Contents

1	Data	a Structure	3
	1.1	Segment Tree	3
		1.1.1 Segment Tree & Lazy Propagation	3
		1.1.2 Quadtree	5
		1.1.3 Mergesort Segtree	6
		1.1.4 Persistent Segtree	7
	1.2	Fenwick Tree	9
	1.2	1.2.1 Fenwick Tree 1D	9
	1.0		10
	1.3		11
			11
		±	13
	1.4		16
		1.4.1 Merge Sort & Vector	16
		1.4.2 Merge Sort	۱7
	1.5	Sparse Table	18
	1.6	SQRT Decomposition	19
		1.6.1 Array	19
		J	21
2	Gra	ph 2	23
			23
			23
		· · · · · · · · · · · · · · · · · · ·	24
			24
	2.2		24
	۷.۷		24
			24 24
	0.0		
	2.3		25
		•	25
	2.4		25
			25
			25
	2.5		25
		2.5.1 Maximum Bipartite Matching	25
		2.5.2 Maximum Flow	27
		2.5.3 Minimum Cost Maximum Flow	29
		2.5.4 Minimum Cut	31
	2.6		32
			32
			33
		1	35
		7 0 1	37
		7 0 1	37
	0.7		37
	2.7		37
		2.7.1 2-SAT 3	37
3	Dvn	namic Programming	88
9	- J 11		. •

	3.1	3.1.1 Divide and Counquer	38 38 40
			41
			41
	3.2	<u> </u>	41
	3.3	•	44
	3.4		46
4	Strii		47
	4.1		47
	4.2		48
	4.3		48
	4.4		50
	4.5	O Company of the comp	50
	4.6	Suffix Array & LCP	50
	4.7	Suffix Tree	52
_			
5			53
	5.1		53
			53
			53
			53
			54
			55
	5.2		55
	5.3		56
	5.4		57
		1	57
		1	57
	5.5		57
	5.6	Combinatorics	57
6	Geo	metry	58
U	6.1	·	58
	6.2	<u>*</u>	58
	6.3	•	58
	6.4		58
	0.7		58
			58
	6.5		58
	6.6		58
			58
	6.7		
	6.8	Circle Sweep	58

Chapter 1

Data Structure

1.1 Segment Tree

1.1.1 Segment Tree & Lazy Propagation

```
class segtree
 3
         const static int N=100000;
4 5
         int tr[4*N], lazy[4*N];
     public:
         segtree(){};
 7
         void clear()
 8
 9
              memset(tr, 0, sizeof(tr));
10
              memset(lazy, 0, sizeof(lazy));
11
         void build(int no, int 1, int r, vector<int>&data)
12
13
              if ( l==r )
14
15
                  tr[no]=data[1];
16
17
                  return;
18
19
              int nxt=no*2;
20
              int mid=(1+r)/2;
21
              build(nxt, 1, mid, data);
22
              build(nxt+1, mid+1, r, data);
23
              tr[no] = tr[nxt] + tr[nxt+1];
24
25
         void propagate(int no, int 1, int r)
26
         {
27
              if (!lazy[no])
                  return;
29
30
              tr[no]+=(r-l+1)*lazy[no];
              if(1!=r)
31
32
              {
                  int nxt=no*2;
                  lazy[nxt]+=lazy[no];
34
                  lazy[nxt+1]+=lazy[no];
35
36
37
              lazy[no]=0;
38
39
         void update(int no, int 1, int r, int i, int j, int x)
40
              propagate(no, 1, r);
41
42
              if(l>j || r<i)
43
                  return;
              if(l>=i && r<=j)
45
46
                  lazy[no]=x;
                  propagate(no, 1, r);
47
48
                  return;
49
50
              int nxt=no*2;
```

```
int mid=(l+r)/2;
update(nxt, l, mid, i, j, x);
update(nxt+1, mid+1, r, i, j, x);
51
52
53
54
55
56
57
58
                     tr[no]=tr[nxt]+tr[nxt+1];
               int \ query(int \ no, \ int \ l, \ int \ r, \ int \ i, \ int \ j)
                     propagate(no, 1, r);
if(l>j || r<i)</pre>
59
60
                           return 0;
                     if(l>=i && r<=j)
    return tr[no];</pre>
61
62
63
                     int nxt=no*2;
64
                     int mid=(1+r)/2;
                     int ql=query(nxt, 1, mid, i, j);
int qr=query(nxt+1, mid+1, r, i, j);
65
66
67
                     return (ql+qr);
68
              }
69
```

