TRIE

1. REPRESENTATION, SEARCH AND INSERT

```
#include <bits/stdc++.h>
using namespace std;

const int ALPHABET_SIZE = 26;


struct TrieNode
{
        struct TrieNode *children[ALPHABET_SIZE];


        bool isEndOfWord;
};


struct TrieNode *getNode(void)
{
        struct TrieNode *pNode = new TrieNode;

        pNode->isEndOfWord = false;

        for (int i = 0; i < ALPHABET_SIZE; i++)
                pNode->children[i] = NULL;

        return pNode;
}


void insert(struct TrieNode *root, string key)
```

```
{
        struct TrieNode *pCrawl = root;

        for (int i = 0; i < key.length(); i++)
        {
                int index = key[i] - 'a';
                if (!pCrawl->children[index])
                        pCrawl->children[index] = getNode();

                pCrawl = pCrawl->children[index];
        }


        pCrawl->isEndOfWord = true;
}


bool search(struct TrieNode *root, string key)
{
        struct TrieNode *pCrawl = root;

        for (int i = 0; i < key.length(); i++)
        {
                int index = key[i] - 'a';
                if (!pCrawl->children[index])
                        return false;

                pCrawl = pCrawl->children[index];
        }

        return (pCrawl != NULL && pCrawl->isEndOfWord);
}

int main()
{
```

```
        string keys[] = {"bad", "bat", "geeks", "geeks", "cat", "cut"};

        int n = sizeof(keys)/sizeof(keys[0]);

        struct TrieNode *root = getNode();


        for (int i = 0; i < n; i++)
                insert(root, keys[i]);


        search(root, "bat")? cout << "Yes\n" :
                                        cout << "No\n";
        search(root, "gee")? cout << "Yes\n" :
                                        cout << "No\n";
        return 0;
}
```

2. DELETE

```
#include <bits/stdc++.h>
using namespace std;

const int ALPHABET_SIZE = 26;


struct TrieNode
{
        struct TrieNode *children[ALPHABET_SIZE];


        bool isEndOfWord;
```

```
};


struct TrieNode *getNode(void)
{
        struct TrieNode *pNode = new TrieNode;

        pNode->isEndOfWord = false;

        for (int i = 0; i < ALPHABET_SIZE; i++)
                pNode->children[i] = NULL;

        return pNode;
}


void insert(struct TrieNode *root, string key)
{
        struct TrieNode *pCrawl = root;

        for (int i = 0; i < key.length(); i++)
        {
                int index = key[i] - 'a';
                if (!pCrawl->children[index])
                        pCrawl->children[index] = getNode();

                pCrawl = pCrawl->children[index];
        }


        pCrawl->isEndOfWord = true;
}


bool search(struct TrieNode *root, string key)
```

```
{
        struct TrieNode *pCrawl = root;

        for (int i = 0; i < key.length(); i++)
        {
                int index = key[i] - 'a';
                if (!pCrawl->children[index])
                        return false;

                pCrawl = pCrawl->children[index];
        }

        return (pCrawl != NULL && pCrawl->isEndOfWord);
}

bool isEmpty(TrieNode* root)
{
    for (int i = 0; i < ALPHABET_SIZE; i++)
        if (root->children[i])
            return false;
    return true;

}

TrieNode* remove(TrieNode* root, string key, int i)
{

    if (!root)
        return NULL;


    if (i == key.size()) {


        if (root->isEndOfWord)
```

```
            root->isEndOfWord = false;


        if (isEmpty(root)) {
            delete (root);
            root = NULL;
        }

        return root;
    }


    int index = key[i] - 'a';
    root->children[index] =
          remove(root->children[index], key, i + 1);


    if (isEmpty(root) && root->isEndOfWord == false) {
        delete (root);
        root = NULL;
    }

    return root;
}



int main()
{

         string keys[] = {"an", "and", "ant", "bad", "bat", "zoo"};

         int n = sizeof(keys)/sizeof(keys[0]);

         struct TrieNode *root = getNode();
```

```
        for (int i = 0; i < n; i++)
                insert(root, keys[i]);

        root = remove(root, "zoo", 0);

        search(root, "zoo")? cout << "zoo --- " << "Yes\n" :
                                        cout << "zoo --- " << "No\n";

        return 0;
}
```

3. COUNT DISTINCT ROWS IN A BINARY MATRIX

```
#include <bits/stdc++.h>
using namespace std;
#define ROW 4
#define COL 3


class Node
{
        public:
        bool isEndOfCol;
        Node *child[2];
} ;


Node* newNode()
{
        Node* temp = new Node();
        temp->isEndOfCol = 0;
        temp->child[0] = temp->child[1] = NULL;
```

```
        return temp;
}


bool insert(Node** root, int (*M)[COL],
                                int row, int col )
{

        if (*root == NULL)
                *root = newNode();



        if (col < COL)
                return insert (&((*root)->child[M[row][col]]),
                                                                        M, row, col
+ 1);

        else
        {

                if (!((*root)->isEndOfCol))
                        return (*root)->isEndOfCol = 1;



                return 0;
        }
}




int  countDis(int (*M)[COL])
{
        Node* root = NULL;
        int i;
```

```
    int cnt = 0;

        for (i = 0; i < ROW; ++i)
            if (insert(&root, M, i, 0))
                cnt++;

        return cnt;
}


int main()
{
        int M[ROW][COL] = {{1, 0, 0},
                {1, 1, 1},
                {1, 0, 0},
                {0, 1, 0}};

        cout << countDis(M);

        return 0;
}
```