

Algorithm Code Book

Tanvir Hasan Anick

December 4, 2015

Contents

1	Data Structure	4
1.1	Trie	4
1.1.1	Static Trie	4
1.2	RMQ	5
1.2.1	Bit	5
1.2.2	Square Root Decompostion	6
1.2.3	MO's Algorithm	7
1.2.4	Segment Tree	9
1.2.5	Sliding Window RMQ	17
1.2.6	Sparse Table	18
1.3	Ternary Bit Mask	18
1.4	Largest Rectangle in Histogram	19
1.4.1	Largest rectangle in histogram using Stack	19
1.4.2	Largest rectangle in histogram using Segment Tree	19
1.5	Least common ancestor (LCA)	20
1.6	Union Find	22
2	Graph Theory	23
2.1	DFS	23
2.1.1	Bicoloring	23
2.1.2	Cycle Finding	23
2.2	Topological Sort	23
2.3	Strongly Connected Component	24
2.4	Havel Hakimi	25
2.5	Articulation Point/Bridge	26
2.5.1	Find Articulation Point:	28
2.5.2	Find Bridge version 1:	28
2.5.3	Find Bridge version 2:	29
2.6	Stable Marriage Problem	30
2.7	2 SAT	30
2.8	Heavy Light Decomposition	33
3	Flow networks, matching	38
3.1	Max Flow	38
3.2	Maximum Bipartite Matching	40

4	Dynamic programming	41
4.1	Longest Increasing sub sequence	41
4.1.1	LIS1	41
4.1.2	LIS(Stanford)	42
4.2	Edit Distance	43
4.3	Distinct Sub Sequence	45
4.4	Knuth Optimization	45
4.5	Max Sum	46
4.5.1	2D Max Sum 1	46
4.5.2	2D Max sum 2	46
4.5.3	3D Max sum	47
4.6	Minimum Vertex Cover	48
4.7	Nth Permutation Of a String	49
5	Strings	51
5.1	KMP	51
5.2	Aho Corasick	52
5.2.1	Aho Corasick with Dynamic Trie	52
5.2.2	Aho Corasick with Static Trie	54
5.3	Manacher's Algorithm	56
6	Math	58
6.1	Pick's Theorem:	58
6.2	Reduce Ratio	58
6.3	Floyd's Cycle Finding algorithm	58
6.4	Angle between clock hand	59
6.5	Big Integer	59
7	Game Theory	63
7.1	Min Max Algorithm	63
7.2	Grundy Number	64
8	Number Theory	65
8.1	Greatest common divisor (GCD)	65
8.2	Least common multiple (LCM)	65
8.3	Big Mod	65
8.3.1	Iterative version	65
8.3.2	Recursive version	65
8.4	Sieve of Eratosthenes	66
8.4.1	Sieve	66
8.4.2	Bitwise Sieve	66
8.5	Euler's totient function	67
8.5.1	Euler phi sieve version	67
8.5.2	Euler phi function	67
8.6	Number of Divisors	68
8.6.1	NOD sieve version	68
8.6.2	NOD function	68
8.7	Summation of Divisors	69
8.8	Extended Euclid GCD	69
8.9	Modular multiplicative inverse	69

8.9.1	Modular inverse using EGCD	69
8.9.2	Modular multiplicative inverse using Euler Theorem	69
8.10	NCR	70
8.10.1	Lucas Theorem	70
9	Computational geometry	71
9.1	Geo Library (Stanford)	71
9.2	Convex Hull	76
9.2.1	Convex Hull 1	76
9.2.2	Convex Hull (Stanford)	77
9.3	Determine a point inside polygon or not	79
9.4	Integer point in a Segment or lattice point	80
9.5	Segment Intersection	80
9.6	Verify Convex polygon	81
9.7	Closest Pair Algorithm	81

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
32         idx
33         while (idx != z) // at some iteration idx (y) will become z
34         {
35             sum -= Bit[idx]; // subtract Bit frequency which is between y
36             and "the same path"
37             idx -= (idx & -idx);
38         }
39     }
40 }

```

```

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     int 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   We were given M queries, we will re-order the queries in a particular
7   order and then process them.
8   Clearly, this is an off-line algorithm. Each query has L and R, we will
9   call them opening and closing.
10  Let us divide the given input array into Sqrt(N) blocks.
11  Each block will be N / Sqrt(N) = Sqrt(N) size.
12  Each opening has to fall in one of these blocks.
13  Each closing has to fall in one of these blocks.
14
15  All the queries are first ordered in ascending order of their block
16  number (block number is the block in which its opening falls).
17  Ties are ordered in ascending order of their R value.
18
19  */
20 #include<bits/stdc++.h>
21 using namespace std;
22 #define Mx 30005
23 #define MxNum 1000005
24 int BlockSize;
25 int Answer;
26 int Freq[MxNum],Num[Mx];
27 struct info
28 {
29     int L,R,qno;
30     info(int L=0,int R=0,int qno=0):L(L),R(R),qno(qno) {}
31     bool operator<(const info &a) const
32     {
33         if(L/BlockSize!=a.L/BlockSize) return L/BlockSize<a.L/BlockSize;
34         return R<a.R;
35     }
36 }

```



```

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<=y)
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 - 1. **
   Initially all elements are zero. Now you have to deal with two types of
   **operations
3  **1.Increase the numbers between indices i and j (inclusive) by 1. This **is
   **represented by the command '0 i j'.
4  **2.Answer how many numbers between indices i and j (inclusive) are **
   divisible by 3. This is represented by the command '1 i j'.
5  */
6  #include<bits/stdc++.h>
7  using namespace std;
8  #define Max 100010
9  int Tree[8*Max][4];
10 int lazy[8*Max];
11 int temp[4];
12 void build(int left,int right,int indx)

```

```

13 {
14     if (left==right)
15     {
16         Tree[indx][0]=1;
17         Tree[indx][1]=Tree[indx][2]=lazy[indx]=0;
18         return;
19     }
20     int mid=(left+right)/2;
21     build(left, mid, 2*indx);
22     build(mid+1, right, 2*indx+1);
23     for (int i=0; i<3; i++)
24     {
25         Tree[indx][i]=Tree[2*indx][i]+Tree[2*indx+1][i];
26     }
27 }
28 void update(int left, int right, int indx, int x, int y, int add)
29 {
30     if (lazy[indx])
31     {
32         int lazy_val=lazy[indx];
33         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, y);

```

```

74 }
75 int main()
76 {
77     int x,y;
78     int test;
79     scanf("%d",&test);
80     for(int t=1;t<=test;t++)
81     {
82         memset(lazy,0,sizeof(lazy));
83         int N,Q;
84         scanf("%d %d",&N,&Q);
85         build(0,N-1,1);
86         printf("Case %d:\n",t);
87         for(int i=0;i<Q;i++)
88         {
89             int d;
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):Lcnt(Lcnt),
23     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     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

```
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*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;

```

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 array
10       elements
11       The queue will store indexes of useful elements in every window and
12       it will
13       maintain decreasing order of values from front to rear in Qi, i.e.,
14       arr[Qi.front()] to arr[Qi.rear()] are sorted in increasing order
15     **/
16     vector<int>MinWindow;
17     deque<int>Q;
18     int i;
19     /* Process first k (or first window) elements of array */
20     for (i=0;i<k;i++)
21     {
22         /** For very element, the previous largest elements are useless so
23         remove them from Qi
24         while (!Q.empty() and A[i]<=A[Q.back()])Q.pop_back();
25         Q.push_back(i);
26     }
27     /** Process rest of the elements, i.e., from arr[k] to arr[n-1]
28     while(i<N)
29     {
30         /** The element at the front of the queue is the smallest element of
31         previous window, so insert it result
32         MinWindow.push_back(A[Q.front()]);
33
34         /** Remove the elements which are out of this window
35         while (!Q.empty() and Q.front()-i<=k)Q.pop_front();
36
37         /** Remove all elements larger than the currently
38         being added element (remove useless elements)
39         while (!Q.empty() and A[i]<=A[Q.back()])Q.pop_back();
40
41         /** Add current element at the rear of Qi
42         Q.push_back(i);
43         i++;
44     }
45     /** insert the minimum element of last window
46     MinWindow.push_back(A[Q.front()]);
47     return MinWindow;
48 }
49 int main()
50 {
51     int A[]={100,10, -1, 2,-3,-4,10, 1,100,20};
52     vector<int>a=SlidingRMQ(A,10,2);
53     for (int i=0;i<a.size();i++)cout<<a[i]<<" ";
54     return 0;
55 }
```

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 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     more_bit[0] = 3;
16     for(int i = 1; i < 10; i++) more_bit[i] = 3 * more_bit[i - 1];
17 }
```

1.4 Largest Rectangle in Histogram

1.4.1 Largest rectangle in histogram using Stack

```
1 int GetMaxArea(int *hist, int n)
2 {
3     stack<int> s;
4
5     int max_area = 0;
6     int tp;
7     int area_with_top;
8     int i = 0;
9     while (i < n)
10    {
11        if (s.empty() || hist[s.top()] <= hist[i])
12            s.push(i++);
13        else
14        {
15            tp = s.top();
16            s.pop();
17            area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
18            if (max_area < area_with_top)
19                max_area = area_with_top;
20        }
21    }
22    while (s.empty() == false)
23    {
24        tp = s.top();
25        s.pop();
26        area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
27
28        if (max_area < area_with_top)
29            max_area = area_with_top;
30    }
31    return max_area;
32 }
```

