

Algorithm Code Book

Tanvir Hasan Anick

July 8, 2015

Contents

1	Data Structure	3
1.1	Trie	3
1.1.1	Static Trie	3
1.2	RMQ	4
1.2.1	Bit	4
1.2.2	Square Root Decomposition	5
1.2.3	MO's Algorithm	6
1.2.4	Segment Tree	8
1.2.5	Sliding Window RMQ	17
1.2.6	Sparse Table	18
1.3	Heavy Light Decomposition	18
1.4	Ternary Bit Mask	25
2	Graph Theory	27
2.1	DFS	27
2.1.1	Bicoloring	27
2.1.2	Cycle Finding	27
2.2	Topological Sort	27
3	Flow networks/ matching	29
3.1	Max Flow	29
4	Dynamic programming	34
4.1	Edit Distance	34
5	Strings	39
5.1	KMP	39
5.2	Aho Corasick	40
5.2.1	Aho Corasick with Dynamic Trie	40
5.2.2	Aho Corasick with Static Trie	42
5.3	Manacher's Algorithm	45
6	Computational geometry	46

7	Math	47
7.1	Reduce Ratio	47
8	Number Theory	48
8.1	NCR	48
8.1.1	Lucas Theorem	48

Chapter 1

Data Structure

1.1 Trie

1.1.1 Static Trie

```
1 #define Max 10005
2 int getId(char c)
3 {
4     return c>='a'?c-'a':c-'A'+26;
5 }
6 struct Trie
7 {
8     struct Tree
9     {
10         int Next[52];
11         bool word;
12         void clear()
13         {
14             word=false;
15             memset(Next,-1,sizeof(Next));
16         }
17     }T[Max];
18     int ptr;
19     void clear()
20     {
21         ptr=1;
22         T[0].clear();
23         memset(T[0].Next,0,sizeof(T[0].Next));
24     }
25     void Insert(const char *str)
26     {
27         int p=0;
28         for(int i=0;str[i];i++)
29         {
30             int id=getId(str[i]);
31             if(T[p].Next[id]<=0)
32             {
33                 T[p].Next[id]=ptr;
```

```

34         T[ptr++].clear();
35     }
36     p=T[p].Next[id];
37 }
38 T[p].word=true;
39 }
40 bool Search(const char *str)
41 {
42     int p=0;
43     for(int i=0;str[i];i++)
44     {
45         int id=getId(str[i]);
46         if(T[p].Next[id]>0)
47         {
48             p=T[p].Next[id];
49         }
50         else return false;
51     }
52     return T[p].word;
53 }
54 };
55 Trie A;

```

1.2 RMQ

1.2.1 Bit

1D Bit

```

1 #define MaxVal 100000
2 int Bit[MaxVal];
3 /**find sum from 1 to idx**/
4 int read(int idx)
5 {
6     int sum = 0;
7     while (idx > 0)
8     {
9         sum += Bit[idx];
10        idx -= (idx & -idx);
11    }
12    return sum;
13 }
14 /**update value ind to MaxVal**/
15 void update(int idx ,int val)
16 {
17     while (idx <= MaxVal)
18     {
19         Bit[idx] += val;
20         idx += (idx & -idx);
21     }
22 }
23
24 /**Find the value of idx**/
25 int readSingle(int idx)
26 {
27     int sum = Bit[idx]; /// sum will be decreased

```

```

28     if (idx > 0)    /// special case
29     {
30         int z = idx - (idx & -idx); /// make z first
31         idx--; /// idx is no important any more, so instead y, you
           can use idx
32         while (idx != z)    /// at some iteration idx (y) will become
           z
33         {
34             sum -= Bit[idx]; /// substruct Bit frequency which is
           between y and "the same path"
35             idx -= (idx & -idx);
36         }
37     }
38     return sum;
39 }

```

2D Bit

```

1 void updatey(int x , int y , int val)
2 {
3     while (y <= max_y)
4     {
5         tree[x][y] += val;
6         y += (y & -y);
7     }
8 }
9 void update(int x , int y , int val)
10 {
11     while (x <= max_x)
12     {
13         updatey(x , y , val); /// this function should update array
           tree[x]
14         x += (x & -x);
15     }
16 }

```

1.2.2 Square Root Decompostion

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int sz=100005;
4 const int inf=(1<<28);
5 template<typename t> t MIN3(t a,t b, t c)
6 {
7     return min(a,min(b,c));
8 }
9 int BLOCK[400];
10 int arr[sz];
11 int getId(int indx,int blockSZ)
12 {
13     return indx/blockSZ;
14 }
15 void init(int sz)
16 {
17     for(int i=0; i<=sz; i++)BLOCK[i]=inf;
18 }
19 void update(int val,int indx,int blockSZ)

```

```

20 {
21     int id=getId(indx, blockSZ);
22     BLOCK[id]=min(BLOCK[id], val);
23 }
24 int query(int L, int R, int blockSZ)
25 {
26     int lid=getId(L, blockSZ);
27     int rid=getId(R, blockSZ);
28     if(lid==rid)
29     {
30         int ret=inf;
31         for(int i=L; i<=R; i++)ret=min(ret, arr[i]);
32         return ret;
33     }
34     m1=inf, m2=inf, m3=inf;
35     for(int i=L; i<(lid+1)*blockSZ; i++)m1=min(m1, arr[i]);
36     for(int i=lid+1; i<rid; i++)m2=min(m2, BLOCK[i]);
37     for(int i=rid*blockSZ; i<=R; i++)m3=min(m3, arr[i]);
38     return MIN3(m1, m2, m3);
39 }
40 int main()
41 {
42     int N, Q;
43     scanf("%d %d", &N, &Q);
44     int blockSZ=sqrt(N);
45     init(blockSZ);
46     for(int i=0; i<N; i++)
47     {
48         int x;
49         scanf("%d", &x);
50         arr[i]=x;
51         update(x, i, blockSZ);
52     }
53     while(Q--)
54     {
55         int x, y;
56         scanf("%d %d", &x, &y);
57         printf("%d\n", query(x, y, blockSZ));
58     }
59     return 0;
60 }

```

1.2.3 MO's Algorithm

```

1  /**
2   MO's Algorithm
3   problem: http://www.spoj.com/problems/DQUERY
4
5   MOs algorithm is just an order in which we process the queries
6   .
7   We were given M queries, we will re-order the queries in a
8   particular order and then process them.
9   Clearly, this is an off-line algorithm. Each query has L and R,
10  we will call them opening and closing.
11  Let us divide the given input array into Sqrt(N) blocks.
12  Each block will be N / Sqrt(N) = Sqrt(N) size.
13  Each opening has to fall in one of these blocks.

```

```

11 Each closing has to fall in one of these blocks.
12
13 All the queries are first ordered in ascending order of their
    block number (block number is the block in which its opening
    falls).
14 Ties are ordered in ascending order of their R value.
15
16 */
17 #include <bits/stdc++.h>
18 using namespace std;
19 #define Mx 30005
20 #define MxNum 1000005
21 int BlockSize;
22 int Answer;
23 int Freq[MxNum], Num[Mx];
24 struct info
25 {
26     int L,R,qno;
27     info(int L=0,int R=0,int qno=0):L(L),R(R),qno(qno){};
28     bool operator<(const info &a) const
29     {
30         if(L/BlockSize!=a.L/BlockSize) return L/BlockSize<a.L/
            BlockSize;
31         return R<a.R;
32     }
33 } Query[200005];
34 int StoreAnswer[200005];
35 void Add(int indx)
36 {
37     Freq[Num[indx]]++;
38     if(Freq[Num[indx]]==1) Answer++;
39 }
40 void Remove(int indx)
41 {
42     Freq[Num[indx]]--;
43     if(Freq[Num[indx]]==0) Answer--;
44 }
45 int main()
46 {
47     int N;
48     scanf("%d",&N);
49     BlockSize=sqrt(N);
50     for(int i=0;i<N;i++)
51     {
52         scanf("%d",&Num[i]);
53     }
54     int Q;
55     scanf("%d",&Q);
56     for(int i=0;i<Q;i++)
57     {
58         int x,y;
59         scanf("%d %d",&x,&y);
60         Query[i]=info(x-1,y-1,i);
61     }
62     sort(Query,Query+Q);
63     int currentL=0,currentR=0;
64     Answer=0;

```



```

65     for (int i=0; i<Q; i++)
66     {
67         int L=Query[i].L;
68         int R=Query[i].R;
69         while (currentL<L)
70         {
71             Remove(currentL);
72             currentL++;
73         }
74         while (currentL>L)
75         {
76             Add(currentL-1);
77             currentL--;
78         }
79         while (currentR<=R)
80         {
81             Add(currentR);
82             currentR++;
83         }
84         while (currentR>R+1)
85         {
86             Remove(currentR-1);
87             currentR--;
88         }
89         StoreAnswer[Query[i].qno]=Answer;
90     }
91     for (int i=0; i<Q; i++)
92     {
93         printf("%d\n", StoreAnswer[i]);
94     }
95     return 0;
96 }

```

1.2.4 Segment Tree

Lazy Propagation1

