Derived Anagram Chain Algorithm

Problem Definition

Derived anagram: word B is a derived anagram of word A if B can be formed by adding exactly one letter to A and rearranging all letters.

Example chain: `abc → abck → abcek → abcelk → baclekt` (length 5)

Input: Dictionary file + start word
Output: All longest chains starting from the start word

Algorithm Overview
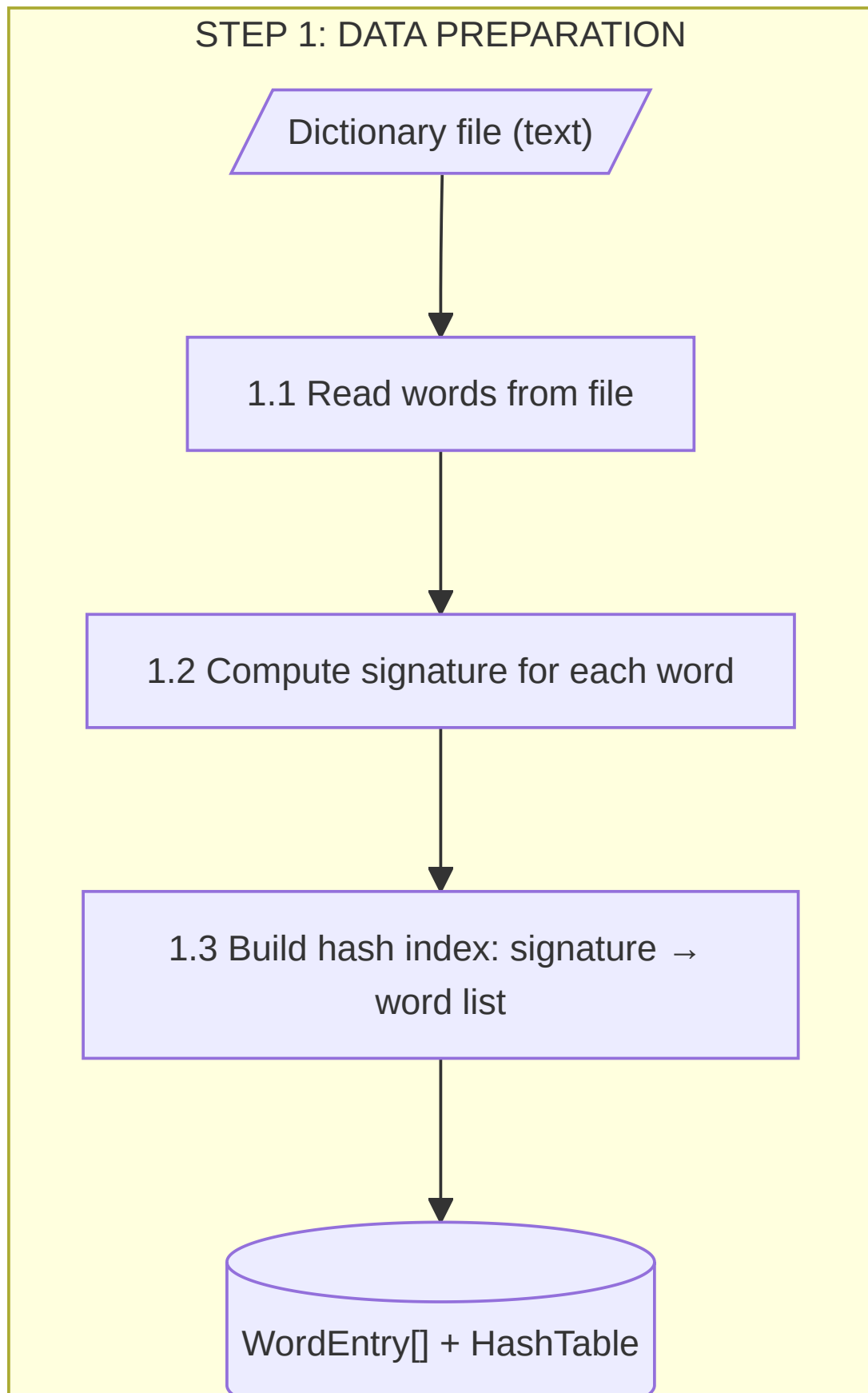
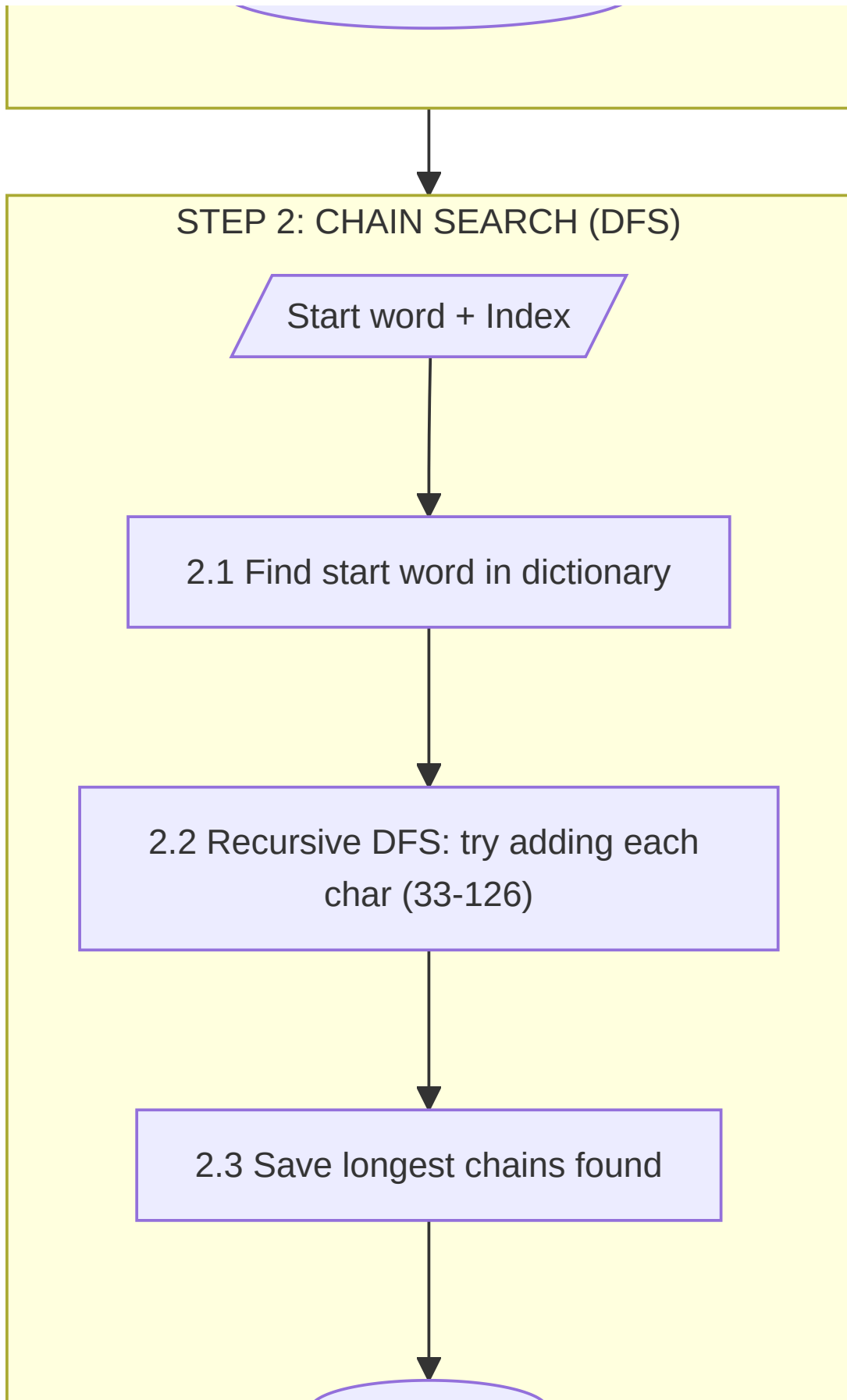## STEP 1: DATA PREPARATION

Dictionary file (text)