1.4.2 Largest rectangle in histogram using Segment Tree

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 #define MN 1000005
4 typedef long long LLD;
5 LLD Histogram[MN], N;
6 template<class T> T MAX3(T a, T b, T c)
7 {
8     return max(a, max(b, c));    //maximum of 3 number
9 }
10 struct info
11 {
12     int Num;
13     int pos;
14     info(int Num=0, int pos=0):Num(Num), pos(pos) {}
15 };
16 info Tree[3*MN];
17 info marge(const info &L, const info &R)
18 {
19     if (L.Num < R.Num) return L;
20     return R;
21 }
22 void build(int L, int R, int indx)
```

```

23 {
24     if (L==R)
25     {
26         Tree[indx]=info (Histogram [L] ,L) ;
27         return ;
28     }
29     int mid=(L+R)>>1;
30     build (L,mid,2*indx) ;
31     build (mid+1,R,2*indx+1) ;
32     Tree [indx]=marge ( Tree [2*indx] , Tree [2*indx+1]) ;
33 }
34 info query (int L,int R,int indx ,int x,int y)
35 {
36     if (L==x&&R==y) return Tree [indx] ;
37     int mid=(L+R)>>1;
38     if (y<=mid) return query (L,mid,2*indx ,x,y) ;
39     else if (x>mid) return query (mid+1,R,2*indx+1,x,y) ;
40     return marge ( query (L,mid,2*indx ,x,mid) , query (mid+1,R,2*indx+1,mid+1,y) ) ;
41 }
42 LLD GetMaximumArea (int L,int R)
43 {
44     if (L>R) return -1;
45     if (L==R) return Histogram [L] ;
46     info p=query (1,N,1 ,L,R) ;
47     LLD a=GetMaximumArea (L,p.pos-1) ;
48     LLD b=GetMaximumArea (p.pos+1,R) ;
49     LLD c=(LLD)p.Num*(R-L+1) ;
50     return MAX3(a,b,c) ;
51 }
52 int main ()
53 {
54     while (scanf ("%d",&N) and N)
55     {
56         for (int i=1; i<=N; i++)scanf ("%lld",&Histogram [i]) ;
57         build (1,N,1) ;
58         printf ("%lld\n" ,GetMaximumArea (1,N) ) ;
59     }
60     return 0;
61 }

```

1.5 Least common ancestor (LCA)

```

1 vector<pp>Graph [2005];
2 int Level [2005];
3 int Lavel_cost [2005];
4 int sparse_table [2005][20];
5 int sparse_cost [2005][20];
6 void init ()
7 {
8     mem (parents,-1);
9     mem (sparse_table,-1);
10    mem (sparse_cost,0);
11 }
12 void dfs (int from,int u,int deep)
13 {
14     Level [u]=deep;
15     parents [u]=from;
16     int sz=Graph [u].size ();
17     for (int i=0; i<sz; i++)
18     {

```

```

19         int v=Graph[u][i].second;
20         int c=Graph[u][i].first;
21         if(v==from) continue;
22         Cost[v]=c;
23         dfs(u,v,deep+1);
24     }
25 }
26 void Sparse(int N)
27 {
28     for(int i=0; i<N; i++)
29     {
30         sparse_cost[i][0]=Cost[i];
31         sparse_table[i][0]=parents[i];
32     }
33     for(int j=1; (1<j)<N; j++)
34     {
35         for(int i=0; i<N; i++)
36         {
37             if(sparse_table[i][j-1]!=-1)
38             {
39                 int a=sparse_table[i][j-1];
40                 sparse_table[i][j]=sparse_table[a][j-1];
41                 sparse_cost[i][j]=sparse_cost[i][j-1]+sparse_cost[a][j-1];
42             }
43         }
44     }
45 }
46 int LCA(int p,int q)
47 {
48     if(Level[p]<Level[q]) swap(p,q);
49     int Log=log2(Level[p])+1;
50     int ans=0;
51     for(int i=Log; i>=0; i--)
52     {
53         if((Level[p]-(1<i))>=Level[q])
54         {
55             ans+=sparse_cost[p][i];
56             p=sparse_table[p][i];
57         }
58     }
59     if(p==q) return ans;
60     for(int i=Log; i>=0; i--)
61     {
62         if(sparse_table[p][i]!=-1&&sparse_table[p][i]!=sparse_table[q][i])
63         {
64             ans+=sparse_cost[p][i]+sparse_cost[q][i];
65             p=sparse_table[p][i],q=sparse_table[q][i];
66         }
67     }
68     ans+=Cost[p]+Cost[q];
69     ///Lca will be parent[p];
70     return ans;
71 }

```

1.6 Union Find

```
1 int parent[10000];
2 void init(int N)
3 {
4     for(int i=0;i<=N;i++)
5         parent[i]=i;
6 }
7 int Find(int r)
8 {
9     if(parent[r]==r) return r;
10    return parent[r]=Find(parent[r]);
11 }
12 void make_parent(int p, int q)
13 {
14     parent[p]=q;
15 }
16 void check(int x, int y)
17 {
18     int p=Find(x);
19     int q=Find(y);
20     if(p!=q)
21     {
22         make_parent(p,q);
23         printf("Make friend\n");
24         return;
25     }
26     printf("they allready friend\n");
27     return;
28 }
```

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     return 0;
41 }

```

2.3 Strongly Connected Component

```

1 typedef vector<int> vint;
2 vint G[100000+5];
3 vint G2[100000+5];
4 vint sorted;
5 vint cycle;
6 bool color[100000+5];
7 void dfs(int u)
8 {
9     color[u]=false;
10    for(int i=0; i<G[u].size(); i++)
11    {
12        int v=G[u][i];
13        if(color[v])
14        {
15            dfs(v);
16        }
17    }
18    sorted.pb(u);
19 }
20 void dfs2(int u)
21 {
22     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 $d_1, d_2, d_3, \dots, d_n$. Is it possible to make a graph which have no cycle and different two node will be connected with one Edge?

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

$$d_i \geq d_{i+1}$$

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

```

1 #include<stdio.h>
2 #include<queue>
3 #include<vector>
4 using namespace std;
5 int main()
6 {
7     int N;
8     while (scanf("%d",&N) and N)
9     {
10         priority_queue<int>Q;
11         bool Ok=true;
12         int Odd_Node=0;
13         for (int i=0; i<N; i++)
14         {
15             int x;
16             scanf("%d",&x);
17             if (x>=N or x<0)Ok=false;
18             Odd_Node+=(x%2);
19             Q.push(x);
20         }
21         Ok&=(Odd_Node%2==0); //Handshaking Theorem
22         for (int i=0; i<N and Ok; i++)
23         {
24             int k=Q.top();
25             Q.pop();
26             vector<int> v;
27             for (int j=0; j<k and Ok; j++)
28             {
29                 int x=Q.top();
30                 Q.pop();
31                 x--;
32                 Ok&=(x>=0);
33                 v.push_back(x);
34             }
35             for (int j=0; j<k and Ok; j++)
36             {
37                 Q.push(v[j]);
38             }
39         }
40         if (Ok) printf("Possible\n");
41         else printf("Not possible\n");
42     }
43     return 0;
44 }

```

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

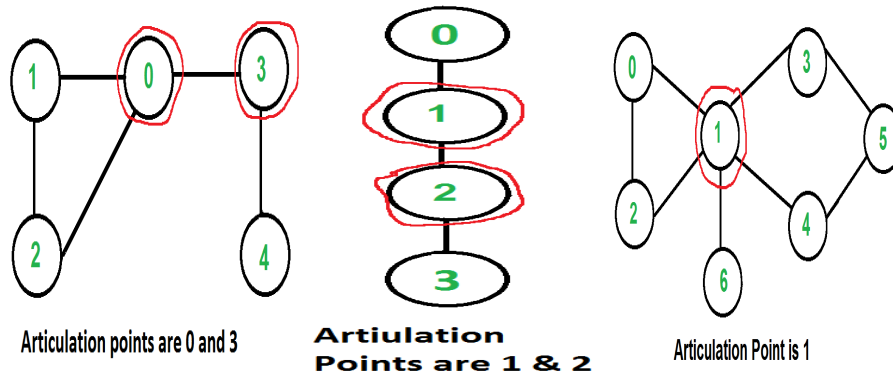


Figure 2.1: Articulation Point

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 .

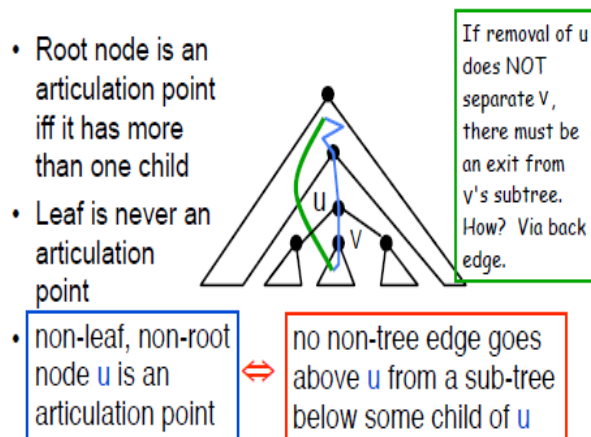


Figure 2.2: Articulation Point

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:

```
1 vector<int>Graph[10000];
2 bool visit[10000];
3 int arti[100000];
4 int discover[100000], Back[100000];
5 int predfn;
6 int source;
7 int child_of_root;
8 int cnt=0;
9 void reset()
10 {
11     memset(visit, false, sizeof(visit));
12     memset(arti, false, sizeof(arti));
13     predfn=child_of_root=0;
14 }
15 void articulation(int v)
16 {
17     visit[v]=true;
18     predfn++;
19     discover[v]=Back[v]=predfn;
20     for(int i=0; i<Graph[v].size(); i++)
21     {
22         int w=Graph[v][i];
23         if(!visit[w])
24         {
25             articulation(w);
26             Back[v]=min(Back[v], Back[w]);
27             if(Back[w]>=discover[v]&&v!=source)
28             {
29                 arti[v]=true;
30             }
31             else if(v==source)
32             {
33                 child_of_root++;
34                 if(child_of_root==2)
35                 {
36                     arti[v]=true;
37                 }
38             }
39         }
40         else
41         {
42             Back[v]=min(Back[v], discover[w]);
43         }
44     }
45 }
```

2.5.2 Find Bridge version 1:

