Algorithm Code Book

Tanvir Hasan Anick

November 12, 2015

Contents

Chapter 1

Data Structure

1.1 Trie

1.1.1 Static Trie

```
#define Max 10005
int getId(char c)
{
    return c>='a'?c-'a':c-'A'+26;
}
struct Trie
{
    struct Tree
    {
        int Next[52];
        bool word;
        void clear()
        {
            word=false;
            memset(Next,-1,sizeof(Next));
        }
    } T[Max];
    int ptr;
    void clear()
    {
        ptr=1;
        T[0].clear();
        memset(T[0].Next,0,sizeof(T[0].Next));
    }
    void Insert(const char *str)
    {
        int p=0;
        for(int i=0; str[i]; i++)
        {
            int id=getId(str[i]);
            if(T[p].Next[id]<=0)
            {
                T[p].Next[id]=ptr;
                T[ptr++].clear();
            }
            p=T[p].Next[id];
        }
        T[p].word=true;
    }
    bool Search(const char *str)
```

```
    {
        int p=0;
        for(int i=0; str[i]; i++)
        {
            int id=getId(str[i]);
            if(T[p].Next[id]>0)
            {
                p=T[p].Next[id];
            }
            else return false;
        }
        return T[p].word;
    }
};
Trie A;
```

1.2 RMQ

1.2.1 Bit

1D Bit

```
#define MaxVal 100000
int Bit[MaxVal];
/**find sum from 1 to idx**/
int read(int idx)
{
    int sum = 0;
    while (idx > 0)
    {
        sum += Bit[idx];
        idx -= (idx & -idx);
    }
    return sum;
}
/**update value ind to MaxVal**/
void update(int idx ,int val)
{
    while (idx <= MaxVal)
    {
        Bit[idx] += val;
        idx += (idx & -idx);
    }
}

/**Find the value of idx**/
int readSingle(int idx)
{
    int sum = Bit[idx]; /// sum will be decreased
    if (idx > 0)  /// special case
    {
        int z = idx - (idx & -idx); /// make z first
        idx--; /// idx is no important any more, so instead y, you can use idx
        while (idx != z)  /// at some iteration idx (y) will become z
        {
            sum -= Bit[idx];/// substruct Bit frequency which is between y and "the same path"
            idx -= (idx & -idx);
        }
    }
```

```
    return sum;
}
```

2D Bit

```
void updatey(int x , int y , int val)
{
    while (y <= max_y)
    {
        tree[x][y] += val;
        y += (y & -y);
    }
}
void update(int x , int y , int val)
{
    while (x <= max_x)
    {
        updatey(x , y , val);// this function should update array tree[x]
        x += (x & -x);
    }
}
```

1.2.2 Square Root Decompostion

```
#include<bits/stdc++.h>
using namespace std;
const int sz=100005;
const int inf=(1<<28);
template<typename t> t MIN3(t a,t b, t c)
{
    return min(a,min(b,c));
}
int BLOCK[400];
int arr[sz];
int getId(int indx,int blockSZ)
{
    return indx/blockSZ;
}
void init(int sz)
{
    for(int i=0; i<=sz; i++)BLOCK[i]=inf;
}
void update(int val,int indx,int blockSZ)
{
    int id=getId(indx,blockSZ);
    BLOCK[id]=min(BLOCK[id],val);
}
int query(int L,int R,int blockSZ)
{
    int lid=getId(L,blockSZ);
    int rid=getId(R,blockSZ);
    if(lid==rid)
    {
        int ret=inf;
        for(int i=L; i<=R; i++)ret=min(ret,arr[i]);
        return ret;
    }
    int m1=inf,m2=inf,m3=inf;
    for(int i=L; i<(lid+1)*blockSZ; i++)m1=min(m1,arr[i]);
    for(int i=lid+1; i<rid; i++)m2=min(m2,BLOCK[i]);
    for(int i=rid*blockSZ; i<=R; i++)m3=min(m3,arr[i]);
```

```
    return MIN3(m1,m2,m3);
}
int main()
{
    int N,Q;
    scanf("%d %d",&N,&Q);
    int blockSZ=sqrt(N);
    init(blockSZ);
    for(int i=0; i<N; i++)
    {
        int x;
        scanf("%d",&x);
        arr[i]=x;
        update(x,i,blockSZ);
    }
    while(Q--)
    {
        int x,y;
        scanf("%d %d",&x,&y);
        printf("%d\n",query(x,y,blockSZ));
    }
    return 0;
}
```

1.2.3 MO's Algorithm

```
/**
    MO's Algorithm
    problem: http://www.spoj.com/problems/DQUERY

    MOs algorithm is just an order in which we process the queries.
    We were given M queries, we will re-order the queries in a particular order and then process them.
    Clearly, this is an off-line algorithm. Each query has L and R, we will call them opening and closing.
    Let us divide the given input array into Sqrt(N) blocks.
    Each block will be N / Sqrt(N) = Sqrt(N) size.
    Each opening has to fall in one of these blocks.
    Each closing has to fall in one of these blocks.

    All the queries are first ordered in ascending order of their block number (block number is the block in which its opening falls).
    Ties are ordered in ascending order of their R value.

**/
#include<bits/stdc++.h>
using namespace std;
#define Mx 30005
#define MxNum 1000005
int BlockSize;
int Answer;
int Freq[MxNum],Num[Mx];
struct info
{
    int L,R,qno;
    info(int L=0,int R=0,int qno=0):L(L),R(R),qno(qno) {};
    bool operator<(const info &a)const
    {
        if(L/BlockSize!=a.L/BlockSize)return L/BlockSize<a.L/BlockSize;
        return R<a.R;
    }
```

```
} Query[200005];
int StoreAnswer[200005];
void Add(int indx)
{
    Freq[Num[indx]]++;
    if(Freq[Num[indx]]==1)Answer++;
}
void Remove(int indx)
{
    Freq[Num[indx]]--;
    if(Freq[Num[indx]]==0)Answer--;
}
int main()
{
    int N;
    scanf("%d",&N);
    BlockSize=sqrt(N);
    for(int i=0; i<N; i++)
    {
        scanf("%d",&Num[i]);
    }
    int Q;
    scanf("%d",&Q);
    for(int i=0; i<Q; i++)
    {
        int x,y;
        scanf("%d %d",&x,&y);
        Query[i]=info(x-1,y-1,i);
    }
    sort(Query,Query+Q);
    int currentL=0,currentR=0;
    Answer=0;
    for(int i=0; i<Q; i++)
    {
        int L=Query[i].L;
        int R=Query[i].R;
        while(currentL<L)
        {
            Remove(currentL);
            currentL++;
        }
        while(currentL>L)
        {
            Add(currentL-1);
            currentL--;
        }
        while(currentR<=R)
        {
            Add(currentR);
            currentR++;
        }
        while(currentR>R+1)
        {
            Remove(currentR-1);
            currentR--;
        }
        StoreAnswer[Query[i].qno]=Answer;
    }
    for(int i=0; i<Q; i++)
    {
        printf("%d\n",StoreAnswer[i]);
```

```
    }
    return 0;
}
```

1.2.4 Segment Tree

Lazy Propagration1

```
/**
**You are given an array of N elements, which are initially all 0. After **
    that you will be given C commands. They are
**0 p q v - you have to add v to all numbers in the range **of p to q (
    inclusive), where p and q are two indexes of the array.
**1 p q - output a line containing a single integer which is the sum of all
    **the array elements between p and q (inclusive)
*/
#include<bits/stdc++.h>
using namespace std;
typedef long long LLD;
LLD tree[3*100005];
LLD lazy[3*100005];
void update(int left,int right,int index,int x,int y,int value)
{
    if(x<=left&&y>=right)
    {
        tree[index]+=(LLD)(right-left+1)*value;
        lazy[index]+=value;
        return;
    }
    int mid=(left+right)/2;
    if(lazy[index]!=0)
    {
        tree[2*index]+=(LLD)(mid-left+1)*lazy[index];
        tree[2*index+1]+=(LLD)(right-mid)*lazy[index];
        lazy[2*index]+=lazy[index];
        lazy[2*index+1]+=lazy[index];
        lazy[index]=0;
    }
    if(x<=mid)
    {
        update(left,mid,2*index,x,y,value);
    }
    if(y>mid)
    {
        update(mid+1,right,2*index+1,x,y,value);
    }
    tree[index]=tree[2*index]+tree[2*index+1];
}
LLD query(int left,int right,int index,int x,int y)
{
    LLD a1=0,a2=0;
    if(x<=left&&y>=right)
    {
        return tree[index];
    }
    int mid=(left+right)/2;
    if(lazy[index]!=0)
    {
        tree[2*index]+=(LLD)(mid-left+1)*lazy[index];
        tree[2*index+1]+=(LLD)(right-mid)*lazy[index];
        lazy[2*index]+=lazy[index];
```

```
        lazy[2*index+1]+=lazy[index];
        lazy[index]=0;
    }
    if(x<=mid)
    {
        a1=query(left,mid,2*index,x,y);
    }
    if(y>mid)
    {
        a2=query(mid+1,right,2*index+1,x,y);
    }
    return (a1+a2);
}
int main()
{
    int test,t;
    scanf("%d",&test);
    for(t=1;t<=test;t++)
    {
        memset(tree,0,sizeof(tree));
        memset(lazy,0,sizeof*lazy);
        int s,q;
        scanf("%d %d",&s,&q);
        while(q--)
        {
            int x,y,v,dec;
            scanf("%d",&dec);
            if(dec)
            {
                scanf("%d %d",&x,&y);
                LLD ans=query(0,s-1,1,x-1,y-1);
                printf("%lld\n",ans);
            }
            else
            {
                scanf("%d %d %d",&x,&y,&v);
                update(0,s-1,1,x-1,y-1,v);
            }
        }
    }
    return 0;
}
```

Lazy Propagration2

```
/*
**You have an array with n elements which is indexed from 0 to n - 1. **
    Initially all elements are zero. Now you have to deal with two types of
    **operations
**1.Increase the numbers between indices i and j (inclusive) by 1. This **is
     **represented by the command '0 i j'.
**2.Answer how many numbers between indices i and j (inclusive) are **
    divisible by 3. This is represented by the command '1 i j'.
*/
#include<bits/stdc++.h>
using namespace std;
#define Max 100010
int Tree[8*Max][4];
int lazy[8*Max];
int temp[4];
void build(int left,int right,int indx)
```

```
{
    if(left==right)
    {
        Tree[indx][0]=1;
        Tree[indx][1]=Tree[indx][2]=lazy[indx]=0;
        return;
    }
    int mid=(left+right)/2;
    build(left,mid,2*indx);
    build(mid+1,right,2*indx+1);
    for(int i=0;i<3;i++)
    {
        Tree[indx][i]=Tree[2*indx][i]+Tree[2*indx+1][i];
    }
}
void update(int left,int right,int indx,int x,int y,int add)
{
    if(lazy[indx])
    {
        int lazy_val=lazy[indx];
        lazy[2*indx]=(lazy[2*indx]+lazy_val)%3;
        lazy[2*indx+1]=(lazy[2*indx+1]+lazy_val)%3;
        for(int i=0;i<3;i++)temp[(lazy_val+i)%3]=Tree[indx][i];
        for(int i=0;i<3;i++)Tree[indx][i]=temp[i];
        lazy[indx]=0;
    }
    if(left>y||right<x)return;
    if(x<=left&&right<=y)
    {
        for(int i=0;i<3;i++)
        {
            temp[(i+add)%3]=Tree[indx][i];
        }
        for(int i=0;i<3;i++)Tree[indx][i]=temp[i];
        lazy[2*indx]=(lazy[2*indx]+add)%3;
        lazy[2*indx+1]=(lazy[2*indx+1]+add)%3;
        return;
    }
    int mid=(left+right)/2;
    update(left,mid,2*indx,x,y,add);
    update(mid+1,right,2*indx+1,x,y,add);
    for(int i=0;i<3;i++)
    {
        Tree[indx][i]=Tree[2*indx][i]+Tree[2*indx+1][i];
    }
}
int query(int left,int right,int indx,int x,int y)
{
    if(lazy[indx])
    {
        int lazy_val=lazy[indx];
        lazy[2*indx]=(lazy[2*indx]+lazy_val)%3;
        lazy[2*indx+1]=(lazy[2*indx+1]+lazy_val)%3;
        for(int i=0;i<3;i++)temp[(lazy_val+i)%3]=Tree[indx][i];
        for(int i=0;i<3;i++)Tree[indx][i]=temp[i];
        lazy[indx]=0;
    }
    if(left>y||right<x)return 0;
    if(x<=left&&right<=y)return Tree[indx][0];
    int mid=(left+right)/2;
    return query(left,mid,2*indx,x,y)+query(mid+1,right,2*indx+1,x,y);
```

```
}
int main()
{
    int x,y;
    int test;
    scanf("%d",&test);
    for(int t=1;t<=test;t++)
    {
        memset(lazy,0,sizeof(lazy));
        int N,Q;
        scanf("%d %d",&N,&Q);
        build(0,N-1,1);
        printf("Case %d:\n",t);
        for(int i=0;i<Q;i++)
        {
            int d;
            scanf("%d %d %d",&d,&x,&y);
            if(d==0)
            {
                update(0,N-1,1,x,y,1);
            }
            else printf("%d\n",query(0,N-1,1,x,y));
        }
    }
    return 0;
}
```