```

1  /**
2  **You are given an array of N elements, which are initially all 0.
   After **that you will be given C commands. They are
3  **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.
4  **1 p q - output a line containing a single integer which is the
   sum of all **the array elements between p and q (inclusive)
5  */
6  #include<bits/stdc++.h>
7  using namespace std;
8  typedef long long LLD;
9  LLD tree[3*100005];
10 LLD lazy[3*100005];
11 void update(int left, int right, int index, int x, int y, int value)
12 {
13     if (x<=left&&right<=y)
14     {
15         tree[index]+=(LLD)(right-left+1)*value;
16         lazy[index]+=value;
17         return;

```

```

18     }
19     int mid=(left+right)/2;
20     if(lazy[index]!=0)
21     {
22         tree[2*index]+=(LLD)(mid-left+1)*lazy[index];
23         tree[2*index+1]+=(LLD)(right-mid)*lazy[index];
24         lazy[2*index]+=lazy[index];
25         lazy[2*index+1]+=lazy[index];
26         lazy[index]=0;
27     }
28     if(x<=mid)
29     {
30         update(left,mid,2*index,x,y,value);
31     }
32     if(y>mid)
33     {
34         update(mid+1,right,2*index+1,x,y,value);
35     }
36     tree[index]=tree[2*index]+tree[2*index+1];
37 }
38 LLD query(int left,int right,int index,int x,int y)
39 {
40     LLD a1=0,a2=0;
41     if(x<=left&&right)
42     {
43         return tree[index];
44     }
45     int mid=(left+right)/2;
46     if(lazy[index]!=0)
47     {
48         tree[2*index]+=(LLD)(mid-left+1)*lazy[index];
49         tree[2*index+1]+=(LLD)(right-mid)*lazy[index];
50         lazy[2*index]+=lazy[index];
51         lazy[2*index+1]+=lazy[index];
52         lazy[index]=0;
53     }
54     if(x<=mid)
55     {
56         a1=query(left,mid,2*index,x,y);
57     }
58     if(y>mid)
59     {
60         a2=query(mid+1,right,2*index+1,x,y);
61     }
62     return (a1+a2);
63 }
64 int main()
65 {
66     int test,t;
67     scanf("%d",&test);
68     for(t=1;t<=test;t++)
69     {
70         memset(tree,0,sizeof(tree));
71         memset(lazy,0,sizeof*lazy);
72         int s,q;
73         scanf("%d %d",&s,&q);
74         while(q--)

```

```

75     {
76         int x,y,v,dec;
77         scanf("%d",&dec);
78         if(dec)
79         {
80             scanf("%d %d",&x,&y);
81             LLD ans=query(0,s-1,1,x-1,y-1);
82             printf("%lld\n",ans);
83         }
84         else
85         {
86             scanf("%d %d %d",&x,&y,&v);
87             update(0,s-1,1,x-1,y-1,v);
88         }
89     }
90 }
91 return 0;
92 }

```

Lazy Propagation2

```

1  /*
2  **You have an array with n elements which is indexed from 0 to n -
3  1. **Initially all elements are zero. Now you have to deal with
4  two types of **operations
5  **1.Increase the numbers between indices i and j (inclusive) by 1.
6  This **is **represented by the command '0 i j'.
7  **2.Answer how many numbers between indices i and j (inclusive) are
8  **divisible by 3. This is represented by the command '1 i j'.
9  */
10 #include<bits/stdc++.h>
11 using namespace std;
12 #define Max 100010
13 int Tree[8*Max][4];
14 int lazy[8*Max];
15 int temp[4];
16 void build(int left,int right,int indx)
17 {
18     if(left==right)
19     {
20         Tree[indx][0]=1;
21         Tree[indx][1]=Tree[indx][2]=lazy[indx]=0;
22         return;
23     }
24     int mid=(left+right)/2;
25     build(left,mid,2*indx);
26     build(mid+1,right,2*indx+1);
27     for(int i=0;i<3;i++)
28     {
29         Tree[indx][i]=Tree[2*indx][i]+Tree[2*indx+1][i];
30     }
31 }
32 void update(int left,int right,int indx,int x,int y,int add)
33 {
34     if(lazy[indx])
35     {
36         int lazy_val=lazy[indx];
37         lazy[2*indx]=(lazy[2*indx]+lazy_val)%3;

```

```

34     lazy[2*indx+1]=(lazy[2*indx+1]+lazy_val)%3;
35     for(int i=0;i<3;i++)temp[(lazy_val+i)%3]=Tree[indx][i];
36     for(int i=0;i<3;i++)Tree[indx][i]=temp[i];
37     lazy[indx]=0;
38 }
39 if(left>y||right<x) return;
40 if(x<=left&&right<=y)
41 {
42     for(int i=0;i<3;i++)
43     {
44         temp[(i+add)%3]=Tree[indx][i];
45     }
46     for(int i=0;i<3;i++)Tree[indx][i]=temp[i];
47     lazy[2*indx]=(lazy[2*indx]+add)%3;
48     lazy[2*indx+1]=(lazy[2*indx+1]+add)%3;
49     return;
50 }
51 int mid=(left+right)/2;
52 update(left,mid,2*indx,x,y,add);
53 update(mid+1,right,2*indx+1,x,y,add);
54 for(int i=0;i<3;i++)
55 {
56     Tree[indx][i]=Tree[2*indx][i]+Tree[2*indx+1][i];
57 }
58 }
59 int query(int left,int right,int indx,int x,int y)
60 {
61     if(lazy[indx])
62     {
63         int lazy_val=lazy[indx];
64         lazy[2*indx]=(lazy[2*indx]+lazy_val)%3;
65         lazy[2*indx+1]=(lazy[2*indx+1]+lazy_val)%3;
66         for(int i=0;i<3;i++)temp[(lazy_val+i)%3]=Tree[indx][i];
67         for(int i=0;i<3;i++)Tree[indx][i]=temp[i];
68         lazy[indx]=0;
69     }
70     if(left>y||right<x) return 0;
71     if(x<=left&&right<=y) return Tree[indx][0];
72     int mid=(left+right)/2;
73     return query(left,mid,2*indx,x,y)+query(mid+1,right,2*indx+1,x,
74 y);
75 }
76 int main()
77 {
78     int x,y;
79     int test;
80     scanf("%d",&test);
81     for(int t=1;t<=test;t++)
82     {
83         memset(lazy,0,sizeof(lazy));
84         int N,Q;
85         scanf("%d %d",&N,&Q);
86         build(0,N-1,1);
87         printf("Case %d:\n",t);
88         for(int i=0;i<Q;i++)
89         {

```

```

90         scanf("%d %d %d",&d,&x,&y);
91         if(d==0)
92         {
93             update(0,N-1,1,x,y,1);
94         }
95         else printf("%d\n",query(0,N-1,1,x,y));
96     }
97 }
98 return 0;
99 }

```

Segment Tree Variant 1

```

1  /**
2  **Give a array Of N numbers. Finding Maximum cumulative number
   frequency in **the range.
3  **input:
4  **10 4
5  **1 1 1 3 3 3 3 2 2 2
6  **1 5
7  **1 6
8  **1 7
9  **Output:
10 **3
11 **3
12 **4
13 **2
14 */
15 #include<bits/stdc++.h>
16 using namespace std;
17 typedef long long LLD;
18 #define MAX 50005
19 struct info
20 {
21     int Lcnt,Rcnt,Max,Lnum,Rnum;
22     info(int Lcnt=0,int Rcnt=0,int Max=0,int Lnum=0,int Rnum=0):
23         Lcnt(Lcnt),Rcnt(Rcnt),Max(Max),Lnum(Lnum),Rnum(Rnum){};
24 };
25 info Tree[3*MAX];
26 int arr[MAX];
27 info marge(const info &L,const info &R)
28 {
29     info ret;
30     if(L.Rnum==R.Lnum)
31     {
32         ret.Max=max(L.Rcnt+R.Lcnt,max(L.Max,R.Max));
33     }
34     else ret.Max=max(L.Max,R.Max);
35     ret.Lnum=L.Lnum;
36     ret.Rnum=R.Rnum;
37     if(L.Lnum==R.Lnum) ret.Lcnt=L.Lcnt+R.Lcnt;
38     else ret.Lcnt=L.Lcnt;
39     if(L.Rnum==R.Rnum) ret.Rcnt=L.Rcnt+R.Rcnt;
40     else ret.Rcnt=R.Rcnt;
41     return ret;
42 }
43 void build(int L,int R,int indx)
44 {

```

```

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

```

1  /**
2  **You are given a sequence A of N (N <= 50000) integers between
   -10000 and 10000.
3  **On this sequence you have to apply M (M <= 50000) operations:
4  **modify the i-th element in the sequence or for given x y print
   max{Ai + Ai+1 + .. + Aj | x<=i<=j<=y }.
5  **/
6  #include<bits/stdc++.h>
7  using namespace std;
8  typedef long long LLD;
9  template<class T> T MAX3(T a,T b,T c) {return max(a,max(b,c));}
10 LLD Inf=(1ll<<60);