```
1 vector<int>Graph[200];
2 int Back[205], Discover[205];
3 bool visit[205];
4 bool bridge[205][205];
5 int brcount;
6 void reset(int n)
7 {
```

```

8     for (int i=0;i<=n;i++)Graph[i].clear();
9     memset(visit,false,sizeof(visit));
10    memset(bridge,false,sizeof(false));
11    brcount=0;
12 }
13 void find_bridge(int u, int parent, int depth)
14 {
15     visit[u] = true;
16     Discover[u] = Back[u] = depth;
17
18     for (int i=0 ; i<Graph[u].size() ; i++)
19     {
20         int v = Graph[u][i];
21
22         if (visit[v] && v!=parent)
23         {
24             Back[u] = min(Back[u],Discover[v]);
25         }
26         if (!visit[v])
27         {
28             find_bridge(v, u, depth+1);
29             Back[u] = min(Back[u],Back[v]);
30             if (Back[v]>Discover[u])
31             {
32                 brcount++;
33                 bridge[u][v] = bridge[v][u] = true;
34             }
35         }
36     }
37 }
38 }

```

2.5.3 Find Bridge version 2:

```

1 void find_bridge(int node, int parent)
2 {
3     discovery_time[node] = bedge[node] = ++T;
4     int to, i, connected = adj[node].size();
5     for(i = 0; i < connected; i++)
6     {
7         to = adj[node][i];
8         if(to == parent) continue;
9         if(!discovery_time[to])
10        {
11            printf("%d %d\n", node, to);
12            find_bridge(to, node);
13            bedge[node] = min(bedge[node], bedge[to]);
14            if(bedge[to] > discovery_time[node]) printf("%d %d\n", to, node);
15        }
16        else if(discovery_time[node] > discovery_time[to])
17        {
18            printf("%d %d\n", node, to);
19            bedge[node] = min(bedge[node], discovery_time[to]);
20        }
21    }
22 }

```

2.6 Stable Marriage Problem

```
1  const int Max=400;
2  int position [Max] [Max];
3  int Husband [Max];
4  queue<int> womenlist [Max];
5  void stable_marage (int N)
6  {
7      mem (Husband, -1);
8      queue<int> freeman;
9      for (int i=1; i<=N; i++) freeman.push(i);
10     while (!freeman.empty())
11     {
12         int man=freeman.front();
13         int woman=womenlist[man].front();
14         womenlist[man].pop();
15         if (Husband[woman]==-1)
16         {
17             Husband[woman]=man;
18             freeman.pop();
19         }
20         else
21         {
22             int herhusband=Husband[woman];
23             if (position[woman][man]<position[woman][herhusband])
24             {
25                 Husband[woman]=man;
26                 freeman.pop();
27                 freeman.push(herhusband);
28             }
29         }
30     }
31 }
```

2.7 2 SAT

```
1  /*
2  A or B
3  A=>B
4  Algo Steps:
5  1. Construct Graph with A->B & B->A Edges
6  2. Find the Strongly Connected component of Graph & ranked them.
7  3. Check A & A is not in the same connected component. If in same group
   then, there is no solution
8  4. If there is solution exist, then solution will be higher rank between A &
   A node.
9  */
10 #include<bits/stdc++.h>
11 using namespace std;
12 #define pb push_back
13 #define mem(x,y) memset(x,y,sizeof(x))
14 typedef vector<int> vint;
15 vint Graph1[5*8000];
16 vint Graph2[5*8000];
17 bool visit[5*8000];
18 int color[5*8000];
19 vint ans;
20 struct info
21 {
22     int u,v;
```

```

23     info(int u,int v):u(u),v(v) {};
24 };
25 vector<info>Edges;
26 vint topsort;
27 void init(int N)
28 {
29     Edges.clear();
30     for(int i=0; i<=3*N; i++)
31     {
32         Graph1[i].clear();
33         Graph2[i].clear();
34     }
35 }
36 void construct_graph(int N)
37 {
38     int sz=Edges.size();
39     for(int i=0; i<sz; i++)
40     {
41         int u=Edges[i].u;
42         int v=Edges[i].v;
43         int a,b,acomp,bcomp;
44         if(u>0)
45         {
46             a=2*u;
47             acomp=2*u+1;
48         }
49         else
50         {
51             a=2*(-u)+1;
52             acomp=2*(-u);
53         }
54         if(v>0)
55         {
56             b=2*v;
57             bcomp=2*v+1;
58         }
59         else
60         {
61             b=2*(-v)+1;
62             bcomp=2*(-v);
63         }
64         Graph1[acomp].pb(b);
65         Graph1[bcomp].pb(a);
66         Graph2[b].pb(acomp);
67         Graph2[a].pb(bcomp);
68     }
69 }
70 void dfs(int u)
71 {
72     if(visit[u])return;
73     visit[u]=true;
74     int sz=Graph1[u].size();
75     for(int i=0; i<sz; i++)
76     {
77         int v=Graph1[u][i];
78         dfs(v);
79     }
80     topsort.pb(u);
81 }
82 void dfs2(int u,int Rank)
83 {

```



```

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[acom]) 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[acom]) 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);

```

```

145     SCC(N);
146     printf("Case %d: ",t);
147     if(is2satpossible(N))
148     {
149         cout<<"Yes"<<endl;
150         solution(N);
151         int sz=ans.size();
152         printf("%d",sz);
153         for(int i=0; i<sz; i++)printf(" %d",ans[i]);
154         printf("\n");
155     }
156     else cout<<"No"<<endl;
157 }
158 return 0;
159 }

```

2.8 Heavy Light Decomposition

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define pp pair<int,int>
4  #define pb push_back
5  const int Max=10000;
6  struct info
7  {
8      int v,cost;
9      info(int v=0,int cost=0):v(v),cost(cost){};
10 };
11 vector<pp>edges;
12 vector<info>Graph[Max+5];
13 int Tree[5*Max+5],BaseArray[Max+5],SubTreeSize[Max+5];
14 int ChainHead[Max+5],ChainNum[Max+5],PosInBaseArray[Max+5],ChainNo;
15 int Level[Max+5],Parent[Max+5],SparseTable[Max+5][16];
16 int ptr;
17 void init(int N)
18 {
19     for(int i=0; i<=N; i++)
20     {
21         Graph[i].clear(),ChainHead[i]=-1;
22         for(int j=0; j<=15; j++)SparseTable[i][j]=-1;
23     }
24     edges.clear();
25     ptr=ChainNo=0;
26 }
27 void buildSegmentTree(int l,int r,int indx)
28 {
29     if(l==r)
30     {
31         Tree[indx]=BaseArray[l];
32         return;
33     }
34     int mid=(l+r)>>1;
35     int lindx=indx<<1;
36     int rindx=lindx|1;
37     buildSegmentTree(l,mid,lindx);
38     buildSegmentTree(mid+1,r,rindx);
39     Tree[indx]=max(Tree[lindx],Tree[rindx]);
40 }
41 void updateSegmentTree(int l,int r,int indx,int update_indx,int value)
42 {

```

```

43     if ( l==r )
44     {
45         Tree [ indx ] = value ;
46         return ;
47     }
48     int mid = ( l + r ) >> 1 ;
49     int lindx = indx << 1 ;
50     int rindx = lindx | 1 ;
51     if ( update_indx <= mid ) updateSegmentTree ( l , mid , lindx , update_indx , value ) ;
52     else updateSegmentTree ( mid + 1 , r , rindx , update_indx , value ) ;
53     Tree [ indx ] = max ( Tree [ lindx ] , Tree [ rindx ] ) ;
54 }
55 int querySegmentTree ( int l , int r , int indx , int x , int y )
56 {
57     if ( l > y || r < x ) return 0 ;
58     if ( x <= l && y >= r ) return Tree [ indx ] ;
59     int mid = ( l + r ) >> 1 ;
60     int lindx = indx << 1 ;
61     int rindx = lindx | 1 ;
62     int c1 = 0 , c2 = 0 ;
63     if ( x <= mid ) c1 = querySegmentTree ( l , mid , lindx , x , y ) ;
64     if ( y > mid ) c2 = querySegmentTree ( mid + 1 , r , rindx , x , y ) ;
65     return max ( c1 , c2 ) ;
66 }
67 void dfs ( int from , int u , int depth )
68 {
69     Level [ u ] = depth ;
70     Parent [ u ] = from ;
71     SubTreeSize [ u ] = 1 ;
72     int sz = Graph [ u ] . size ( ) ;
73     for ( int i = 0 ; i < sz ; i ++ )
74     {
75         int v = Graph [ u ] [ i ] . v ;
76         if ( v == from ) continue ;
77         dfs ( u , v , depth + 1 ) ;
78         SubTreeSize [ u ] += SubTreeSize [ v ] ;
79     }
80 }
81 void sparseTable ( int N )
82 {
83     for ( int i = 0 ; i <= N ; i ++ ) SparseTable [ i ] [ 0 ] = Parent [ i ] ;
84     for ( int j = 1 ; ( 1 << j ) <= N ; j ++ )
85     {
86         for ( int i = 0 ; i <= N ; i ++ )
87         {
88             if ( SparseTable [ i ] [ j - 1 ] != - 1 )
89             {
90                 int a = SparseTable [ i ] [ j - 1 ] ;
91                 SparseTable [ i ] [ j ] = SparseTable [ a ] [ j - 1 ] ;
92             }
93         }
94     }
95 }
96 int LCA ( int p , int q )
97 {
98     if ( Level [ p ] < Level [ q ] ) swap ( p , q ) ;
99     int Log = log2 ( Level [ p ] ) + 1 ;
100    for ( int i = Log ; i >= 0 ; i -- )
101    {
102        if ( ( Level [ p ] - ( 1 << i ) ) >= Level [ q ] ) p = SparseTable [ p ] [ i ] ;
103    }