Segment Tree Variant 1

```
/**
**Give a array Of N numbers. Finding Maximum cumulative number frequency in
    **the range.
**input:
**10 4
**1 1 1 3 3 3 3 2 2 2
**1 5
**1 6
**1 7
**Output:
**3
**3
**4
**2
*/
#include<bits/stdc++.h>
using namespace std;
typedef long long LLD;
#define MAX 50005
struct info
{
    int Lcnt,Rcnt,Max,Lnum,Rnum;
    info(int Lcnt=0,int Rcnt=0,int Max=0,int Lnum=0,int Rnum=0):Lcnt(Lcnt),
    Rcnt(Rcnt),Max(Max),Lnum(Lnum),Rnum(Rnum){};
};
info Tree[3*MAX];
int arr[MAX];
info marge(const info &L,const info &R)
{
    info ret;
    if(L.Rnum==R.Lnum)
    {
```

```
31          ret.Max=max(L.Rcnt+R.Lcnt,max(L.Max,R.Max));
32      }
33      else ret.Max=max(L.Max,R.Max);
34      ret.Lnum=L.Lnum;
35      ret.Rnum=R.Rnum;
36      if(L.Lnum==R.Lnum)ret.Lcnt=L.Lcnt+R.Lcnt;
37      else ret.Lcnt=L.Lcnt;
38      if(L.Rnum==R.Rnum)ret.Rcnt=L.Rcnt+R.Rcnt;
39      else ret.Rcnt=R.Rcnt;
40      return ret;
41  }
42  void build(int L,int R,int indx)
43  {
44      if(L==R)
45      {
46          Tree[indx]=info(1,1,1,arr[L],arr[R]);
47          return;
48      }
49      int mid=(L+R)>>1;
50      build(L,mid,2*indx);
51      build(mid+1,R,2*indx+1);
52      Tree[indx]=marge(Tree[2*indx],Tree[2*indx+1]);
53  }
54  info query(int L,int R,int indx,int x,int y)
55  {
56      if(L>=x&&R<=y)return Tree[indx];
57      int mid=(L+R)>>1;
58      info c1,c2;
59      if(x<=mid)c1=query(L,mid,2*indx,x,y);
60      if(y>mid)c2=query(mid+1,R,2*indx+1,x,y);
61      return marge(c1,c2);
62  }
63  int main()
64  {
65      int test;
66      scanf("%d",&test);
67      for(int t=1;t<=test;t++)
68      {
69          int N,C,Q;
70          scanf("%d %d %d",&N,&C,&Q);
71          for(int i=0;i<N;i++)
72          {
73              int x;
74              scanf("%d",&arr[i+1]);
75          }
76          build(1,N,1);
77          printf("Case %d:\n",t);
78          while(Q--)
79          {
80              int x,y;
81              scanf("%d %d",&x,&y);
82              printf("%d\n",query(1,N,1,x,y).Max);
83          }
84      }
85      return 0;
86  }
```

Segment Tree Variant 2

```
/**
**You are given a sequence A of N (N <= 50000) integers between -10000 and 10000.
**On this sequence you have to apply M (M <= 50000) operations:
**modify the i-th element in the sequence or for given x y print max{Ai + Ai+1 + .. + Aj | x<=i<=j<=y }.
**/
#include<bits/stdc++.h>
using namespace std;
typedef long long LLD;
template<class T> T MAX3(T a,T b,T c) {return max(a,max(b,c));}
LLD Inf=(1ll<<60);
#define MN 50005
struct info
{
    LLD prefixSum;
    LLD suffixSum;
    LLD Total;
    LLD TotalMax;
    info(int pre=-Inf,int suff=-Inf,int total=-Inf,int totalmax=-Inf):
    prefixSum(pre),suffixSum(suff),Total(total),TotalMax(totalmax){};
};
info marge(const info &a,const info &b)
{
    info ret;
    ret.Total=a.Total+b.Total;
    ret.prefixSum=max(a.prefixSum,a.Total+b.prefixSum);
    ret.suffixSum=max(a.suffixSum+b.Total,b.suffixSum);
    ret.TotalMax=MAX3(a.TotalMax,b.TotalMax,a.suffixSum+b.prefixSum);
    return ret;
}
LLD arr[MN];
info Tree[3*MN];
void build(int L,int R,int indx)
{
    if(L==R)
    {
        Tree[indx]=info(arr[L],arr[L],arr[L],arr[L]);
        return;
    }
    int mid=(L+R)>>1;
    build(L,mid,2*indx);
    build(mid+1,R,2*indx+1);
    Tree[indx]=marge(Tree[2*indx],Tree[2*indx+1]);
}
void update(int L,int R,int indx,int x,LLD val)
{
    if(L==R)
    {
        Tree[indx]=info(val,val,val,val);
        return;
    }
    int mid=(L+R)>>1;
    if(x<=mid)update(L,mid,2*indx,x,val);
    else update(mid+1,R,2*indx+1,x,val);
    Tree[indx]=marge(Tree[2*indx],Tree[2*indx+1]);
}
info query(int L,int R,int indx,int x,int y)
{
```

```
    if(L==x and y==R)return Tree[indx];
    int mid=(L+R)>>1;
    if(y<=mid)return query(L,mid,2*indx,x,y);
    else if(x>mid)return query(mid+1,R,2*indx+1,x,y);
    return marge(query(L,mid,2*indx,x,mid),query(mid+1,R,2*indx+1,mid+1,y));
}
int main()
{
    #ifdef _ANICK_
    //f_input;
    #endif // _ANICK_
    int N;
    scanf("%d",&N);
    for(int i=1;i<=N;i++)scanf("%lld",&arr[i]);
    build(1,N,1);
    int Q;
    scanf("%d",&Q);
    while(Q--)
    {
        int t,x,y;
        scanf("%d %d %d",&t,&x,&y);
        if(t)printf("%lld\n",query(1,N,1,x,y).TotalMax);
        else update(1,N,1,x,y);
    }
    return 0;
}
```

Segment Tree Variant 3

```
/**
**Given a bracket sequence.
** On a bracket word one can do the following operations:
**replacement -- changes the i-th bracket into the opposite one
**check -- if the word is a correct bracket expression
**/
#include<bits/stdc++.h>
using namespace std;
typedef long long LLD;
#define MAX 50005
struct info
{
    int sum,sub;
    info(int sum=0,int sub=0):sum(sum),sub(sub){};
};
info Tree[4*MAX];
char inp[MAX];
info marge(const info &L,const info &R)
{
    info ret;
    ret.sum= L.sum+R.sum;
    ret.sub=L.sub;
    ret.sub=min(ret.sub,L.sum+R.sub);
    return ret;
}
void build(int L,int R,int indx)
{
    if(L==R)
    {
        int x;
        if(inp[L]=='(')x=1;
        else x=-1;
```

```
33          Tree[indx]=info(x,x);
34          return;
35      }
36      int mid=(L+R)>>1;
37      build(L,mid,2*indx);
38      build(mid+1,R,2*indx+1);
39      Tree[indx]=marge(Tree[2*indx],Tree[2*indx+1]);
40  }
41  void update(int L,int R,int indx,int x)
42  {
43      if(L==R)
44      {
45          int x;
46          if(inp[L]=='(')x=1;
47          else x=-1;
48          Tree[indx]=info(x,x);
49          return;
50      }
51      int mid=(L+R)>>1;
52      if(x<=mid)update(L,mid,2*indx,x);
53      else update(mid+1,R,2*indx+1,x);
54      Tree[indx]=marge(Tree[2*indx],Tree[2*indx+1]);
55  }
56  info query(int L,int R,int indx,int x,int y)
57  {
58      if(L==x&&R==y)return Tree[indx];
59      int mid=(L+R)>>1;
60      if(y<=mid)return query(L,mid,2*indx,x,y);
61      else if(x>mid)return query(mid+1,R,2*indx+1,x,y);
62      else return marge(query(L,mid,2*indx,x,mid),query(mid+1,R,2*indx+1,mid
        +1,y));
63  }
64  int main()
65  {
66      int N,t=1;
67      while(scanf("%d",&N)==1)
68      {
69          scanf("%s",inp);
70          build(0,N-1,1);
71          int Q;
72          printf("Test %d:\n",t++);
73          scanf("%d",&Q);
74          while(Q--)
75          {
76              int x;
77              scanf("%d",&x);
78              if(x)
79              {
80                  if(inp[x-1]=='(')inp[x-1]=')';
81                  else inp[x-1]='(';
82                  update(0,N-1,1,x-1);
83              }
84              else
85              {
86                  info y=query(0,N-1,1,0,N-1);
87                  if(y.sum==0&&y.sub>=0)printf("YES\n");
88                  else printf("NO\n");
89              }
90          }
91      }
92      return 0;
```

```
}
```

1.2.5 Sliding Window RMQ

```
/**
    every K size window RMQ
    Calculate in O(N+K) time
**/
#include<bits/stdc++.h>
using namespace std;
vector<int>SlidingRMQ(int *A,int N,int k)
{
    /** Create a Double Ended Queue, Qi that will store indexes of array elements
        The queue will store indexes of useful elements in every window and it will
        maintain decreasing order of values from front to rear in Qi, i.e.,
        arr[Qi.front[]] to arr[Qi.rear()] are sorted in increasing order
    **/
    vector<int>MinWindow;
    deque<int>Q;
    int i;
    /* Process first k (or first window) elements of array */
    for(i=0;i<k;i++)
    {
        /// For very element, the previous largest elements are useless so
        /// remove them from Qi
        while(!Q.empty() and A[i]<=A[Q.back()])Q.pop_back();
        Q.push_back(i);
    }
    /// Process rest of the elements, i.e., from arr[k] to arr[n-1]
    while(i<N)
    {
        /// The element at the front of the queue is the smallest element of
        /// previous window, so insert it result
        MinWindow.push_back(A[Q.front()]);

        /// Remove the elements which are out of this window
        while(!Q.empty() and Q.front()<=i-k)Q.pop_front();

        /// Remove all elements larger than the currently
        /// being added element (remove useless elements)
        while(!Q.empty() and A[i]<=A[Q.back()])Q.pop_back();

        /// Add current element at the rear of Qi
        Q.push_back(i);
        i++;
    }
    /// insert the minimum element of last window
    MinWindow.push_back(A[Q.front()]);
    return MinWindow;
}
int main()
{
    int A[]={100,10, -1, 2,-3,-4,10, 1,100,20};
    vector<int>a=SlidingRMQ(A,10,2);
    for(int i=0;i<a.size();i++)cout<<a[i]<<" ";
    return 0;
}
```

1.2.6 Sparse Table

```
/**
    Compute sparse table in O(NlogN)
    query in O(1)
    Ref link: https://www.topcoder.com/community/data-science/data-science-tutorials/range-minimum-query-and-lowest-common-ancestor/
**/
#include<bits/stdc++.h>
using namespace std;
#define Max 10000005
int rmq[24][Max];
int A[Max];
void Compute_ST(int N)
{
    for (int i = 0; i < N; ++i)rmq[0][i] = i;
    for (int k = 1; (1 << k) < N; ++k)
    {
        for (int i = 0; i + (1 << k) <= N; i++)
        {
            int x = rmq[k - 1][i];
            int y = rmq[k - 1][i + (1 << k - 1)];
            rmq[k][i] = A[x] <= A[y] ? x : y;
        }
    }
}

int RMQ(int i, int j)
{
    int k = log2(j-i);
    int x = rmq[k][i];
    int y = rmq[k][j - (1 << k) + 1];
    return A[x] <= A[y] ? x : y;
}

int main()
{

    return 0;
}
```

1.3 Ternary Bit Mask

```

int more_bit[10];
int get_bit(int mask , int pos)
{
    return (mask / more_bit[pos]) % 3;
}
int set_bit(int mask, int pos , int bit)
{
    int tmp = (mask / more_bit[pos]) % 3;
    mask -= tmp * more_bit[pos];
    mask += bit * more_bit[pos];
    return mask;
}
void init(void){
    more_bit[0] = 3;
    for(int i = 1; i < 10; i++) more_bit[i] = 3 * more_bit[i - 1];
}
```

1.4 Largest Rectangle in Histogram

1.4.1 Largest rectangle in histogram using Stack

```
int GetMaxArea(int *hist,int n)
{
    stack<int> s;

    int max_area = 0;
    int tp;
    int area_with_top;
    int i = 0;
    while (i < n)
    {
        if (s.empty() || hist[s.top()] <= hist[i])
            s.push(i++);
        else
        {
            tp = s.top();
            s.pop();
            area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
            if (max_area < area_with_top)
                max_area = area_with_top;
        }
    }
    while (s.empty() == false)
    {
        tp = s.top();
        s.pop();
        area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);

        if (max_area < area_with_top)
            max_area = area_with_top;
    }
    return max_area;
}
```

1.4.2 Largest rectangle in histogram using Segment Tree

```
#include<bits/stdc++.h>
using namespace std;
#define MN 1000005
typedef long long LLD;
LLD Histogram[MN],N;
template<class T> T MAX3(T a,T b,T c)
{
    return max(a,max(b,c));    ///maximum of 3 number
}
struct info
{
    int Num;
    int pos;
    info(int Num=0,int pos=0):Num(Num),pos(pos) {};
};
info Tree[3*MN];
info marge(const info &L,const info &R)
{
    if(L.Num<R.Num)return L;
    return R;
}
void build(int L,int R,int indx)
```

```
{
    if(L==R)
    {
        Tree[indx]=info(Histogram[L],L);
        return;
    }
    int mid=(L+R)>>1;
    build(L,mid,2*indx);
    build(mid+1,R,2*indx+1);
    Tree[indx]=marge(Tree[2*indx],Tree[2*indx+1]);
}
info query(int L,int R,int indx,int x,int y)
{
    if(L==x&&R==y)return Tree[indx];
    int mid=(L+R)>>1;
    if(y<=mid)return query(L,mid,2*indx,x,y);
    else if(x>mid)return query(mid+1,R,2*indx+1,x,y);
    return marge(query(L,mid,2*indx,x,mid),query(mid+1,R,2*indx+1,mid+1,y));
}
LLD GetMaximumArea(int L,int R)
{
    if(L>R)return -1;
    if(L==R)return Histogram[L];
    info p=query(1,N,1,L,R);
    LLD a=GetMaximumArea(L,p.pos-1);
    LLD b=GetMaximumArea(p.pos+1,R);
    LLD c=(LLD)p.Num*(R-L+1);
    return MAX3(a,b,c);
}
int main()
{
    while(scanf("%d",&N) and N)
    {
        for(int i=1; i<=N; i++)scanf("%lld",&Histogram[i]);
        build(1,N,1);
        printf("%lld\n",GetMaximumArea(1,N));
    }
    return 0;
}
```

