Theory of Computation

Homework 1

The Baker-Bird Two-Dimensional Pattern Matching

Ilyong Cho (2009-20899)

1 Introduction

Two-dimensional pattern matching problem is to find all occurences of the pattern P in the text T. when the pattern P and the text T is two-dimensional rectangluar array. We can see such a problem in some method to find an object in a digital picture. and we can find it in the detection of special conditions in a middle of a board game, such as "go" and "chess".

Two-dimensional pattern matching problem can be defined more formally as follow.

> If P is a u by v rectangular array of elements of alphabet $\sum$ and T is a m by n array of the same type, the problem is to find all pairs (i, j) such that
>
> $$S[i-u+k, j-v+l] = P[k, l]$$
>
> for all k and l such that $1 \leq k \leq u$ and $1 \leq l \leq v$. [1]

In this homework, I had implemented the BAKER-BIRD to find two-dimensional pattern matching. and I had to implement the AHO-CORASICK and the KNUTH-MORRIS-PRATT algorithm to implement the BAKER-BIRD . The specifications of this homework are as follows.

- $\sum = \{a, b, c, \ldots, z\}$.
- The size of the pattern P is $m \times m$.
- The size of the text T is $n \times n$.
- $1 \leq m \leq n \leq 100$.
- Use extra space $O(|\sum|m^2 + n)$.

The last specification requres interleaving the AHO-CORASICK and the KNUTH-MORRIS-PRATT in the column-matching step in the BAKER-BIRD .

2 The Baker-Bird Algorithm

2.1 The Baker-Bird Algorithm

Input: $m \times m$ pattern array P and $n \times n$ text array T.
Output: A list of all positions (i, j) where P "matches" T.

Phase 1: Preprocessing pattern array P

1. Regard each row of pattern P as independent patterns and preprocess the patterns to build the Aho-Corasick data structure
2. Identify the distinct rows of P and assign each a unique index. Let the distinct rows be $X_1, \ldots, X_q$.
3. Represent P as follow

$$P = \begin{pmatrix} X_p(1) \\ \vdots \\ X_p(m) \end{pmatrix}$$

where $p(1) \ldots p(m)$ in $\{1, \ldots, q\}^*$.

Phase 2: The row matching step

Compute the following array R using Aho-Corasick .

$$R = \begin{pmatrix} R_{1,1} & \cdots & R_{1,n} \\ \vdots & & \vdots \\ R_{n,1} & \cdots & R_{n,n} \end{pmatrix}$$

in $(\{1, \ldots, q\}^*)^*$.

defined by $R_{i,j} = k$ if and only if X_k matches $T_{i,j-m+1} \cdots T_{i,j}$, else $R_{i,j} = 0$

Phase 3: The column matching step

Compute the following array S using Knuth-Morris-Pratt .

$$S = \begin{pmatrix} S_{1,1} & \cdots & S_{1,n} \\ \vdots & & \vdots \\ S_{n,1} & \cdots & S_{n,n} \end{pmatrix}$$

in $(\{0, 1\}^*)^*$.

defined by $S_{i,j} = 1$ if and only if $p(1) \cdots p(m)$ matches $S_{i-m+1,j} \cdots S_{i,j}$, else $S_{i,j} = 0$. [1]

2.2 Time Complexity of The Baker-Bird Algorithm

The total time complexity of the BAKER-BIRD is $O(|\sum|m^2+n^2)$. When it use array representation for the Aho-Corasick data structure.

- Build the Aho-Corasick data structure in ***phase 1*** - $O(|\sum|m^2)$.
- Row matching(construct R) in ***phase 2*** - $O(n^2)$.
- KMP preprocessing in ***phase 3*** - $O(m)$.
- Column matching(construct S) in ***phase 3*** - $O(n^2)$.

2.3 Space Complexity of The Baker-Bird Algorithm

The total space complexity of the BAKER-BIRD is $O(|\sum|m^2 + n)$. When it use 2D array representation for the AHO-CORASICK and interleaves ***phase 2*** and ***phase 3***. Interleaving means that at each time a row of R is computed, runs n KNUTH-MORRIS-PRATT one for each column of R. So we only need $O(n)$ extra space to maintain the table R.