```

```

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
120  * size.
121  * The following if condition is failed for leaf nodes.
122  * When the if condition passes, we expand the chain to special child.
123  * In the second for loop we recursively call the function on all normal
124  * nodes.
125  * chainNo++ ensures that we are creating a new chain for each normal child.
126  */
127 void heavyLightDecompositon(int from,int curNode,int cost)
128 {
129     if(ChainHead[ChainNo]==-1)ChainHead[ChainNo]=curNode; /// Assign chain
130     head
131     ChainNum[curNode]=ChainNo;
132     PosInBaseArray[curNode]=ptr; /// Position of this node in baseArray
133     which we will use in Segtree
134     BaseArray[ptr++]=cost;
135     int sc=-1,nextCost;
136     int sz=Graph[curNode].size();
137     for(int i=0; i<sz; i++) /// Loop to find special child
138     {
139         int v=Graph[curNode][i].v;
140         if(v==from) continue;
141         if(sc==-1||SubTreeSize[sc]<SubTreeSize[v])
142         {
143             sc=v;
144             nextCost=Graph[curNode][i].cost;
145         }
146     }
147     if(sc!=-1)heavyLightDecompositon(curNode,sc,nextCost); /// Expand the
148     chain
149     for(int i=0; i<sz; i++)
150     {
151         int v=Graph[curNode][i].v;
152         int cost=Graph[curNode][i].cost;
153         if(v==from||sc==v) continue;
154         ChainNo++;
155         heavyLightDecompositon(curNode,v,cost);
156     }
157 }
158 void updateTree(int ith,int val)
159 {
160     pp a=edges[ith];
161     int u=a.first,v=a.second;
162     int indx=PosInBaseArray[u];
163     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

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define pb push_back
4 #define MN 1000
5 typedef vector< vector<int> > vint2D;
6 const int inf=(1<<29);
7 vint2D graph;
8 int Cost[MN][MN];
9 int parent[MN+5];
10 int flow;
11 void init(int N)
12 {
13     graph=vint2D(N);
14     memset(Cost,0,sizeof(Cost));
15 }
16 void AddEdge(int u,int v,int cost)
17 {
18     graph[u].pb(v);
19     graph[v].pb(u);
20     Cost[u][v]+=cost;
21     Cost[v][u]+=cost;
22 }
23 bool augmenting_path(int source,int sink)
24 {
25     memset(parent,-1,sizeof(parent));
26     queue<int>Q;
27     Q.push(source);
28     while(!Q.empty())
29     {
30         int u=Q.front();
31         Q.pop();
32         int sz=graph[u].size();
33         for(int i=0; i<sz; i++)
34         {
35             int v=graph[u][i];
36             if(parent[v]==-1 and Cost[u][v]>0)
37             {
38                 parent[v]=u;
39                 Q.push(v);
40                 if(v==sink) return true;
41             }
42         }
43     }
```

```

43     }
44     return false;
45 }
46 void path(int v,int source)
47 {
48     int u=parent[v];
49     flow=min(flow, Cost[u][v]);
50     if(source!=u) path(u, source);
51     Cost[u][v]-=flow;
52     Cost[v][u]+=flow;
53     return;
54 }
55 int max_flow(int source,int sink)
56 {
57     int ret=0;
58     while(augmenting_path(source,sink))
59     {
60         flow=inf;
61         path(sink,source);
62         ret+=flow;
63     }
64     return ret;
65 }
66 int main()
67 {
68     int test;
69     scanf("%d",&test);
70     while(test--)
71     {
72         int P,S,C,M;
73         scanf("%d %d %d %d",&P,&S,&C,&M);
74         init(P+S+5);
75         int superSource=0,SuperSikn=P+S+1;
76         for(int i=1; i<=P; i++)AddEdge(superSource,i,1);
77         for(int i=1; i<=S; i++)AddEdge(P+1,SuperSikn,C);
78         for(int i=0; i<M; i++)
79         {
80             int x,y;
81             scanf("%d %d",&x,&y);
82             AddEdge(x,P+y,(1<<30));
83         }
84         printf("%d\n",max_flow(superSource,SuperSikn));
85     }
86     return 0;
87 }

```


3.2 Maximum Bipartite Matching

```
1 bool khun(int u)
2 {
3     int sz=graph[u].size();
4     for(int i=0; i<sz; i++)
5     {
6         int v=graph[u][i];
7         if(seen[v]) continue;
8         seen[v]=true;
9         if(Right[v]==-1||khun(Right[v]))
10        {
11            Right[v]=u;
12            Left[u]=v;
13            return true;
14        }
15    }
16    return false;
17 }
18 int bipartite_matching(int m)
19 {
20     mem(Left,-1);
21     mem(Right,-1);
22     int cnt=0;
23     for(int i=0; i<m; i++)
24     {
25         mem(seen,false);
26         if(khun(i)) cnt++;
27     }
28     return cnt;
29 }
```

Chapter 4

Dynamic programming

4.1 Longest Increasing sub sequence

4.1.1 LIS1

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 const int inf = 2000000000; // a large value as infinity
4
5 int n; // the number of items in the sequence
6 int Sequence[32]; // the sequence of integers
7 int L[32]; // L[] as described in the algorithm
8 int I[32]; // I[] as described in the algorithm
9 vector<int> LIS;
10 void takeInput()
11 {
12     scanf("%d", &n); // how many numbers in the sequence ?
13     for( int i = 0; i < n; i++ ) // take the sequence
14         scanf("%d", &Sequence[i]);
15 }
16
17 int LisNlogK() // which runs the NlogK LIS algorithm
18 {
19     int i; // auxiliary variable for iteration
20
21     I[0] = -inf; // I[0] = -infinite
22     for( i = 1; i <= n; i++ ) // observe that i <= n are given
23         I[i] = inf; // I[1 to n] = infinite
24
25     int LisLength = 0; // keeps the maximum position where a data is
26                         // inserted
27
28     for( i = 0; i < n; i++ ) // iterate left to right
29     {
30         int low, high, mid; // variables to perform binary search
31         low = 0; // minimum position where we to put data in I[]
32         high = LisLength; // the maximum position
33
34         while( low <= high ) // binary search to find the correct position
35         {
36             mid = ( low + high ) / 2;
37             if( I[mid] < Sequence[i] )
38                 low = mid + 1;
39             else
40                 high = mid - 1;
```

```

40     }
41     // observe the binary search carefully, when the binary search ends
42     // low > high and we put our item in I[low]
43     I[low] = Sequence[i];
44     L[i]=low;
45     if( LisLength < low ) // LisLength contains the maximum position
46         LisLength = low;
47 }
48
49
50 i=0;
51 for( int j=0; j<n; j++)
52 {
53     if(L[j]>L[i])
54     {
55         i=j;
56     }
57 }
58 LIS.push_back(Sequence[i]);
59 for( int j=i-1; j>=0; j--)
60 {
61     if(Sequence[j]<Sequence[i]&&L[j]==L[i]-1)
62     {
63         i=j;
64         LIS.push_back(Sequence[j]);
65     }
66 }
67 reverse(LIS.begin(),LIS.end());
68 return LisLength; // return the result
69 }
70 int main()
71 {
72     takeInput();
73     int result = LisNlogK();
74     printf("The LIS length is %d\n", result);
75     for(int i=0; i<LIS.size(); i++)printf("%d ",LIS[i]);
76     return 0;
77 }

```

4.1.2 LIS(Stanford)

```

1 // Given a list of numbers of length n, this routine extracts a
2 // longest increasing subsequence.
3 //
4 // Running time: O(n log n)
5 //
6 // INPUT: a vector of integers
7 // OUTPUT: a vector containing the longest increasing subsequence
8
9 #include <iostream>
10 #include <vector>
11 #include <algorithm>
12 using namespace std;
13 typedef vector<int> VI;
14 typedef pair<int, int> PII;
15 typedef vector<PII> VPII;
16
17 #define STRICTLY_INCREASNG
18
19 VI LongestIncreasingSubsequence(VI v)
20 {

```

```

21     VPII best;
22     VI dad(v.size(), -1);
23
24     for (int i = 0; i < v.size(); i++)
25     {
26 #ifdef STRICTLY_INCREASNG
27         PII item = make_pair(v[i], 0);
28         VPII::iterator it = lower_bound(best.begin(), best.end(), item);
29         item.second = i;
30 #else
31         PII item = make_pair(v[i], i);
32         VPII::iterator it = upper_bound(best.begin(), best.end(), item);
33 #endif
34         if (it == best.end())
35         {
36             dad[i] = (best.size() == 0 ? -1 : best.back().second);
37             best.push_back(item);
38         }
39         else
40         {
41             dad[i] = dad[it->second];
42             *it = item;
43         }
44     }
45
46     VI ret;
47     for (int i = best.back().second; i >= 0; i = dad[i])
48         ret.push_back(v[i]);
49     reverse(ret.begin(), ret.end());
50     return ret;
51 }

```

4.2 Edit Distance

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int dp[88][88];
4 int N,M,step;
5 char S1[88],S2[88];
6 int solve(int i,int j)
7 {
8     if(i==N and j==M) return 0;
9     if(i==N) return M-j;
10    if(j==M) return N-i;
11    int &ret=dp[i][j];
12    if(ret!=-1) return ret;
13    ret=(1<<28);
14    if(S1[i]==S2[j]) ret=solve(i+1,j+1);
15    else
16    {
17        ret=min(ret, solve(i, j+1)+1);
18        ret=min(ret, solve(i+1, j)+1);
19        ret=min(ret, solve(i+1, j+1)+1);
20    }
21    return ret;
22 }
23 void pathPrint(int i,int j,int del,int ins,int st)
24 {
25     if(i==N&&j==M) return ;
26     if(i==N)