↓

1.1 Read words from file

↓

1.2 Compute signature for each word

↓

1.3 Build hash index: signature → word list

↓

WordEntry[] + HashTable

STEP 2: CHAIN SEARCH (DFS)

Start word + Index

2.1 Find start word in dictionary

2.2 Recursive DFS: try adding each char (33-126)

2.3 Save longest chains found

ChainResults

STEP 3: OUTPUT RESULTS

Found chains

3.1 Print chain count and max length

3.2 Print each chain:
word1 → word2 → word3
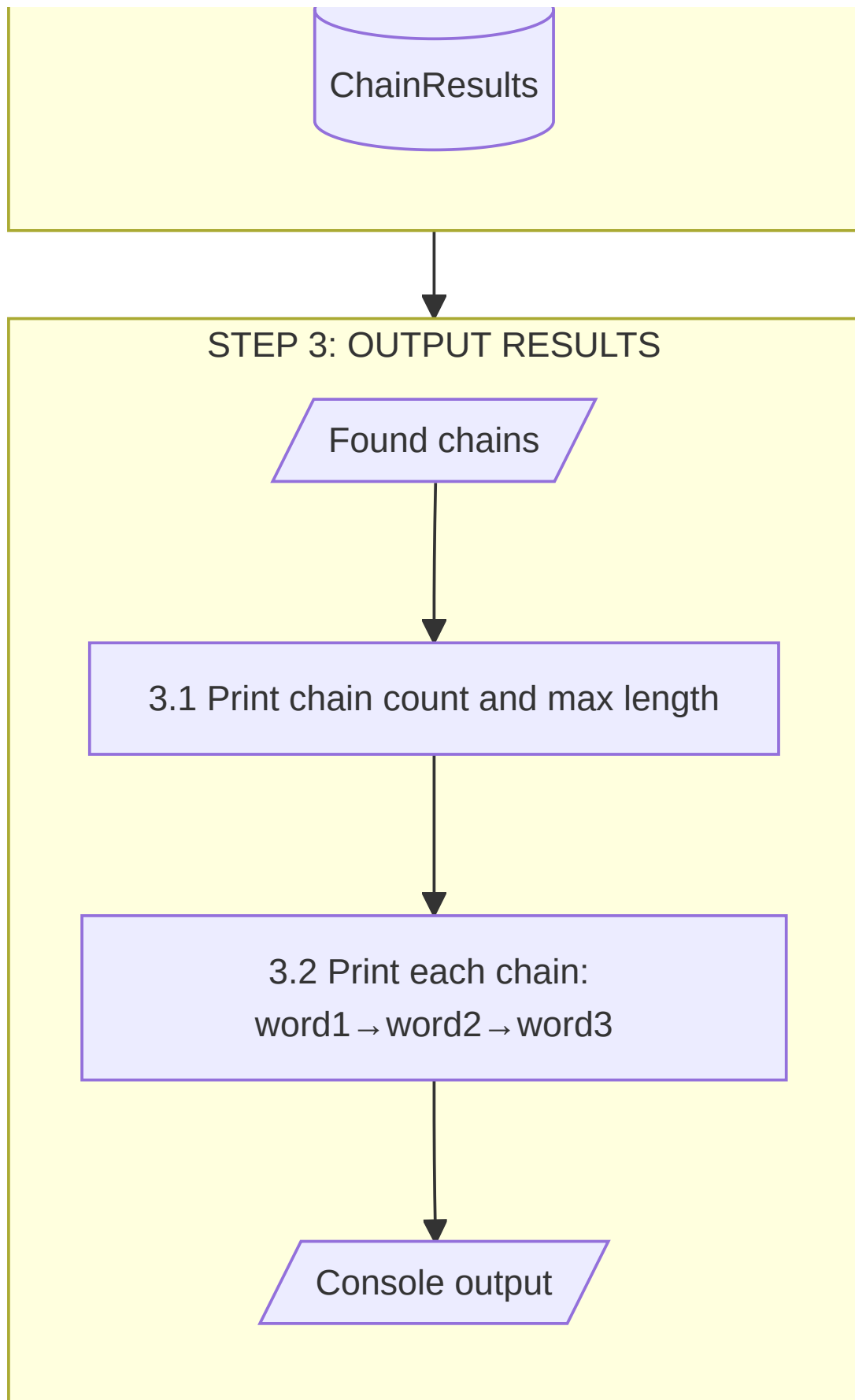
Console output

Step	Input	Process	Output
STEP 1	Dictionary file	Load → Signatures → Hash table	`entries[] + HashTable`
STEP 2	Index + start word	DFS search all paths	`ChainResults`
STEP 3	Found chains	Format and print	Console output