```

```

11 #define MN 50005
12 struct info
13 {
14     LLD prefixSum;
15     LLD suffixSum;
16     LLD Total;
17     LLD TotalMax;
18     info (int pre==-Inf, int suff==-Inf, int total=-Inf, int totalmax=-
        Inf): prefixSum(pre), suffixSum(suff), Total(total), TotalMax(
        totalmax) {};
19 };
20 info marge(const info &a, const info &b)
21 {
22     info ret;
23     ret.Total=a.Total+b.Total;
24     ret.prefixSum=max(a.prefixSum, a.Total+b.prefixSum);
25     ret.suffixSum=max(a.suffixSum+b.Total, b.suffixSum);
26     ret.TotalMax=MAX3(a.TotalMax, b.TotalMax, a.suffixSum+b.prefixSum
        );
27     return ret;
28 }
29 LLD arr[MN];
30 info Tree[3*MN];
31 void build(int L, int R, int indx)
32 {
33     if (L==R)
34     {
35         Tree[indx]=info(arr[L], arr[L], arr[L], arr[L]);
36         return;
37     }
38     int mid=(L+R)>>1;
39     build(L, mid, 2*indx);
40     build(mid+1, R, 2*indx+1);
41     Tree[indx]=marge(Tree[2*indx], Tree[2*indx+1]);
42 }
43 void update(int L, int R, int indx, int x, LLD val)
44 {
45     if (L==R)
46     {
47         Tree[indx]=info(val, val, val, val);
48         return;
49     }
50     int mid=(L+R)>>1;
51     if (x<=mid) update(L, mid, 2*indx, x, val);
52     else update(mid+1, R, 2*indx+1, x, val);
53     Tree[indx]=marge(Tree[2*indx], Tree[2*indx+1]);
54 }
55 info query(int L, int R, int indx, int x, int y)
56 {
57     if (L==x and y==R) return Tree[indx];
58     int mid=(L+R)>>1;
59     if (y<=mid) return query(L, mid, 2*indx, x, y);
60     else if (x>mid) return query(mid+1, R, 2*indx+1, x, y);
61     return marge(query(L, mid, 2*indx, x, mid), query(mid+1, R, 2*indx+1,
        mid+1, y));
62 }
63 int main()

```

```

64 {
65     #ifdef _ANICK_
66         //f_input;
67     #endif // _ANICK_
68     int N;
69     scanf("%d",&N);
70     for(int i=1;i<=N;i++)scanf("%lld",&arr[i]);
71     build(1,N,1);
72     int Q;
73     scanf("%d",&Q);
74     while(Q--)
75     {
76         int t,x,y;
77         scanf("%d %d %d",&t,&x,&y);
78         if(t) printf("%lld\n",query(1,N,1,x,y).TotalMax);
79         else update(1,N,1,x,y);
80     }
81     return 0;
82 }

```

Segment Tree Variant 3

```

1  /**
2  **Given a bracket sequence.
3  ** On a bracket word one can do the following operations:
4  **replacement — changes the i-th bracket into the opposite one
5  **check — if the word is a correct bracket expression
6  **/
7  #include<bits/stdc++.h>
8  using namespace std;
9  typedef long long LLD;
10 #define MAX 50005
11 struct info
12 {
13     int sum,sub;
14     info(int sum=0,int sub=0):sum(sum),sub(sub){};
15 };
16 info Tree[4*MAX];
17 char inp[MAX];
18 info marge(const info &L,const info &R)
19 {
20     info ret;
21     ret.sum= L.sum+R.sum;
22     ret.sub=L.sub;
23     ret.sub=min(ret.sub,L.sum+R.sub);
24     return ret;
25 }
26 void build(int L,int R,int indx)
27 {
28     if(L==R)
29     {
30         int x;
31         if(inp[L]=='(')x=1;
32         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 *
63         indx + 1, mid + 1, y));
64 }
65 int main()
66 {
67     int N, t = 1;
68     while (scanf("%d", &N) == 1)
69     {
70         scanf("%s", inp);
71         build(0, N - 1, 1);
72         int Q;
73         printf("Test %d:\n", t++);
74         scanf("%d", &Q);
75         while (Q--)
76         {
77             int x;
78             scanf("%d", &x);
79             if (x)
80             {
81                 if (inp[x - 1] == '(') inp[x - 1] = ')';
82                 else inp[x - 1] = '(';
83                 update(0, N - 1, 1, x - 1);
84             }
85             else
86             {
87                 info y = query(0, N - 1, 1, 0, N - 1);
88                 if (y.sum == 0 && y.sub >= 0) printf("YES\n");
89                 else printf("NO\n");
90             }
91         }
92     }
93     return 0;

```

93 }

1.2.5 Sliding Window RMQ

```
1  /**
2     every K size window RMQ
3     Calculate in O(N+K) time
4  */
5  #include <bits/stdc++.h>
6  using namespace std;
7  vector<int> SlidingRMQ(int *A, int N, int k)
8  {
9      /** Create a Double Ended Queue, Qi that will store indexes of
10         array elements
11         The queue will store indexes of useful elements in every
12         window and it will
13         maintain decreasing order of values from front to rear in
14         Qi, i.e.,
15         arr[Qi.front()] to arr[Qi.rear()] are sorted in increasing
16         order
17     */
18     vector<int> MinWindow;
19     deque<int> Q;
20     int i;
21     /* Process first k (or first window) elements of array */
22     for (i=0; i<k; i++)
23     {
24         /// For very element, the previous largest elements are
25         /// useless so
26         /// remove them from Qi
27         while (!Q.empty() and A[i]<=A[Q.back()]) Q.pop_back();
28         Q.push_back(i);
29     }
30     /// Process rest of the elements, i.e., from arr[k] to arr[n-1]
31     while (i<N)
32     {
33         /// The element at the front of the queue is the smallest
34         /// element of
35         /// previous window, so insert it result
36         MinWindow.push_back(A[Q.front()]);
37
38         /// Remove the elements which are out of this window
39         while (!Q.empty() and Q.front()<=i-k) Q.pop_front();
40
41         /// Remove all elements larger than the currently
42         /// being added element (remove useless elements)
43         while (!Q.empty() and A[i]<=A[Q.back()]) Q.pop_back();
44
45         /// Add current element at the rear of Qi
46         Q.push_back(i);
47         i++;
48     }
49     /// insert the minimum element of last window
50     MinWindow.push_back(A[Q.front()]);
51     return MinWindow;
52 }
53 int main()
```

```

48 {
49     int A[]={100,10, -1, 2,-3,-4,10, 1,100,20};
50     vector<int>a=SlidingRMQ(A,10,2);
51     for (int i=0;i<a.size();i++)cout<<a[i]<<" ";
52     return 0;
53 }

```

1.2.6 Sparse Table

```

1  /**
2   Compute sparse table in O(NlogN)
3   query in O(1)
4   Ref link: https://www.topcoder.com/community/data-science/data-science-tutorials/range-minimum-query-and-lowest-common-ancestor/
5  */
6  #include<bits/stdc++.h>
7  using namespace std;
8  #define Max 10000005
9  int rmq[24][Max];
10 int A[Max];
11 void Compute_ST(int N)
12 {
13     for (int i = 0; i < N; ++i)rmq[0][i] = i;
14     for (int k = 1; (1 << k) < N; ++k)
15     {
16         for (int i = 0; i + (1 << k) <= N; i++)
17         {
18             int x = rmq[k - 1][i];
19             int y = rmq[k - 1][i + (1 << k - 1)];
20             rmq[k][i] = A[x] <= A[y] ? x : y;
21         }
22     }
23 }
24
25 int RMQ(int i, int j)
26 {
27     int k = log2(j-i);
28     int x = rmq[k][i];
29     int y = rmq[k][j - (1 << k) + 1];
30     return A[x] <= A[y] ? x : y;
31 }
32
33 int main()
34 {
35
36     return 0;
37 }

```

1.3 Heavy Light Decomposition

```

1  /*
2   Tanvir Hasan Anick
3   University of Asia pacific
4  */
5  /**Header file**/

```

```

6 #include<stdio>
7 #include<iomanip>
8 #include<cstring>
9 #include<cmath>
10 #include<cstdlib>
11 #include<cctype>
12 #include<algorithm>
13 #include<string>
14 #include<vector>
15 #include<queue>
16 #include<map>
17 #include<set>
18 #include<sstream>
19 #include<stack>
20 #include<list>
21 #include<iostream>
22 #include<assert.h>
23
24 /**Define file I/O **/
25 #define f_input freopen("input.txt","r",stdin)
26 #define f_output freopen("output.txt","w",stdout)
27
28 /**Define memory set function**/
29 #define mem(x,y) memset(x,y,sizeof(x))
30 #define CLEAR(x) memset(x,0,sizeof(x))
31
32 /**Define function and object**/
33 #define pb push_back
34 #define Sort(v) sort(v.begin(),v.end())
35 #define RSort(v) sort(v.rbegin(),v.rend())
36 #define CSort(v,C) sort(v.begin(),v.end(),C)
37 #define all(v) (v).begin(),(v).end()
38 #define sqr(x) ((x)*(x))
39 #define find_dist(a,b) sqrt(sqr(a.x-b.x)+sqr(a.y-b.y))
40
41 /**Define constant value**/
42 #define ERR 1e-9
43 #define pi (2*acos(0))
44 #define PI 3.141592653589793
45
46 /**Define input**/
47 #define scanint(a) scanf("%d",&a)
48 #define scanLLD(a) scanf("%lld",&a)
49 #define scanstr(s) scanf("%s",s)
50 #define scanline(l) scanf("%[^\n]",l);
51
52 /**Define Bitwise operation**/
53 #define check(n, pos) (n & (1<<(pos)))
54 #define biton(n, pos) (n | (1<<(pos)))
55 #define bitoff(n, pos) (n & ~(1<<(pos)))
56
57 /**Define color**/
58 #define WHITE 0
59 #define GREY 1
60 #define BLACK 2
61
62 /**Sync off with stdio**/

