CodeWork

SUBSTRING MATCHING

Rabin karp Hashing :

```
void HashValue(string p)
{
   for(int i=0;i<p.size();i++)
   {
      value+=aa[p[i]]*pow(prime,i);
   }
}
int chackSubString(string s,string p)
{
   int count = 0;
   int len1 = p.size();
   int len2 = s.size();
   ll temp = 0;
   for(int i=0;i<len1;i++)
   {
      temp +=aa[s[i]] * pow(prime,i);
   }
   if(temp == value) count++;
   for(int i=len1;i<len2;i++)
   {
      /// main algorithm
        temp= temp - aa[s[i-len1]];
      temp= temp/prime;
      temp= temp + aa[s[i]]*pow(prime,len1-1);
      if(temp == value) count++;
   }
```

Hashing sub Stirng

```
typedef long long int ll;
int prime = 13;
ll base[1000];
ll hash1[1000];
ll hash2[1000];
void power()
```

```
{
    base[0] = 1;
    for(int i=1;i<1000;i++) base[i] = base[i-1]*prime;
}
void HashValue(string s,string p)
{
    int len1 = s.size(),len2 = p.size();
    hash1[0] = s[0];
    for(int i=1;i<len1;i++) hash1[i] = hash1[i-1]*prime + s[i];
    hash2[0] = p[0];
    for(int i=1;i<len2;i++) hash2[i] = hash2[i-1]*prime + p[i];
}
ll getHash(int i,int j)
{
    if(i==0) return hash1[j];
    return hash1[j] - hash1[i-1] * base[j-i+1];
    /// here base value is sub of j and i but in sum one value is missing
    /// so that reason j-i with add one...j-i+1;
}
int main()
{
    power();
    string s,p;
    cin >> s >> p;
    HashValue(s,p);
    ll len1 = s.size();
    ll len2 = p.size();
    ll HashVal = hash2[len2-1];
    int count = 0;
    for(int i=0;i<len1;i++)
    {
        ll temp = getHash(i,i+len2-1);
        if(temp== HashVal) count++;
    }
    cout << count << endl;
}
```

Link List Concept:

```
#include <bits/stdc++.h>
using namespace std;
struct Node{
    int data;
```

```
    Node *next;
};
struct Node *head;

void Insert(int x)
{
    struct Node *temp = new Node();
    temp ->data = x;
    temp ->next = head;
    head = temp;
}
void print()
{
    struct Node *temp = head;
    while(temp!=NULL)
    {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
void Num_insert(int num,int n)
{
    struct Node *temp = new Node();
    temp->data = num;
    temp->next = NULL;
    if(n==1)
    {
        temp->next = head;
        head = temp;
        return;
    }
    Node *temp1 = head;
    for(int i=0;i<n-2;i++) temp1 = temp1->next;
    temp->next = temp1->next;
    temp1->next = temp;
}

void Delete(int n)
{
    Node *temp1,*temp2;
    temp1 = head;
    for(int i=0;i<n-2;i++) temp1 = temp1->next;
```

```
    temp2 = temp1->next;;
    temp1->next = temp2->next;
    //delete temp2;
}

void Reverse()
{
    Node *next,*current,*pre;
    pre = NULL;
    current = head;
    while(current!=NULL)
    {
        next = current->next;
        current->next = pre;
        pre = current;
        current = next;
    }
    head = pre;
}

int main()
{
    head = NULL;
    int n;
    cin >> n;
    while(n--)
    {
        int num;
        cin >> num;
        Insert(num);
//        print();
    }
    print();
    int num;
//    cin >> num >> n;
//    Num_insert(num,n);
//    print();
//    cin >> n;
//    Delete(n);
//    print();
    Reverse();
    print();
}
```

BIT(Binary Index Tree)

Build tree: Right most bit one is 2^(index) is the size of the index.
Example:

1 = 0001 so size is 2^0 = 1
2 = 0010 so size is 2^1 = 2
3 = 0011 so size is 2^0 = 1
6 = 0110 so size is 2^1 = 2