```

```

27 {
28     for(int k=j; k<M; k++,i++)
29     {
30         printf("%d Insert %d,%c\n",st++,i-del+1+ins,S2[k]);
31     }
32     return ;
33 }
34 if(j==M)
35 {
36     for(; i<N; i++)
37     {
38         printf("%d Delete %d\n",st++,i-del+1+ins);
39         del++;
40     }
41     return ;
42 }
43 int ret = solve(i,j);
44 int tmp;
45 if(S1[i]==S2[j])
46 {
47     tmp=solve(i+1,j+1);
48     if(ret==tmp)
49     {
50         pathPrint(i+1,j+1,del,ins,st);
51         return ;
52     }
53 }
54 tmp=solve(i,j+1)+1;
55 if(tmp==ret)
56 {
57     printf("%d Insert %d,%c\n",st,i-del+1+ins,S2[j]);
58     pathPrint(i,j+1,del,ins+1,st+1);
59     return ;
60 }
61 tmp=solve(i+1,j)+1;
62 if(tmp==ret)
63 {
64     printf("%d Delete %d\n",st,i-del+1+ins);
65     pathPrint(i+1,j,del+1,ins,st+1);
66     return ;
67 }
68 tmp=solve(i+1,j+1)+1;
69 if(tmp==ret)
70 {
71     printf("%d Replace %d,%c\n",st,i-del+1+ins,S2[j]);
72     pathPrint(i+1,j+1,del,ins,st+1);
73     return ;
74 }
75 return ;
76 }
77 int main()
78 {
79     bool New=false;
80     while(gets(S1))
81     {
82         gets(S2);
83         if(New)printf("\n");
84         New=true;
85         N=strlen(S1);
86         M=strlen(S2);
87         memset(dp,-1,sizeof(dp));

```

```

88     step=solve(0,0);
89     printf("%d\n",step);
90     pathPrint(0,0,0,0,1);
91 }
92 return 0;
93 }

```

4.3 Distinct Sub Sequence

```

1 LLD solve()
2 {
3     dp[0]=1;
4     mem( Last_occurance ,0);
5     for( int i=1;s[i];i++)
6     {
7         dp[i]=2*dp[i-1];
8         int x=s[i]-'A';
9         if( Last_occurance[x]) dp[i]=dp[ Last_occurance[x]-1];
10        Last_occurance[x]=i;
11        dp[i]=dp[i];
12    }
13    return dp[ strlen(s+1)];
14 }

```

4.4 Knuth Optimization

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int dp[505][505];
4 int positionK[505][505];
5 int Csum[505];
6 int main()
7 {
8     int N;
9     while( scanf("%d",&N)==1)
10    {
11        for( int i=1; i<=N; i++)
12        {
13            scanf("%d",&Csum[i]);
14            Csum[i]+=Csum[i-1];
15        }
16        for( int L=1; L<=N; L++)
17        {
18            for( int i=1; i<=N; i++)
19            {
20                int j=L+i-1;
21                if(j>N) continue;
22                if(L==1)
23                {
24                    dp[i][j]=0;
25                    positionK[i][j]=i;
26                }
27                else
28                {
29                    dp[i][j]=(1<<28);
30                    for( int k=positionK[i][j-1]; k<=positionK[i+1][j]; k++)
31                    {
32                        int cost=dp[i][k-1];
33                        cost+=dp[k+1][j];
34                        cost+=(Csum[j]-Csum[k]);

```

```

35         cost+=(Csum[k-1]-Csum[i-1]);
36         if(cost<dp[i][j])
37         {
38             dp[i][j]=cost;
39             positionK[i][j]=k;
40         }
41     }
42 }
43 }
44 }
45 printf("%d\n",dp[1][N]);
46 }
47 return 0;
48 }

```

4.5 Max Sum

4.5.1 2D Max Sum 1

Complexity $O(n^4)$

```

1 precalculation:
2 for(int i=1;i<=N;i++)
3 {
4     for(int j=1;j<=N;j++)
5     {
6         Mat[i][j]=Mat[i-1][j]+Mat[i][j-1]-Mat[i-1][j-1]+A[i][j];
7     }
8 }
9 Query:
10 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

```

1 int arr[SZ][SZ];
2 int input(void);
3 int max_sum(int n);
4 int main()
5 {
6     int test, n;
7     scanf("%d", &test);
8     while(test--)
9     {
10         n = input();
11         printf("%d\n", max_sum(n));
12         if(test)
13             printf("\n");
14     }
15     return 0;
16 }
17
18 int input(void)
19 {
20     int i, len, r = 1, n;
21     char str[26];
22     scanf("%s", &str);
23     len = strlen(str);
24     for(i = 1; i <= len; i++)
25     {

```

```

26     n = str[i - 1] - '0';
27     if(n)
28         arr[r][i] = n;
29     else
30         arr[r][i] = -700;
31 }
32 r++;
33 while(r <= len)
34 {
35     scanf("%s", &str);
36     for(i = 1; i <= len; i++)
37     {
38         n = str[i - 1] - '0';
39         if(n)
40             arr[r][i] = n;
41         else
42             arr[r][i] = -700;
43     }
44     r++;
45 }
46 return len;
47 }
48
49 int max_sum(int n)
50 {
51     int i, j, c1, c2, r, sum = 0, m = 0;
52     for(i = 0; i <= n; i++)
53         arr[i][0] = arr[0][i] = 0;
54     for(i = 1; i <= n; i++)
55         for(j = 1; j <= n; j++)
56             arr[i][j] += arr[i][j - 1];
57     for(c1 = 1; c1 <= n; c1++)
58         for(c2 = c1; c2 <= n; c2++)
59         {
60             sum = 0;
61             for(r = 1; r <= n; r++)
62             {
63                 sum += arr[r][c2] - arr[r][c1 - 1];
64                 if(sum < 0)
65                     sum = 0;
66                 else if(sum > m)
67                     m = sum;
68             }
69         }
70     return m;
71 }

```

4.5.3 3D Max sum

Complexity $O(n^4)$

Cumulative Sum formula: $S(x,y,z) = \text{value}(x,y,z) + S(x-1,y,z) + S(x,y-1,z) + S(x,y,z-1) - S(x-1,y-1,z) - S(x,y-1,z-1) - S(x-1,y,z-1) + S(x-1,y-1,z-1)$

Query Formula: $S([x1,y1,z1] \text{ to } [x2,y2,z2]) = S(x2,y2,z2) + S(x1,y1,z2) + S(x1,y2,z1) + S(x1,y1,z1) - S(x2,y2,z1) - S(x2,y1,z2) - S(x1,y2,z2) - S(x1,y1,z1)$

```

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

```



```

2 {
3
4     for (int j = n; j <= N; j++)
5     {
6         for (int k = n; k <= N; k++)
7         {
8             int x2=i ,y2=j ,z2=k;
9             int x1=i-n ,y1=j-n ,z1=k-n;
10            int temp=[Query formul]
11                ans=max( ans , temp ) ;
12        }
13    }
14 }

```

4.6 Minimum Vertex Cover

```

1 #define MAXN 100002
2 int dp[MAXN][5];
3 int par[MAXN];
4 vector<int>edges[MAXN];
5 int f(int u,int isGuard)
6 {
7     if(edges[u].size()==0)return 0;
8     if(dp[u][isGuard]!=-1) return dp[u][isGuard];
9     int sum=0;
10    for(int i=0; i<(int)edges[u].size(); i++)
11    {
12        int v=edges[u][i];
13        if(v!=par[u])
14        {
15            par[v]=u;
16            if(isGuard==0) sum+=f(v,1);
17            else sum+=min(f(v,1),f(v,0));
18        }
19    }
20    return dp[u][isGuard]=sum+isGuard;
21 }
22 int main()
23 {
24     memset(dp,-1,sizeof(dp));
25     int n;
26     scanf("%d",&n);
27     for(int i=1; i<n; i++)
28     {
29         int u,v;
30         scanf("%d%d",&u,&v);
31         edges[u].push_back(v);
32         edges[v].push_back(u);
33     }
34     int ans=0;
35     ans=min(f(1,1),f(1,0));
36     printf("%d\n",ans);
37     return 0;
38 }

```

4.7 Nth Permutation Of a String

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 #define mem(x,y) memset(x,y,sizeof(x))
4 typedef long long LLD;
5 LLD FAC[22];
6 int Freq[27];
7 string S;
8 void FACTORIAL()
9 {
10     FAC[0]=1;
11     for(int i=1; i<=20; i++)FAC[i]=FAC[i-1]*i;
12 }
13 LLD Permutation(int n)
14 {
15     LLD per=FAC[n];
16     for(int i=0; i<26; i++)per/=FAC[Freq[i]];
17     return per;
18 }
19 string NthPermutation(int nth)
20 {
21     string ans="";
22     int len=S.size();
23     while(len)
24     {
25         LLD upto=0;
26         for(int i=0; i<26; i++)
27         {
28             if(Freq[i])
29             {
30                 Freq[i]-=1;
31                 LLD now=Permutation(len-1);
32                 if(upto+now>=nth)
33                 {
34                     nth-=upto;
35                     ans+=('a'+i);
36                     len--;
37                     break;
38                 }
39                 else
40                 {
41                     upto+=now;
42                     Freq[i]+=1;
43                 }
44             }
45         }
46     }
47     return ans;
48 }
49 int main()
50 {
51     FACTORIAL();
52     int test;
53     scanf("%d",&test);
54     for(int t=1; t<=test; t++)
55     {
56         mem(Freq,0);
57         char inp[100];
58         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

```
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

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 through 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