1.5 Least common ancestor (LCA)

```
vector<pp>Graph[2005];
int Level[2005];
int Lavel_cost[2005];
int sparse_table[2005][20];
int sparse_cost[2005][20];
void init()
{
    mem(parents,-1);
    mem(sparse_table,-1);
    mem(sparse_cost,0);
}
void dfs(int from,int u,int deep)
{
    Level[u]=deep;
    parents[u]=from;
    int sz=Graph[u].size();
    for(int i=0; i<sz; i++)
    {
```

```
        int v=Graph[u][i].second;
        int c=Graph[u][i].first;
        if(v==from)continue;
        Cost[v]=c;
        dfs(u,v,deep+1);
    }
}
void Sparse(int N)
{
    for(int i=0; i<N; i++)
    {
        sparse_cost[i][0]=Cost[i];
        sparse_table[i][0]=parents[i];
    }
    for(int j=1; (1<<j)<N; j++)
    {
        for(int i=0; i<N; i++)
        {
            if(sparse_table[i][j-1]!=-1)
            {
                int a=sparse_table[i][j-1];
                sparse_table[i][j]=sparse_table[a][j-1];
                sparse_cost[i][j]=sparse_cost[i][j-1]+sparse_cost[a][j-1];
            }
        }
    }
}
int LCA(int p,int q)
{
    if(Level[p]<Level[q])swap(p,q);
    int Log=log2(Level[p])+1;
    int ans=0;
    for(int i=Log; i>=0; i--)
    {
        if((Level[p]-(1<<i))>=Level[q])
        {
            ans+=sparse_cost[p][i];
            p=sparse_table[p][i];
        }
    }
    if(p==q)return ans;
    for(int i=Log; i>=0; i--)
    {
        if(sparse_table[p][i]!=-1&&sparse_table[p][i]!=sparse_table[q][i])
        {
            ans+=sparse_cost[p][i]+sparse_cost[q][i];
            p=sparse_table[p][i],q=sparse_table[q][i];
        }
    }
    ans+=Cost[p]+Cost[q];
    ///Lca will be parent[p];
    return ans;
}
```

1.6 Union Find

```
int parent[10000];
void init(int N)
{
      for(int i=0;i<=N;i++)
         parent[i]=i;
}
int Find(int r)
{
      if(parent[r]==r)return r;
      return parent[r]=Find(parent[r]);
}
void make_parent(int p,int q)
{
      parent[p]=q;
}
void check(int x,int y)
{
      int p=Find(x);
      int q=Find(y);
      if(p!=q)
      {
            make_parent(p,q);
            printf("Make friend\n");
            return;
      }
      printf("they allready friend\n");
      return;
}
```

Chapter 2

Graph Theory

2.1 DFS

2.1.1 Bicoloring

```
///color will be initial with -1
int color[20005];
bool dfs(int u,int c)
{
    if(color[u]==c)return true;
    if(color[u]==(1-c))return false;
    color[u]=c;
    bool ret=true;
    for(auto v:graph[u])ret&=dfs(v,1-c);
    return ret;
}
```

2.1.2 Cycle Finding

```
int color[20005];
bool dfs(int u)
{
    color[u]=GREY;
    bool no_cycle=true;
    for(auto v:graph[u])
    {
        if(color[v]==WHITE)
        {
            no_cycle=dfs(v);
        }
        else if(color[v]==GREY)return false;
    }
    color[u]=BLACK;
    return no_cycle;
}
```

2.2 Topological Sort

```
#include<bits/stdc++.h>
using namespace std;
#define WHITE 0
#define GREY 1
#define BLACK 2
vector<int> graph[100005];
```

```
vector<int> ans;
int visit[100005];
bool dfs(int u)
{
    visit[u]=GREY;
    bool no_cycle=true;
    int sz=graph[u].size();
    for(int i=0;i<sz;i++)
    {
        int v=graph[u][i];
        if(visit[v]==WHITE)
        {
            no_cycle=dfs(v);
        }
        else if(visit[v]==GREY)return false;
    }
    visit[u]=BLACK;
    ans.push_back(u);
    return no_cycle;
}
bool topsort(int N)
{
    ans.clear();
    memset(visit,false,sizeof(visit));
    int no_cycle=true;
    for(int i=0;i<N;i++)
    {
        if(visit[i]==WHITE)no_cycle&=dfs(i);
    }
    return no_cycle;
}
int main()
{
    return 0;
}
```

2.3 Strongly Connected Component

```
typedef vector<int> vint;
vint G[100000+5];
vint G2[100000+5];
vint sorted;
vint cycle;
bool color[100000+5];
void dfs(int u)
{
    color[u]=false;
    for(int i=0; i<G[u].size(); i++)
    {
        int v=G[u][i];
        if(color[v])
        {
            dfs(v);
        }
    }
    sorted.pb(u);
}
void dfs2(int u)
{
    cycle.pb(u);
```

```
23      color[u]=false;
24      for(int i=0; i<G2[u].size(); i++)
25      {
26          int v=G2[u][i];
27          if(color[v])
28          {
29              dfs2(v);
30          }
31      }
32  }
33  void SCC(int N)
34  {
35      mem(color,true);
36      for(int i=1; i<=N; i++)
37      {
38          if(color[i])
39          {
40              dfs(i);
41          }
42      }
43      mem(color,true);
44      for(int i=sorted.size()-1; i>=0; i--)
45      {
46          int u=sorted[i];
47          if(color[u])
48          {
49              cycle.clear();
50              dfs2(u);
51              print_vector(cycle);
52          }
53      }
54  }
```

2.4 Havel Hakimi

Given N degree d1,d2,d3.....dn. Is it possible to make a graph which have no cycle and different two node will be connected with one Edge?

$S = d_1, d_2, d_3, \cdots, d_n$

$d_i \geq d_{i+1}$

1. If any $d_i \geq n$ then fail
2. If there is an odd number of odd degrees then fail
3. If there is a $d_i < 0$ then fail
4. If all $d_i = 0$ then report success
5. Reorder S into non-increasing order
6. Let $k = d_1$
7. Remove d_1 from S.
8. Subtract 1 from the first k terms remaining of the new sequence
9. Go to step 3 above

```cpp
#include<stdio.h>
#include<queue>
#include<vector>
using namespace std;
int main()
{
    int N;
    while(scanf("%d",&N) and N)
    {
        priority_queue<int>Q;
        bool Ok=true;
        int Odd_Node=0;
        for(int i=0; i<N; i++)
        {
            int x;
            scanf("%d",&x);
            if(x>=N or x<0)Ok&=false;
            Odd_Node+=(x%2);
            Q.push(x);
        }
        Ok&=(Odd_Node%2==0); ///Handshaking Theorem
        for(int i=0; i<N and Ok; i++)
        {
            int k=Q.top();
            Q.pop();
            vector<int> v;
            for(int j=0; j<k and Ok; j++)
            {
                int x=Q.top();
                Q.pop();
                x--;
                Ok&=(x>=0);
                v.push_back(x);
            }
            for(int j=0; j<k and Ok; j++)
            {
                Q.push(v[j]);
            }
        }
        if(Ok)printf("Possible\n");
        else printf("Not possible\n");
    }
    return 0;
}
```

2.5 Articulation Point/Bridge

Articulation point: A vertex in an undirected connected graph is an articulation point (or cut vertex) iff removing it (and edges through it) disconnects the graph. Articulation points represent vulnerabilities in a connected network single points whose failure would split the network into 2 or more disconnected components. They are useful for designing reliable networks. For a disconnected undirected graph, an articulation point is a vertex removing which increases number of connected components. Following are some example graphs with articulation points encircled with red color.

A O(V+E) algorithm to find all Articulation Points:

The idea is to use DFS (Depth First Search). In DFS, we follow vertices in tree form called DFS tree. In DFS tree, a vertex u is parent of another vertex v, if v is discovered

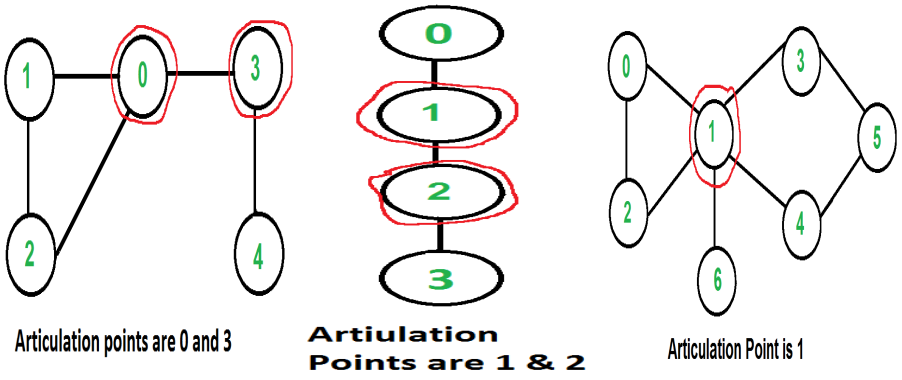

Figure 2.1: Articulation Point

by u (obviously v is an adjacent of u in graph). In DFS tree, a vertex u is articulation point if one of the following two conditions is true.

1. u is root of DFS tree and it has at least two children.
2. u is not root of DFS tree and it has a child v such that no vertex in subtree rooted with v has a back edge to one of the ancestors (in DFS tree) of u.

u is not root of DFS tree and it has a child v such that no vertex in subtree rooted with v has a back edge to one of the ancestors (in DFS tree) of u.

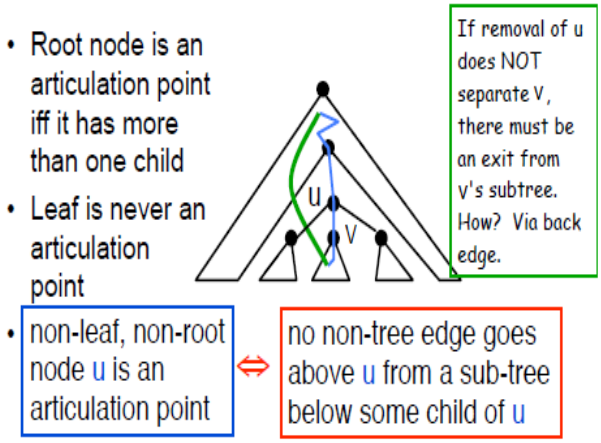

Figure 2.2: Articulation Point

We do DFS traversal of given graph with additional code to find out Articulation Points (APs). In DFS traversal, we maintain a parent[] array where parent[u] stores parent of vertex u. Among the above mentioned two cases, the first case is simple to detect. For every vertex, count children. If currently visited vertex u is root (parent[u] is NIL) and has more than two children, print it. How to handle second case? The second case is trickier. We maintain an array disc[] to store discovery time of vertices. For every node u, we need to find out the earliest visited vertex (the vertex with minimum discovery time) that can be reached from subtree rooted with u. So we maintain an additional array low[] which is defined as follows.

low[u] = min(disc[u], disc[w])
where w is an ancestor of u and there is a back edge from some descendant of u to w.

2.5.1 Find Articulation Point:

```
vector<int>Graph[10000];
bool visit[10000];
int arti[100000];
int discover[100000],Back[100000];
int predfn;
int source;
int child_of_root;
int cnt=0;
void reset()
{
    memset(visit,false,sizeof(visit));
    memset(arti,false,sizeof(arti));
    predfn=child_of_root=0;
}
void articulation(int v)
{
    visit[v]=true;
    predfn++;
    discover[v]=Back[v]=predfn;
    for(int i=0;i<Graph[v].size();i++)
    {
        int w=Graph[v][i];
        if(!visit[w])
        {
            articulation(w);
            Back[v]=min(Back[v],Back[w]);
            if(Back[w]>=discover[v]&&v!=source)
            {
                arti[v]=true;
            }
            else if(v==source)
            {
                child_of_root++;
                if(child_of_root==2)
                {
                    arti[v]=true;
                }
            }
        }
        else
        {
            Back[v]=min(Back[v],discover[w]);
        }
    }
}
```

2.5.2 Find Bridge version 1:

```
vector<int>Graph[200];
int Back[205],Discover[205];
bool visit[205];
bool bridge[205][205];
int brcount;
void reset(int n)
{
```

```
    for(int i=0;i<=n;i++)Graph[i].clear();
    memset(visit,false,sizeof(visit));
    memset(bridge,false,sizeof(false));
    brcount=0;
}
void find_bridge(int u, int parent, int depth)
{
    visit[u] = true;
    Discover[u] = Back[u] = depth;

    for (int i=0 ; i<Graph[u].size() ; i++)
    {
        int v = Graph[u][i];

        if (visit[v] && v!=parent)
        {
            Back[u] = min(Back[u],Discover[v]);
        }
        if (!visit[v])
        {
            find_bridge(v, u, depth+1);
            Back[u] = min(Back[u],Back[v]);
            if (Back[v]>Discover[u])
            {
                brcount++;
                bridge[u][v] = bridge[v][u] = true;
            }
        }
    }

}
```

2.5.3 Find Bridge version 2:

```
void find_bridge(int node, int parent)
{
    discovery_time[node] = bedge[node] = ++T;
    int to, i, connected = adj[node].size();
    for(i = 0; i < connected; i++)
    {
        to = adj[node][i];
        if(to == parent) continue;
        if(!discovery_time[to])
        {
            printf("%d %d\n", node, to);
            find_bridge(to, node);
            bedge[node] = min(bedge[node], bedge[to]);
            if(bedge[to] > discovery_time[node]) printf("%d %d\n", to, node)
    ;
        }
        else if(discovery_time[node] > discovery_time[to])
        {
            printf("%d %d\n", node, to);
            bedge[node] = min(bedge[node], discovery_time[to]);
        }
    }
}
```

2.6 Stable Marriage Problem

```
const int Max=400;
int position[Max][Max];
int Husband[Max];
queue<int>womenlist[Max];
void stable_marage(int N)
{
    mem(Husband,-1);
    queue<int>freeman;
    for(int i=1; i<=N; i++)freeman.push(i);
    while(!freeman.empty())
    {
        int man=freeman.front();
        int woman=womenlist[man].front();
        womenlist[man].pop();
        if(Husband[woman]==-1)
        {
            Husband[woman]=man;
            freeman.pop();
        }
        else
        {
            int herhusband=Husband[woman];
            if(position[woman][man]<position[woman][herhusband])
            {
                Husband[woman]=man;
                freeman.pop();
                freeman.push(herhusband);
            }
        }
    }
}
```