```

```

63 #define __ cin.sync_with_stdio(false);\
64     cin.tie();
65
66 using namespace std;
67
68 /**Typedef**/
69 typedef vector<int> vint;
70 typedef vector< vint > vint2D;
71 typedef vector<string> vstr;
72 typedef vector<char> vchar;
73 typedef vector< vchar > vchar2D;
74 typedef queue<int> Qi;
75 typedef queue< Qi > Qii;
76 typedef map<int, int> Mii;
77 typedef map<string, int> Msi;
78 typedef map<int, string> Mis;
79 typedef stack<int> stk;
80 typedef pair<int, int> pp;
81 typedef pair<int, pp > ppp;
82 typedef long long int LLD;
83 const int inf=0x7FFFFFFF;
84
85 /**Template & structure**/
86 namespace my{
87 struct point_int{int x,y;point_int(){}point_int(int a,int b){x=a,y=
    b;}}; //Point for x,y (int) coordinate in 2D space
88 struct point_double{double x,y;point_double(){}point_double(double
    a,double b){x=a,y=b;}}; //Point for x,y (double) coordinate in
    2D space
89 struct Node{int v,w;Node() {}bool operator<(const Node &a)const{
    return w>a.w;}Node(int _v, int _w){v=_v,w=_w;}}; //Node for
    Dijkstra
90 template<class T> gcd(T a,T b){return b == 0 ? a : gcd(b, a % b);}
91 template<typename T> lcm(T a, T b) {return a / gcd(a,b) * b;}
92 template<class T>T big_mod(T n,T p,T m){if(p==0)return (T)1;T x=
    big_mod(n,p/2,m);x=(x*x)%m;if(p&1)x=(x*n)%m;return x;}
93 template<class T>T multiplication(T n,T p,T m){if(p==0)return (T)0;
    T x=multiplication(n,p/2,m);x=(x+x)%m;if(p&1)x=(x+n)%m;return x
    ;}
94 template<class T>T my_pow(T n,T p){if(p==0)return 1;T x=my_pow(n,p
    /2);x=(x*x);if(p&1)x=(x*n);return x;} //n to the power p
95 template <class T> double getdist(T a, T b){return sqrt((a.x - b.x)
    * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));} // distance
    between a & b
96 template <class T> T extract(string s, T ret) {stringstream ss(s);
    ss >> ret; return ret;} // extract words or numbers from a line
97 template <class T> string toString(T n) {stringstream ss; ss << n;
    return ss.str();} // convert a number to string
98 template<class T> inline T Mod(T n,T m) {return (n%m+m)%m;} //For
    Positive Negative No.
99 template<class T> T MIN3(T a,T b,T c) {return min(a,min(b,c));} //
    minimum of 3 number
100 template<class T> T MAX3(T a,T b,T c) {return max(a,max(b,c));} //
    maximum of 3 number
101 template <class T> void print_vector(T &v){int sz=v.size();if(sz)
    cout<<v[0];for(int i = 1; i < sz; i++)cout << ' '<<v[i];cout<<
    endl;} // prints all elements in a vector

```

```

102 bool isVowel(char ch){ ch=toupper(ch); if(ch=='A' || ch=='U' || ch=='I'
    || ch=='O' || ch=='E') return true; return false;}
103 bool isConsonant(char ch){if (isalpha(ch) && !isVowel(ch)) return
    true; return false;}}
104 /**Shortcut input function**/
105 int read_int(){int n;scanf("%d",&n);return n;}
106 int read_LLD(){LLD n;scanf("%lld",&n);return n;}
107 inline int buffer_input() { char inp[1000]; scanstr(inp); return
    atoi(inp); }
108
109 /**Direction**/
110 //int col[8] = {0, 1, 1, 1, 0, -1, -1, -1};int row[8] = {1, 1, 0,
    -1, -1, -1, 0, 1}; //8 Direction
111 //int col[4] = {1, 0, -1, 0};int row[4] = {0, 1, 0, -1}; //4
    Direction
112 //int dx[]={2,1,-1,-2,-2,-1,1,2};int dy
    []={1,2,2,1,-1,-2,-2,-1};//Knight Direction
113 //int dx[]={-1,-1,+0,+1,+1,+0};int dy[]={-1,+1,+2,+1,-1,-2}; //
    Hexagonal Direction
114
115
116 /*****Ajaira Jinish Sesh
    *****/
117
118 const int Max=10000;
119 struct info
120 {
121     int v, cost;
122     info(int v=0,int cost=0):v(v),cost(cost) {};
123 };
124 vector<pp>edges;
125 vector<info>Graph[Max+5];
126 int Tree[5*Max+5],BaseArray[Max+5],SubTreeSize[Max+5];
127 int ChainHead[Max+5],ChainNum[Max+5],PosInBaseArray[Max+5],ChainNo;
128 int Level[Max+5],Parent[Max+5],SparseTable[Max+5][16];
129 int ptr;
130 void init(int N)
131 {
132     for(int i=0;i<=N;i++)
133     {
134         Graph[i].clear(),ChainHead[i]=-1;
135         for(int j=0;j<=15;j++)SparseTable[i][j]=-1;
136     }
137     edges.clear();
138     ptr=ChainNo=0;
139 }
140 void buildSegmentTree(int l,int r,int indx)
141 {
142     if(l==r)
143     {
144         Tree[indx]=BaseArray[l];
145         return;
146     }
147     int mid=(l+r)>>1;
148     int lindx=indx<<1;
149     int rindx=lindx|1;
150     buildSegmentTree(l,mid,lindx);

```

```

151     buildSegmentTree(mid+1,r,rindx);
152     Tree[indx]=max(Tree[lindx],Tree[rindx]);
153 }
154 void updateSegmentTree(int l,int r,int indx,int update_indx,int
    value)
155 {
156     if(l==r)
157     {
158         Tree[indx]=value;
159         return;
160     }
161     int mid=(l+r)>>1;
162     int lindx=indx<<1;
163     int rindx=indx|1;
164     if(update_indx<=mid)updateSegmentTree(l,mid,lindx,update_indx,
        value);
165     else updateSegmentTree(mid+1,r,rindx,update_indx,value);
166     Tree[indx]=max(Tree[lindx],Tree[rindx]);
167 }
168 int querySegmentTree(int l,int r,int indx,int x,int y)
169 {
170     if(l>y||r<x)return 0;
171     if(x<=l&&y>=r)return Tree[indx];
172     int mid=(l+r)>>1;
173     int lindx=indx<<1;
174     int rindx=indx|1;
175     int c1=0,c2=0;
176     if(x<=mid)c1=querySegmentTree(l,mid,lindx,x,y);
177     if(y>mid)c2=querySegmentTree(mid+1,r,rindx,x,y);
178     return max(c1,c2);
179 }
180 void dfs(int from,int u,int depth)
181 {
182     Level[u]=depth;
183     Parent[u]=from;
184     SubTreeSize[u]=1;
185     int sz=Graph[u].size();
186     for(int i=0;i<sz;i++)
187     {
188         int v=Graph[u][i].v;
189         if(v==from)continue;
190         dfs(u,v,depth+1);
191         SubTreeSize[u]+=SubTreeSize[v];
192     }
193 }
194 void sparseTable(int N)
195 {
196     for(int i=0;i<=N;i++)SparseTable[i][0]=Parent[i];
197     for(int j=1;(1<j)<=N;j++)
198     {
199         for(int i=0;i<=N;i++)
200         {
201             if(SparseTable[i][j-1]!=-1)
202             {
203                 int a=SparseTable[i][j-1];
204                 SparseTable[i][j]=SparseTable[a][j-1];
205             }