Find the query : Tree build first rightmost bit one(1) change than find the range sum of the index.
Example: if index 11 than 11 binary is 1011

1011 = 11
1010 = 10
1000 = 81
5 = 101
-5 = 011
5&-5 = 001
5+(5&-5) = 6

----------code----------

```
/*
Here input start in 0
But BitTree array start with 1
*/
int getSum(int i)
{
   int sum = 0;
   while(i>0)
   {
      sum+=BitTree[i];
      /// reverse the BIT Tree
      i-=i&-i;
   }
   return sum;
}
void updateBIT(int i,int n,int val)
{
   /// update and insert are same;
   i++;
   while(i<=n)
   {
```

```
        BitTree[i]+=val;
        /// forward the BIT Tree
        i +=i&(-i);
    }
}
```

Segment Tree Range Minimum:

1.partial overlap
2.total overlap
3.no overlap

Find child Node :
node*2+1
node*2+2

```
int st[1000];
int input[1000];
void CreateSegmentTree(int l,int h,int pos)
{
    if(l==h)
    {
        st[pos] = input[h];
        return;
    }
    int mid = (l+h)/2;
    CreateSegmentTree(l,mid,pos*2+1);
    CreateSegmentTree(mid+1,h,pos*2+2);
    st[pos] = min(st[pos*2+1],st[pos*2+2]);
}

int rangeBit(int qlow,int qhigh,int low,int high,int pos)
{
    ///total overlap
    if(qlow<=low && high<=qhigh)
    {
        return st[pos];
    }
```

```
    /// no overlap
    if(qlow>high || qhigh<low) return 10000000;
    int mid = (low + high)/2;
    return min(rangeBit(qlow,qhigh,low,mid,pos*2+1),rangeBit(qlow,qhigh,mid+1,high,pos*2+2));
}
```

KMP Algorithm(sub String):

```
String s,p;
int fre[1000010];
void process(string p)
{
    int len = p.size();
    int i =0,j = -1;
    fre[i] = j;
    while(i<len)
    {
        while(j>=0 && p[i]!=p[j]) j = fre[j];
        i++;
        j++;
        fre[i] = j;
    }
}
int knp(string s,string p)
{
    int len1 = s.size();
    int len2 = p.size();
    int i=0,j=0,count = 0;
    while(i<len1)
    {
        while(j>=0 && s[i] != p[j]) j = fre[j];
        i++,j++;
        if(j == len2)
        {
            count++;
            /// here the subString not begin for the pattern.
            j = fre[j];
            /// match previous substring index;
        }
```

```
    }
    return count;
}
```
----------if palindrome--------
Case is match the suffix and prefix of the reverse of the given String
Example:

S =aaaa(bb)

P = (bb)aaaa (reverse of the string S)

Answer= Sizeof(s) * 2 - (match of the reverse subSting length

Trie:

```
struct trieNode{
    trieNode *childNode[26];
    bool isEndOfWord;
};
trieNode *getNode()
{
    struct trieNode *pNode = new trieNode;
    pNode->isEndOfWord = false;
    for(int i =0;i<26;i++)
    {
        pNode->childNode[i] = NULL;
    }
    return pNode;
}
void insert(struct trieNode *root,string key)
{
    struct trieNode *pNode =root;
    for(int i=0;i<key.length();i++)
    {
        int index = key[i] - 'a';
////        if(!pNode -> childNode[index])
            if(pNode -> childNode[index]==NULL)
            pNode->childNode[index] = getNode();
        pNode = pNode->childNode[index];
    }
    pNode->isEndOfWord = true;
}
bool search(struct trieNode *root,string key)
{
```

```
    struct trieNode *pNode = root;
    for(int i=0;i<key.length();i++)
    {
        int index = key[i] - 'a';
        if(!pNode->childNode[index])
            return false;
        pNode = pNode->childNode[index];
    }
    return (pNode!=NULL && pNode->isEndOfWord);
}
```