Data Structures
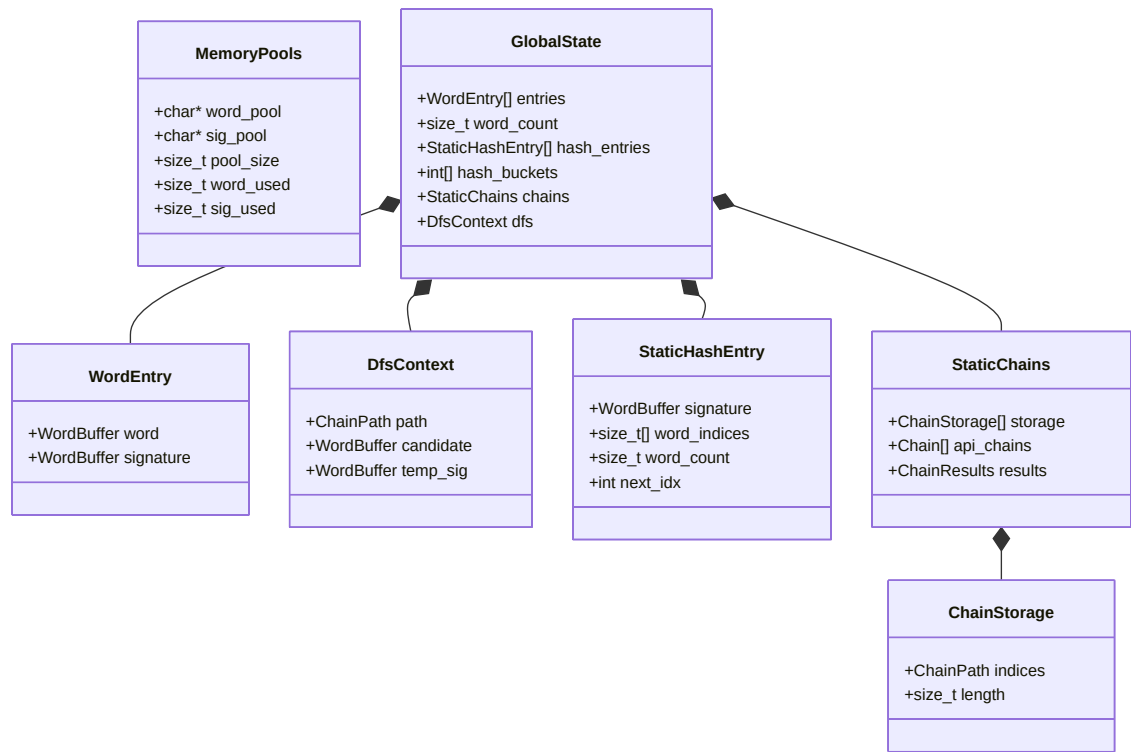
STEP 1: Data Preparation

1.1 Dictionary Loading

```
entries[0].word  = "abcdg"    entries[0].signature  = "abcdg"
entries[1].word  = "abcd"     entries[1].signature  = "abcd"
..
entries[8].word  = "bafced"   entries[8].signature  = "abcdef"   (sorted)
entries[9].word  = "akjpqwmn" entries[9].signature  = "ajkmnpqw" (sorted)
entries[10].word = "abcelk"   entries[10].signature = "abcekl"   (sorted)
entries[11].word = "baclekt"  entries[11].signature = "abceklt"  (sorted)
```