```

```

206     }
207 }
208 }
209 int LCA(int p, int q)
210 {
211     if (Level[p] < Level[q]) swap(p, q);
212     int Log = log2(Level[p]) + 1;
213     for (int i = Log; i >= 0; i--)
214     {
215         if ((Level[p] - (1 << i)) >= Level[q]) p = SparseTable[p][i];
216     }
217     if (p == q) return p;
218     for (int i = Log; i >= 0; i--)
219     {
220         if (SparseTable[p][i] != -1 && SparseTable[p][i] != SparseTable[q][i])
221         {
222             p = SparseTable[p][i], q = SparseTable[q][i];
223         }
224     }
225     return Parent[p];
226 }
227 /**
228  * Actual HL-Decomposition part
229  * Initially all entries of chainHead[] are set to -1.
230  * So when ever a new chain is started, chain head is correctly
231  * assigned.
232  * As we add a new node to chain, we will note its position in the
233  * baseArray.
234  * In the first for loop we find the child node which has maximum
235  * sub-tree size.
236  * The following if condition is failed for leaf nodes.
237  * When the if condition passes, we expand the chain to special
238  * child.
239  * In the second for loop we recursively call the function on all
240  * normal nodes.
241  * chainNo++ ensures that we are creating a new chain for each
242  * normal child.
243  */
244 void heavyLightDecompositon(int from, int curNode, int cost)
245 {
246     if (ChainHead[ChainNo] == -1) ChainHead[ChainNo] = curNode; // Assign chain head
247     ChainNum[curNode] = ChainNo;
248     PosInBaseArray[curNode] = ptr; // Position of this node in
249     baseArray which we will use in Segtree
250     BaseArray[ptr++] = cost;
251     int sc = -1, nextCost;
252     int sz = Graph[curNode].size();
253     for (int i = 0; i < sz; i++) // Loop to find special child
254     {
255         int v = Graph[curNode][i].v;
256         if (v == from) continue;
257         if (sc == -1 || SubTreeSize[sc] < SubTreeSize[v])
258         {
259             sc = v;
260             nextCost = Graph[curNode][i].cost;

```



```

254     }
255 }
256 if (sc != -1) heavyLightDecompositon (curNode , sc , nextCost); ///
Expand the chain
257 for (int i=0; i<sz; i++)
258 {
259     int v=Graph[curNode][i].v;
260     int cost=Graph[curNode][i].cost;
261     if (v==from || sc==v) continue;
262     ChainNo++;
263     heavyLightDecompositon (curNode , v , cost);
264 }
265 }
266 void updateTree (int ith , int val)
267 {
268     pp a=edges [ith];
269     int u=a.first , v=a.second;
270     int indx=PosInBaseArray [u];
271     if (Level[u]<Level[v]) indx=PosInBaseArray [v];
272     updateSegmentTree (0 , ptr-1 , 1 , indx , val);
273 }
274 /**
275  * query-up:
276  * It takes two nodes u and v , condition is that v is an ancestor
277  * of u
278  * We query the chain in which u is present till chain head , then
279  * move to next chain up
280  * We do that way till u and v are in the same chain , we query for
281  * that part of chain and break
282  */
283 int queryUp (int u , int v)
284 {
285     if (u==v) return 0;
286     int uchain , vchain=ChainNum [v] , ans=-1;
287     while (true)
288     {
289         uchain=ChainNum [u];
290         if (uchain==vchain)
291         {
292             if (u==v) // Both u and v are in the same chain ,
293                 // so we need to query from u to v , update answer and break.
294                 break; // We break because we came from u up
295             // till v , we are done
296             ans=max (ans , querySegmentTree (0 , ptr-1 , 1 , PosInBaseArray [v]
297                 +1 , PosInBaseArray [u]));
298             break;
299         }
300         int uchainhead=ChainHead [uchain];
301         ans=max (ans , querySegmentTree (0 , ptr-1 , 1 , PosInBaseArray [
302             uchainhead] , PosInBaseArray [u]));
303         // Above is call to segment tree query
304         // function . We do from chainHead of u till u . That is the whole
305         // chain from
306         u=Parent [uchainhead];
307     }
308     return ans;
309 }

```

```

301 int queryTree(int u, int v)
302 {
303     int lca=LCA(u,v);
304     return max(queryUp(u,lca),queryUp(v,lca));
305 }
306 int main()
307 {
308     #ifdef _ANICK_
309     //f_input;
310     #endif // _ANICK_
311     --
312     int test;
313     cin>>test;
314     while(test--)
315     {
316         int N;
317         cin>>N;
318         init(N);
319         for(int i=0;i<N-1;i++)
320         {
321             int u,v,c;
322             cin>>u>>v>>c;
323             u--,v--;
324             Graph[u].pb(info(v,c));
325             Graph[v].pb(info(u,c));
326             edges.pb(pp(u,v));
327         }
328         dfs(-1,0,0);
329         sparseTable(N);
330         heavyLightDecompositon(-1,0,-1);
331         buildSegmentTree(0,ptr-1,1);
332         string ch;
333         int x,y;
334         while(true)
335         {
336             cin>>ch;
337             if(ch[0]=='D') break;
338             cin>>x>>y;
339             if(ch[0]=='Q') printf("%d\n",queryTree(x-1,y-1));
340             else if(ch[0]=='C') updateTree(x-1,y);
341         }
342     }
343     return 0;
344 }

```

1.4 Ternary Bit Mask

```

1
2 int more_bit[10];
3 int get_bit(int mask , int pos)
4 {
5     return (mask / more_bit[pos]) % 3;
6 }
7 int set_bit(int mask, int pos , int bit)
8 {
9     int tmp = (mask / more_bit[pos]) % 3;
10    mask -= tmp * more_bit[pos];

```

```
11     mask += bit * more_bit[pos];
12     return mask;
13 }
14 void init(void)
15 {
16     more_bit[0] = 3;
17     for(int i = 1; i < 10; i++) more_bit[i] = 3 * more_bit[i - 1];
18 }
```

Chapter 2

Graph Theory

2.1 DFS

2.1.1 Bicoloring

```
1 ///color will be initial with -1
2 int color[20005];
3 bool dfs(int u, int c)
4 {
5     if (color[u]==c) return true;
6     if (color[u]==(1-c)) return false;
7     color[u]=c;
8     bool ret=true;
9     for (auto v:graph[u]) ret&=dfs(v,1-c);
10    return ret;
11 }
```

2.1.2 Cycle Finding

```
1 int color[20005];
2 bool dfs(int u)
3 {
4     color[u]=GREY;
5     bool no_cycle=true;
6     for (auto v:graph[u])
7     {
8         if (color[v]==WHITE)
9         {
10             no_cycle=dfs(v);
11         }
12         else if (color[v]==GREY) return false;
13     }
14     color[u]=BLACK;
15     return no_cycle;
16 }
```

2.2 Topological Sort

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define WHITE 0
4 #define GREY 1
5 #define BLACK 2
6 vector<int> graph[100005];
7 vector<int> ans;
8 int visit[100005];
9 bool dfs(int u)
10 {
11     visit[u]=GREY;
12     bool no_cycle=true;
13     int sz=graph[u].size();
14     for(int i=0;i<sz;i++)
15     {
16         int v=graph[u][i];
17         if(visit[v]==WHITE)
18         {
19             no_cycle=dfs(v);
20         }
21         else if(visit[v]==GREY) return false;
22     }
23     visit[u]=BLACK;
24     ans.push_back(u);
25     return no_cycle;
26 }
27 bool topsort(int N)
28 {
29     ans.clear();
30     memset(visit, false, sizeof(visit));
31     int no_cycle=true;
32     for(int i=0;i<N;i++)
33     {
34         if(visit[i]==WHITE) no_cycle&=dfs(i);
35     }
36     return no_cycle;
37 }
38 int main()
39 {
40     #ifdef _ANICK_
41     //f_input;
42     #endif // _ANICK_
43     return 0;
44 }

```

Chapter 3

Flow networks/ matching

3.1 Max Flow

```
1  /*
2     Tanvir Hasan Anick
3     University of Asia pacific
4  */
5  /**Header file**/
6  #include<cstdio>
7  #include<iomanip>
8  #include<cstring>
9  #include<cmath>
10 #include<cstdlib>
11 #include<cctype>
12 #include<algorithm>
13 #include<string>
14 #include<vector>
15 #include<queue>
16 #include<map>
17 #include<set>
18 #include<sstream>
19 #include<stack>
20 #include<list>
21 #include<iostream>
22 #include<assert.h>
23
24 /**Define file I/O **/
25 #define f_input freopen("input.txt","r",stdin)
26 #define f_output freopen("output.txt","w",stdout)
27
28 /**Define memory set function**/
29 #define mem(x,y) memset(x,y,sizeof(x))
30 #define CLEAR(x) memset(x,0,sizeof(x))
31
32 /**Define function and object**/
33 #define pb push_back
34 #define Sort(v) sort(v.begin(),v.end())
35 #define RSort(v) sort(v.rbegin(),v.rend())
36 #define CSort(v,C) sort(v.begin(),v.end(),C)
```