2.7 2 SAT

```
/*
A or B
A =>B
Algo Steps:
1. Construct Graph with A ->B & B ->A Edges
2. Find the Strongly Connected component of Graph & ranked them.
3. Check A & A  is not in the same connected component. If in same group
    then, there is no solution
4.If there is solution exist, then solution will be higher rank between A &
    A node.
*/
#include<bits/stdc++.h>
using namespace std;
#define pb push_back
#define mem(x,y) memset(x,y,sizeof(x))
typedef vector<int> vint;
vint Graph1[5*8000];
vint Graph2[5*8000];
bool visit[5*8000];
int color[5*8000];
vint ans;
struct info
{
    int u,v;
```

```
    info(int u,int v):u(u),v(v) {};
};
vector<info>Edges;
vint topsort;
void init(int N)
{
    Edges.clear();
    for(int i=0; i<=3*N; i++)
    {
        Graph1[i].clear();
        Graph2[i].clear();
    }
}
void construct_graph(int N)
{
    int sz=Edges.size();
    for(int i=0; i<sz; i++)
    {
        int u=Edges[i].u;
        int v=Edges[i].v;
        int a,b,acomp,bcomp;
        if(u>0)
        {
            a=2*u;
            acomp=2*u+1;
        }
        else
        {
            a=2*(-u)+1;
            acomp=2*(-u);
        }
        if(v>0)
        {
            b=2*v;
            bcomp=2*v+1;
        }
        else
        {
            b=2*(-v)+1;
            bcomp=2*(-v);
        }
        Graph1[acomp].pb(b);
        Graph1[bcomp].pb(a);
        Graph2[b].pb(acomp);
        Graph2[a].pb(bcomp);
    }
}
void dfs(int u)
{
    if(visit[u])return;
    visit[u]=true;
    int sz=Graph1[u].size();
    for(int i=0; i<sz; i++)
    {
        int v=Graph1[u][i];
        dfs(v);
    }
    topsort.pb(u);
}
void dfs2(int u,int Rank)
{
```

```
84      if(visit[u])return;
85      visit[u]=true;
86      color[u]=Rank;
87      int sz=Graph2[u].size();
88      for(int i=0; i<sz; i++)
89      {
90          int v=Graph2[u][i];
91          dfs2(v,Rank);
92      }
93  }
94  void SCC(int N)
95  {
96      topsort.clear();
97      mem(visit,false);
98      for(int i=2; i<=2*N+1; i++)
99      {
100         if(visit[i]==false)dfs(i);
101     }
102     int r=1;
103     mem(visit,false);
104     for(int i=topsort.size()-1; i>=0; i--)
105     {
106         if(visit[topsort[i]]==false)dfs2(topsort[i],r++);
107     }
108 }
109 bool is2satpossible(int N)
110 {
111     for(int i=1; i<=N; i++)
112     {
113         int a=2*i;
114         int acomp=2*i+1;
115         if(color[a]==color[acomp])return false;
116     }
117     return true;
118 }
119 void solution(int N)
120 {
121     ans.clear();
122     for(int i=1; i<=N; i++)
123     {
124         int a=2*i;
125         int acomp=2*i+1;
126         if(color[a]>color[acomp])ans.pb(i);
127     }
128 }
129 int main()
130 {
131     int test;
132     scanf("%d",&test);
133     for(int t=1; t<=test; t++)
134     {
135         int M,N;
136         scanf("%d %d",&M,&N);
137         init(N);
138         for(int i=0; i<M; i++)
139         {
140             int u,v;
141             scanf("%d %d",&u,&v);
142             Edges.pb(info(u,v));
143         }
144         construct_graph(N);
```

```
        SCC(N);
        printf("Case %d: ",t);
        if(is2satpossible(N))
        {
            cout<<"Yes"<<endl;
            solution(N);
            int sz=ans.size();
            printf("%d",sz);
            for(int i=0; i<sz; i++)printf(" %d",ans[i]);
            printf("\n");
        }
        else cout<<"No"<<endl;
    }
    return 0;
}
```

2.8 Heavy Light Decomposition

```
#include<bits/stdc++.h>
using namespace std;
#define pp pair<int,int>
#define pb push_back
const int Max=10000;
struct info
{
    int v,cost;
    info(int v=0,int cost=0):v(v),cost(cost) {};
};
vector<pp>edges;
vector<info>Graph[Max+5];
int Tree[5*Max+5],BaseArray[Max+5],SubTreeSize[Max+5];
int ChainHead[Max+5],ChainNum[Max+5],PosInBaseArray[Max+5],ChainNo;
int Level[Max+5],Parent[Max+5],SparseTable[Max+5][16];
int ptr;
void init(int N)
{
    for(int i=0; i<=N; i++)
    {
        Graph[i].clear(),ChainHead[i]=-1;
        for(int j=0; j<=15; j++)SparseTable[i][j]=-1;
    }
    edges.clear();
    ptr=ChainNo=0;
}
void buildSegmentTree(int l,int r,int indx)
{
    if(l==r)
    {
        Tree[indx]=BaseArray[l];
        return;
    }
    int mid=(l+r)>>1;
    int lindx=indx<<1;
    int rindx=lindx|1;
    buildSegmentTree(l,mid,lindx);
    buildSegmentTree(mid+1,r,rindx);
    Tree[indx]=max(Tree[lindx],Tree[rindx]);
}
void updateSegmentTree(int l,int r,int indx,int update_indx,int value)
{
```

```
    if(l==r)
    {
        Tree[indx]=value;
        return;
    }
    int mid=(l+r)>>1;
    int lindx=indx<<1;
    int rindx=lindx|1;
    if(update_indx<=mid)updateSegmentTree(l,mid,lindx,update_indx,value);
    else updateSegmentTree(mid+1,r,rindx,update_indx,value);
    Tree[indx]=max(Tree[lindx],Tree[rindx]);
}
int querySegmentTree(int l,int r,int indx,int x,int y)
{
    if(l>y||r<x)return 0;
    if(x<=l&&y>=r)return Tree[indx];
    int mid=(l+r)>>1;
    int lindx=indx<<1;
    int rindx=lindx|1;
    int c1=0,c2=0;
    if(x<=mid)c1=querySegmentTree(l,mid,lindx,x,y);
    if(y>mid)c2=querySegmentTree(mid+1,r,rindx,x,y);
    return max(c1,c2);
}
void dfs(int from,int u,int depth)
{
    Level[u]=depth;
    Parent[u]=from;
    SubTreeSize[u]=1;
    int sz=Graph[u].size();
    for(int i=0; i<sz; i++)
    {
        int v=Graph[u][i].v;
        if(v==from)continue;
        dfs(u,v,depth+1);
        SubTreeSize[u]+=SubTreeSize[v];
    }
}
void sparseTable(int N)
{
    for(int i=0; i<=N; i++)SparseTable[i][0]=Parent[i];
    for(int j=1; (1<<j)<=N; j++)
    {
        for(int i=0; i<=N; i++)
        {
            if(SparseTable[i][j-1]!=-1)
            {
                int a=SparseTable[i][j-1];
                SparseTable[i][j]=SparseTable[a][j-1];
            }
        }
    }
}
int LCA(int p,int q)
{
    if(Level[p]<Level[q])swap(p,q);
    int Log=log2(Level[p])+1;
    for(int i=Log; i>=0; i--)
    {
        if((Level[p]-(1<<i))>=Level[q])p=SparseTable[p][i];
    }
```

```
104     if(p==q)return p;
105     for(int i=Log; i>=0; i--)
106     {
107         if(SparseTable[p][i]!=-1&&SparseTable[p][i]!=SparseTable[q][i])
108         {
109             p=SparseTable[p][i],q=SparseTable[q][i];
110         }
111     }
112     return Parent[p];
113 }
114 /**
115  * Actual HL-Decomposition part
116  * Initially all entries of chainHead[] are set to -1.
117  * So when ever a new chain is started, chain head is correctly assigned.
118  * As we add a new node to chain, we will note its position in the baseArray
        .
119  * In the first for loop we find the child node which has maximum sub-tree
        size.
120  * The following if condition is failed for leaf nodes.
121  * When the if condition passes, we expand the chain to special child.
122  * In the second for loop we recursively call the function on all normal
        nodes.
123  * chainNo++ ensures that we are creating a new chain for each normal child.
124  **/
125 void heavyLightDecompositon(int from,int curNode,int cost)
126 {
127     if(ChainHead[ChainNo]==-1)ChainHead[ChainNo]=curNode; /// Assign chain
        head
128     ChainNum[curNode]=ChainNo;
129     PosInBaseArray[curNode]=ptr; /// Position of this node in baseArray
        which we will use in Segtree
130     BaseArray[ptr++]=cost;
131     int sc=-1,nextCost;
132     int sz=Graph[curNode].size();
133     for(int i=0; i<sz; i++) /// Loop to find special child
134     {
135         int v=Graph[curNode][i].v;
136         if(v==from)continue;
137         if(sc==-1||SubTreeSize[sc]<SubTreeSize[v])
138         {
139             sc=v;
140             nextCost=Graph[curNode][i].cost;
141         }
142     }
143     if(sc!=-1)heavyLightDecompositon(curNode,sc,nextCost); /// Expand the
        chain
144     for(int i=0; i<sz; i++)
145     {
146         int v=Graph[curNode][i].v;
147         int cost=Graph[curNode][i].cost;
148         if(v==from||sc==v)continue;
149         ChainNo++;
150         heavyLightDecompositon(curNode,v,cost);
151     }
152 }
153 void updateTree(int ith,int val)
154 {
155     pp a=edges[ith];
156     int u=a.first,v=a.second;
157     int indx=PosInBaseArray[u];
158     if(Level[u]<Level[v])indx=PosInBaseArray[v];
```

```
159     updateSegmentTree(0,ptr-1,1,indx,val);
160 }
161 /**
162  * query_up:
163  * It takes two nodes u and v, condition is that v is an ancestor of u
164  * We query the chain in which u is present till chain head, then move to
        next chain up
165  * We do that way till u and v are in the same chain, we query for that part
         of chain and break
166  **/
167 int queryUp(int u,int v)
168 {
169     if(u==v)return 0;
170     int uchain,vchain=ChainNum[v],ans=-1;
171     while(true)
172     {
173         uchain=ChainNum[u];
174         if(uchain==vchain)
175         {
176             if(u==v)        /// Both u and v are in the same chain, so we
        need to query from u to v, update answer and break.
177                 break;      /// We break because we came from u up till v,
        we are done
178             ans=max(ans,querySegmentTree(0,ptr-1,1,PosInBaseArray[v]+1,
        PosInBaseArray[u]));
179             break;
180         }
181         int uchainhead=ChainHead[uchain];
182         ans=max(ans,querySegmentTree(0,ptr-1,1,PosInBaseArray[uchainhead],
        PosInBaseArray[u]));
183         /// Above is call to segment tree query function. We do from
        chainHead of u till u. That is the whole chain from
184         u=Parent[uchainhead];
185     }
186     return ans;
187 }
188 int queryTree(int u,int v)
189 {
190     int lca=LCA(u,v);
191     return max(queryUp(u,lca),queryUp(v,lca));
192 }
193 int main()
194 {
195     int test;
196     cin>>test;
197     while(test--)
198     {
199         int N;
200         cin>>N;
201         init(N);
202         for(int i=0; i<N-1; i++)
203         {
204             int u,v,c;
205             cin>>u>>v>>c;
206             u--,v--;
207             Graph[u].pb(info(v,c));
208             Graph[v].pb(info(u,c));
209             edges.pb(pp(u,v));
210         }
211         dfs(-1,0,0);
212         sparseTable(N);
```

```
213         heavyLightDecompositon(-1,0,-1);
214         buildSegmentTree(0,ptr-1,1);
215         string ch;
216         int x,y;
217         while(true)
218         {
219             cin>>ch;
220             if(ch[0]=='D')break;
221             cin>>x>>y;
222             if(ch[0]=='Q')printf("%d\n",queryTree(x-1,y-1));
223             else if(ch[0]=='C')updateTree(x-1,y);
224         }
225     }
226     return 0;
227 }
```

Chapter 3

Flow networks, matching

3.1 Max Flow

```
#include<bits/stdc++.h>
using namespace std;
#define pb push_back
#define MN 1000
typedef vector< vector<int> > vint2D;
const int inf=(1<<29);
vint2D graph;
int Cost[MN][MN];
int parent[MN+5];
int flow;
void init(int N)
{
    graph=vint2D(N);
    memset(Cost,0,sizeof(Cost));
}
void AddEdge(int u,int v,int cost)
{
    graph[u].pb(v);
    graph[v].pb(u);
    Cost[u][v]+=cost;
    Cost[v][u]+=cost;
}
bool augmenting_path(int source,int sink)
{
    memset(parent,-1,sizeof(parent));
    queue<int>Q;
    Q.push(source);
    while(!Q.empty())
    {
        int u=Q.front();
        Q.pop();
        int sz=graph[u].size();
        for(int i=0; i<sz; i++)
        {
            int v=graph[u][i];
            if(parent[v]==-1 and Cost[u][v]>0)
            {
                parent[v]=u;
                Q.push(v);
                if(v==sink)return true;
            }
        }
```

```
    }
    return false;
}
void path(int v,int source)
{
    int u=parent[v];
    flow=min(flow,Cost[u][v]);
    if(source!=u)path(u,source);
    Cost[u][v]-=flow;
    Cost[v][u]+=flow;
    return;
}
int max_flow(int source,int sink)
{
    int ret=0;
    while(augmenting_path(source,sink))
    {
        flow=inf;
        path(sink,source);
        ret+=flow;
    }
    return ret;
}
int main()
{
    int test;
    scanf("%d",&test);
    while(test--)
    {
        int P,S,C,M;
        scanf("%d %d %d %d",&P,&S,&C,&M);
        init(P+S+5);
        int superSource=0,SuperSikn=P+S+1;
        for(int i=1; i<=P; i++)AddEdge(superSource,i,1);
        for(int i=1; i<=S; i++)AddEdge(P+1,SuperSikn,C);
        for(int i=0; i<M; i++)
        {
            int x,y;
            scanf("%d %d",&x,&y);
            AddEdge(x,P+y,(1<<30));
        }
        printf("%d\n",max_flow(superSource,SuperSikn));
    }
    return 0;
}
```

3.2 Maximum Bipartite Matching

```
bool khun(int u)
{
    int sz=graph[u].size();
    for(int i=0; i<sz; i++)
    {
        int v=graph[u][i];
        if(seen[v])continue;
        seen[v]=true;
        if(Right[v]==-1||khun(Right[v]))
        {
            Right[v]=u;
            Left[u]=v;
            return true;
        }
    }
    return false;
}
int bipartite_matching(int m)
{
    mem(Left,-1);
    mem(Right,-1);
    int cnt=0;
    for(int i=0; i<m; i++)
    {
        mem(seen,false);
        if(khun(i))cnt++;
    }
    return cnt;
}
```