1.1.2 Quadtree

```
class quadtree
2.
3
           //needs to be NxN
          const static int N=100000;
5
          int tr[16*N];
 6
      public:
7
          quadtree(){};
8
          void build(int node, int l1, int r1, int l2, int r2, vector< vector<int> >data)
9
10
               if(11==12 \&\& r1==r2)
11
12
                    tr[node]=data[l1][r1];
13
                    return;
14
15
               int nxt=node*4;
               int midl=(11+12)/2;
16
17
               int midr=(r1+r2)/2;
18
               build(nxt-2, l1, r1, midl, midr, data);
19
20
               build (nxt-1, \ midl+1, \ r1 \, , \ l2 \, , \ midr \, , \ data);
21
               build(nxt, l1, midr+1, midl, r2, data);
22
               build(nxt+1, midl+1, midr+1, l2, r2, data);
23
24
               tr[node] = tr[nxt-2] + tr[nxt-1] + tr[nxt] + tr[nxt+1];
25
26
          void update(int node, int 11, int r1, int 12, int r2, int i, int j, int x)
27
28
               if (l1>l2 || r1>r2)
29
                    return;
30
               if (i>l2 || j>r2 || i<l1 || j<r1)
31
                    return:
               if(i==11 && i==12 && j==r1 && j==r2)
32
33
34
                    tr[node]=x;
35
                    return;
36
37
               int nxt=node*4;
38
               int midl=(l1+l2)/2;
               int midr=(r1+r2)/2;
39
40
41
               update(nxt-2, l1, r1, midl, midr, i, j, x);
               update(nxt-1, midl+1, r1, l2, midr, i, j, x);
42
               update(nxt, 11, midr+1, midl, r2, i, j, x);
update(nxt+1, midl+1, midr+1, 12, r2, i, j, x);
43
44
45
46
               tr[node]=tr[nxt-2]+tr[nxt-1]+tr[nxt]+tr[nxt+1];
47
48
           int query(int node, int 11, int r1, int 12, int r2, int i1, int j1, int i2, int j2)
49
50
               if(i1>l2 \ || \ j1>r2 \ || \ i2<l1 \ || \ j2<r1 \ || \ i1>i2 \ || \ j1>j2)
51
                    return 0;
52
               if(i1<=l1 && j1<=r1 && l2<=i2 && r2<=j2)
53
                    return tr[node];
54
               int nxt=node*4;
               int midl=(11+12)/2;
55
56
               int midr=(r1+r2)/2;
57
58
               int q1=query(nxt-2, l1, r1, midl, midr, i1, j1, i2, j2);
               int q2=query(nxt-1, midl+1, r1, l2, midr, i1, j1, i2, j2);
int q3=query(nxt, l1, midr+1, midl, r2, i1, j1, i2, j2);
int q4=query(nxt+1, midl+1, midr+1, l2, r2, i1, j2, i2, j2);
59
60
61
62
          }
63
```

1.1.3 Mergesort Segtree

```
1
    class mergesort_segtree
2
3
          const static int N=100000;
4
          vector<int>tr[4*N];
5
     public:
6
          mergesort_segtree(){};
7
          void build(int no, int 1, int r, vector<int>&data)
8
9
              if(l==r)
10
              {
                   tr[no].push_back(data[1]);
11
12
                   return;
13
14
              int nxt=no*2;
15
              int mid=(1+r)/2;
              build(nxt, 1, mid, data);
16
17
              build(nxt+1, mid+1, r, data);
18
              tr[no].resize(tr[nxt].size()+tr[nxt+1].size());
              merge(tr[nxt].begin(), tr[nxt].end(), tr[nxt+1].begin(), tr[nxt+1].end(), tr[no].begin());
19
20
          //how many numbers in (i, j) are greater or equal than k int query(int no, int 1, int r, int i, int j, int k)
21
22
23
24
              if (r<i || l>j)
25
                  return 0;
26
              if(1 >= i \&\& r <= j)
27
                  return (int)(tr[no].end()-upper_bound(tr[no].begin(), tr[no].end(), k));
28
              int nxt=no*2;
29
              int mid=(1+r)/2;
30
              int ql=query(nxt, 1, mid, i, j, k);
31
              int qr=query(nxt+1, mid+1, r, i, j, k);
32
              return ql+qr;
33
34
     };
```