```

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     for(int i=0; str[i]; i++)
69     {
70         int id=getID(str[i]);
71         while(p->next[id]==NULL&& p!=root)p=p->fail;
72         if(p->next[id]!=NULL)p=p->next[id];
73         Trie *tp=p;
74         while(tp!=root)
75         {
76             if(tp->stringMap>0)ans[tp->stringMap]++;
77             tp=tp->fail;
78         }
79     }
80 }
81 void Delete(Trie *u)
82 {
83     if(u==NULL)return;
84     for(int i=0; i<Max; i++)Delete(u->next[i]);
85     delete u;
86 }
87
88 int main()
89 {
90     int test;
91     scanf("%d",&test);
92     for(int t=1; t<=test; t++)
93     {

```

```

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

```

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     }
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 void AhoCorasick(const Trie &A, const char *str)
82 {
83     int p=root;
84     for(int i=0; str[i]; i++)
85     {
86         int id=getID(str[i]);
87         while(A.T[p].Next[id]==NULL&&p!=root)p=A.T[p].fail;
88         if(p!=NULL&&A.T[p].Next[id]!=NULL)p=A.T[p].Next[id];
89         int tp=p;
90         while(tp!=root)
91         {
92             if(A.T[tp].stringMap>0)ans[A.T[tp].stringMap]++;

```



```

94         tp=A.T[tp].fail;
95     }
96 }
97 }
98 Trie A;
99 int main()
100 {
101     #ifdef _ANICK_
102         freopen("input.txt","r",stdin);
103     #endif // _ANICK_
104     int test;
105     scanf("%d",&test);
106     for(int t=1; t<=test; t++)
107     {
108         Map.clear();
109         v.clear();
110         memset(ans,0,sizeof(ans));
111         A.clear();
112         int N;
113         scanf("%d",&N);
114         scanf("%s",text);
115         int cnt=1;
116         for(int i=0; i<N; i++)
117         {
118             scanf("%s",inp);
119             if(Map.find(inp)==Map.end())Map[inp]=cnt++;
120             A.Insert(inp,Map[inp]);
121             v.push_back(Map[inp]);
122         }
123         A.ComputeFailure();
124         AhoCorasick(A,text);
125         printf("Case %d:\n",t);
126         for(int i=0; i<N; i++)
127         {
128             printf("%d\n",ans[v[i]]);
129         }
130     }
131     return 0;
132 }

```

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     int P[t.size()],c=0,r=0,i,i_mirror,n=t.size()-1;
18     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 Pick's Theorem:

$$Area = \frac{B}{2} + I - 1$$

6.2 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 }
```

6.3 Floyd's Cycle Finding algorithm

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 #define pp pair<int,int>
4 int Z,L,M,I;
5 int f(int L)
6 {
7     return (Z*L+I)%M;
8 }
9 pp CycleFinding()
10 {
11     ///L here initial seed
12     int hare,tortoise,lambda,meu;
13     bool cyclefind=false;
14     hare=tortoise=L;
15     while(!cyclefind)
16     {
17         tortoise=f(tortoise);
18         hare=f(hare);
19         hare=f(hare);
20         if(hare==tortoise) cyclefind=true;
```

```

21     }
22     hare=L;
23     meu=0;
24     while(hare!=tortoise)
25     {
26         meu++;
27         hare=f(hare);
28         tortoise=f(tortoise);
29     }
30     int i=0;
31     hare=L;
32     while(i<=meu)
33     {
34         i++;
35         hare=f(hare);
36     }
37     tortoise=f(hare);
38     lambda=1;
39     while(hare!=tortoise)
40     {
41         tortoise=f(tortoise);
42         lambda++;
43     }
44     return {meu,lambda}; //meu is starting index and lambda is cycle length
45 }
46 int main()
47 {
48     int t=1;
49     while(scanf("%d %d %d %d",&Z,&I,&M,&L) and (Z or I or M or L))
50     {
51         pp a=CycleFinding();
52         cout<<"Cycle starts from index "<<a.first<<"\nCycle length is "<<a.
second<<endl;
53     }
54     return 0;
55 }

```

6.4 Angle between clock hand

```

1 double angle(int h, int m)
2 {
3     double hAngle = 0.5D * (h * 60 + m);
4     double mAngle = 6 * m;
5     double angle = abs(hAngle - mAngle);
6     angle = min(angle, 360 - angle);
7     return angle;
8 }

```

6.5 Big Integer

```

1 struct Bigint
2 {
3     // representations and structures
4     string a; // to store the digits
5     int sign; // sign = -1 for negative numbers, sign = 1 otherwise
6
7     // constructors
8     Bigint() {} // default constructor
9     Bigint( string b )
10    {

```

```

11     (*this) = b;    // constructor for string
12 }
13 Bigint(long long num)
14 {
15     if(num<0)sign=-1;
16     else sign=1;
17     if(num==0)a.push_back('0');
18     while(num)
19     {
20         a.push_back(num%10+'0');
21         num/=10;
22     }
23 }
24 // some helpful methods
25 int size()    // returns number of digits
26 {
27     return a.size();
28 }
29 Bigint inverseSign()    // changes the sign
30 {
31     sign *= -1;
32     return (*this);
33 }
34 Bigint normalize( int newSign )    // removes leading 0, fixes sign
35 {
36     for( int i = a.size() - 1; i > 0 && a[i] == '0'; i-- )
37         a.erase(a.begin() + i);
38     sign = ( a.size() == 1 && a[0] == '0' ) ? 1 : newSign;
39     return (*this);
40 }
41
42 // assignment operator
43 void operator = ( string b )    // assigns a string to Bigint
44 {
45     a = b[0] == '-' ? b.substr(1) : b;
46     reverse( a.begin(), a.end() );
47     this->normalize( b[0] == '-' ? -1 : 1 );
48 }
49
50 // conditional operators
51 bool operator < ( const Bigint &b ) const    // less than operator
52 {
53     if( sign != b.sign ) return sign < b.sign;
54     if( a.size() != b.a.size() )
55         return sign == 1 ? a.size() < b.a.size() : a.size() > b.a.size();
56 ;
57     for( int i = a.size() - 1; i >= 0; i-- ) if( a[i] != b.a[i] )
58         return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
59     return false;
60 }
61 bool operator == ( const Bigint &b ) const    // operator for equality
62 {
63     return a == b.a && sign == b.sign;
64 }
65 // mathematical operators
66 Bigint operator + ( Bigint b )    // addition operator overloading
67 {
68     if( sign != b.sign ) return (*this) - b.inverseSign();
69     Bigint c;
70     for(int i = 0, carry = 0; i<a.size() || i<b.size() || carry; i++)

```

```

71         carry+=(i<a.size() ? a[i]-48 : 0)+(i<b.a.size() ? b.a[i]-48 : 0)
72     ;
73     c.a += (carry % 10 + 48);
74     carry /= 10;
75     }
76     return c.normalize(sign);
77 }
78 BigInt operator - ( BigInt b )    // subtraction operator overloading
79 {
80     if( sign != b.sign ) return (*this) + b.inverseSign();
81     int s = sign;
82     sign = b.sign = 1;
83     if( (*this) < b ) return ((b - (*this)).inverseSign()).normalize(-s)
84 ;
85     BigInt c;
86     for( int i = 0, borrow = 0; i < a.size(); i++ )
87     {
88         borrow = a[i] - borrow - (i < b.size() ? b.a[i] : 48);
89         c.a += borrow >= 0 ? borrow + 48 : borrow + 58;
90         borrow = borrow >= 0 ? 0 : 1;
91     }
92     return c.normalize(s);
93 }
94 BigInt operator * (BigInt b)    // multiplication operator overloading
95 {
96     BigInt c("0");
97     for( int i = 0, k = a[i] - 48; i < a.size(); i++, k = a[i] - 48 )
98     {
99         while(k-->0) c = c + b; // ith digit is k, so, we add k times
100        b.a.insert(b.a.begin(), '0'); // multiplied by 10
101    }
102    return c.normalize(sign * b.sign);
103 }
104 BigInt operator / ( BigInt b )    // division operator overloading
105 {
106     if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
107     BigInt c("0"), d;
108     for( int j = 0; j < a.size(); j++ ) d.a += "0";
109     int dSign = sign * b.sign;
110     b.sign = 1;
111     for( int i = a.size() - 1; i >= 0; i-- )
112     {
113         c.a.insert( c.a.begin(), '0' );
114         c = c + a.substr( i, 1 );
115         while( !( c < b ) ) c = c - b, d.a[i]++;
116     }
117     return d.normalize(dSign);
118 }
119 BigInt operator % ( BigInt b )    // modulo operator overloading
120 {
121     if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
122     BigInt c("0");
123     b.sign = 1;
124     for( int i = a.size() - 1; i >= 0; i-- )
125     {
126         c.a.insert( c.a.begin(), '0' );
127         c = c + a.substr( i, 1 );
128         while( !( c < b ) ) c = c - b;
129     }
130     return c.normalize(sign);
131 }

