# Tries

# Data Structure

# Lesson 4

## PHONE DIRECTORY

Given a list of contacts which exist in a phone directory and a query string str. The task is to implement search query for the phone directory. Run a search query for each prefix p of the query string str(i.e from index 1 to str length) that prints all the distinct recommended contacts which have the same prefix as our query (p) in lexicographical order. Please refer the explanation part for better understanding.

**Example:**

**Input:**

1

3

geeikistest geeksforgeeks geeksfortest

geeips

**Output:**

geeikistest geeksforgeeks geeksfortest

geeikistest geeksforgeeks geeksfortest

geeikistest geeksforgeeks geeksfortest

geeikistest

0

0

**ALGORITHM: create a trie,** insert all the strings in trie, search of all prefixes of the given string and print all the words with given prefixes from the trie.

Time complexity:O(N*max_word_length)

**CODE:**

```
struct node{
    struct node* children[26];
    bool eow;
    vector<string>v;
};

struct node* getnode(void)
{
    node *p=new node;
    for(int i=0;i<26;i++)
    {
        p->children[i]=NULL;
    }
    p->eow=false;
    return p;
}

void insert(struct node* root, string s)
{
        node *p=root;
```

```
for(int i=0;i<s.length();i++)
    {   int index=s[i]-'a';
        if(!p->children[index])
        {
            p->children[index]=getnode();
        }
        p=p->children[index];
    }
    p->eow=true;
    (p->v).push_back(s);
}
 void printwords(struct node *root)
 {
    if(root->eow)
     {
         cout<<root->v[0]<<" ";
     }
     for(int j=0;j<26;j++)
     {
         node *child=root->children[j];
    if(child)
          printwords(child);
     }
 }
 bool search(struct node *root,string s)
 {  node *p=root;
```

```
    for(int i=0;i<s.length();i++)
    {
        int index=s[i]-'a';
        if(!p->children[index])
     return false;
        p=p->children[index];
    }
    printwords(p);
    return true;
}
void phonedirectory(string arr[],int n, string s)
{  node *root=getnode();
    for(int i=0;i<n;i++)
    {
   insert(root,arr[i]);
    }
    string a="";
    for(int i=0;i<s.length();i++)
    {
        a+=s[i];
        if(!search(root,a))
          cout<<"0";
        cout<<endl;
    }
}
```

# MOST FREQUENT WORD IN ARRAY OF STRINGS

**Given an array arr containing N words consisting of lowercase characters. Your task is to find the most frequent word in the array. If multiple words have same frequency, then print the word whose first occurence occurs last in the array as compared to the other strings with same frequency.**

**Example 1:**

**Input:**
**N = 3**
**arr[] = {geeks,for,geeks}**
**Output: geeks**
**Explanation: "geeks" comes 2 times.**

**ALGORITHM:**

**BRUTE FORCE:** A simple solution is to run two loops and count occurrences of every word. Time complexity of this solution is O(n * n * MAX_WORD_LEN).

**Hashing approach:**

Traverse the array , keep track of word and it's frequency. Next step includes iterate over it and find out the word with maximum frequency.

**CODE:**

```
string mostFrequentWord(string arr[], int n) {
unordered_map<string,int>s;
for(int i=0;i<n;i++)
{
    s[arr[i]]++;
}
string ans;
int max=0;
for(int i=0;i<n;i++)
{
    if(s[arr[i]]>=max && s[arr[i]]!=-1)
    {
        max=s[arr[i]];
        ans=arr[i];
        s[arr[i]]=-1;
    }
}
return ans;
}
```

**USING TRIES:**

**An efficient solution** is to use tries ,insert all the strings in tries. Take a pair for storing frequency of strings and index of last ocurrence of string of max frequency.