```

37 #define all(v) (v).begin(),(v).end()
38 #define sqr(x) ((x)*(x))
39 #define find_dist(a,b) sqrt(sqr(a.x-b.x)+sqr(a.y-b.y))
40
41 /**Define constant value**/
42 #define ERR 1e-9
43 #define pi (2*acos(0))
44 #define PI 3.141592653589793
45
46 /**Define input**/
47 #define scanint(a) scanf("%d",&a)
48 #define scanLLD(a) scanf("%lld",&a)
49 #define scanstr(s) scanf("%s",s)
50 #define scanline(l) scanf("%[^\n]",l);
51
52 /**Define Bitwise operation**/
53 #define check(n, pos) (n & (1<<(pos)))
54 #define biton(n, pos) (n | (1<<(pos)))
55 #define bitoff(n, pos) (n & ~(1<<(pos)))
56
57 /**Define color**/
58 #define WHITE 0
59 #define GREY 1
60 #define BLACK 2
61
62 /**Sync off with stdio**/
63 #define -- cin.sync_with_stdio(false);\
64         cin.tie();
65 /**Debug tools**/
66 #define what_is(x) cerr<<(#x)<<" is "<<x<<endl
67 using namespace std;
68
69 /**Typedef**/
70 typedef vector<int> vint;
71 typedef vector<vint> vint2D;
72 typedef vector<string> vstr;
73 typedef vector<char> vchar;
74 typedef vector<vchar> vchar2D;
75 typedef queue<int> Qi;
76 typedef queue<Qi> Qii;
77 typedef map<int,int> Mii;
78 typedef map<string,int> Msi;
79 typedef map<int,string> Mis;
80 typedef stack<int> stk;
81 typedef pair<int,int> pp;
82 typedef pair<int,pp> ppp;
83 typedef long long int LLD;
84 const int inf=0x7FFFFFFF;
85
86 /**Template & structure**/
87 struct point_int{int x,y;point_int(){}point_int(int a,int b){x=a,y=
    b;}}; //Point for x,y (int) coordinate in 2D space
88 struct point_double{double x,y;point_double(){}point_double(double
    a,double b){x=a,y=b;}}; //Point for x,y (double) coordinate in
    2D space
89 struct Node{int v,w;Node() {}bool operator<(const Node &a)const{
    return w>a.w;}Node(int _v,int _w){v=_v,w=_w;}}; //Node for

```

```

Dijkstra
90 namespace my{
91 template<class T>T gcd(T a,T b){return b == 0 ? a : gcd(b, a % b);}
92 template<typename T>T lcm(T a, T b) {return a / gcd(a,b) * b;}
93 template<class T>T big_mod(T n,T p,T m){if(p==0)return (T)1;T x=
    big_mod(n,p/2,m);x=(x*x)%m;if(p&1)x=(x*n)%m;return x;}
94 template<class T>T multiplication(T n,T p,T m){if(p==0)return (T)0;
    T x=multiplication(n,p/2,m);x=(x+x)%m;if(p&1)x=(x+n)%m;return x
    ;}
95 template<class T>T my_pow(T n,T p){if(p==0)return 1;T x=my_pow(n,p
    /2);x=(x*x);if(p&1)x=(x*n);return x;} //n to the power p
96 template <class T> double getdist(T a, T b){return sqrt((a.x - b.x)
    * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));} // distance
    between a & b
97 template <class T> T extract(string s, T ret) {stringstream ss(s);
    ss >> ret; return ret;} // extract words or numbers from a line
98 template <class T> string toString(T n) {stringstream ss; ss << n;
    return ss.str();} // convert a number to string
99 template<class T> inline T Mod(T n,T m) {return (n%m+m)%m;} //For
    Positive Negative No.
100 template<class T> T MIN3(T a,T b,T c) {return min(a,min(b,c));} //
    minimum of 3 number
101 template<class T> T MAX3(T a,T b,T c) {return max(a,max(b,c));} //
    maximum of 3 number
102 template <class T> void print_vector(T &v){int sz=v.size();if(sz)
    cout<<v[0];for(int i = 1; i < sz; i++)cout << ' ' <<v[i];cout<<
    endl;} // prints all elements in a vector
103 bool isVowel(char ch){ ch=toupper(ch); if(ch=='A' || ch=='U' || ch=='I'
    || ch=='O' || ch=='E') return true; return false;}
104 bool isConsonant(char ch){if (isalpha(ch) && !isVowel(ch)) return
    true; return false;}
105 /**Shortcut input function**/
106 int read_int(){int n;scanf("%d",&n);return n;}
107 int read_LLD(){LLD n;scanf("%lld",&n);return n;}
108 inline int buffer_input() { char inp[1000]; scanstr(inp); return
    atoi(inp); }
109
110 /**Direction**/
111 //int col[8] = {0, 1, 1, 1, 0, -1, -1, -1};int row[8] = {1, 1, 0,
    -1, -1, -1, 0, 1}; //8 Direction
112 //int col[4] = {1, 0, -1, 0};int row[4] = {0, 1, 0, -1}; //4
    Direction
113 //int dx[]={2,1,-1,-2,-2,-1,1,2};int dy
    []={1,2,2,1,-1,-2,-2,-1}; //Knight Direction
114 //int dx[]={-1,-1,+0,+1,+1,+0};int dy[]={-1,+1,+2,+1,-1,-2}; //
    Hexagonal Direction
115
116
117 /*****Ajaira Jinish Sesh
    *****/
118 #define MN 1000
119 vint2D graph;
120 int Cost[MN][MN];
121 int parent[MN+5];
122 int flow;
123 void init(int N)
124 {

```



```

125     graph=vint2D(N);
126     mem(Cost,0);
127 }
128 void AddEdge(int u,int v,int cost)
129 {
130     graph[u].pb(v);
131     graph[v].pb(u);
132     Cost[u][v]+=cost;
133     Cost[v][u]+=cost;
134 }
135 bool augmenting_path(int source,int sink)
136 {
137     mem(parent,-1);
138     queue<int>Q;
139     Q.push(source);
140     while(!Q.empty())
141     {
142         int u=Q.front();
143         Q.pop();
144         int sz=graph[u].size();
145         for(int i=0;i<sz;i++)
146         {
147             int v=graph[u][i];
148             if(parent[v]==-1 and Cost[u][v]>0)
149             {
150                 parent[v]=u;
151                 Q.push(v);
152                 if(v==sink) return true;
153             }
154         }
155     }
156     return false;
157 }
158 void path(int v,int source)
159 {
160     int u=parent[v];
161     flow=min(flow,Cost[u][v]);
162     if(source!=u) path(u,source);
163     Cost[u][v]-=flow;
164     Cost[v][u]+=flow;
165     return;
166 }
167 int max_flow(int source,int sink)
168 {
169     int ret=0;
170     while(augmenting_path(source,sink))
171     {
172         flow=inf;
173         path(sink,source);
174         ret+=flow;
175     }
176     return ret;
177 }
178 int main()
179 {
180     #ifdef _ANICK_
181     //f_input;

```

```

182 #endif // _ANICK_
183 int test;
184 scanf("%d",&test);
185 while(test--)
186 {
187     int P,S,C,M;
188     scanf("%d %d %d %d",&P,&S,&C,&M);
189     init(P+S+5);
190     int superSource=0,SuperSikn=P+S+1;
191     for(int i=1;i<=P;i++)AddEdge(superSource,i,1);
192     for(int i=1;i<=S;i++)AddEdge(P+1,SuperSikn,C);
193     for(int i=0;i<M;i++)
194     {
195         int x,y;
196         scanf("%d %d",&x,&y);
197         AddEdge(x,P+y,(1<30));
198     }
199     printf("%d\n",max_flow(superSource,SuperSikn));
200 }
201 return 0;
202 }

```

Chapter 4

Dynamic programming

4.1 Edit Distance

```
1  /*
2     Tanvir Hasan Anick
3     University of Asia pacific
4  */
5  /**Header file**/
6  #include<cstdio>
7  #include<iomanip>
8  #include<cstring>
9  #include<cmath>
10 #include<cstdlib>
11 #include<cctype>
12 #include<algorithm>
13 #include<string>
14 #include<vector>
15 #include<queue>
16 #include<map>
17 #include<set>
18 #include<sstream>
19 #include<stack>
20 #include<list>
21 #include<iostream>
22 #include<assert.h>
23
24 /**Define file I/O **/
25 #define f_input freopen("input.txt","r",stdin)
26 #define f_output freopen("output.txt","w",stdout)
27
28 /**Define memory set function**/
29 #define mem(x,y) memset(x,y,sizeof(x))
30 #define CLEAR(x) memset(x,0,sizeof(x))
31
32 /**Define function and object**/
33 #define pb push_back
34 #define Sort(v) sort(v.begin(),v.end())
35 #define RSort(v) sort(v.rbegin(),v.rend())
36 #define CSort(v,C) sort(v.begin(),v.end(),C)
```