```

```
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

```
1  /**
2   * There is N stone of pile , every player in each move can remove Move[i]
3   * stone . If Any move there is no Stone remain
4   * these player will be lost .
5   */
6  #include <bits/stdc++.h>
7  using namespace std;
8  int Move[100], k, n;
9  int dp[1000000+5];
10 bool is_valid(int x)
11 {
12     return x >= 0;
13 }
14 bool solve(int sum)
15 {
16     if(sum == 0) return false;
17     if(dp[sum] != -1) return dp[sum];
18     for(int i = 1; i <= k; i++)
19     {
20         if(is_valid(sum - Move[i]) and !solve(sum - Move[i])) return dp[sum] = true;
21     }
22     return dp[sum] = false;
23 }
24 int main()
25 {
26     while(scanf("%d", &n) == 1)
27     {
28         scanf("%d", &k);
29         for(int i = 1; i <= k; i++)
30         {
31             scanf("%d", &Move[i]);
32         }
33         memset(dp, -1, sizeof(dp));
34         if(solve(n)) printf("A\n");
35         else printf("B\n");
36     }
37     return 0;
38 }
```


7.2 Grundy Number

```
1 const int MX = 1000005 ;
2 int grunday[MX] , rev[6] = { 1 , 4 , 27 , 256 , 3125 , 46656 } ; // in a single
   move n*n can be removed
3 int Cal(int x)
4 {
5     if( x == 0 ) return 0 ; // base case , grunday is always zero
6     int &ret = grunday[x];
7     if( ret != -1 ) return ret;
8     set<int> s ;
9     int i ;
10    for ( i = 0 ; i < 6 ; i++ )
11    {
12        if( x - rev[i] >= 0 )
13            s.insert( Cal(x-(rev[i])) );
14    }
15    int ans = 0 ;
16    while(s.find(ans) != s.end()) ans++;
17    return ret = ans ;
18 }
19 int main()
20 {
21     // ios_base::sync_with_stdio(0); cin.tie(0);
22     memo(grunday,-1);
23     int cs , t ;
24     scanf("%d",&t);
25     for ( cs = 1 ; cs <= t ; cs++ )
26     {
27         int n , i , ans = 0 , x ;
28         scanf("%d",&n);
29         for ( i = 0 ; i < n ; i++ )
30         {
31             scanf("%d",&x);
32             ans ^= Cal(x);
33         }
34         if( ans ) puts("Little Chef"); // first mover win
35         else puts("Head Chef");
36     }
37
38     return 0;
39 }
```

Chapter 8

Number Theory

8.1 Greatest common divisor (GCD)

```
1 int gcd(int a, int b)
2 {
3     return b == 0 ? a : gcd(b, a % b);
4 }
```

8.2 Least common multiple (LCM)

```
1 int lcm(int a, int b)
2 {
3     return (a / gcd(a, b)) * b;
4 }
```

8.3 Big Mod

8.3.1 Iterative version

```
1 LLD bigmod(LLD B, LLD P, LLD M)
2 {
3     LLD R=1;
4     while(P>0)
5     {
6         if(P%2==1)
7         {
8             R=(R*B)%M;
9         }
10        P/=2;
11        B=(B*B)%M;
12    }
13    return R;
14 }
```

8.3.2 Recursive version

```
1 int big_mod(int n, int p, int m)
2 {
3     if(!n) return 1;
4     int x=(n, p>>1, m);
5     x=(x*x)%m;
6     if(p&1) x=(x*n)%m;
7     return x;
8 }
```

8.4 Sieve of Eratosthenes

8.4.1 Sieve

```
1  const int Max=1000000;
2  bool prime_check [Max+5];
3  vint prime;
4  void sieve ()
5  {
6      int i , j ;
7      int sq=sqrt (Max) ;
8      for ( i=2; i*i<=Max; i++)
9      {
10         if (!prime_check [ i ])
11         {
12             prime.pb ( i ) ;
13             for ( j=i*i ; j<=Max; j+=i )
14             {
15                 prime_check [ j ]=true ;
16             }
17         }
18     }
19     for ( i=sq+1; i<=Max; i++)
20     {
21         if (!prime_check [ i ]) prime.pb ( i ) ;
22     }
23 }
```

8.4.2 Bitwise Sieve

```
1  #include<stdio.h>
2  #include<math.h>
3  #define check(N,pos) (N&(1<<(pos)))
4  #define Set(N,pos) N|(1<<(pos))
5  const int M=100000;
6  int status [M/32+2]={0};
7  void sieve ()
8  {
9      int sqN=(int) sqrt (M) ;
10     int i , j , cnt=1;
11     puts ("2");
12     for ( i=3; i<=sqN; i+=2)
13     {
14         if (check (status [ i >> 5 ], i & 31) == 0)
15         {
16             for ( j=i*i ; j<=M; j+=2*i )
17             {
18                 status [ j >> 5 ] = Set ( status [ j >> 5 ], j & 31 );
19             }
20         }
21     }
22     for ( i=3; i<=M; i+=2)
23     {
24         if (check (status [ i >> 5 ], i & 31) == 0)
25         {
26             printf ("%d\n" , i );
27         }
28     }
29 }
```

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

```
1 int phi[10000];
2 const int M=1000;
3 void Generate_phi()
4 {
5     int i, j;
6     phi[1]=1;
7     for (i=2; i<M; i++)
8     {
9         if (!phi[i])
10        {
11            phi[i]=i-1;
12            for (j=i+i; j<M; j+=i)
13            {
14                if (!phi[j]) phi[j]=j;
15                phi[j]=phi[j]/i*(i-1);
16            }
17        }
18    }
19 }
```

8.5.2 Euler phi function

```
1 int phi (int n)
2 {
3     int ret = n;
4     for (int i = 2; i * i <= n; i++)
5     {
6         if (n % i == 0)
7         {
8             while (n % i == 0)
9             {
10                n /= i;
11            }
12            ret -= ret / i;
13        }
14    }
15    // this case will happen if n is a prime number
16    // in that case we won't find any prime that divides n
17    // that's less or equal to sqrt(n)
18    if (n > 1) ret -= ret / n;
19    return ret;
20 }
```

8.6 Number of Divisors

If a number

$$N = P_1^{k_1} * P_2^{k_2} * P_3^{k_3} * P_n^{k_n}$$

Number of divisor:

$$f(n) = (k_1 + 1) * (k_2 + 1) * (k_3 + 1) * (k_n + 1)$$

8.6.1 NOD sieve version

```
1 int nod[100000+5];
2 void Generate()
3 {
4     nod[1]=1;
5     for(int i=2; i<=100000; i++)
6     {
7         if(!nod[i])    //here checking i is prime or not ???
8         {
9             nod[i]=2;
10            for(int j=i+i; j<=100000; j+=i)
11            {
12                if(!nod[j]) nod[j]=1;
13                int n=j, cnt=0;
14                while(!(n%i))
15                {
16                    cnt++;
17                    n/=i;
18                }
19                nod[j]*=(cnt+1);
20            }
21        }
22    }
23 }
```

8.6.2 NOD function

```
1 int find_divisor(int n){
2     int i, ans=1, count=1;
3     while(n%2==0)
4     {
5         n/=2;
6         count++;
7     }
8     ans*=count;
9     for(i=3; i*i<=n; i+=2)
10    {
11        count=1;
12        while(n%i==0)
13        {
14            n/=i;
15            count++;
16        }
17        ans*=count;
18    }
19    if(n>1)
20    {
21        ans*=2;
22    }
23    return ans;
24 }
```

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

```
1 typedef pair<int , int> pii;
2 #define x first
3 #define y second
4
5 pii extendedEuclid(int a, int b) // returns x, y | ax + by =gcd(a,b)
6 {
7     if(b == 0) return pii(1, 0);
8     else
9     {
10         pii d = extendedEuclid(b, a % b);
11         return pii(d.y, d.x - d.y * (a / b));
12     }
13 }
```

8.9 Modular multiplicative inverse

8.9.1 Modular inverse using EGCD

```
1 int modularInverse(int a, int n)
2 {
3     pii ret = extendedEuclid(a, n);
4     return ((ret.x % n) + n) % n;
5 }
```

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 $\varphi(m)$ is Euler's totient function. This follows from the fact that a belongs to the multiplicative group $(\frac{\mathbb{Z}}{m\mathbb{Z}})$ 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

```
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 }
```

Chapter 9

Computational geometry

9.1 Geo Library (Stanford)

```
1 // C++ routines for computational geometry.
2
3 #include <iostream>
4 #include <vector>
5 #include <cmath>
6 #include <cassert>
7 #define MPI (2*acos(0))
8 using namespace std;
9
10 double INF = 1e100;
11 double EPS = 1e-12;
12
13 struct PT {
14     double x, y;
15     PT() {}
16     PT(double x, double y) : x(x), y(y) {}
17     PT(const PT &p) : x(p.x), y(p.y) {}
18     PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
19     PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
20     PT operator * (double c) const { return PT(x*c, y*c); }
21     PT operator / (double c) const { return PT(x/c, y/c); }
22 };
23
24 double dot(PT p, PT q) { return p.x*q.x+p.y*q.y; }
25 double dist2(PT p, PT q) { return dot(p-q,p-q); }
26 double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
27 ostream &operator<<(ostream &os, const PT &p) {
28     os << "(" << p.x << ", " << p.y << ")";
29 }
30
31 // rotate a point CCW or CW around the origin
32 PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
33 PT RotateCW90(PT p) { return PT(p.y,-p.x); }
34 PT RotateCCW(PT p, double t) {
35     return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
36 }
37
38 // project point c onto line through a and b
39 // assuming a != b
40 PT ProjectPointLine(PT a, PT b, PT c) {
41     return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
42 }
```



```

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=d-c; 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;

```