Chapter 4

Dynamic programming

4.1 Longest Increasing sub sequence

4.1.1 LIS1

```
#include<bits/stdc++.h>
using namespace std;
const int inf = 2000000000; // a large value as infinity

int n; // the number of items in the sequence
int Sequence[32]; // the sequence of integers
int L[32]; // L[] as described in the algorithm
int I[32]; // I[] as described in the algorithm
vector<int>LIS;
void takeInput()
{
    scanf("%d", &n); // how many numbers in the sequence ?
    for( int i = 0; i < n; i++ ) // take the sequence
        scanf("%d", &Sequence[i]);
}

int LisNlogK()   // which runs the NlogK LIS algorithm
{
    int i; // auxilary variable for iteration

    I[0] = -inf; // I[0] = -infinite
    for( i = 1; i <= n; i++ ) // observe that i <= n are given
        I[i] = inf; // I[1 to n] = infinite

    int LisLength = 0; // keeps the maximum position where a data is
    inserted

    for( i = 0; i < n; i++ )   // iterate left to right
    {
        int low, high, mid; // variables to perform binary search
        low = 0; // minimum position where we to put data in I[]
        high = LisLength; // the maximum position

        while( low <= high )   // binary search to find the correct position
        {
            mid = ( low + high ) / 2;
            if( I[mid] < Sequence[i] )
                low = mid + 1;
            else
                high = mid - 1;
```

```
        }
        // observe the binary search carefully, when the binary search ends
        // low > high and we put our item in I[low]
        I[low] = Sequence[i];
        L[i]=low;
        if( LisLength < low ) // LisLength contains the maximum position
            LisLength = low;
    }


    i=0;
    for(int j=0; j<n; j++)
    {
        if(L[j]>L[i])
        {
            i=j;
        }
    }
    LIS.push_back(Sequence[i]);
    for(int j=i-1; j>=0; j--)
    {
        if(Sequence[j]<Sequence[i]&&L[j]==L[i]-1)
        {
            i=j;
            LIS.push_back(Sequence[j]);
        }
    }
    reverse(LIS.begin(),LIS.end());
    return LisLength; // return the result
}
int main()
{
    takeInput();
    int result = LisNlogK();
    printf("The LIS length is %d\n", result);
    for(int i=0; i<LIS.size(); i++)printf("%d ",LIS[i]);
    return 0;
}
```

4.1.2 LIS(Stanford)

```
// Given a list of numbers of length n, this routine extracts a
// longest increasing subsequence.
//
// Running time: O(n log n)
//
//   INPUT: a vector of integers
//   OUTPUT: a vector containing the longest increasing subsequence

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
typedef vector<int> VI;
typedef pair<int,int> PII;
typedef vector<PII> VPII;

#define STRICTLY_INCREASNG

VI LongestIncreasingSubsequence(VI v)
{
```

```
    VPII best;
    VI dad(v.size(), -1);

    for (int i = 0; i < v.size(); i++)
    {
#ifdef STRICTLY_INCREASNG
        PII item = make_pair(v[i], 0);
        VPII::iterator it = lower_bound(best.begin(), best.end(), item);
        item.second = i;
#else
        PII item = make_pair(v[i], i);
        VPII::iterator it = upper_bound(best.begin(), best.end(), item);
#endif
        if (it == best.end())
        {
            dad[i] = (best.size() == 0 ? -1 : best.back().second);
            best.push_back(item);
        }
        else
        {
            dad[i] = dad[it->second];
            *it = item;
        }
    }

    VI ret;
    for (int i = best.back().second; i >= 0; i = dad[i])
        ret.push_back(v[i]);
    reverse(ret.begin(), ret.end());
    return ret;
}
```

4.2 Edit Distance

```
#include<bits/stdc++.h>
using namespace std;
int dp[88][88];
int N,M,step;
char S1[88],S2[88];
int solve(int i,int j)
{
    if(i==N and j==M)return 0;
    if(i==N)return M-j;
    if(j==M)return N-i;
    int &ret=dp[i][j];
    if(ret!=-1)return ret;
    ret=(1<<28);
    if(S1[i]==S2[j])ret=solve(i+1,j+1);
    else
    {
        ret=min(ret,solve(i,j+1)+1);
        ret=min(ret,solve(i+1,j)+1);
        ret=min(ret,solve(i+1,j+1)+1);
    }
    return ret;
}
void pathPrint(int i,int j,int del,int ins,int st)
{
    if(i==N&&j==M) return ;
    if(i==N)
```

```
    {
        for(int k=j; k<M; k++,i++)
        {
            printf("%d Insert %d,%c\n",st++,i-del+1+ins,S2[k]);
        }
        return ;
    }
    if(j==M)
    {
        for(; i<N; i++)
        {
            printf("%d Delete %d\n",st++,i-del+1+ins);
            del++;
        }
        return ;
    }
    int ret = solve(i,j);
    int tmp;
    if(S1[i]==S2[j])
    {
        tmp=solve(i+1,j+1);
        if(ret==tmp)
        {
            pathPrint(i+1,j+1,del,ins,st);
            return ;
        }
    }
    tmp=solve(i,j+1)+1;
    if(tmp==ret)
    {
        printf("%d Insert %d,%c\n",st,i-del+1+ins,S2[j]);
        pathPrint(i,j+1,del,ins+1,st+1);
        return ;
    }
    tmp=solve(i+1,j)+1;
    if(tmp==ret)
    {
        printf("%d Delete %d\n",st,i-del+1+ins);
        pathPrint(i+1,j,del+1,ins,st+1);
        return ;
    }
    tmp=solve(i+1,j+1)+1;
    if(tmp==ret)
    {
        printf("%d Replace %d,%c\n",st,i-del+1+ins,S2[j]);
        pathPrint(i+1,j+1,del,ins,st+1);
        return ;
    }
    return ;
}
int main()
{
    bool New=false;
    while(gets(S1))
    {
        gets(S2);
        if(New)printf("\n");
        New=true;
        N=strlen(S1);
        M=strlen(S2);
        memset(dp,-1,sizeof(dp));
```

```
        step=solve(0,0);
        printf("%d\n",step);
        pathPrint(0,0,0,0,1);
    }
    return 0;
}
```

4.3 Distinct Sub Sequence

```
LLD solve()
{
    dp[0]=1;
    mem(Last_occurance,0);
    for(int i=1;s[i];i++)
    {
        dp[i]=2*dp[i-1];
        int x=s[i]-'A';
        if(Last_occurance[x])dp[i]-=dp[Last_occurance[x]-1];
        Last_occurance[x]=i;
        dp[i]=dp[i];
    }
    return dp[strlen(s+1)];
}
```

4.4 Knuth Optimization

```
#include<bits/stdc++.h>
using namespace std;
int dp[505][505];
int positionK[505][505];
int Csum[505];
int main()
{
    int N;
    while(scanf("%d",&N)==1)
    {
        for(int i=1; i<=N; i++)
        {
            scanf("%d",&Csum[i]);
            Csum[i]+=Csum[i-1];
        }
        for(int L=1; L<=N; L++)
        {
            for(int i=1; i<=N; i++)
            {
                int j=L+i-1;
                if(j>N)continue;
                if(L==1)
                {
                    dp[i][j]=0;
                    positionK[i][j]=i;
                }
                else
                {
                    dp[i][j]=(1<<28);
                    for(int k=positionK[i][j-1]; k<=positionK[i+1][j]; k++)
                    {
                        int cost=dp[i][k-1];
                        cost+=dp[k+1][j];
                        cost+=(Csum[j]-Csum[k]);
```

```
                            cost+=(Csum[k-1]-Csum[i-1]);
                            if(cost<dp[i][j])
                            {
                                dp[i][j]=cost;
                                positionK[i][j]=k;
                            }
                        }
                    }
                }
            }
            printf("%d\n",dp[1][N]);
        }
        return 0;
}
```

4.5 Max Sum

4.5.1 2D Max Sum 1

Complexity $O(n^4)$

```
precalculation:
for(int i=1;i<=N;i++)
{
    for(int j=1;j<=N;j++)
    {
        Mat[i][j]=Mat[i-1][j]+Mat[i][j-1]-Mat[i-1][j-1]+A[i][j];
    }
}
Query:
Sum=Mat[x2][y2]-Mat[x1-1][y2]-Mat[x2][y1-1]+Mat[x1-1][y1-1]
```

4.5.2 2D Max sum 2

Complexity n^3

```
int arr[SZ][SZ];
int input(void);
int max_sum(int n);
int main()
{
    int test, n;
    scanf("%d", &test);
    while(test--)
    {
        n = input();
        printf("%d\n", max_sum(n));
        if(test)
            printf("\n");
    }
    return 0;
}

int input(void)
{
    int i, len, r = 1, n;
    char str[26];
    scanf(" %s", &str);
    len = strlen(str);
    for(i = 1; i <= len; i++)
    {
```

```
        n = str[i - 1] - '0';
        if(n)
            arr[r][i] = n;
        else
            arr[r][i] = -700;
    }
    r++;
    while(r <= len)
    {
        scanf("%s", &str);
        for(i = 1; i <= len; i++)
    {
        n = str[i - 1] - '0';
        if(n)
            arr[r][i]= n;
        else
            arr[r][i] = -700;
    }
        r++;
    }
    return len;
}

int max_sum(int n)
{
    int i, j, c1, c2, r, sum = 0,m = 0;
    for(i = 0; i <= n; i++)
        arr[i][0] = arr[0][i] = 0;
    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            arr[i][j] += arr[i][j - 1];
    for(c1 = 1; c1 <= n; c1++)
        for(c2 = c1; c2 <= n; c2++)
        {
            sum = 0;
            for(r = 1; r <= n; r++)
            {
                sum += arr[r][c2] - arr[r][c1 - 1];
                if(sum < 0)
                    sum = 0;
                else if(sum > m)
                    m = sum;
            }
        }
    return m;
}
```

4.5.3 3D Max sum

Complexity $O(n^4)$

Cumulative Sum formula: S(x,y,z)=value(x,y,z) + S(x-1,y,z) + S(x,y-1,z)+S(x,y,z-1)-S(x-1,y-1,z)-S(x,y-1,z-1)-S(x-1,y,z-1) +S(x-1,y-1,z-1)

Query Formula: S([x1,y1,z1] to [x2,y2,z2])=S(x2,y2,z2)+S(x1,y1,z2)+S(x1,y2,z1)+S(x1,y1,z1)-S(x2,y2,z1)-S(x2,y1,z2)-S(x1,y2,z2) -S(x1,y1,z1)

```
for(int i = n; i <= N; i++)
```

```
{

    for(int j = n; j <= N; j++)
    {
        for(int k = n; k <= N; k++)
        {
            int x2=i,y2=j,z2=k;
            int x1=i-n,y1=j-n,z1=k-n;
            int temp=[Query formul]
                    ans=max(ans,temp);
        }
    }
}
```

4.6 Minimum Vertex Cover

```
#define MAXN 100002
int dp[MAXN][5];
int par[MAXN];
vector<int>edges[MAXN];
int f(int u,int isGuard)
{
    if(edges[u].size()==0)return 0;
    if(dp[u][isGuard]!=-1) return dp[u][isGuard];
    int sum=0;
    for(int i=0; i<(int)edges[u].size(); i++)
    {
        int v=edges[u][i];
        if(v!=par[u])
        {
            par[v]=u;
            if(isGuard==0) sum+=f(v,1);
            else sum+=min(f(v,1),f(v,0));
        }
    }
    return dp[u][isGuard]=sum+isGuard;
}
int main()
{
    memset(dp,-1,sizeof(dp));
    int n;
    scanf("%d",&n);
    for(int i=1; i<n; i++)
    {
        int u,v;
        scanf("%d%d",&u,&v);
        edges[u].push_back(v);
        edges[v].push_back(u);
    }
    int ans=0;
    ans=min(f(1,1),f(1,0));
    printf("%d\n",ans);
    return 0;
}
```

4.7 Nth Permutation Of a String

```
#include<bits/stdc++.h>
using namespace std;
#define mem(x,y) memset(x,y,sizeof(x))
typedef long long LLD;
LLD FAC[22];
int Freq[27];
string S;
void FACTORIAL()
{
    FAC[0]=1;
    for(int i=1; i<=20; i++)FAC[i]=FAC[i-1]*i;
}
LLD Permutation(int n)
{
    LLD per=FAC[n];
    for(int i=0; i<26; i++)per/=FAC[Freq[i]];
    return per;
}
string NthPermutation(int nth)
{
    string ans="";
    int len=S.size();
    while(len)
    {
        LLD upto=0;
        for(int i=0; i<26; i++)
        {
            if(Freq[i])
            {
                Freq[i]-=1;
                LLD now=Permutation(len-1);
                if(upto+now>=nth)
                {
                    nth-=upto;
                    ans+=('a'+i);
                    len--;
                    break;
                }
                else
                {
                    upto+=now;
                    Freq[i]+=1;
                }
            }
        }
    }
    return ans;
}
int main()
{
    FACTORIAL();
    int test;
    scanf("%d",&test);
    for(int t=1; t<=test; t++)
    {
        mem(Freq,0);
        char inp[100];
        LLD Nth;
```

```
59          scanf("%s %lld",inp,&Nth);
60          S=inp;
61          for(int i=0; inp[i]; i++)Freq[inp[i]-'a']++;
62          string Ans="Impossible";
63          if(Permutation(S.size())>=Nth)
64          {
65              Ans=NthPermutation(Nth);
66          }
67          printf("Case %d: %s\n",t,Ans.c_str());
68      }
69      return 0;
70  }
```

Chapter 5

Strings

5.1 KMP

Tutorial

```
#include<bits/stdc++.h>
using namespace std;
char TXT[10000000],ptr[10000000];
vector<int> compute_prefix(const char *p)
{
    int m=strlen(p+1);
    vector<int> prefix(m+1);
    prefix[1]=0;
    int k=0;
    for(int i=2; i<=m; i++)
    {
        while(k>0 and p[k+1]!=p[i])k=prefix[k];
        if(p[k+1]==p[i])k=k+1;
        prefix[i]=k;
    }
    return prefix;
}
vector<int> KMP_match(const char *txt,const char *ptrn)
{
    int n=strlen(txt+1);
    int m=strlen(ptrn+1);
    vector<int> Prefix=compute_prefix(ptrn);
    vector<int>Match_position;
    int q=0;
    for(int i=1; i<=n; i++)
    {
        while(q>0 and ptrn[q+1]!=txt[i])q=Prefix[q];
        if(ptrn[q+1]==txt[i])q=q+1;
        if(q==m)
        {
            Match_position.push_back(i-m);
            q=Prefix[q];
        }
    }
    return Match_position;
}
int main()
{
    scanf("%s %s",TXT+1,ptr+1);
    vector<int> Match_position=KMP_match(TXT,ptr);
    for(int i=0; i<Match_position.size(); i++)
```

```
    {
        if(!i)printf("%d",Match_position[i]);
        else printf(" %d",Match_position[i]);
    }
    return 0;
}
```

5.2 Aho Corasick

Algorithm: We can divide the algorithm into 2 steps:

1. constructing a finite state automaton
    - constructing a Trie with input words
    - finding the right fall function for each node of trie
2. moving throuhg automaton - reading each character from input, storing the results. If an algorithm finds right words in some step, (more words can end in the same place), the nodes light up and the numbers by their indices are incremented.