1.2 Signature Computation (Counting Sort)

```c
static void sort_chars(char *s, size_t len)
{
    int counts[CHAR_COUNT_SIZE] = {0};
    size_t i;
    size_t pos;
    int c;

    ASSERT_NOT_NULL(s);

    for (i = 0; i < len; i++)
    {
        ASSERT_MSG((unsigned char)s[i] < CHAR_COUNT_SIZE, "char out of range");
        counts[(unsigned char)s[i]]++;
    }

    pos = 0;
    for (c = 0; c < CHAR_COUNT_SIZE; c++)
    {
        while (counts[c]-- > 0)
        {
            s[pos++] = (char)c;
        }
    }
}
```
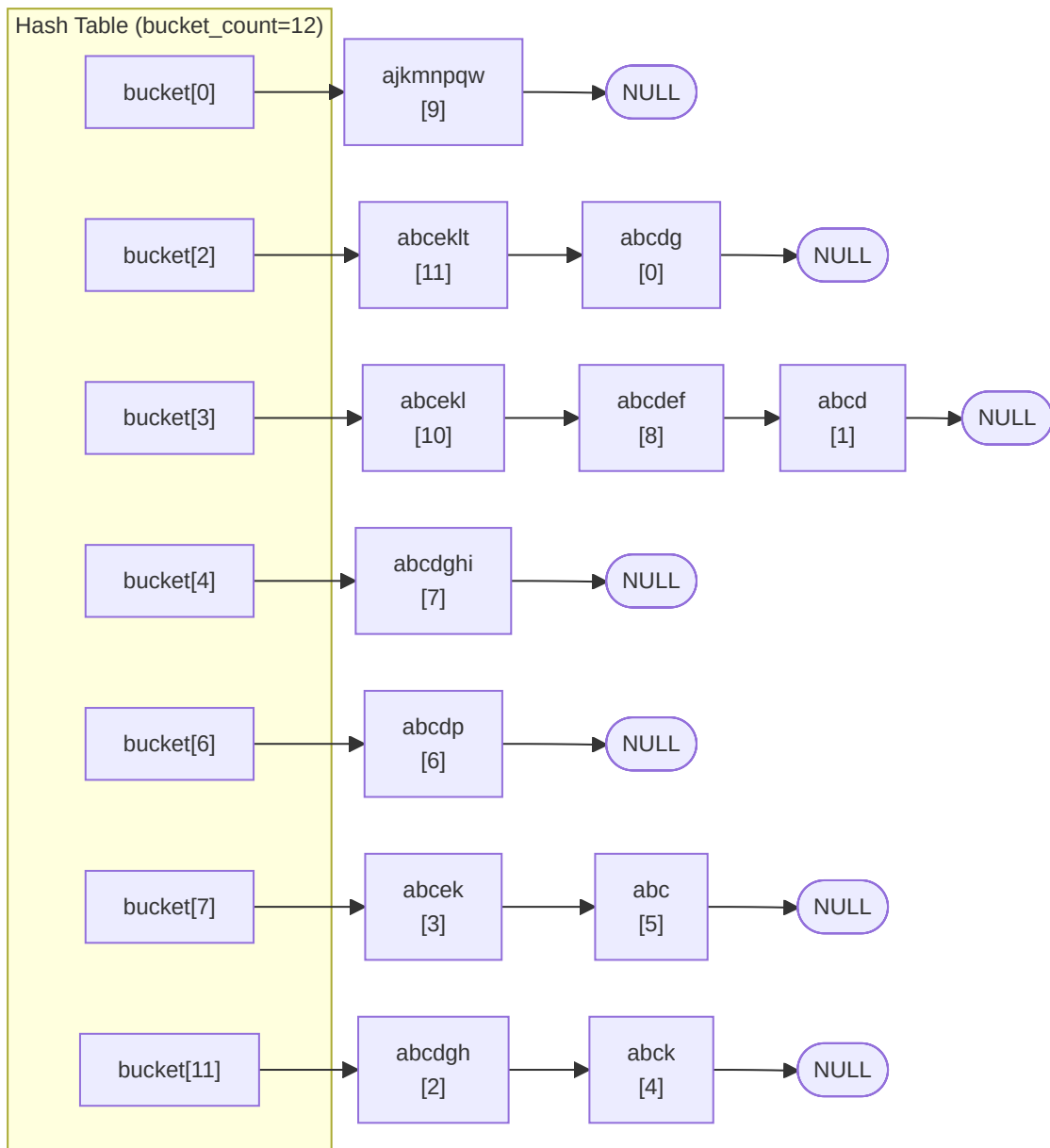
1.3 Hash Table