Time Complexity: O(n * MAX_WORD_LEN)

```
struct node{
    struct node *children[26];
    bool eow;
    pair<int,int>compare;
```

```
};

struct node* getnode(void)
{
  node *p=new node;
     for(int i=0;i<26;i++)
  {
         p->children[i]=NULL;
     }
     p->eow=false;
     p->compare={0,-1};
     return p;
   }
   bool insert(struct node* root, string s,int a,pair<int,int>&x)
   {
     node *p=root;
     for(int i=0;i<s.length();i++)
     {
         int index=s[i]-'a';
         if(!p->children[index])
         {
             p->children[index]=getnode();
         }
         p=p->children[index];
   }
   p->eow=true;
   p->compare.first++;
if(p->compare.second==-1)
   p->compare.second=a;
 if(x.first==p->compare.first)
 {
     if(p->compare.second>x.second)
     {
         x=p->compare;
         return true;
     }
     return false;
 }
 if(p->compare.first>x.first)
 {
     x=p->compare;
     return true;
 }
 return false;
}

string mostFrequentWord(string arr[], int n) {
   node *root=getnode();
```

```
    pair<int,int>x={0,-1};
    string ans="";
    for(int i=0;i<n;i++)
     {
         if(insert(root,arr[i],i,x))
           ans=arr[i];
     }
     return ans;
}
```

## PALINDROME PAIRS

Given an array of strings arr[] of size N, find if there exists 2 strings arr[i] and arr[j] such that arr[i]+arr[j] is a palindrome i.e the concatenation of string arr[i] and arr[j] results into a palindrome.

Example 1:

```
Input:
N = 6
arr[] = {"geekf", "geeks", "or","keeg", "abc", "bc"}
Output: 1
Explanation: There is a pair "geekf"
and "keeg".
```

## APPROACH:

**SIMPLE SOLUTION:**

Consider each pair one by one, check if any of the pairs forms a palindrome after concatenating them. Return true, if any such pair exists. Else, return false.

Time Complexity :O(N^2*K) ,Here n is the number of the words in the list and k is the maximum length that is checked for a palindrome.

**Effective solution USING TRIES:**

Create  a empty trie,insert reverse of all strings in trie, then search each string whether it forms a palindrome with the string in trie or not.

**Time Complexity: O(N*K^2)**

**Where n is the number of words in the list and k is the maximum length that is checked for palindrome.**

**CODE:**

```
struct node{
   struct node *children[26];
   bool eow;
};

struct node* getnode(void)
{
   struct node *p=new node;
```

```
  for(int i=0;i<26;i++)
  {
     p->children[i]=NULL;
  }
  p->eow=false;
  return p;
}

void insert(struct node *root,string s)
{  struct node *p=root;
for(int i=s.length()-1;i>=0;i--)
   {
  int index=s[i]-'a';
     if(!p->children[index])
         p->children[index]=getnode();
     p=p->children[index];
   }
    p->eow=true;
   }
   bool palindrome(string s)
   {   int n=s.length();
     for(int i=0;i<=n/2;i++)
  {
         if(s[i]!=s[n-i-1])
           return false;
     }
```

```
      return true;
  }
  bool search(struct node *root,string s)
  {
   node *p=root;
   int i;
string a="";
   for(i=0;i<s.length();i++)
   {
       int index=s[i]-'a';
       if(!p->children[index])
          return false;
       p=p->children[index];
   }

    if(p->eow)
{
    return true;
   }

   else{
       for(int i=0;i<26;i++)
       {
           if(p->children[i])
            { a.push_back(i+'a');
            p=p->children[i];
          }
         }
    }
      bool res=palindrome(a);
      return res;
   }
   class Solution{
   public:

   // Function to check if a palindrome pair exists
bool palindromepair(int N, string arr[])
   {
     struct node *root=getnode();
     for(int i=0;i<N;i++)
     {
         insert(root,arr[i]);
     }
     bool ans;
     for(int i=0;i<N;i++)
     {
    ans=search(root,arr[i]);
         if(ans)
           return true;
     }
     return false;
   }
```

# MINIMUM XOR VALUE PAIR

**Given an array of integers of size N find minimum xor of any 2 elements.**

**Example 1:**

**Input**:

**N = 3**

**arr[] = {9,5,3}**

**Output: 6**

**Explanation:**

**There are 3 pairs -**

**9^5 = 12**

**5^3 = 6**

**9^3 = 10**

**Therefore output is 6.**

## ALGORITHM:

A Simple Solution is generate all pairs of given array and compute XOR their values. Finally return minimum XOR value. This solution takes O(N^2) time.

**An Efficient solution** ,can solve this problem in O(nlogn) time,sort the given array and check for consecutive pair.

A further more **Efficient solution** can solve the above problem in O(n) time under the assumption that integers take fixed number of bits to store. The idea is to use Trie Data Structure. Below is algorithm.

1). Create an empty trie. Every node of trie contains two children

for 0 and 1 bits.