```

37 #define all(v) (v).begin(),(v).end()
38 #define sqr(x) ((x)*(x))
39 #define find_dist(a,b) sqrt(sqr(a.x-b.x)+sqr(a.y-b.y))
40
41 /**Define constant value**/
42 #define ERR 1e-9
43 #define pi (2*acos(0))
44 #define PI 3.141592653589793
45
46 /**Define input**/
47 #define scanint(a) scanf("%d",&a)
48 #define scanLLD(a) scanf("%lld",&a)
49 #define scanstr(s) scanf("%s",s)
50 #define scanline(l) scanf("%[^\n]",l);
51
52 /**Define Bitwise operation**/
53 #define check(n, pos) (n & (1<<(pos)))
54 #define biton(n, pos) (n | (1<<(pos)))
55 #define bitoff(n, pos) (n & ~(1<<(pos)))
56
57 /**Define color**/
58 #define WHITE 0
59 #define GREY 1
60 #define BLACK 2
61
62 /**Sync off with stdio**/
63 #define -- cin.sync_with_stdio(false);\
64         cin.tie();
65 /**Debug tools**/
66 #define what_is(x) cerr<<(#x)<<" is "<<x<<endl
67 using namespace std;
68
69 /**Typedef**/
70 typedef vector<int> vint;
71 typedef vector<vint> vint2D;
72 typedef vector<string> vstr;
73 typedef vector<char> vchar;
74 typedef vector<vchar> vchar2D;
75 typedef queue<int> Qi;
76 typedef queue<Qi> Qii;
77 typedef map<int,int> Mii;
78 typedef map<string,int> Msi;
79 typedef map<int,string> Mis;
80 typedef stack<int> stk;
81 typedef pair<int,int> pp;
82 typedef pair<int,pp> ppp;
83 typedef long long int LLD;
84 const int inf=0x7FFFFFFF;
85
86 /**Template & structure**/
87 struct point_int{int x,y;point_int(){}point_int(int a,int b){x=a,y=
    b;}}; //Point for x,y (int) coordinate in 2D space
88 struct point_double{double x,y;point_double(){}point_double(double
    a,double b){x=a,y=b;}}; //Point for x,y (double) coordinate in
    2D space
89 struct Node{int v,w;Node() {}bool operator<(const Node &a)const{
    return w>a.w;}Node(int _v,int _w){v=_v,w=_w;}}; //Node for

```

```

Dijkstra
90 namespace my{
91 template<class T>T gcd(T a,T b){return b == 0 ? a : gcd(b, a % b);}
92 template<typename T>T lcm(T a, T b) {return a / gcd(a,b) * b;}
93 template<class T>T big_mod(T n,T p,T m){if(p==0)return (T)1;T x=
    big_mod(n,p/2,m);x=(x*x)%m;if(p&1)x=(x*n)%m;return x;}
94 template<class T>T multiplication(T n,T p,T m){if(p==0)return (T)0;
    T x=multiplication(n,p/2,m);x=(x+x)%m;if(p&1)x=(x+n)%m;return x
    ;}
95 template<class T>T my_pow(T n,T p){if(p==0)return 1;T x=my_pow(n,p
    /2);x=(x*x);if(p&1)x=(x*n);return x;} //n to the power p
96 template <class T> double getdist(T a, T b){return sqrt((a.x - b.x)
    * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));} // distance
    between a & b
97 template <class T> T extract(string s, T ret) {stringstream ss(s);
    ss >> ret; return ret;} // extract words or numbers from a line
98 template <class T> string toString(T n) {stringstream ss; ss << n;
    return ss.str();} // convert a number to string
99 template<class T> inline T Mod(T n,T m) {return (n%m+m)%m;} //For
    Positive Negative No.
100 template<class T> T MIN3(T a,T b,T c) {return min(a,min(b,c));} //
    minimum of 3 number
101 template<class T> T MAX3(T a,T b,T c) {return max(a,max(b,c));} //
    maximum of 3 number
102 template <class T> void print_vector(T &v){int sz=v.size();if(sz)
    cout<<v[0];for(int i = 1; i < sz; i++)cout << ' ' <<v[i];cout<<"
    \n";} // prints all elements in a vector
103 bool isVowel(char ch){ ch=toupper(ch); if(ch=='A' || ch=='U' || ch=='I'
    || ch=='O' || ch=='E') return true; return false;}
104 bool isConsonant(char ch){if (isalpha(ch) && !isVowel(ch)) return
    true; return false;}
105 /**Shortcut input function**/
106 int read_int(){int n;scanf("%d",&n);return n;}
107 int read_LLD(){LLD n;scanf("%lld",&n);return n;}
108 inline int buffer_input() { char inp[1000]; scanstr(inp); return
    atoi(inp); }
109
110 /**Direction**/
111 //int col[8] = {0, 1, 1, 1, 0, -1, -1, -1};int row[8] = {1, 1, 0,
    -1, -1, -1, 0, 1}; //8 Direction
112 //int col[4] = {1, 0, -1, 0};int row[4] = {0, 1, 0, -1}; //4
    Direction
113 //int dx[]={2,1,-1,-2,-2,-1,1,2};int dy
    []={1,2,2,1,-1,-2,-2,-1}; //Knight Direction
114 //int dx[]={-1,-1,+0,+1,+1,+0};int dy[]={-1,+1,+2,+1,-1,-2}; //
    Hexagonal Direction
115
116
117 /*****Ajaira Jinish Sesh
    *****/
118 int dp[88][88];
119 int N,M,step;
120 char S1[88],S2[88];
121 int solve(int i,int j)
122 {
123     if(i==N and j==M)return 0;
124     if(i==N)return M-j;

```

```

125     if (j==M) return N-i;
126     int &ret=dp[i][j];
127     if (ret!=-1) return ret;
128     ret=(1<<28);
129     if (S1[i]==S2[j]) ret=solve(i+1,j+1);
130     else
131     {
132         ret=min(ret, solve(i, j+1)+1);
133         ret=min(ret, solve(i+1, j)+1);
134         ret=min(ret, solve(i+1, j+1)+1);
135     }
136     return ret;
137 }
138 void pathPrint(int i, int j, int del, int ins, int st)
139 {
140     if (i==N&&j==M) return ;
141     if (i==N)
142     {
143         for (int k=j; k<M; k++, i++)
144         {
145             printf("%d Insert %d,%c\n", st++, i-del+1+ins, S2[k]);
146         }
147         return ;
148     }
149     if (j==M)
150     {
151         for (; i<N; i++)
152         {
153             printf("%d Delete %d\n", st++, i-del+1+ins);
154             del++;
155         }
156         return ;
157     }
158     int ret = solve(i, j);
159     int tmp;
160     if (S1[i]==S2[j])
161     {
162         tmp=solve(i+1, j+1);
163         if (ret==tmp)
164         {
165             pathPrint(i+1, j+1, del, ins, st);
166             return ;
167         }
168     }
169     tmp=solve(i, j+1)+1;
170     if (tmp==ret)
171     {
172         printf("%d Insert %d,%c\n", st, i-del+1+ins, S2[j]);
173         pathPrint(i, j+1, del, ins+1, st+1);
174         return ;
175     }
176     tmp=solve(i+1, j)+1;
177     if (tmp==ret)
178     {
179         printf("%d Delete %d\n", st, i-del+1+ins);
180         pathPrint(i+1, j, del+1, ins, st+1);
181         return ;

```

```

182     }
183     tmp=solve ( i+1,j+1)+1;
184     if (tmp==ret)
185     {
186         printf ("%d Replace %d,%c\n",st,i-del+1+ins,S2[j]);
187         pathPrint (i+1,j+1,del,ins,st+1);
188         return ;
189     }
190     return ;
191 }
192 int main()
193 {
194     #ifdef _ANICK_
195     //f_input;
196     #endif // _ANICK_
197     bool New=false;
198     while (gets (S1))
199     {
200         gets (S2);
201         if (New) printf ("\n");
202         New=true;
203         N=strlen (S1);
204         M=strlen (S2);
205         mem(dp,-1);
206         step=solve (0,0);
207         printf ("%d\n",step);
208         pathPrint (0,0,0,0,1);
209     }
210     return 0;
211 }

```

Chapter 5

Strings

5.1 KMP

Tutorial

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 char TXT[10000000], ptr[10000000];
4 vector<int> compute_prefix(const char *p)
5 {
6     int m=strlen(p+1);
7     vector<int> prefix(m+1);
8     prefix[1]=0;
9     int k=0;
10    for(int i=2; i<=m; i++)
11    {
12        while(k>0 and p[k+1]!=p[i]) k=prefix[k];
13        if(p[k+1]==p[i]) k=k+1;
14        prefix[i]=k;
15    }
16    return prefix;
17 }
18 vector<int> KMP_match(const char *txt, const char *ptrn)
19 {
20     int n=strlen(txt+1);
21     int m=strlen(ptrn+1);
22     vector<int> Prefix=compute_prefix(ptrn);
23     vector<int> Match_position;
24     int q=0;
25     for(int i=1; i<=n; i++)
26     {
27         while(q>0 and ptrn[q+1]!=txt[i]) q=Prefix[q];
28         if(ptrn[q+1]==txt[i]) q=q+1;
29         if(q==m)
30         {
31             Match_position.push_back(i-m);
32             q=Prefix[q];
33         }
34     }
```



```

35     return Match_position;
36 }
37 int main()
38 {
39     scanf("%s %s",TXT+1,ptr+1);
40     vector<int> Match_position=KMP.match(TXT,ptr);
41     for(int i=0; i<Match_position.size(); i++)
42     {
43         if(!i)printf("%d",Match_position[i]);
44         else printf(" %d",Match_position[i]);
45     }
46     return 0;
47 }

```

5.2 Aho Corasick

5.2.1 Aho Corasick with Dynamic Trie

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define Max 26
4  int getID(char c)
5  {
6      return c>='a'?c-'a':c-'A';
7  }
8  char inp[1000005];
9  char text[1000005];
10 int ans[5000];
11 map<string,int>Map;
12 vector<int>v;
13 struct Trie
14 {
15     Trie *next[26],*fail;
16     int stringMap;
17     Trie()
18     {
19         stringMap=0;
20         for(int i=0;i<Max;i++)next[i]=NULL;
21         fail=NULL;
22     }
23 };
24 Trie *root;
25 void Insert(const char *str,int M)
26 {
27     Trie *p=root;
28     for(int i=0;str[i];i++)
29     {
30         int id=getID(str[i]);
31         if(p->next[id]==NULL)p->next[id]=new Trie();
32         p=p->next[id];
33     }
34     p->stringMap=M;
35 }
36 void computeFailure()
37 {

```

