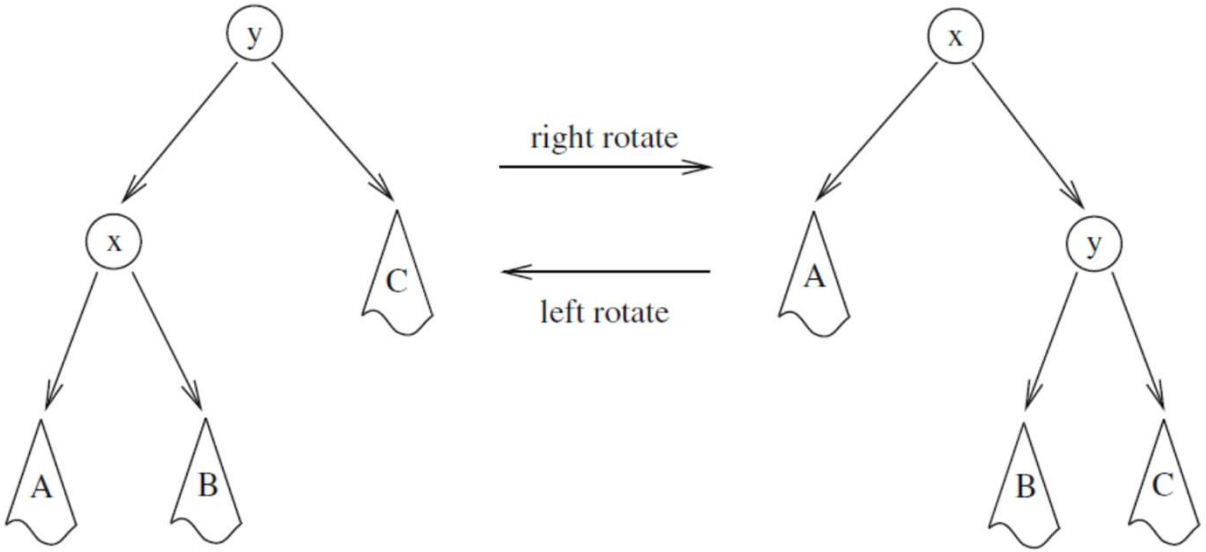
✗ 2. for $1 \leq i \leq |s|$
   (a) set $c \leftarrow s[i]$
   (b) set $A_c \leftarrow A_c + 1$

✓ 2. For each character c in string s, increment A_c.

Figures

- An effective way of conveying the intricacies of data structure and even quite simple structure can require complex de

✓ A single rotation can be used to bring a node one level closer to the root. In a left-rotation, a parent node x and its right child y are exchanged as follows: as B is the left child of y, assign B to be the new right child of x and assign x to be the new left child of y. The left child of x and the right child of y remain unchanged. The complementary operation is a right-rotation. Left- and right-rotations are shown in the following diagram.



Notation

- Mathematical notation is preferable to programming notation of presentation of algorithms
  - Use "x_i" rather than "x[i]"
  - Don't use "*" or "x" to denote multiplication
  - Avoid using constructs from specific programming language such as ==, a=b=0, a++, and for (i=0; i<n; i++)
  - Block-bounding statements such as begin and end are usually unnecessary
  - Nesting can be shown by indentation or by the numbering style
  - Take care with variable names of more than one character – don't use "pg" if it might be interpreted as "p x q"

Environment of Algorithms

- The steps that comprise an algorithm are only part of its description. The other part is its environment: data structure on which it operates, input and output data types, and, in some cases, factors such as properties of the underlying operating system and hardware
- Specify the types of all variables, other than trivial items such as counters.
- Describes expected input and out, including assumption about the correctness of the input
- State any limitation of the algorithm
- Discuss possible errors that are not explicitly captured by the algorithm

Environment of Algorithms

- **Describe data structures carefully**

  ✓ Each element is a triple

  ```
  (string, length, positions)
  ```

  in which `positions` is a sequence of byte offsets at which `string` has been observed.

- **Be consistent**
  - When presenting several algorithms for the same task, they should as fare as possible be defined over the same input and output

Asymptotic Cost

- Performance of algorithm is often measured by asympototic analysis
- For algorithms that operate on static data structure, it may be appropriate to consider the cost of creating that data structurre


Q
&
A