5.2.1 Aho Corasick with Dynamic Trie

```
#include<bits/stdc++.h>
using namespace std;
#define Max 26
int getID(char c)
{
    return c>='a'?c-'a':c-'A';
}
char inp[1000005];
char text[1000005];
int ans[5000];
map<string,int>Map;
vector<int>v;
struct Trie
{
    Trie *next[26],*fail;
    int stringMap;
    Trie()
    {
        stringMap=0;
        for(int i=0; i<Max; i++)next[i]=NULL;
        fail=NULL;
    }
};
Trie *root;
void Insert(const char *str,int M)
{
    Trie *p=root;
    for(int i=0; str[i]; i++)
    {
        int id=getID(str[i]);
        if(p->next[id]==NULL)p->next[id]=new Trie();
        p=p->next[id];
    }
    p->stringMap=M;
}
```

```
void computeFailure()
{
    Trie *u,*prefix;
    queue<Trie*>Q;
    Q.push(root);
    while(!Q.empty())
    {
        u=Q.front(); ///Take a new node
        Q.pop();
        for(int i=0; i<Max; i++)
        {
            if(u->next[i]!=NULL) ///select fail position of ith node of parent u
            {
                prefix=u->fail; /// Going to u node fail position/ prefix position
                while(prefix!=NULL)
                {
                    if(prefix->next[i]!=NULL) ///if match found
                    {
                        u->next[i]->fail=prefix->next[i];
                        break;
                    }
                    prefix=prefix->fail; /// match not found, going to upper child prefix position
                }
                if(prefix==NULL)u->next[i]->fail=root;
                Q.push(u->next[i]);
            }
        }
    }
}
void AhoCorasick(const char *str)
{
    Trie *p=root;
    for(int i=0; str[i]; i++)
    {
        int id=getID(str[i]);
        while(p->next[id]==NULL&&p!=root)p=p->fail;
        if(p->next[id]!=NULL)p=p->next[id];
        Trie *tp=p;
        while(tp!=root)
        {
            if(tp->stringMap>0)ans[tp->stringMap]++;
            tp=tp->fail;
        }
    }
}
void Delete(Trie *u)
{
    if(u==NULL)return;
    for(int i=0; i<Max; i++)Delete(u->next[i]);
    delete u;
}

int main()
{
    int test;
    scanf("%d",&test);
    for(int t=1; t<=test; t++)
    {
```

```
        Map.clear();
        v.clear();
        memset(ans,0,sizeof(ans));
        root=new Trie();
        int N;
        scanf("%d",&N);
        scanf("%s",text);
        int cnt=1;
        for(int i=0; i<N; i++)
        {
            scanf("%s",inp);
            if(Map.find(inp)==Map.end())Map[inp]=cnt++;
            Insert(inp,Map[inp]);
            v.push_back(Map[inp]);
        }
        computeFailure();
        AhoCorasick(text);
        printf("Case %d:\n",t);
        for(int i=0; i<N; i++)
        {
            printf("%d\n",ans[v[i]]);
        }
        Delete(root);
    }
    return 0;
}
```

5.2.2 Aho Corasick with Static Trie

```
#include<bits/stdc++.h>
using namespace std;
#define root 0
#define NuLL -1
#define Max 248878
#define MC 26
int ans[10000];
char text[1000005];
char inp[100000];
map<string,int>Map;
vector<int> v;
int getID(const char c)
{
    return c>='a'?c-'a':c-'A';
}
struct Trie
{
    struct node
    {
        int Next[26],fail;
        int stringMap;
        void clear()
        {
            memset(Next,-1,sizeof(Next));
            fail=-1;
            stringMap=0;
        }
    } T[Max];
    int ptr;
    void clear()
    {
        ptr=1;
```

```
33          T[0].clear();
34      }
35      void Insert(char *str,int M)
36      {
37          int p=0;
38          for(int i=0; str[i]; i++)
39          {
40              int id=getID(str[i]);
41              if(T[p].Next[id]==-1)
42              {
43                  T[p].Next[id]=ptr;
44                  T[ptr++].clear();
45              }
46              int q=p;
47              p=T[p].Next[id];
48          }
49          T[p].stringMap=M;
50      }
51      void ComputeFailure()
52      {
53          queue<int>Q;
54          Q.push(root);
55          int u,prefix;
56          while(!Q.empty())
57          {
58              u=Q.front();
59              Q.pop();
60              for(int i=0; i<MC; i++)
61              {
62                  if(T[u].Next[i]!=NuLL)
63                  {
64                      int now=T[u].Next[i];
65                      prefix=T[u].fail;
66                      while(prefix!=NuLL)
67                      {
68                          if(T[prefix].Next[i]!=NuLL)
69                          {
70                              T[now].fail=T[prefix].Next[i];
71                              break;
72                          }
73                          prefix=T[prefix].fail;
74                      }
75                      if(prefix==NuLL)T[now].fail=root;
76                      Q.push(now);
77                  }
78              }
79          }
80      }
81  };
82  void AhoCorasick(const Trie &A, const char *str)
83  {
84      int p=root;
85      for(int i=0; str[i]; i++)
86      {
87          int id=getID(str[i]);
88          while(A.T[p].Next[id]==NuLL&&p!=root)p=A.T[p].fail;
89          if(p!=NuLL&&A.T[p].Next[id]!=NuLL)p=A.T[p].Next[id];
90          int tp=p;
91          while(tp!=root)
92          {
93              if(A.T[tp].stringMap>0)ans[A.T[tp].stringMap]++;
```

```
            tp=A.T[tp].fail;
        }
    }
}
Trie A;
int main()
{
#ifdef _ANICK_
    freopen("input.txt","r",stdin);
#endif // _ANICK_
    int test;
    scanf("%d",&test);
    for(int t=1; t<=test; t++)
    {
        Map.clear();
        v.clear();
        memset(ans,0,sizeof(ans));
        A.clear();
        int N;
        scanf("%d",&N);
        scanf("%s",text);
        int cnt=1;
        for(int i=0; i<N; i++)
        {
            scanf("%s",inp);
            if(Map.find(inp)==Map.end())Map[inp]=cnt++;
            A.Insert(inp,Map[inp]);
            v.push_back(Map[inp]);
        }
        A.ComputeFailure();
        AhoCorasick(A,text);
        printf("Case %d:\n",t);
        for(int i=0; i<N; i++)
        {
            printf("%d\n",ans[v[i]]);
        }
    }
    return 0;
}
```

5.3 Manacher's Algorithm

```
#include<bits/stdc++.h>
using namespace std;
string s, t;
char str[1000005];
void prepare_string()
{
    int i;
    t="^#";
    for(i=0; i<s.size(); i++)
        t+=s[i],t+="#";
    t+="$";
}

int manacher()
{
    prepare_string();
    int P[t.size()],c=0,r=0,i,i_mirror,n=t.size()-1;
    for(i=1; i<n; i++)
```

```
19      {
20          i_mirror=(2*c)-i;
21          P[i]=r>i?min(r-i,P[i_mirror]):0;
22          while(t[i+1+P[i]]==t[i-1-P[i]]) P[i]++;
23          if(i+P[i]>r)
24          {
25              c=i;
26              r=i+P[i];
27          }
28      }
29      return *max_element(P+1,P+n);
30  }
31  int main()
32  {
33      int kase=1;
34      while(scanf(" %s", str)&&str[0]!= 'E')
35      {
36          s=str;
37          printf("Case %d: %d\n", kase++, manacher());
38      }
39      return 0;
40  }
```

Chapter 6

Math

6.1 Reduce Ratio

$\left(\frac{A}{B}\right)$ ratio reduce to $\left(\frac{x}{y}\right)$

```
int main()
{
    int A,B,x,y;
    cin>>A>>B>>x>>y;
    int g=__gcd(x,y);
    x/=g,y/=g;
    int t=min(A/x,B/y);
    cout<<x*t<<" "<<y*t<<endl;
    return 0;
}
```

6.2 Floyd's Cycle Finding algorithm

```
#include<bits/stdc++.h>
using namespace std;
#define pp pair<int,int>
int Z,L,M,I;
int f(int L)
{
    return (Z*L+I)%M;
}
pp CycleFinding()
{
    ///L here initial seed
    int hare,tortoise,lambda,meu;
    bool cyclefind=false;
    hare=tortoise=L;
    while(!cyclefind)
    {
        tortoise=f(tortoise);
        hare=f(hare);
        hare=f(hare);
        if(hare==tortoise)cyclefind=true;
    }
    hare=L;
    meu=0;
    while(hare!=tortoise)
    {
```

```
        meu++;
        hare=f(hare);
        tortoise=f(tortoise);
    }
    int i=0;
    hare=L;
    while(i<=meu)
    {
        i++;
        hare=f(hare);
    }
    tortoise=f(hare);
    lambda=1;
    while(hare!=tortoise)
    {
        tortoise=f(tortoise);
        lambda++;
    }
    return {meu,lambda}; ///meu is starting index and lambda is cycle length
}
int main()
{
    int t=1;
    while(scanf("%d %d %d %d",&Z,&I,&M,&L) and (Z or I or M or L))
    {
        pp a=CycleFinding();
        cout<<"Cycle starts from index "<<a.first<<"\nCycle length is "<<a.
    second<<endl;
    }
    return 0;
}
```

6.3 Angle between clock hand

```
double angle(int h, int m)
{
    double hAngle = 0.5D * (h * 60 + m);
    double mAngle = 6 * m;
    double angle = abs(hAngle - mAngle);
    angle = min(angle, 360 - angle);
    return angle;
}
```

6.4 Big Integer

```
struct Bigint
{
    // representations and structures
    string a; // to store the digits
    int sign; // sign = -1 for negative numbers, sign = 1 otherwise

    // constructors
    Bigint() {} // default constructor
    Bigint( string b )
    {
        (*this) = b;    // constructor for string
    }
    Bigint(long long num)
    {
        if(num<0)sign=-1;
```

```
        else sign=1;
        if(num==0)a.push_back('0');
        while(num)
        {
            a.push_back(num%10+'0');
            num/=10;
        }
    }
    // some helpful methods
    int size()   // returns number of digits
    {
        return a.size();
    }
    Bigint inverseSign()   // changes the sign
    {
        sign *= -1;
        return (*this);
    }
    Bigint normalize( int newSign )   // removes leading 0, fixes sign
    {
        for( int i = a.size() - 1; i > 0 && a[i] == '0'; i-- )
            a.erase(a.begin() + i);
        sign = ( a.size() == 1 && a[0] == '0' ) ? 1 : newSign;
        return (*this);
    }

    // assignment operator
    void operator = ( string b )   // assigns a string to Bigint
    {
        a = b[0] == '-' ? b.substr(1) : b;
        reverse( a.begin(), a.end() );
        this->normalize( b[0] == '-' ? -1 : 1 );
    }

    // conditional operators
    bool operator < ( const Bigint &b ) const   // less than operator
    {
        if( sign != b.sign ) return sign < b.sign;
        if( a.size() != b.a.size() )
            return sign == 1 ? a.size() < b.a.size() : a.size() > b.a.size()
    ;
        for( int i = a.size() - 1; i >= 0; i-- ) if( a[i] != b.a[i] )
                return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
        return false;
    }
    bool operator == ( const Bigint &b ) const   // operator for equality
    {
        return a == b.a && sign == b.sign;
    }
// mathematical operators
    Bigint operator + ( Bigint b )   // addition operator overloading
    {
        if( sign != b.sign ) return (*this) - b.inverseSign();
        Bigint c;
        for(int i = 0, carry = 0; i<a.size() || i<b.size() || carry; i++ )
        {
            carry+=(i<a.size() ? a[i]-48 : 0)+(i<b.a.size() ? b.a[i]-48 : 0)
    ;
            c.a += (carry % 10 + 48);
            carry /= 10;
        }
```

```
75          return c.normalize(sign);
76      }
77      Bigint operator - ( Bigint b )   // subtraction operator overloading
78      {
79          if( sign != b.sign ) return (*this) + b.inverseSign();
80          int s = sign;
81          sign = b.sign = 1;
82          if( (*this) < b ) return ((b - (*this)).inverseSign()).normalize(-s)
    ;
83          Bigint c;
84          for( int i = 0, borrow = 0; i < a.size(); i++ )
85          {
86              borrow = a[i] - borrow - (i < b.size() ? b.a[i] : 48);
87              c.a += borrow >= 0 ? borrow + 48 : borrow + 58;
88              borrow = borrow >= 0 ? 0 : 1;
89          }
90          return c.normalize(s);
91      }
92      Bigint operator * (Bigint b )   // multiplication operator overloading
93      {
94          Bigint c("0");
95          for( int i = 0, k = a[i] - 48; i < a.size(); i++, k = a[i] - 48 )
96          {
97              while(k--) c = c + b; // ith digit is k, so, we add k times
98              b.a.insert(b.a.begin(), '0'); // multiplied by 10
99          }
100         return c.normalize(sign * b.sign);
101     }
102     Bigint operator / ( Bigint b )   // division operator overloading
103     {
104         if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
105         Bigint c("0"), d;
106         for( int j = 0; j < a.size(); j++ ) d.a += "0";
107         int dSign = sign * b.sign;
108         b.sign = 1;
109         for( int i = a.size() - 1; i >= 0; i-- )
110         {
111             c.a.insert( c.a.begin(), '0');
112             c = c + a.substr( i, 1 );
113             while( !( c < b ) ) c = c - b, d.a[i]++;
114         }
115         return d.normalize(dSign);
116     }
117     Bigint operator % ( Bigint b )   // modulo operator overloading
118     {
119         if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
120         Bigint c("0");
121         b.sign = 1;
122         for( int i = a.size() - 1; i >= 0; i-- )
123         {
124             c.a.insert( c.a.begin(), '0');
125             c = c + a.substr( i, 1 );
126             while( !( c < b ) ) c = c - b;
127         }
128         return c.normalize(sign);
129     }
130
131     // output method
132     void print()
133     {
134         if( sign == -1 ) putchar('-');
```

```
135         for( int i = a.size() - 1; i >= 0; i-- ) putchar(a[i]);
136         puts("");
137     }
138 };
```