- $O(|\sum|m^2)$ for Aho-Corasick data structure.
- $O(m)$ for KNUTH-MORRIS-PRATT .
- $O(n)$ for R.

3 Implementation

3.1 Implementation of the Aho-Corasick

The AHO-CORASICK is implemented in the class **AhoCorasick**. 2D array $g(s, x)$ for each state s and character c is used to maintain state. Because each state needs $O(|\sum|)$ space and there are at most $O(m^2)$ states, totally $O(|\sum|m^2)$ space is needed. The 2D array representation yeilds branching time in $O(1)$.

There are three important methods in the class.

SCAN
- Preprocess pattern.
- Make transfer table $g(s, x)$.
- Calling MAKEFAILUREFUNCTION to make failure function and output.
- Return identifiers for each row of the input pattern. If two or more row is the same, then they have the same identifier.
- Time complexity - $O(|\sum|m^2)$ including MAKEFAILUREFUNCTION.

MAKEFAILUREFUNCTION
- Make failure function and output data structure.
- Do breadth-first-search to compute failure function and output table.
- Time complexity - $O(|\sum|m^2)$

SEARCH
- Search text to find pattern matching.
- Return list of pairs, each pair comprise the identifire of matching pattern and the matched index.
- Time complexity - $O(n)$
- This function is called $O(n)$ times during entire execution of the BAKER-BIRD . So the total time complexity is $O(n^2)$.

Note that m is not the sum of all rows of a pattern, but the length of a row in a pattern. Thus total number of characters in 2D pattern is m^2.

3.2 Implementation of the Knuth-Morris-Pratt

The KNUTH-MORRIS-PRATT is implemented in the class **KMP**. To interleave with the AHO-CORASICK in the BAKER-BIRD , I modified the KNUTH-MORRIS-PRATT slightly, so that can process pattern matching step by step.

There are two important methods in the class.

PRIFIX
- Preprocess pattern.
- Time complexity - $O(m)$.

KMPSEARCHSTEP
- Process KNUTH-MORRIS-PRATT search only one step.

• Get current state and chracter from function parameter and return the next state.
• If the next state is matching state, then return matching flag together.
• Time complexity - $O(1)$.
• To find all occurrences of the pattern matching, This function must be called $O(n)$ time. so the total time complexity to find all the pattern matching is $O(n)$.

3.3 Implementation of the Baker-Bird

The Baker-Bird is implemented in the class **BakerBird**. It finds all occurrences of the 2D pattern matching in $O(|\sum|m^2 + n^2)$ time and uses $O(|\sum|m^2 + n)$ space. The class **BakerBird** has **AhoCorasick** and **KMP** as aggregation. Aho-Corasick is used in the row matching step and Knuth-Morris-Pratt is used in the column matching step.

There are two important methods in the class.

SetPattern
• Preprocess pattern and prepare all the structures for Baker-Bird .
• To prepare Aho-Corasick data structure, call AhoCorasick::Scan().
• To prepare KMP data structure, call KMP::Prefix().
• Time complexity - $O(|\sum|m^2) + O(m) = O(|\sum|m^2)$.
• Total space for data structures for Aho-Corasick and Knuth-Morris-Pratt is $O(|\sum|m^2) + O(m) = O(|\sum|m^2)$.

TwoDimensionalMatching
• Do row matching and column matching interleaving.
• Compute a row of R using Aho-Corasick . $O(n)$ space and $O(n)$ time is needed to compute and save a row of R. Thus total time complexity for row matching is $O(n^2)$.
• Maintain the state vector for n interleaved Knuth-Morris-Pratt . It require $O(n)$ space.
• If one of the n Knuth-Morris-Pratt returns matching flag. It means that a column matching has been made.
• Because only current row of the R and the current states of the set of the n Knuth-Morris-Pratt are needed to interleave row and column matching, We only need totally $O(n)$ space to maintain the R and states of the set of the n Knuth-Morris-Pratt .
• Time complexity - $O(n^2)$ ($O(n)$ for each row and there are n rows).