Hash Table (bucket_count=12)

```
bucket[0]  ──▶  ajkmnpqw [9]  ──▶  NULL

bucket[2]  ──▶  abceklt [11]  ──▶  abcdg [0]  ──▶  NULL

bucket[3]  ──▶  abcekl [10]  ──▶  abcdef [8]  ──▶  abcd [1]  ──▶  NULL

bucket[4]  ──▶  abcdghi [7]  ──▶  NULL

bucket[6]  ──▶  abcdp [6]  ──▶  NULL

bucket[7]  ──▶  abcek [3]  ──▶  abc [5]  ──▶  NULL

bucket[11] ──▶  abcdgh [2]  ──▶  abck [4]  ──▶  NULL
```

FNV-1a Hash Function:

```c
static unsigned long hash_fnv1a(const char *s)
{
    unsigned long h;

    ASSERT_NOT_NULL(s);

    h = FNV_OFFSET_BASIS;

    while (*s)
    {
        h ^= (unsigned char)*s++;
```
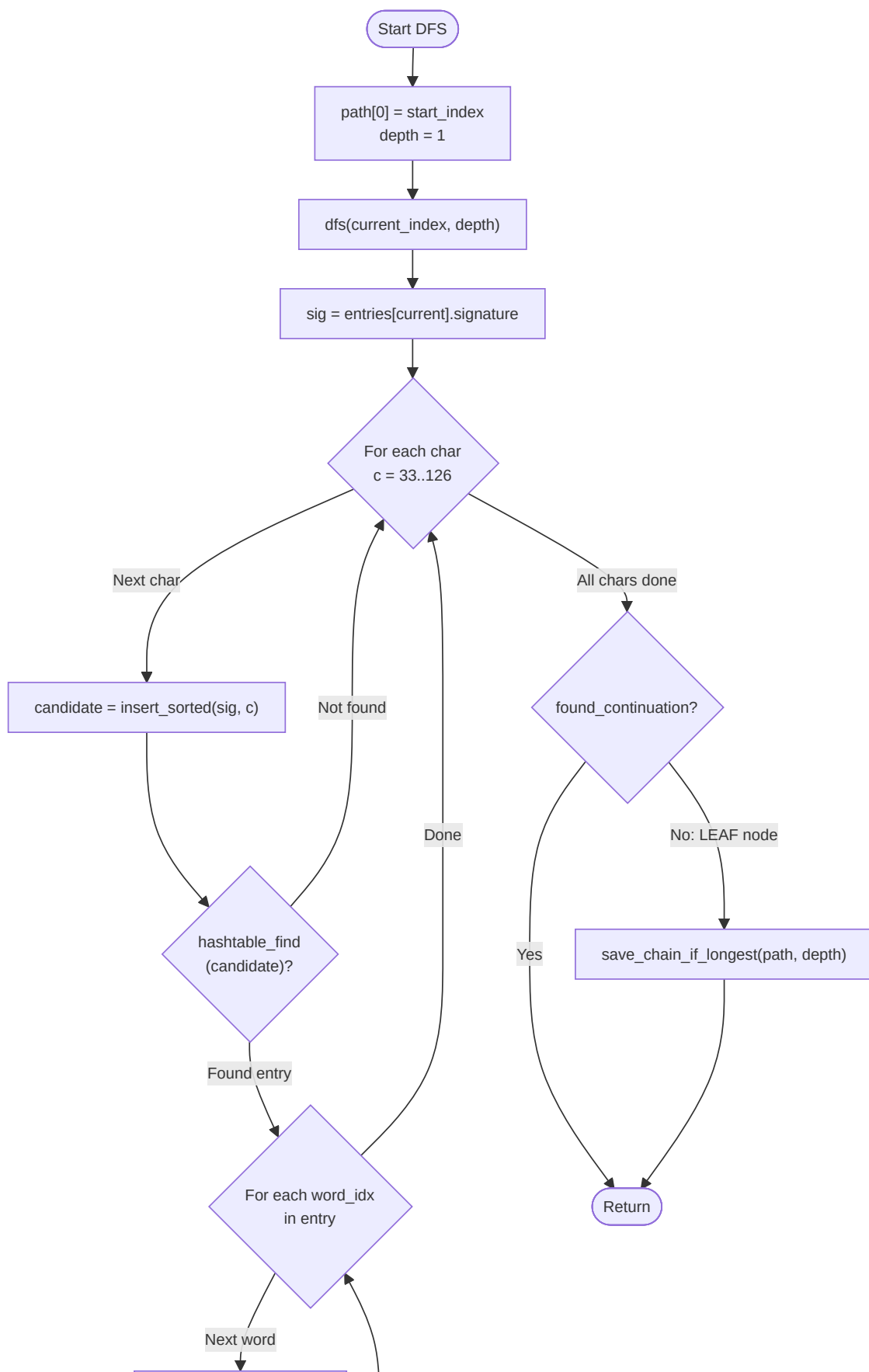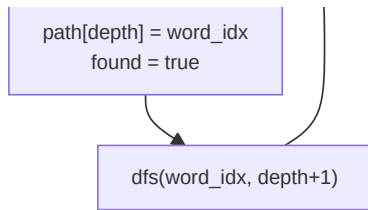
```
        h *= FNV_PRIME;
    }

    return h;
}
```