Binary Search tree

```
#include<bits/stdc++.h>
using namespace std;
struct Node{
    int data;
    Node *left;
    Node *right;
};
Node *createNode(int data)
{
    Node *root = new Node();
    root->data = data;
    root->left = root->right = NULL;
    return root;
}
Node *FindMin(Node *root)
{
    if(root == NULL) return NULL;
    while(root->left != NULL)
    {
        root = root->left;
    }
    return root;
}
Node *Delete(Node *root,int data)
{
    if(root == NULL) return root;
    else if(data < root->data) root->left = Delete(root->left,data);
    else if(data > root->data) root->right = Delete(root->right,data);
```

```
    else
    {
        if(root->left == NULL && root->right == NULL)
        {
            delete root;
            root = NULL;
        }
        else if(root->left == NULL)
        {
            Node *temp = root;
            root = root->right;
            delete temp;
        }
        else if(root->right == NULL)
        {
            Node *temp = root;
            root = root->left;
            delete temp;
        }
        else {
            Node *temp = (root->right);
            root->data = temp->data;
            root->right = Delete(root->right,temp->data);
        }
    }
    return root;
}


Node *InsertNode(Node *root,int data)
{
    if(root == NULL)
    {
        root = createNode(data);
    }
    else if(data<=root->data)
    {
        root->left = InsertNode(root->left,data);
    }
    else{
        root->right = InsertNode(root->right,data);
    }
    return root;
```

```
}

bool Search(Node *root,int data)
{
   if(root == NULL) return NULL;
   else if(root->data == data) return true;
   else if(root->data<data) Search(root->right,data);
   else Search(root->left,data);
}
int main()
{
   int n;
   cin >> n;
   Node *root = NULL   ;
   for(int i=0;i<n;i++)
   {
      int num;
      cin >> num;
      root = InsertNode(root,num);
   }
   int num;
   while(true)
   {
      int test ;
      cin >> test >> num;
      if(test == 1)
      {
         if(Search(root,num)) cout << "data found.\n" << endl;
      else cout << "data not found\n" << endl;
      }
      else{
         Delete(root,num);
         cout << "Delete Complect." << endl;
      }
   }
}
```

Pointer insert-delete

```
#include<bits/stdc++.h>
using namespace std;
struct node {
    int data;
    node *next;
};
struct node *head;
void Insert(int data)
{
    node *temp = new node();
    temp->data = data;
    temp->next = head;
    head = temp;
}
void insertAnyPosition(int n,int data)
{
    node *temp1 = new node();
    temp1 ->data = data;
    temp1 ->next = NULL;
    if(n == 1)
    {
        temp1->next = head;
        head = temp1;
        return;
    }
    node *temp2 = head;
    for(int i = 0;i<n-2;i++) temp2 = temp2->next;
    temp1->next = temp2->next;
    temp2->next = temp1;
}
void Delete(int n)
{
    node *temp1 = head;
    if(n==1)
    {
        head = temp1->next;
        free(temp1);
    }
    for(int i = 0;i<n-2;i++) temp1 = temp1->next;
    node *temp2 = temp1->next;
    temp1->next = temp2->next;
```

```
    free(temp2);
}
void Print()
{
    node *temp = head;
    while(temp!=NULL)
    {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
void ReversePointer()
{
    node *current,*prev,*next;
    current = head;
    prev = NULL;
    while(current!=NULL)
    {
         next = current->next;
         current->next = prev;
         prev = current;
         current = next;
    }
    head = prev;
}
void ReverseRecursive(node *p)
{
    if(p->next == NULL)
    {
        head = p;
        return;
    }
    ReverseRecursive(p->next);
    node *q=p->next;
    q->next = p;
    p->next =NULL;
}

int main()
{
    head = NULL;
    int n;
```

```
    cin >> n;
    for(int i=0;i<n;i++)
    {
        int data;
        cin >> data;
        Insert(data);
//        Print();
//        cout << endl;
    }
    Print();
    insertAnyPosition(4,34);
    Print();
    insertAnyPosition(1,33);
    Print();
    insertAnyPosition(2,14);
    Print();
    insertAnyPosition(2,44);
    Print();
    Delete(2);
    ReversePointer();
    Print();


}
```