Theory of Computation: Homework 1

Author

- Name: 윤준혁
- Student ID: 2023-23475
- Date: 2025-04-07

Execution Environment

- Operating System: Ubuntu 22.04.3 LTS
- Compiler: g++ (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
- Compile option: `std=c++17`

Implementation of the Baker-Bird Algorithm

This section describes the structure, operation, and complexity of the key functions used in the implementation of the Baker-Bird two-dimensional pattern matching algorithm. The algorithm integrates the Aho–Corasick and Knuth–Morris–Pratt (KMP) algorithms, and it is optimized to respect the space complexity constraint of $O(|\Sigma|m2 + n)$ as specified in the assignment.

`char_to_index(char c)`

Converts a given character `c` into an integer index. Based on a fixed alphabet set $\Sigma = \{a–z,\ A–Z,\ 0–9\}$, each of the 62 characters is mapped to an integer in the range [0, 61], enabling constant-time branching in the trie structure.

- `'a'` to `'z'` → 0 to 25
- `'A'` to `'Z'` → 26 to 51
- `'0'` to `'9'` → 52 to 61

`insert_pat_row(const string &s, int r)`

Inserts a pattern row string `s` into the Aho–Corasick trie, assigning a unique label `r` to the terminal node. The index `r` denotes a unique ID for each distinct row in the pattern.

- Traverses the trie from the root for each character in `s`.
- Creates a new node if the transition does not exist.
- Marks the terminal node with `out[now] = r`.

`build_fail()`

Constructs the failure links for each node in the Aho–Corasick trie. These links define fallback behavior upon a mismatch, similar in principle to the failure function in the KMP algorithm.

- Uses BFS to traverse all trie nodes.

- Sets each failure link by following the parent's failure link and searching for the appropriate fallback.

`baker_bird(istream &in)`

This is the main function that executes the entire Baker-Bird algorithm. It reads the pattern and text from input, searches for all pattern occurrences, and returns their positions.

1. Pattern Row Preprocessing

Reads the $m \times m$ pattern matrix row by row. Duplicate rows are mapped to the same integer label `r`. An integer array `pat_col` is constructed from the pattern rows, representing the vertical sequence of row labels.

- Time complexity: $O(m^2)$
- Space complexity: $O(m^2)$

2. Aho–Corasick Preprocessing

Inserts each distinct row string into the trie using `insert_pat_row()` and builds the failure function using `build_fail()`.

- Time complexity: $O(|\Sigma| \cdot m^2)$
- Space complexity: $O(|\Sigma| \cdot m^2)$

3. KMP Preprocessing

Creates a prefix table (`pre_pat_col`) for the vertical pattern sequence `pat_col`, to be used in column-wise matching.

- Time complexity: $O(m)$
- Space complexity: $O(m)$

4. Pattern Matching

The text is read and processed one row at a time to reduce memory usage.

Each row is scanned using the Aho–Corasick trie to compute R values for each column.

For each column, the corresponding KMP state is updated dynamically.

When a full match is found, the top-left coordinate is recorded.

- Time complexity: $O(n^2)$
- Space complexity: $O(n)$

`main(int argc, char *argv[])`

Reads the input file path from command-line arguments, runs the Baker-Bird algorithm, and prints all matched positions to standard output.

- Time complexity: $O(|\Sigma|m^2 + n^2)$
- Space complexity: $O(|\Sigma|m^2 + n)$

Checker Program Implementation

The checker program verifies the correctness of the output produced by the Baker-Bird algorithm. It recomputes the answer independently using a brute-force sliding window approach and compares it to the given output.

Algorithm Description

The checker reads the pattern and text from the input file, then iterates over all possible $m \times m$ submatrices in the text to find exact matches.

Steps:

1. For every valid top-left coordinate (i, j) where:

$$0 \leq i \leq n - m, \quad 0 \leq j \leq n - m$$

2. Compare each character in the $m \times m$ region of the text to the pattern:

$$\text{text}[i + x][j + y] \stackrel{?}{=} \text{pattern}[x][y], \quad \forall 0 \leq x, y < m$$

3. If all characters match, store the coordinate (i, j) as a valid match.

Output Comparison

The checker reads the algorithm's output and extracts:

- Line 1: The number of matches `k`
- Next `k` lines: List of matched coordinates

It then compares:

1. Count: The number of coordinates must match.
2. Values: All coordinates must appear in the same order.

If both conditions are met, it prints `yes`. Otherwise, it prints `no`.

Complexity Analysis

- Time complexity:

  For each of the $O(n^2)$ candidate positions, compare $O(m^2)$ characters

$$O(n^2 m^2)$$

- Space complexity:

$$O(m^2 + n^2 + k) = O(m^2 + n^2)$$

  where `k` is the number of matches (at most $O(n^2)$).

Summary

- Implemented the Baker-Bird algorithm using Aho–Corasick (for row matching) and KMP (for column matching).

- Respected space constraints of $O(|\Sigma|m^2 + n)$.
- Developed a brute-force checker for correctness verification.
- Provided time and space complexity analysis for both the main algorithm and the checker.

Example running 1

Input

```
3 8
dyy
vyz
tUG
85NtDdyy
tt8iAvyz
uTuivtUG
GB5sZdyy
vGOqgvyz
pyPy1tUG
qZcZJwtu
IljzwIoU
```

Baker-Bird algorithm output

```
2
0 5
3 5
```

Checker program output

```
yes
```

Example running 2

Input

```
3 6
bzB
wv2
App
HMNbzB
na5wv2
bzBApp
wv2I7z
AppTdV
hTgnxH
```

Baker-Bird algorithm output

```
2
0 3
2 0
```

Checker program output

```
yes
```

Example running 3

Input

```
1 7
T
XWbBDiT
gwSOj9J
T9cCfTT
T3T5TjC
TTTTTTT
YDTIUT7
T7LTcTn
```

Baker-Bird algorithm output

```
19
0 6
2 0
2 5
2 6
3 0
3 2
3 4
4 0
4 1
4 2
4 3
4 4
4 5
4 6
5 2
5 5
6 0
6 3
6 5
```

Checker program output

```
yes
```

Example running 4

Input

```
2 10
FK
Mv
5HEIlyFKQY
FKFKFKMv1Y
MvMvMv3nFK
RvWyWRs7Mv
FKzBFKVidF
MvFKMvlFKY
2GMvTDXMvw
FKXI0yHxFK
MvFKFKCTMv
3MMvMvkLTZ
```

Baker-Bird algorithm output

```
13
0 6
1 0
1 2
1 4
2 8
4 0
4 4
5 2
5 7
7 0
7 8
8 2
8 4
```

Checker program output

```
yes
```