STEP 2: DFS Chain Search

```mermaid
flowchart TD
    Start([Start DFS])
    Start --> Init[path[0] = start_index<br>depth = 1]
    Init --> DFS[dfs(current_index, depth)]
    DFS --> Sig[sig = entries[current].signature]
    Sig --> ForChar{For each char<br>c = 33..126}
    ForChar -->|Next char| Candidate[candidate = insert_sorted(sig, c)]
    ForChar -->|All chars done| FoundCont{found_continuation?}
    Candidate --> HashFind{hashtable_find<br>(candidate)?}
    HashFind -->|Not found| ForChar
    HashFind -->|Found entry| ForWord{For each word_idx<br>in entry}
    ForWord -->|Done| ForChar
    ForWord -->|Next word| Next[ ]
    FoundCont -->|Yes| Return([Return])
    FoundCont -->|No: LEAF node| Save[save_chain_if_longest(path, depth)]
    Save --> Return
```

DFS Trace Example (start: "abck")

```
dfs(4, depth=1) sig="abck"
|
├─ char='e' → candidate="abcek" → FOUND [3]
|   └─ dfs(3, depth=2) sig="abcek"
|       |
|       └─ char='l' → candidate="abcekl" → FOUND [10]
|           └─ dfs(10, depth=3) sig="abcekl"
|               |
|               └─ char='t' → candidate="abceklt" → FOUND [11]
|                   └─ dfs(11, depth=4) sig="abceklt"
|                       |
|                       └─ No continuation found → LEAF
|                          Save chain: [4,3,10,11] length=4
|
└─ Result: "abck" → "abcek" → "abcelk" → "baclekt"
```

Memory Modes

Static Mode (Embedded)

```c
typedef struct {
    WordEntry entries[POOL_MAX_WORDS];
    StaticHashEntry hash_entries[POOL_MAX_HASH_ENTRIES];
    int hash_buckets[POOL_HASH_BUCKETS];
    StaticChains chains;
    DfsContext dfs;
} GlobalState;
```

Dynamic Mode (PC)

```c
typedef struct {
    MemoryPools pools;
    Dictionary *dict;
    HashTable *ht;
    ChainResults *results;
    DfsContext dfs;
} GlobalState;
```

Implementation Files

File	Purpose
anagram_chain_core.c	DFS, hash table, dictionary
anagram_chain_io.c	I/O, timers, validation
config.h	Memory pool constants
global.h	Type definitions
trace.h	Debug tracing
assert.h	Runtime checks
anagram_chain.h	Public API