1.1.4 Persistent Segtree

```
class persistent_segtree
 2
 3
          const static int N=100000;
          int n;
 5
          int tr[N];
 6
          int root[N], L[N], R[N];
 7
          int cnt, id;
 8
     public:
 9
          persistent_segtree() {};
10
          void set(int _n)
11
              memset(tr, 0, sizeof(tr));
memset(root, 0, sizeof(root));
12
13
14
              memset(L, 0, sizeof(L));
15
              memset(R, 0, sizeof(R));
16
              id = 0:
17
              cnt = 1;
18
              n=_n;
19
20
          void build(int no, int 1, int r, vector<int>&data)
21
22
              if(l==r)
23
24
                   tr[no]=data[1];
25
                  return;
26
27
              int mid=(1+r)/2;
28
              L[no]=cnt++;
              R[no] = cnt + +;
29
30
              build(L[no], 1, mid, data);
              build(R[no], mid+1, r, data);
31
32
              tr[no]=tr[ L[no] ]+tr[ R[no] ];
33
34
          int update(int no, int 1, int r, int i, int x)
35
36
              int newno=cnt++;
              tr[newno] = tr[no];
37
38
              L[newno]=L[no];
              R[newno]=R[no];
39
40
              if(l==r)
41
42
                   tr[newno]=x:
43
                  return newno;
44
              int mid=(1+r)/2;
45
46
              if(i \le mid)
47
                  L[newno] = update(L[newno], 1, mid, i, x);
48
49
                  R[newno] = update(R[newno], mid+1, r, i, x);
50
              tr[newno] = tr[ L[newno] ] + tr[ R[newno] ];
51
              return newno;
53
          int query(int no, int 1, int r, int i, int j)
54
55
              if(r<i || l>j)
56
                  return 0;
57
              if (1>=i && r<=j)
58
                  return tr[no];
              int mid=(1+r)/2;
59
60
              int ql=query(L[no], 1, mid, i, j);
61
              int qr=query(R[no], mid+1, r, i, j);
62
              return ql+qr;
63
          //update the i-th value to x.
64
65
          void update(int i, int x)
66
67
              root[id+1]=update(root[id], 0, n-1, i, x);
68
              id++;
69
70
          //returns sum(1, r) after the k-th update.
71
          int query(int 1, int r, int k)
72
          {
73
              return query(root[k], 0, n-1, 1, r);
74
```

75 };

1.2 Fenwick Tree

1.2.1 Fenwick Tree 1D

```
class fenwicktree
 1
2
         #define D(x) x&(-x) const static int N=100000;
 4
 5
          int tr[N], n;
6
7
     public:
         fenwicktree(){};
         void build(int _n)
 8
 9
10
              n=_n;
11
              memset(tr, 0, sizeof(tr));
12
13
          void update(int i, int x)
14
              for(i++; i \le n; i+=D(i))
15
16
                  tr[i]+=x;
17
18
          int query(int i)
19
20
              int ret = 0;
21
              for(i++; i>0; i-=D(i))
22
                  ret+=tr[i];
23
              return ret;
24
25
          int rquery(int 1, int r)
26
27
              return query(r)-query(l-1);
28
         }
29
          void set(int i, int x)
30
         {
31
              update(i, -rquery(i, i)+x);
32
         }
33
          void rset(int 1, int r, int x)
34
              update(1, x);
35
36
              update(r+1, -x);
38
     };
```

1.2.2 Fenwick Tree 2D

```
1
    class fenwicktree
2
3
         #define D(x) x&(-x)
         const static int N=1000;
5
         int tr[N][N], n, m;
6
     public:
         fenwicktree() {};
7
8
         void build(int _n, int _m)
9
10
             n=_n, m=_m;
             memset(tr, 0, sizeof(tr));
11
12
13
         void update(int r, int c, int x)
14
15
              for(int i=r+1; i \le n; i+=D(i))
                  for (int j=c+1; j \le m; j+=D(j))
16
                     tr[i][j]+=x;
17
18
         int query(int r, int c)
19
20
21
              int ret = 0;
22
              for(int i=r+1; i>0; i=D(i))
23
                 for (int j=c+1; j>0; j=D(j))
24
                      ret+=tr[i][j];
25
              return ret;
26
27
         int rquery(int r1, int c1, int r2, int c2)
28
         {
29
              if((r1>r2 \&\& c1>c2) || (r1==r2 \&\& c1>c2) || (r1>r2 \&\& c1==c2))
30
              {
                  swap(r1, r2);
31
32
                  swap(c1, c2);
33
34
              else if (r1<r2 && c1>c2)
35
36
                  swap(c1,c2);
37
              else if(r1>r2 && c1<c2)
38
39
40
                 swap(r1,r2);
41
42
              return query(r2, c2)-query(r1-1, c2)-query(r2, c1-1)+query(r1-1, c1-1);
43
44
         void set(int r, int c, int x)
45
46
             update(r, c, -rquery(r, c, r, c)+x);
47
48
     };
```