Together SetPattern and TwoDimensionalMatching, Total time comlexity of the Baker-Bird is $O(|\sum|m^2 + n^2)$ and space complexity is $O(|\sum|m^2 + n)$.

3.4 Directories and Files

./	
AC	Aho-Corasick
AhoCorasick.h	Aho-Corasick for general purpose
AhoCorasick.cpp	
AhoCorasickA.h	Aho-Corasick for English alphabet
AhoCorasickA.cpp	
CheckAC.cpp	Aho-Corasick check program
Makefile	Makefile for Aho-Corasick
KMP	Knuth-Morris-Pratt
KMP.h	KMP algorithm
KMP.cpp	
CheckKMP.cpp	KMP check program
Makefile	Makefile for KMP
BB	Baker-Bird
BakerBird.h	Baker-Bird algorithm
BakerBird.cpp	
CheckBakerBird.cpp	Baker-Bird check program
Makefile	Makefile for Baker-Bird
BakerBird.cpp	Main program for Homework
Checker.cpp	Cheker program
RandomSetMaker.py	Random test set generator(Python)
Makefile	Makefile for all

4 Checker Program

The checker program is implemented in the source file named "Cheker.cpp". A naïve algorithm is used to verify result of BAKER-BIRD in the checker program. The checker program receives a pair of an input for an original two-dimensional matching problem and the output solved by the BAKER-BIRD . and verifies its result.

It prints "YES" if the result is correct, otherwise prints "NO".

```
procedure NAÏVE-2D-PATTERN-MATCHING(P, T, m, n)
    MatchingList ← ∅
    for i ← 0 to n − m do
        for j ← 0 to n − m do
            flag ← true
            for y ← 0 to m do
                for x ← 0 to m do
                    if T_{i+y,j+x} ≠ P_{y,x} then          ▷ Pattern is not matched
                        flag ← false
                    end if
                end for
            end for
            if flag is true then              ▷ A pattern matching has been found
                add (i + m − 1, j + m − 1) into MatchingList
            end if
        end for
    end for
    return MatchingList
end procedure
```

The time complexity of the naïve algorithm is $O(n^2m^2)$.

5 Experiment

5.1 Experimental Environment

- CPU: Intel® Core$^{\mathrm{TM}}$2 Duo CPU E6750 2.66GHz
- RAM: 3GB
- Network HDD
- OS: Linux kernel 2.6.9
- Complier: gcc 4.0.2

5.2 Data Format

5.2.1 Input

- The first line of the input file contains m and m.
- The following m lines contain a $m \times m$ pattern, one row in each line.
- The following n lines contain a $n \times n$ text, one row in each line.

5.2.2 Output

- The positions of occurrences in row major order
- Ordered by row first and column next.

5.3 Test Set

I made more than 20 test sets to verify correctness of Baker-Bird . 5 by hand and others by random generator program. The random generator program recieves m, n and the number of alphabets k from standard input to generate random pattern P and random text T and make a input file.

I had tested the Baker-Bird using those test sets and verified its result by the naïve-algorithm mentioned in the section 4.

The following is one of the handmade input and its output.

Input	Output
3 20	4 12
aba	4 18
aba	6 2
bab	8 11
ababababababababab	8 17
babababbabababababba	8 19
aaabababbbabababababab	10 5
abababbbababababababab	10 10
ababbabbbababaaababa	10 12
ababbabaababababababba	12 6
babababbaabaabbababa	13 2
bbababbababababababa	13 19
bbbababbababaaababab	15 8
bbbababbababababaababa	16 3
bbababaabababababababa	17 11
ababababababababbaba	17 13
ababbababababababababa	19 3
babbabababababababab	19 5
babababaababbabababababa	
babababababababaabbaba	
ababbababababababababa	
aabababbbbababbaabaa	
aabababbabababbabbbb	
bbabababababababbaba	

References

[1] Theodore P. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM Journal on Computing*, 7(4):533–541, 1978.