DATA STRUCTURE

DAY-7

1.Program for Trie

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define ALPHABET_SIZE 26
 typedef struct TrieNode {
   struct TrieNode* children[ALPHABET_SIZE];
   bool isEndOfWord;
} TrieNode;
 TrieNode* createNode() {
   TrieNode* newNode = (TrieNode*)malloc(sizeof(TrieNode));
   if (newNode) {
      for (int i = 0; i < ALPHABET_SIZE; i++) {
         newNode->children[i] = NULL;
      }
      newNode->isEndOfWord = false;
   }
   return newNode;
}
 void insert(TrieNode* root, const char* word) {
   TrieNode* current = root;
   while (*word) {
      int index = *word - 'a'; // Assuming only lowercase letters
      if (current->children[index] == NULL) {
```

```
            current->children[index] = createNode();
        }
        current = current->children[index];
        word++;
    }
    current->isEndOfWord = true;
}
bool search(TrieNode* root, const char* word) {
    TrieNode* current = root;
    while (*word) {
        int index = *word - 'a'; // Assuming only lowercase letters
        if (current->children[index] == NULL) {
            return false;
        }
        current = current->children[index];
        word++;
    }
    return current != NULL && current->isEndOfWord;
}
 void deleteTrie(TrieNode* root) {
    if (root == NULL) return;
    for (int i = 0; i < ALPHABET_SIZE; i++) {
        if (root->children[i] != NULL) {
            deleteTrie(root->children[i]);
        }
    }
    free(root);
}
```

```
int main() {
    TrieNode* root = createNode();
    insert(root, "hello");
    insert(root, "world");
    insert(root, "trie");
    printf("Searching for 'hello': %s\n", search(root, "hello") ? "Found" : "Not Found");
    printf("Searching for 'world': %s\n", search(root, "world") ? "Found" : "Not Found");
    printf("Searching for 'trie': %s\n", search(root, "trie") ? "Found" : "Not Found");
    printf("Searching for 'test': %s\n", search(root, "test") ? "Found" : "Not Found");
    deleteTrie(root);
    return 0;
}
```

Output:

```
Searching for 'hello': Found
Searching for 'world': Found
Searching for 'trie': Found
Searching for 'test': Not Found
```

2.Program for 2-3 Trie

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define ALPHABET_SIZE 26
#define MAX_CHILDREN 3
typedef struct TrieNode {
    struct TrieNode* children[MAX_CHILDREN][ALPHABET_SIZE];
    bool isEndOfWord;
```

```
    int numChildren;
} TrieNode;
 TrieNode* createNode() {
    TrieNode* node = (TrieNode*)malloc(sizeof(TrieNode));
    for (int i = 0; i < MAX_CHILDREN; i++) {
        for (int j = 0; j < ALPHABET_SIZE; j++) {
            node->children[i][j] = NULL;
        }
    }
    node->isEndOfWord = false;
    node->numChildren = 0;
    return node;
}
 void insert(TrieNode* root, const char* word) {
    TrieNode* node = root;
    while (*word) {
        int index = *word - 'a';
        if (node->numChildren < MAX_CHILDREN) {
            if (node->children[node->numChildren][index] == NULL) {
                node->children[node->numChildren][index] = createNode();
            }
            node = node->children[node->numChildren][index];
            node->numChildren++;
        } else {
            // Handle node with maximum children (not implemented here)
        }
        word++;
    }
```

```
    node->isEndOfWord = true;
}
 bool search(TrieNode* root, const char* word) {
    TrieNode* node = root;
    while (*word) {
        int index = *word - 'a';
        bool found = false;
        for (int i = 0; i < node->numChildren; i++) {
            if (node->children[i][index] != NULL) {
                node = node->children[i][index];
                found = true;
                break;
            }
        }
        if (!found) return false;
        word++;
    }
    return node->isEndOfWord;
}
void deleteTrie(TrieNode* root) {
    if (root == NULL) return;
    for (int i = 0; i < MAX_CHILDREN; i++) {
        for (int j = 0; j < ALPHABET_SIZE; j++) {
            if (root->children[i][j] != NULL) {
                deleteTrie(root->children[i][j]);
            }
        }
    }
```

```
    free(root);
}
 int main() {
    TrieNode* root = createNode();
    insert(root, "hello");
    insert(root, "world");
    printf("Searching for 'hello': %s\n", search(root, "hello") ? "Found" : "Not Found");
    printf("Searching for 'world': %s\n", search(root, "world") ? "Found" : "Not Found");
    printf("Searching for 'test': %s\n", search(root, "test") ? "Found" : "Not Found");

    deleteTrie(root);
    return 0;
}
```

Output:

Searching for 'hello': Not Found

Searching for 'world': Not Found

Searching for 'test': Not Found

3.Program for 2-3-4 Trie

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define ALPHABET_SIZE 26
#define MAX_CHILDREN 4
typedef struct TrieNode {
    struct TrieNode* children[MAX_CHILDREN][ALPHABET_SIZE];
    bool isEndOfWord;
```

```
    int numChildren;
} TrieNode;
 TrieNode* createNode() {
    TrieNode* node = (TrieNode*)malloc(sizeof(TrieNode));
    for (int i = 0; i < MAX_CHILDREN; i++) {
        for (int j = 0; j < ALPHABET_SIZE; j++) {
            node->children[i][j] = NULL;
        }
    }
    node->isEndOfWord = false;
    node->numChildren = 0;
    return node;
}
 void insert(TrieNode* root, const char* word) {
    TrieNode* node = root;
    while (*word) {
        int index = *word - 'a';
        bool found = false;
        for (int i = 0; i < node->numChildren; i++) {
            if (node->children[i][index] != NULL) {
                node = node->children[i][index];
                found = true;
                break;
            }
        }
        if (!found) {
            if (node->numChildren < MAX_CHILDREN) {
                node->children[node->numChildren][index] = createNode();
```

```
            node = node->children[node->numChildren][index];
            node->numChildren++;
        } else {
                printf("Node with maximum children reached; cannot insert '%s'.\n",
word);
            return;
        }
      }
      word++;
   }
   node->isEndOfWord = true;
}
 bool search(TrieNode* root, const char* word) {
   TrieNode* node = root;
   while (*word) {
      int index = *word - 'a';
      bool found = false;
      for (int i = 0; i < node->numChildren; i++) {
         if (node->children[i][index] != NULL) {
            node = node->children[i][index];
            found = true;
            break;
         }
      }
      if (!found) return false;
      word++;
   }
   return node->isEndOfWord;
```

```
}
 void deleteTrie(TrieNode* root) {
   if (root == NULL) return;
   for (int i = 0; i < MAX_CHILDREN; i++) {
      for (int j = 0; j < ALPHABET_SIZE; j++) {
         if (root->children[i][j] != NULL) {
            deleteTrie(root->children[i][j]);
         }
      }
   }
   free(root);
}
 int main() {
   TrieNode* root = createNode();
   insert(root, "hello");
   insert(root, "world");
   insert(root, "trie");
   printf("Searching for 'hello': %s\n", search(root, "hello") ? "Found" : "Not Found");
   printf("Searching for 'world': %s\n", search(root, "world") ? "Found" : "Not Found");
   printf("Searching for 'trie': %s\n", search(root, "trie") ? "Found" : "Not Found");
   printf("Searching for 'test': %s\n", search(root, "test") ? "Found" : "Not Found");
   deleteTrie(root);
   return 0;
}
```

Output:

```
Searching for 'hello': Not Found
Searching for 'world': Not Found
Searching for 'trie': Not Found
```

Searching for 'test': Not Found