1.3 Cartesian Tree

1.3.1 Cartesian Tree

```
//srand(time(NULL))
 1
      int vrand()
 3
     {
 4
           return abs(rand()<<(rand()%31));</pre>
     }
6
7
     struct node
 8
 9
           //x=key, y=priority key, c=tree count
           int x, y, c;
node *L, *R;
10
11
12
           node(){};
13
           node(int _x)
14
                x=_x, y=vrand(), c=0;
15
                L=R=NULL;
16
17
18
      };
19
20
      int cnt(node *root)
21
22
           return root?root->c:0;
23
24
25
    void upd_cnt(node *root)
26
      {
27
            if(root)
28
                root \rightarrow c=1+cnt(root \rightarrow L)+cnt(root \rightarrow R);
29
     }
30
31
    void split(node *root, int x, node *&L, node *&R)
32
33
            if (!root)
34
                L=R=NULL;
35
           else if (x < root \rightarrow x)
36
                split(root->L, x, L, root->L), R=root;
37
38
                split(root \rightarrow R, x, root \rightarrow R, R), L=root;
39
           upd_cnt(root);
40
      }
41
42
      void insert(node *&root, node *it)
43
    {
44
           if (!root)
45
                root=it;
           else if(it->y > root->y)
46
47
                split(root, it\rightarrow x, it\rightarrow L, it\rightarrow R), root=it;
48
49
                insert(it\rightarrow x < root\rightarrow x? root\rightarrow L:root\rightarrow R, it);
50
           upd_cnt(root);
51
52
53
    void merge(node *&root, node *L, node *R)
54
            if (!L || !R)
55
56
                root=L?L:R;
57
           else if (L\rightarrow y > R\rightarrow y)
                merge(L\rightarrow R, L\rightarrow R, R), root=L;
58
59
60
                merge(R\rightarrow L, L, R\rightarrow L), root=R;
61
           upd_cnt(root);
      }
62
63
64
      void erase(node *&root, int x)
65
66
            if(root \rightarrow x==x)
                merge(root, root->L, root->R);
67
68
69
                erase(x < root \rightarrow x? root \rightarrow L: root \rightarrow R, x);
70
           upd_cnt(root);
71
```

```
72
 73
     node *unite(node *L, node *R)
 74
       {
75
76
77
            if(!L || !R)
    return L?L:R;
            if(L\rightarrow y < R\rightarrow y)
 78
                 swap(L, R);
            node *Lt, *Rt;

split(R, L->x, Lt, Rt);

L->L=unite(L->L, Lt);

L->R=unite(L->R, Rt);
 79
 80
81
82
83
            return L;
 84
       }
 85
86
       int find(node *root, int x)
 87
 88
            if (! root)
89
                return 0;
 90
            if(root \rightarrow x==x)
91
                 return 1;
92
            if(x > root \rightarrow x)
 93
               return find(root->R, x);
94
            else
95
                 return find(root->L, x);
 96
       }
97
98
       int findkth(node *root, int x)
99
100
            if(!root)
101
                 return -1;
            int Lc=cnt(root->L);
102
103
            if(x-Lc-1==0)
104
                 return root->x;
105
            if(x>Lc)
106
                return findkth(root->R, x-Lc-1);
107
                 return findkth(root->L, x);
108
109
```