2). Initialize min_xor = INT_MAX, insert arr[0] into trie

3). Traversal all array element one-by-one starting from second.

a. First find minimum setbet difference value in trie do xor of current element with minimum setbit diff that value .

b. update min_xor value if required

c. insert current array element in trie

4). return min_xor

## CODE:

```
#define INT_SIZE 32
struct node{
   struct node* children[2];
   int value;
};

struct node *getnode(void)
{
   node *p=new node;
   p->children[0]=p->children[1]=NULL;
```

```
  p->value=0;
  return p;
}
void insert(struct node *root,int key)
{
  node *p=root;
  for(int i=INT_SIZE-1;i>=0;i--)
  {
     bool bit=(key & (1<<i));
        if(p->children[bit]==NULL)
       p->children[bit]=getnode();
        p=p->children[bit];
     }
     p->value=key;
   }
   int minimumxor(struct node* root,int key)
   {
     node *temp=root;
     for(int i=INT_SIZE-1;i>=0;i--)
     {
     bool bit=(key & (1<<i));
        if(temp->children[bit]!=NULL)
           temp=temp->children[bit];
        else if(temp->children[1-bit]!=NULL)
           temp=temp->children[1-bit];
     }
```

```
        return (key^(temp->value));
    }
    int XOR(int arr[],int n)
    {
  int m=INT_MAX;
        node *root=getnode();
        insert(root,arr[0]);
        for(int i=1;i<n;i++)
        {
            m=min(m,minimumxor(root,arr[i]));
            insert(root,arr[i]);
        }
        return m;
    }
class Solution{
    public:
        int minxorpair(int N, int arr[]){
            int a=XOR(arr,N);
            return a;
        }
    };
```

## MAXIMUM XOR SUBARRAY

**Given an array of integers of size N find subarray with maximum xor. A subbarray is a contiguous part of array.**

**Example 1:**

**Input:**

**N = 4**

**arr[] = {1,2,3,4}**

**Output: 7**

**Explanation:**

**The subarray {3,4} has maximum xor**

**value equal to 7.**

## ALGORITHM:

A Simple Solution is to use two loops to find XOR of all subarrays and return the maximum, Time complexity O(N^2).

**An Efficient Solution,** can solve the above problem in O(n) time under the assumption that integers take fixed number of bits to store. The idea is to use Trie Data Structure. Below is algorithm.

1) Create an empty Trie. Every node of Trie is going to contain two children,

for 0 and 1 value of bit.

2) Initialize pre_xor = 0 and insert into the Trie.

3) Initialize result = minus infinite

4) Traverse the given array and do following for every

array element arr[i].

a) pre_xor = pre_xor ^ arr[i], pre_xor now contains xor of elements from

arr[0] to arr[i].

b) Query the maximum xor value ending with arr[i] from Trie.

c) Update result if the value obtained is more than current value of result.

**CODE:**

```
#define INT_SIZE 32
struct node{
   struct node* children[2];
   int value;
};

struct node *getnode(void)
{
   node *p=new node;
   p->children[0]=p->children[1]=NULL;
   p->value=0;
   return p;
}
void insert(struct node *root,int key)
{
```

```
node *p=root;
for(int i=INT_SIZE-1;i>=0;i--)
{
    bool bit=(key & (1<<i));
        if(p->children[bit]==NULL)
       p->children[bit]=getnode();
        p=p->children[bit];
    }
    p->value=key;
  }
  int maximumxor(struct node* root,int key)
  {
    node *temp=root;
    for(int i=INT_SIZE-1;i>=0;i--)
    {
    bool bit=(key & (1<<i));
        if(temp->children[1-bit]!=NULL)
            temp=temp->children[1-bit];
        else if(temp->children[bit]!=NULL)
            temp=temp->children[bit];
    }
    return (key^(temp->value));
  }
  int XOR(int arr[],int n)
  {
int m=INT_MIN,pre_xor=0;
```

```
    node *root=getnode();
    insert(root,0);
    for(int i=0;i<n;i++)
    {
        pre_xor=pre_xor^arr[i];
        insert(root,pre_xor);
        m=max(m,maximumxor(root,pre_xor));
    }
    return m;
}
class Solution{
public:
    int maxSubarrayXOR(int N, int arr[]){
        XOR(arr,N);
    }
};
```