Chapter 7

Game Theory

7.1 Min Max Algorithm

```
/**
    There is N stone of pile, every player in each move can remove Move[i]
    stone. If Any move there is no Stone remain
    these player will be lost.
**/
#include<bits/stdc++.h>
using namespace std;
int Move[100],k,n;
int dp[1000000+5];
bool is_valid(int x)
{
    return x>=0;
}
bool solve(int sum)
{
    if(sum==0)return false;
    if(dp[sum]!=-1)return dp[sum];
    for(int i=1; i<=k; i++)
    {
        if(is_valid(sum-Move[i]) and !solve(sum-Move[i]))return dp[sum]=true;
    }
    return dp[sum]=false;
}
int main()
{
    while(scanf("%d",&n)==1)
    {
        scanf("%d",&k);
        for(int i=1; i<=k; i++)
        {
            scanf("%d",&Move[i]);
        }
        memset(dp,-1,sizeof(dp));
        if(solve(n))printf("A\n");
        else printf("B\n");
    }
    return 0;
}
```

7.2 Grundy Number

```
const int MX = 1000005 ;
int grunday[MX] , rev[6] = { 1 , 4 , 27, 256, 3125, 46656} ; // in a single
    move n*n can be removed
int Cal(int x)
{
    if( x == 0 ) return 0 ; // base case , grunday is always zero
    int &ret = grunday[x];
    if( ret != -1 ) return ret;
    set <int> s ;
    int i ;
    for (i = 0 ;  i < 6 ; i++ )
    {
        if( x - rev[i] >= 0 )
            s.insert(Cal(x-(rev[i])));
    }
    int ans = 0 ;
    while(s.find(ans) != s.end()) ans++;
    return ret = ans ;
}
int main()
{
    // ios_base::sync_with_stdio(0); cin.tie(0);
    memo(grunday,-1);
    int cs , t ;
    scanf("%d",&t);
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        int n , i , ans = 0 , x ;
        scanf("%d",&n);
        for ( i = 0 ; i < n ; i++ )
        {
            scanf("%d",&x);
            ans ^= Cal(x);
        }
        if( ans ) puts("Little Chef"); // first mover win
        else puts("Head Chef");
    }

    return 0;
}
```

Chapter 8

Number Theory

8.1 Greatest common divisor (GCD)

```
int gcd(int a, int b)
{
    return b == 0 ? a : gcd(b, a % b);
}
```

8.2 Least common multiple (LCM)

```
int lcm(int a, int b)
{
    return (a / gcd(a, b)) * b;
}
```

8.3 Big Mod

8.3.1 Iterative version

```
LLD bigmod(LLD B,LLD P,LLD M)
{
    LLD R=1;
    while(P>0)
    {
        if(P%2==1)
        {
            R=(R*B)%M;
        }
        P/=2;
        B=(B*B)%M;
    }
    return R;
}
```

8.3.2 Recursive version

```
int big_mod(int n, int p, int m)
{
    if(!n) return 0;
    int x=(n, p>>1 , m);
    x= (x*x)%m;
    if(p&1) x=(x*n)%m;
    return x;
}
```

8.4 Sieve of Eratosthenes

8.4.1 Sieve

```
const int Max=1000000;
bool prime_check[Max+5];
vint prime;
void sieve()
{
    int i,j;
    int sq=sqrt(Max);
    for(i=2;i*i<=Max;i++)
    {
        if(!prime_check[i])
        {
            prime.pb(i);
            for(j=i*i;j<=Max;j+=i)
            {
                prime_check[j]=true;
            }
        }
    }
    for(i=sq+1;i<=Max;i++)
    {
        if(!prime_check[i])prime.pb(i);
    }
}
```

8.4.2 Bitwise Sieve

```
#include<stdio.h>
#include<math.h>
#define check(N,pos) (N&(1<<(pos)))
#define Set(N,pos) N|(1<<(pos))
const int M=100000;
int status[M/32+2]={0};
void sieve()
{
    int sqN=(int)sqrt(M);
    int i,j,cnt=1;
    puts("2");
    for(i=3;i<=sqN;i+=2)
    {
        if(check(status[i>>5],i&31)==0)
        {
            for(j=i*i;j<=M;j+=2*i)
            {
                status[j>>5]=Set(status[j>>5],j&31);
            }
        }
    }
    for(i=3;i<=M;i+=2)
    {
        if(check(status[i>>5],i&31)==0)
        {
            printf("%d\n",i);
        }
    }
}
```

8.5 Euler's totient function

Euler's totient function (or Euler's phi function), denoted as $\varphi(n) or \phi(n)$, is an arithmetic function that counts the positive integers less than or equal to n that are relatively prime to n. (These integers are sometimes referred to as totatives of n.) Thus, if n is a positive integer, then $\varphi(n)$ is the number of integers k in the range $1 \leq k \leq n$ for which the greatest common divisor gcd(n, k) = 1.

8.5.1 Euler phi sieve version

```
int phi[10000];
const int M=1000;
void Generate_phi()
{
    int i,j;
    phi[1]=1;
    for(i=2; i<M; i++)
    {
        if(!phi[i])
        {
            phi[i]=i-1;
            for(j=i+i; j<M; j+=i)
            {
                if(!phi[j])phi[j]=j;
                phi[j]=phi[j]/i*(i-1);
            }
        }
    }
}
```

8.5.2 Euler phi function

```
int phi (int n)
{
    int ret = n;
    for (int i = 2; i * i <= n; i++)
    {
        if (n % i == 0)
        {
            while (n % i == 0)
            {
                n /= i;
            }
            ret -= ret / i;
        }
    }
    // this case will happen if n is a prime number
    // in that case we won't find any prime that divides n
    // that's less or equal to sqrt(n)
    if (n > 1) ret -= ret / n;
    return ret;
}
```

8.6 Number of Divisors

If a number

$$N = P1^{k1} * P2^{k2} * P3^{k3}.....Pn^{kn}$$

Number of divisor:

$$f(n) = (k1 + 1) * (k2 + 1) * (k3 + 1) * .....(kn + 1)$$

8.6.1 NOD sieve version

```
int nod[100000+5];
void Generate()
{
    nod[1]=1;
    for(int i=2; i<=100000; i++)
    {
        if(!nod[i])    //here checking i is prime or not ???
        {
            nod[i]=2;
            for(int j=i+i; j<=100000; j+=i)
            {
                if(!nod[j])nod[j]=1;
                int n=j,cnt=0;
                while(!(n%i))
                {
                    cnt++;
                    n/=i;
                }
                nod[j]*=(cnt+1);
            }
        }
    }
}
```

8.6.2 NOD function

```
int find_divisor(int n){
    int i,ans=1,count=1;
    while(n%2==0)
    {
        n/=2;
        count++;
    }
    ans*=count;
    for(i=3; i*i<=n; i+=2)
    {
        count=1;
        while(n%i==0)
        {
            n/=i;
            count++;
        }
        ans*=count;
    }
    if(n>1)
    {
        ans*=2;
    }
    return ans;
}
```

8.7 Summation of Divisors

If the prime factorization of an integer is:

$$P1^{x1} * P2^{x2} * ..... * Pn^{xn}$$

Where $P1, P2, Pn$ are primes, then the summation of divisors is

$$\frac{(P1^{x1}-1)}{(P1-1)} * \frac{(P2^{x2}-1)}{(P2-1)} * ..... * \frac{(Pn^{xn}-1)}{(Pn-1)}$$

8.8 Extended Euclid GCD

```
typedef pair<int, int> pii;
#define x first
#define y second

pii extendedEuclid(int a, int b)   // returns x, y | ax + by =gcd(a,b)
{
    if(b == 0) return pii(1, 0);
    else
    {
        pii d = extendedEuclid(b, a % b);
        return pii(d.y, d.x - d.y * (a / b));
    }
}
```

8.9 Modular multiplicative inverse

8.9.1 Modular inverse using EGCD

```
int modularInverse(int a, int n)
{
    pii ret = extendedEuclid(a, n);
    return ((ret.x % n) + n) % n;
}
```

8.9.2 Modular multiplicative inverse using Euler Theorem

According to Euler's theorem, if a is coprime to m, that is, gcd(a, m) = 1,then

$$a^{\varphi(m)} \equiv 1 \pmod{m}$$

where (m) is Euler's totient function. This follows from the fact that a belongs to the multiplicative group $(\frac{Z}{mZ})$ iff a is coprime to m. Therefore the modular multiplicative inverse can be found directly:

$$a^{\varphi(m)-1} \equiv a^{-1} \pmod{m}$$

. In the special case when m is a prime, the modular inverse is given by the below equation as:

$$a^{-1} \equiv a^{m-2} \pmod{m}$$

.

8.10 NCR

8.10.1 Lucas Theorem

```
/**
    Fine NCR % M when N C M are large number.
    using Lucas theorem.
**/
#include<bits/stdc++.h>
using namespace std;
typedef long long LLD;
LLD mod=1000003;
LLD big_mod(LLD n,LLD p,LLD m)
{
    if(p==0)return (LLD)1;
    LLD x=big_mod(n,p/2,m);
    x=(x*x)%m;
    if(p&1)x=(x*n)%m;
    return x;
}
LLD inverse_modulo(LLD t,LLD m)
{
    return big_mod(t,m-2,m);
}
LLD combi(LLD n, LLD k,LLD m)
{
    if(n<k)
        return 0;
    if(n-k<k)
        return combi(n,n-k,m);
    LLD i,p=1,t=1;
    for(i=n-k+1; i<=n; i++)
        p=(p*i)%m;
    for(i=1; i<=k; i++)
        t=(t*i)%m;
    return (p*inverse_modulo(t,m))%m;
}
LLD lucas(LLD n, LLD k, LLD m)
{
    if(n<k)
        return 0;
    if(k==0 || n==k)
        return 1;
    return (lucas(n/m,k/m,m)*combi(n%m,k%m,m))%m;
}
int main()
{
    return 0;
}
```

Chapter 9

Computational geometry

9.1 Geo Library (Stanford)

```
// C++ routines for computational geometry.

#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>
#define M_PI (2*acos(0))
using namespace std;

double INF = 1e100;
double EPS = 1e-12;

struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
  PT operator * (double c)     const { return PT(x*c,   y*c  ); }
  PT operator / (double c)     const { return PT(x/c,   y/c  ); }
};

double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}
```

```
43
44 // project point c onto line segment through a and b
45 PT ProjectPointSegment(PT a, PT b, PT c) {
46   double r = dot(b-a,b-a);
47   if (fabs(r) < EPS) return a;
48   r = dot(c-a, b-a)/r;
49   if (r < 0) return a;
50   if (r > 1) return b;
51   return a + (b-a)*r;
52 }
53
54 // compute distance from c to segment between a and b
55 double DistancePointSegment(PT a, PT b, PT c) {
56   return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
57 }
58
59 // compute distance between point (x,y,z) and plane ax+by+cz=d
60 double DistancePointPlane(double x, double y, double z,
61                           double a, double b, double c, double d)
62 {
63   return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
64 }
65
66 // determine if lines from a to b and c to d are parallel or collinear
67 bool LinesParallel(PT a, PT b, PT c, PT d) {
68   return fabs(cross(b-a, c-d)) < EPS;
69 }
70
71 bool LinesCollinear(PT a, PT b, PT c, PT d) {
72   return LinesParallel(a, b, c, d)
73       && fabs(cross(a-b, a-c)) < EPS
74       && fabs(cross(c-d, c-a)) < EPS;
75 }
76
77 // determine if line segment from a to b intersects with
78 // line segment from c to d
79 bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
80   if (LinesCollinear(a, b, c, d)) {
81     if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
82       dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
83     if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
84       return false;
85     return true;
86   }
87   if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
88   if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
89   return true;
90 }
91
92 // compute intersection of line passing through a and b
93 // with line passing through c and d, assuming that unique
94 // intersection exists; for segment intersection, check if
95 // segments intersect first
96 PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
97   b=b-a; d=c-d; c=c-a;
98   assert(dot(b, b) > EPS && dot(d, d) > EPS);
99   return a + b*cross(c, d)/cross(b, d);
100 }
101
102 // compute center of circle given three points
103 PT ComputeCircleCenter(PT a, PT b, PT c) {
```

```
104   b=(a+b)/2;
105   c=(a+c)/2;
106   return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c))
        ;
107 }
108
109 // determine if point is in a possibly non-convex polygon (by William
110 // Randolph Franklin); returns 1 for strictly interior points, 0 for
111 // strictly exterior points, and 0 or 1 for the remaining points.
112 // Note that it is possible to convert this into an *exact* test using
113 // integer arithmetic by taking care of the division appropriately
114 // (making sure to deal with signs properly) and then by writing exact
115 // tests for checking point on polygon boundary
116 bool PointInPolygon(const vector<PT> &p, PT q) {
117   bool c = 0;
118   for (int i = 0; i < p.size(); i++){
119     int j = (i+1)%p.size();
120     if ((p[i].y <= q.y && q.y < p[j].y ||
121       p[j].y <= q.y && q.y < p[i].y) &&
122       q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
123       c = !c;
124   }
125   return c;
126 }
127
128 // determine if point is on the boundary of a polygon
129 bool PointOnPolygon(const vector<PT> &p, PT q) {
130   for (int i = 0; i < p.size(); i++)
131     if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
132       return true;
133     return false;
134 }
135
136 // compute intersection of line through points a and b with
137 // circle centered at c with radius r > 0
138 vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
139   vector<PT> ret;
140   b = b-a;
141   a = a-c;
142   double A = dot(b, b);
143   double B = dot(a, b);
144   double C = dot(a, a) - r*r;
145   double D = B*B - A*C;
146   if (D < -EPS) return ret;
147   ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
148   if (D > EPS)
149     ret.push_back(c+a+b*(-B-sqrt(D))/A);
150   return ret;
151 }
152
153 // compute intersection of circle centered at a with radius r
154 // with circle centered at b with radius R
155 vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {
156   vector<PT> ret;
157   double d = sqrt(dist2(a, b));
158   if (d > r+R || d+min(r, R) < max(r, R)) return ret;
159   double x = (d*d-R*R+r*r)/(2*d);
160   double y = sqrt(r*r-x*x);
161   PT v = (b-a)/d;
162   ret.push_back(a+v*x + RotateCCW90(v)*y);
163   if (y > 0)
```

```
164     ret.push_back(a+v*x - RotateCCW90(v)*y);
165   return ret;
166 }
167
168 // This code computes the area or centroid of a (possibly nonconvex)
169 // polygon, assuming that the coordinates are listed in a clockwise or
170 // counterclockwise fashion.  Note that the centroid is often known as
171 // the "center of gravity" or "center of mass".
172 double ComputeSignedArea(const vector<PT> &p) {
173   double area = 0;
174   for(int i = 0; i < p.size(); i++) {
175     int j = (i+1) % p.size();
176     area += p[i].x*p[j].y - p[j].x*p[i].y;
177   }
178   return area / 2.0;
179 }
180
181 double ComputeArea(const vector<PT> &p) {
182   return fabs(ComputeSignedArea(p));
183 }
184
185 PT ComputeCentroid(const vector<PT> &p) {
186   PT c(0,0);
187   double scale = 6.0 * ComputeSignedArea(p);
188   for (int i = 0; i < p.size(); i++){
189     int j = (i+1) % p.size();
190     c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
191   }
192   return c / scale;
193 }
194
195 // tests whether or not a given polygon (in CW or CCW order) is simple
196 bool IsSimple(const vector<PT> &p) {
197   for (int i = 0; i < p.size(); i++) {
198     for (int k = i+1; k < p.size(); k++) {
199       int j = (i+1) % p.size();
200       int l = (k+1) % p.size();
201       if (i == l || j == k) continue;
202       if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
203         return false;
204     }
205   }
206   return true;
207 }
208
209 int main() {
210
211   // expected: (-5,2)
212   cerr << RotateCCW90(PT(2,5)) << endl;
213
214   // expected: (5,-2)
215   cerr << RotateCW90(PT(2,5)) << endl;
216
217   // expected: (-5,2)
218   cerr << RotateCCW(PT(2,5),M_PI/2) << endl;
219
220   // expected: (5,2)
221   cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;
222
223   // expected: (5,2) (7.5,3) (2.5,1)
224   cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7)) << " "
```

```
225       << ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7)) << " "
226       << ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;
227
228   // expected: 6.78903
229   cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;
230
231   // expected: 1 0 1
232   cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
233        << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
234        << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;
235
236   // expected: 0 0 1
237   cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
238        << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
239        << LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;
240
241   // expected: 1 1 1 0
242   cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << " "
243        << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << " "
244        << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << " "
245        << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;
246
247   // expected: (1,2)
248   cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) <<
       endl;
249
250   // expected: (1,1)
251   cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;
252
253   vector<PT> v;
254   v.push_back(PT(0,0));
255   v.push_back(PT(5,0));
256   v.push_back(PT(5,5));
257   v.push_back(PT(0,5));
258
259   // expected: 1 1 1 0 0
260   cerr << PointInPolygon(v, PT(2,2)) << " "
261        << PointInPolygon(v, PT(2,0)) << " "
262        << PointInPolygon(v, PT(0,2)) << " "
263        << PointInPolygon(v, PT(5,2)) << " "
264        << PointInPolygon(v, PT(2,5)) << endl;
265
266   // expected: 0 1 1 1 1
267   cerr << PointOnPolygon(v, PT(2,2)) << " "
268        << PointOnPolygon(v, PT(2,0)) << " "
269        << PointOnPolygon(v, PT(0,2)) << " "
270        << PointOnPolygon(v, PT(5,2)) << " "
271        << PointOnPolygon(v, PT(2,5)) << endl;
272
273   // expected: (1,6)
274   //           (5,4) (4,5)
275   //           blank line
276   //           (4,5) (5,4)
277   //           blank line
278   //           (4,5) (5,4)
279   vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
280   for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
281   u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
282   for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
283   u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
284   for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
```

```
  u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;

  // area should be 5.0
  // centroid should be (1.1666666, 1.166666)
  PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
  vector<PT> p(pa, pa+4);
  PT c = ComputeCentroid(p);
  cerr << "Area: " << ComputeArea(p) << endl;
  cerr << "Centroid: " << c << endl;

  return 0;
}
```