```

38     Trie *u,*prefix;
39     queue<Trie*>Q;
40     Q.push(root);
41     while(!Q.empty())
42     {
43         u=Q.front(); //Take a new node
44         Q.pop();
45         for(int i=0;i<Max;i++)
46         {
47             if(u->next[i]!=NULL) //select fail position of ith
node of parent u
48             {
49                 prefix=u->fail; // Going to u node fail position/
prefix position
50                 while(prefix!=NULL)
51                 {
52                     if(prefix->next[i]!=NULL) //if match found
53                     {
54                         u->next[i]->fail=prefix->next[i];
55                         break;
56                     }
57                     prefix=prefix->fail; // match not found, going
to upper child prefix position
58                 }
59                 if(prefix==NULL)u->next[i]->fail=root;
60                 Q.push(u->next[i]);
61             }
62         }
63     }
64 }
65 void AhoCorasick(const char *str)
66 {
67     Trie *p=root;
68     int cnt=0;
69     for(int i=0;str[i];i++)
70     {
71         int id=getID(str[i]);
72         while(p->next[id]==NULL&&p!=root)p=p->fail, cnt++;
73         if(p->next[id]!=NULL)p=p->next[id];
74         Trie *tp=p;
75         while(tp!=root)
76         {
77             cnt++;
78             if(tp->stringMap>0)ans[tp->stringMap]++;
79             tp=tp->fail;
80         }
81     }
82 }
83 void Delete(Trie *u)
84 {
85     if(u==NULL)return;
86     for(int i=0;i<Max;i++)Delete(u->next[i]);
87     delete u;
88 }
89
90 int main()
91 {

```

```

92     int test;
93     scanf("%d",&test);
94     for(int t=1;t<=test;t++)
95     {
96         Map.clear();
97         v.clear();
98         memset(ans,0,sizeof(ans));
99         root=new Trie();
100        int N;
101        scanf("%d",&N);
102        scanf("%s",text);
103        int cnt=1;
104        for(int i=0;i<N;i++)
105        {
106            scanf("%s",inp);
107            if(Map.find(inp)==Map.end())Map[inp]=cnt++;
108            Insert(inp,Map[inp]);
109            v.push_back(Map[inp]);
110        }
111        computeFailure();
112        AhoCorasick(text);
113        printf("Case %d:\n",t);
114        for(int i=0;i<N;i++)
115        {
116            printf("%d\n",ans[v[i]]);
117        }
118        Delete(root);
119    }
120    return 0;
121 }

```

5.2.2 Aho Corasick with Static Trie

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define root 0
4  #define NuLL -1
5  #define Max 248878
6  #define MC 26
7  int ans[10000];
8  char text[1000005];
9  char inp[100000];
10 map<string,int>Map;
11 vector<int> v;
12 int getID(const char c)
13 {
14     return c>='a'?c-'a':c-'A';
15 }
16 struct Trie
17 {
18     struct node
19     {
20         int Next[26],fail;
21         int stringMap;
22         void clear()
23         {
24             memset(Next,-1,sizeof(Next));

```

```

25         fail=-1;
26         stringMap=0;
27     }
28 }T[Max];
29 int ptr;
30 void clear()
31 {
32     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         if(p<0)
49         {
50             while(1);
51         }
52     }
53     T[p].stringMap=M;
54 }
55 void ComputeFailure()
56 {
57     queue<int>Q;
58     Q.push(root);
59     int u,prefix;
60     int cnt=0,cnt2=0;
61     while(!Q.empty())
62     {
63         u=Q.front();
64         Q.pop();
65         for(int i=0;i<MC;i++)
66         {
67             if(T[u].Next[i]!=NuLL)
68             {
69                 int now=T[u].Next[i];
70                 prefix=T[u].fail;
71                 while(prefix!=NuLL)
72                 {
73                     cnt2++;
74                     if(T[prefix].Next[i]!=NuLL)
75                     {
76                         T[now].fail=T[prefix].Next[i];
77                         break;
78                     }
79                     prefix=T[prefix].fail;
80                 }
81                 if(prefix==NuLL)T[now].fail=root;

```

```

82         Q.push(now);
83     }
84 }
85 }
86 }
87 };
88 void AhoCorasick(const Trie &A, const char *str)
89 {
90     int p=root;
91     int cnt1=0,cnt2=0;
92     for(int i=0;str[i];i++)
93     {
94         int id=getID(str[i]);
95         while(A.T[p].Next[id]==NULL&&p!=root)p=A.T[p].fail;
96         if(p!=NULL&&A.T[p].Next[id]!=NULL)p=A.T[p].Next[id];
97         int tp=p;
98         while(tp!=root)
99         {
100             if(A.T[tp].stringMap>0)ans[A.T[tp].stringMap]++;
101             tp=A.T[tp].fail;
102         }
103     }
104 }
105 Trie A;
106 int main()
107 {
108     #ifdef _ANICK_
109         freopen("input.txt","r",stdin);
110     #endif // _ANICK_
111     int test;
112     scanf("%d",&test);
113     for(int t=1;t<=test;t++)
114     {
115         Map.clear();
116         v.clear();
117         memset(ans,0,sizeof(ans));
118         A.clear();
119         int N;
120         scanf("%d",&N);
121         scanf("%s",text);
122         int cnt=1;
123         for(int i=0;i<N;i++)
124         {
125             scanf("%s",inp);
126             if(Map.find(inp)==Map.end())Map[inp]=cnt++;
127             A.Insert(inp,Map[inp]);
128             v.push_back(Map[inp]);
129         }
130         A.ComputeFailure();
131         AhoCorasick(A,text);
132         printf("Case %d:\n",t);
133         for(int i=0;i<N;i++)
134         {
135             printf("%d\n",ans[v[i]]);
136         }
137     }
138     return 0;

```

139 }

5.3 Manacher's Algorithm

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 string s, t;
4 char str[1000005];
5 void prepare_string()
6 {
7     int i;
8     t = "^#";
9     for(i = 0; i < s.size(); i++)
10         t += s[i], t += "#";
11     t += "$";
12 }
13
14 int manacher()
15 {
16     prepare_string();
17
18     int P[t.size()], c = 0, r = 0, i, i_mirror, n = t.size() - 1;
19
20     for(i = 1; i < n; i++)
21     {
22         i_mirror = (2 * c) - i;
23
24         P[i] = r > i? min(r - i, P[i_mirror]) : 0;
25
26         while(t[i + 1 + P[i]] == t[i - 1 - P[i]])
27             P[i]++;
28
29         if(i + P[i] > r)
30         {
31             c = i;
32             r = i + P[i];
33         }
34     }
35     return *max_element(P + 1, P + n);
36 }
37
38 int main()
39 {
40     int kase = 1;
41     while(scanf("%s", str) && str[0] != 'E')
42     {
43         s = str;
44         printf("Case %d: %d\n", kase++, manacher());
45     }
46     return 0;
47 }
```

Chapter 6

Computational geometry

Chapter 7

Math

7.1 Reduce Ratio

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

```
1 int main()
2 {
3     int A,B,x,y;
4     cin>>A>>B>>x>>y;
5     int g=__gcd(x,y);
6     x/=g,y/=g;
7     int t=min(A/x,B/y);
8     cout<<x*t<<" "<<y*t<<endl;
9     return 0;
10 }
```


Chapter 8

Number Theory

8.1 NCR

8.1.1 Lucas Theorem

```
1  /**
2   * Fine NCR % M when N C M are large number.
3   * using Lucas theorem.
4   */
5  #include <bits/stdc++.h>
6  using namespace std;
7  typedef long long LLD;
8  LLD mod=1000003;
9  LLD big_mod(LLD n,LLD p,LLD m)
10 {
11     if(p==0)return (LLD) 1;
12     LLD x=big_mod(n,p/2,m);
13     x=(x*x)%m;
14     if(p&1)x=(x*n)%m;
15     return x;
16 }
17 LLD inverse_modulo(LLD t,LLD m)
18 {
19     return big_mod(t,m-2,m);
20 }
21 LLD combi(LLD n, LLD k,LLD m)
22 {
23     if(n<k)
24         return 0;
25     if(n-k<k)
26         return combi(n,n-k,m);
27     LLD i ,p=1,t=1;
28     for(i=n-k+1; i<=n; i++)
29         p=(p*i)%m;
30     for(i=1; i<=k; i++)
31         t=(t*i)%m;
32     return (p*inverse_modulo(t,m))%m;
33 }
```

```

34 LLD lucas(LLD n, LLD k, LLD m)
35 {
36     if (n < k)
37         return 0;
38     if (k == 0 || n == k)
39         return 1;
40     return (lucas(n/m, k/m, m) * combi(n%m, k%m, m)) % m;
41 }
42 int main()
43 {
44     return 0;
45 }

```