```

285 u = CircleCircleIntersection (PT(1,1), PT(8,8), 5, 5);
286 for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
287 u = CircleCircleIntersection (PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
288 for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
289 u = CircleCircleIntersection (PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
290 for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
291
292 // area should be 5.0
293 // centroid should be (1.1666666, 1.1666666)
294 PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
295 vector<PT> p(pa, pa+4);
296 PT c = ComputeCentroid(p);
297 cerr << "Area: " << ComputeArea(p) << endl;
298 cerr << "Centroid: " << c << endl;
299
300 return 0;
301 }

```

9.2 Convex Hull

9.2.1 Convex Hull 1

```

1 typedef long long LLD;
2 struct POINT
3 {
4     double x,y;
5     POINT(LLD x=0,LLD y=0):x(x),y(y) {};
6 };
7
8 POINT P0;
9
10 int area2 (POINT p0,POINT p1,POINT p2)
11 {
12     return (p2.x-p0.x)*(p1.y-p0.y)-(p1.x-p0.x)*(p2.y-p0.y);
13 }
14 int dist2 (POINT a,POINT b)
15 {
16     int dx=a.x-b.x,dy=a.y-b.y;
17     return dx*dx+dy*dy;
18 }
19
20 ///Compare version 1
21 bool angle_comp (POINT a,POINT b)
22 {
23     if (area2 (P0,a,b)<0)return true;
24     else if (area2 (P0,a,b)==0)return dist2 (P0,a)<dist2 (P0,b);
25     return false;
26 }
27
28 ///Compare version 2
29
30 bool angle_comp2 (POINT a,POINT b)
31 {
32     if (area2 (P0,a,b)==0)
33     {
34         return dist2 (P0,a)<dist2 (P0,b);
35     }
36     int d1x=a.x-P0.x,d1y=a.y-P0.y;
37     int d2x=b.x-P0.x,d2y=b.y-P0.y;
38     return (atan2 ((double)d1y,(double)d1x)-atan2 ((double)d2y,(double)d2x))
39     <0;

```

```

39 }
40
41 vector<POINT> Find_convex_hull (vector<POINT>P)
42 {
43     int i,N=(int)P.size();
44     int po=0;
45     for ( i=0; i<N; i++)
46     {
47         if(P[i].y < P[po].y || (P[i].y== P[po].y && P[i].x < P[po].x))po=i;
48     }
49     P0 = P[po];
50     sort(P.begin(),P.end(),angle_comp);
51     POINT prev(0,0),now(0,0);
52     stack<POINT> S;
53     S.push(P0);
54     i=1;
55     while(i<N)
56     {
57         if(S.size()<2)
58         {
59             S.push(P[i++]);
60         }
61         else
62         {
63             now = S.top();
64             S.pop();
65             prev=S.top();
66             S.push(now);
67             if ( area2 (prev,now,P[i]) <0)
68                 S.push(P[i++]);
69             else S.pop();
70         }
71     }
72
73     vector<POINT> ConvexHull;
74     ConvexHull.push_back(P0);
75     while(!S.empty())
76     {
77
78         ConvexHull.push_back(S.top());
79         S.pop();
80     }
81     return ConvexHull;
82 }

```

9.2.2 Convex Hull (Stanford)

```

1 // Compute the 2D convex hull of a set of points using the monotone chain
2 // algorithm. Eliminate redundant points from the hull if REMOVE_REDUNDANT
3 // is
4 // #defined.
5 // Running time: O(n log n)
6 //
7 // INPUT:   a vector of input points, unordered.
8 // OUTPUT:  a vector of points in the convex hull, counterclockwise,
9 //          starting
10 //          with bottommost/leftmost point
11 #define REMOVE_REDUNDANT
12

```

```

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 #ifndef REMOVEDREDUNDANT
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)
43         *(c.y-b.y) <= 0);
44 }
45 #endif
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]) >=
54             0) up.pop_back();
55         while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <=
56             0) dn.pop_back();
57         up.push_back(pts[i]);
58         dn.push_back(pts[i]);
59     }
60     pts = dn;
61     for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);
62
63     #ifndef REMOVEDREDUNDANT
64     if (pts.size() <= 2) return;
65     dn.clear();
66     dn.push_back(pts[0]);
67     dn.push_back(pts[1]);
68     for (int i = 2; i < pts.size(); i++)
69     {
70         if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
71         dn.push_back(pts[i]);
72     }
73     #endif

```

```

70     }
71     if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1]))
72     {
73         dn[0] = dn.back();
74         dn.pop_back();
75     }
76     pts = dn;
77 #endif
78 }

```

9.3 Determine a point inside polygon or not

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 struct point
4 {
5     int x,y;
6     point() {}
7     point(int a,int b)
8     {
9         x=a,y=b;
10    }
11 };
12 int trun(point p0,point p1,point p2)
13 {
14     int result=(p2.x-p0.x)*(p1.y-p0.y)-(p1.x-p0.x)*(p2.y-p0.y);
15     if(result>0) return 1;
16     if(result < 0) return -1;
17     return result;
18 }
19 bool on_segment(point pi,point pj,point pk)
20 {
21     int minx=min(pi.x,pj.x);
22     int maxx=max(pi.x,pj.x);
23     int miny=min(pi.y,pj.y);
24     int maxy=max(pi.y,pj.y);
25     if(pk.x>=minx&&pk.x<=maxx&&pk.y>=miny&&pk.y<=maxy) return true;
26     return false;
27 }
28 bool Segment_intersect(point p1,point p2,point p3,point p4)
29 {
30     int d1=trun(p3,p4,p1);
31     int d2=trun(p3,p4,p2);
32     int d3=trun(p1,p2,p3);
33     int d4=trun(p1,p2,p4);
34     if((d1*d2<0)&&(d3*d4<0)) return true;
35     if(d1==0&&on_segment(p3,p4,p1)) return true;
36     if(d2==0&&on_segment(p3,p4,p2)) return true;
37     if(d3==0&&on_segment(p1,p2,p3)) return true;
38     if(d4==0&&on_segment(p1,p2,p4)) return true;
39     return false;
40 }
41 bool in_poligon(vector<point>p,point given,point inf2)
42 {
43     int intersect=0;
44     int N=p.size();
45     for(int i=0; i<N; i++)
46     {
47         point p1=p[i];
48         point p2=p[(i+1)%N];

```



```

49         if (Segment_intersect (p1 , p2 , given , inf2 ))
50         {
51             intersect++;
52         }
53     }
54     return intersect%2==1;
55 }
56 int main()
57 {
58     int N;
59     while (scanf ("%d" , &N) && N)
60     {
61         int Max = -10005;
62         vector<point> poligon ;
63         for (int i = 0; i < N; i++)
64         {
65             int x , y;
66             scanf ("%d %d" , &x , &y);
67             poligon . push_back ( point (x , y) );
68             Max = max (x , Max);
69         }
70         int x , y;
71         scanf ("%d %d" , &x , &y);
72         if (in_poligon ( poligon , point (x , y) , point (Max , y) ))
73         {
74             printf ("T\n");
75         }
76         else
77         {
78             printf ("F\n");
79         }
80     }
81     return 0;
82 }

```

9.4 Integer point in a Segment or lattice point

```

1 int IntegerPointsOnSegment (const point &P1, const point &P2)
2 {
3     point P;
4     P.x = abs (P1.x - P2.x);
5     P.y = abs (P1.y - P2.y);
6     if (P.x == 0) return P.y - 1;
7     if (P.y == 0) return P.x - 1;
8     return (--gcd (P.x , P.y)) - 1;
9 }

```

9.5 Segment Intersection

```

1 int Direction (const point_int &O, const point_int &A, const point_int &B) //
2 checks rotation
3 {
4     int rotation = (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
5     if (rotation == 0) return 0;
6     return rotation > 0 ? 1 : -1;
7 }
8 bool On_Segment (point_int p, point_int q, point_int r) // q point lies
9 between p and r
10 {
11     if (q.x <= max (p.x , r.x) && q.x >= min (p.x , r.x) &&

```

```

10         q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
11         return true;
12     return false;
13 }
14 bool Segment_Intersect(point_int p1, point_int p2, point_int q1, point_int
    q2)
15 {
16     int d1 = Direction(p1, p2, q1);
17     int d2 = Direction(p1, p2, q2);
18     int d3 = Direction(q1, q2, p1);
19     int d4 = Direction(q1, q2, p2);
20     if(d1 != d2 && d3 != d4) return true;
21     if(!d1 && On_Segment(p1, q1, p2)) return true;
22     if(!d2 && On_Segment(p1, q2, p2)) return true;
23     if(!d3 && On_Segment(q1, p1, q2)) return true;
24     if(!d4 && On_Segment(q1, p2, q2)) return true;
25     return false;
26 }

```

9.6 Verify Convex polygon

```

1 int trun(point p0, point p1, point p2)
2 {
3     int result = (p2.x - p0.x) * (p1.y - p0.y) - (p1.x - p0.x) * (p2.y - p0.y);
4     return result;
5 }
6 bool isConvex(int n, vector<point> v)
7 {
8     int pos = 0, neg = 0;
9     for(int i = 0; i < n; i++)
10     {
11         int prev = (i + n - 1) % n, next = (i + 1) % n;
12         point A = v[i];
13         point B = v[prev];
14         point C = v[next];
15         int pv = trun(A, B, C);
16         if(pv > 0) pos++;
17         else
18         {
19             if(pv < 0) neg++;
20         }
21     }
22     return (pos == 0) || (neg == 0);
23 }

```

9.7 Closest Pair Algorithm

```

1 const int MAX = 100005;
2 struct point
3 {
4     int x, y, i;
5 };
6 point arr[MAX], sortedY[MAX];
7 bool flag[MAX];
8 template <class T> int getdist(T a, T b)
9 {
10     return max(abs(a.x - b.x), abs(a.y - b.y));
11 }
12 bool compareX(const point &a, const point &b)
13 {

```

```

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 }

```