9.2 Convex Hull

9.2.1 Convex Hull 1

```
typedef long long LLD;
struct POINT
{
    double x,y;
    POINT(LLD x=0,LLD y=0):x(x),y(y) {};
};

POINT P0;

int area2(POINT p0,POINT p1,POINT p2)
{
    return (p2.x-p0.x)*(p1.y-p0.y)-(p1.x-p0.x)*(p2.y-p0.y);
}
int dist2(POINT a,POINT b)
{
    int dx=a.x-b.x,dy=a.y-b.y;
    return dx*dx+dy*dy;
}

///Compare version 1
bool angle_comp(POINT a,POINT b)
{
    if(area2(P0,a,b)<0)return true;
    else if(area2(P0,a,b)==0)return dist2(P0,a)<dist2(P0,b);
    return false;
}

///Compare version 2

bool angle_comp2(POINT a,POINT b)
{
    if(area2(P0,a,b)==0)
    {
        return dist2(P0,a)<dist2(P0,b);
    }
    int d1x=a.x-P0.x,d1y=a.y-P0.y;
    int d2x=b.x-P0.x,d2y=b.y-P0.y;
    return (atan2((double)d1y,(double)d1x)-atan2((double)d2y,(double)d2x))
    <0;
```

```
}

vector<POINT> Find_convex_hull(vector<POINT>P)
{
    int i,N=(int)P.size();
    int po=0;
    for ( i=0; i<N; i++)
    {
        if(P[i].y < P[po].y || (P[i].y== P[po].y && P[i].x < P[po].x))po=i;
    }
    P0 = P[po];
    sort(P.begin(),P.end(),angle_comp);
    POINT prev(0,0),now(0,0);
    stack<POINT> S;
    S.push(P0);
    i=1;
    while(i<N)
    {
        if(S.size()<2)
        {
            S.push(P[i++]);
        }
        else
        {
            now = S.top();
            S.pop();
            prev=S.top();
            S.push(now);
            if(area2(prev,now,P[i])<0)
                S.push(P[i++]);
            else S.pop();
        }
    }

    vector<POINT> ConvexHull;
    ConvexHull.push_back(P0);
    while(!S.empty())
    {

        ConvexHull.push_back(S.top());
        S.pop();
    }
    return ConvexHull;
}
```

9.2.2 Convex Hull (Stanford)

```
// Compute the 2D convex hull of a set of points using the monotone chain
// algorithm.  Eliminate redundant points from the hull if REMOVE_REDUNDANT
    is
// #defined.
//
// Running time: O(n log n)
//
//   INPUT:   a vector of input points, unordered.
//   OUTPUT:  a vector of points in the convex hull, counterclockwise,
    starting
//            with bottommost/leftmost point

#define REMOVE_REDUNDANT

```

```
13 typedef double T;
14 const T EPS = 1e-7;
15 struct PT
16 {
17     T x, y;
18     PT() {}
19     PT(T x, T y) : x(x), y(y) {}
20     bool operator<(const PT &rhs) const
21     {
22         return make_pair(y,x) < make_pair(rhs.y,rhs.x);
23     }
24     bool operator==(const PT &rhs) const
25     {
26         return make_pair(y,x) == make_pair(rhs.y,rhs.x);
27     }
28 };
29
30 T cross(PT p, PT q)
31 {
32     return p.x*q.y-p.y*q.x;
33 }
34 T area2(PT a, PT b, PT c)
35 {
36     return cross(a,b) + cross(b,c) + cross(c,a);
37 }
38
39 #ifdef REMOVE_REDUNDANT
40 bool between(const PT &a, const PT &b, const PT &c)
41 {
42     return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y
    )*(c.y-b.y) <= 0);
43 }
44 #endif
45
46 void ConvexHull(vector<PT> &pts)
47 {
48     sort(pts.begin(), pts.end());
49     pts.erase(unique(pts.begin(), pts.end()), pts.end());
50     vector<PT> up, dn;
51     for (int i = 0; i < pts.size(); i++)
52     {
53         while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >=
     0) up.pop_back();
54         while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <=
     0) dn.pop_back();
55         up.push_back(pts[i]);
56         dn.push_back(pts[i]);
57     }
58     pts = dn;
59     for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);
60
61 #ifdef REMOVE_REDUNDANT
62     if (pts.size() <= 2) return;
63     dn.clear();
64     dn.push_back(pts[0]);
65     dn.push_back(pts[1]);
66     for (int i = 2; i < pts.size(); i++)
67     {
68         if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back()
    ;
69         dn.push_back(pts[i]);
```

```
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1]))
    {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
#endif
}
```

9.3 Determine a point inside polygon or not

```
#include<bits/stdc++.h>
using namespace std;
struct point
{
    int x,y;
    point() {}
    point(int a,int b)
    {
        x=a,y=b;
    }
};
int trun(point p0,point p1,point p2)
{
    int result=(p2.x-p0.x)*(p1.y-p0.y)-(p1.x-p0.x)*(p2.y-p0.y);
    if(result>0)return 1;
    if(result < 0)return -1;
    return result;
}
bool on_segment(point pi,point pj,point pk)
{
    int minx=min(pi.x,pj.x);
    int maxx=max(pi.x,pj.x);
    int miny=min(pi.y,pj.y);
    int maxy=max(pi.y,pj.y);
    if(pk.x>=minx&&pk.x<=maxx&&pk.y>=miny&&pk.y<=maxy)return true;
    return false;
}
bool Segment_intersect(point p1,point p2,point p3,point p4)
{
    int d1=trun(p3,p4,p1);
    int d2=trun(p3,p4,p2);
    int d3=trun(p1,p2,p3);
    int d4=trun(p1,p2,p4);
    if((d1*d2<0)&&(d3*d4<0))return true;
    if(d1==0&&on_segment(p3,p4,p1))return true;
    if(d2==0&&on_segment(p3,p4,p2))return true;
    if(d3==0&&on_segment(p1,p2,p3))return true;
    if(d4==0&&on_segment(p1,p2,p4))return true;
    return false;
}
bool in_poligon(vector<point>p,point given,point inf2)
{
    int intersect=0;
    int N=p.size();
    for(int i=0; i<N; i++)
    {
        point p1=p[i];
        point p2=p[(i+1)%N];
```

```
        if(Segment_intersect(p1,p2,given,inf2))
        {
            intersect++;
        }
    }
    return intersect%2==1;
}
int main()
{
    int N;
    while(scanf("%d",&N)&&N)
    {
        int Max=-10005;
        vector<point>poligon;
        for(int i=0; i<N; i++)
        {
            int x,y;
            scanf("%d %d",&x,&y);
            poligon.push_back(point(x,y));
            Max=max(x,Max);
        }
        int x,y;
        scanf("%d %d",&x,&y);
        if(in_poligon(poligon,point(x,y),point(Max,y)))
        {
            printf("T\n");
        }
        else
        {
            printf("F\n");
        }
    }
    return 0;
}
```

9.4 Integer point in a Segment or lattice point

```
int IntegerPointsOnSegment(const point &P1, const point &P2)
{
    point P;
    P.x=abs(P1.x-P2.x);
    P.y=abs(P1.y-P2.y);
    if(P.x==0) return P.y-1;
    if(P.y==0) return P.x-1;
    return (__gcd(P.x,P.y))-1;
}
```

9.5 Segment Intersection

```
int Direction(const point_int &O, const point_int &A, const point_int &B) //
    checks rotation
{
    int rotation = (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
    if(rotation == 0) return 0;
    return rotation > 0? 1 : -1;
}
bool On_Segment(point_int p, point_int q, point_int r) // q point lies
    between p and r
{
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
```

```
            q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
        return true;
    return false;
}
bool Segment_Intersect(point_int p1, point_int p2, point_int q1, point_int
    q2)
{
    int d1 = Direction(p1, p2, q1);
    int d2 = Direction(p1, p2, q2);
    int d3 = Direction(q1, q2, p1);
    int d4 = Direction(q1, q2, p2);
    if(d1 != d2 && d3 != d4)return true;
    if(!d1 && On_Segment(p1, q1, p2)) return true;
    if(!d2 && On_Segment(p1, q2, p2)) return true;
    if(!d3 && On_Segment(q1, p1, q2)) return true;
    if(!d4 && On_Segment(q1, p2, q2)) return true;
    return false;
}
```

9.6 Verify Convex polygon

```
int trun(point p0,point p1,point p2)
{
    int result=(p2.x-p0.x)*(p1.y-p0.y)-(p1.x-p0.x)*(p2.y-p0.y);
    return result;
}
bool isConvex(int n,vector<point>v)
{
    int pos=0,neg =0;
    for(int i = 0; i < n; i++)
    {
        int prev = (i + n -1) % n , next = (i  + 1) % n;
        point A=v[i];
        point B=v[prev];
        point C=v[next];
        int pv=trun(A,B,C);
        if(pv > 0) pos++;
        else
        {
            if(pv < 0) neg++;
        }
    }
    return (pos == 0) || (neg == 0);
}
```

9.7 Closest Pair Algorithm

```
const int MAX=100005;
struct point
{
    int x, y, i;
};
point arr[MAX], sortedY[MAX];
bool flag[MAX];
template <class T> int getdist(T a, T b)
{
    return max(abs(a.x - b.x), abs(a.y - b.y));
}
bool compareX(const point &a, const point &b)
{
```

```
14      return a.x < b.x;
15  }
16  bool compareY(const point &a, const point &b)
17  {
18      return a.y < b.y;
19  }
20
21  int closest_pair(point X[], point Y[], int n)
22  {
23      int left_call, right_call, mindist;
24      if(n == 1) return inf;
25      if(n == 2) return getdist(X[0], X[1]);
26      int n1, n2, ns, j, m = n / 2, i;
27      point xL[m + 1], xR[m + 1], yL[m + 1], yR[m + 1], Xm = X[m - 1], yS[n];
28      for(i = 0; i < m; i++)
29      {
30          xL[i] = X[i];
31          flag[X[i].i] = 0;
32      }
33      for(; i < n; i++)
34      {
35          xR[i - m] = X[i];
36          flag[X[i].i] = 1;
37      }
38      for(i = n2 = n1 = 0; i < n; i++)
39      {
40          if(!flag[Y[i].i]) yL[n1++] = Y[i];
41          else yR[n2++] = Y[i];
42      }
43      left_call = closest_pair(xL, yL, n1);
44      right_call = closest_pair(xR, yR, n2);
45      mindist = min(left_call, right_call);
46      for(i = ns = 0; i < n; i++)
47          if((Y[i].x - Xm.x) < mindist)
48              yS[ns++] = Y[i];
49      for(i = 0; i < ns; i++)
50          for(j = i + 1; j < ns && (yS[j].y - yS[i].y) < mindist; j++)
51              mindist = min(mindist, getdist(yS[i], yS[j]));
52      return mindist;
53  }
54
55  int Find_closestpair(int n)
56  {
57      sort(arr, arr + n, compareX);
58      sort(sortedY, sortedY + n, compareY);
59      int ans = closest_pair(arr, sortedY, n);
60      return ans;
61  }
```