1.3.2 Implicit Cartesian Tree

```
//srand(time(NULL))
2
     int vrand()
3
    {
          return abs(rand()<<(rand()%31));</pre>
5
    }
6
7
    struct node
8
9
          //basic treap: x=key, y=priority key, c=tree count;
10
          int x, y, c;
          //treap operations: v=max(x), lazy=lazy value of propagation, rev=reversed
11
12
          int v, lazy, rev;
13
14
          node *L, *R;
          node(){};
15
16
          node(int _x)
17
18
              x=_x, y=vrand();
L=R=NULL;
19
20
               v=x;
               lazy=0;
21
22
               rev=0;
23
24
     };
25
26
      //updating functions
27
     inline int get_cnt(node *root)
28
29
          return root?root->c:0;
     }
30
31
      inline void upd_cnt(node *root)
32
33
34
          if(root)
35
               root \rightarrow c=1+get\_cnt(root \rightarrow L)+get\_cnt(root \rightarrow R);
36
     }
37
      inline void push (node *&root)
38
39
40
          if(root && root->rev)
41
42
               root \rightarrow rev = 0;
43
               swap(root->L, root->R);
44
               if (root->L)
45
                   root->L->rev^=1;
46
               if(root->R)
47
                   root \rightarrow R \rightarrow rev^=1;
48
          }
49
50
51
    inline void propagate(node *&root)
53
          if(root)
54
          {
55
               if (!root->lazy)
56
                   return;
57
               int lazy=root->lazy;
               root->x+=lazy;
58
59
60
               if (root->L)
                  root->L->lazy=lazy;
61
62
               if (root->R)
63
                   root->R->lazy=lazy;
64
               root \rightarrow lazy = 0;
65
66
     }
67
68
     inline int get_max(node *root)
69
70
          return root?root->v:-INF;
71
72
73
      inline void upd_max(node *root)
74
```

```
75
           if(root)
 76
                root->v=max(root->x, max(get_max(root->L), get_max(root->R)));
 77
 78
 79
      inline void update(node *root)
 80
      {
 81
           propagate(root);
 82
           upd_cnt(root);
 83
           upd_max(root);
 84
      }
 85
 86
       void merge(node *&root, node *L, node *R)
 87
 88
           push(L);
 89
           push(R);
 90
           if (!L || !R)
 91
               root=L?L:R;
 92
           else if (L\rightarrow y > R\rightarrow y)
               merge(L\rightarrow R, L\rightarrow R, R), root=L;
 93
 94
           else
 95
                merge(R\rightarrow L, L, R\rightarrow L), root=R;
 96
           update(root);
 97
 98
 99
     void split(node *root, node *&L, node *&R, int x, int add=0)
100
101
            if (!root)
102
                return void(L=R=NULL);
103
           push(root);
104
            int ix=add+get_cnt(root->L); //implicit key
105
           if(x \le ix)
106
                split(root \rightarrow L, L, root \rightarrow L, x, add), R=root;
107
108
                split(root->R, root->R, R, x, add+1+get_cnt(root->L)), L=root;
109
           update(root);
110
      }
111
112
       //insert function
113
      void insert(node *&root, int pos, int x)//(insert x at position pos)
114
115
           node *R1, *R2;
116
           split(root, R1, R2, pos);
117
           merge(R1, R1, new node(x));
118
           merge(root, R1, R2);
119
120
121
      //erase value x
122
      void erase_x(node *&root, int x)
123
124
           if (!root)
125
                return;
126
           if(root \rightarrow x = = x)
127
                merge(root, root->L, root->R);
128
129
                erase_x(x < root \rightarrow x? root \rightarrow L:root \rightarrow R, x);
130
           update(root);
131
132
133
      //erase kth value
134
      void erase_kth(node *&root, int x)
135
136
           if (!root)
137
                return:
138
           int Lc=get_cnt(root->L);
139
           if(x-Lc-1==0)
140
                merge(root, root \rightarrow L, root \rightarrow R);
141
            else if (x>Lc)
142
                erase kth(root->R, x-Lc-1);
143
            else
144
                erase_kth(root->L, x);
           update(root);
145
146
147
148
      //add x to [l,r]
       inline void paint(node *&root, int 1, int r, int x)
149
150
151
           node *R1, *R2, *R3;
```

```
split(root, R1, R2, 1);
152
           split (R2, R2, R3, r-l+1);
153
154
           R2 \rightarrow lazy = x;
155
           propagate(R2);
156
157
           merge(root, R1, R2);
158
           merge(root, root, R3);
159
160
161
       //max range query [1,r]
162
       inline int rquery(node *&root, int 1, int r)
163
           node *R1, *R2, *R3;
split(root, R1, R2, 1);
164
165
           split (R2, R2, R3, r-l+1);
166
167
           int ret=R2->v;
           merge(root, R1, R2);
merge(root, root, R3);
168
169
170
           return ret;
171
172
173
       inline void reverse(node *&root, int 1, int r)//reverse elements [1, r]
174
           node *R1, *R2, *R3;
175
176
           split(root, R1, R2, 1);
           split (R2, R2, R3, r-1+1);
R2->rev^=1;
177
178
179
           merge(root, R1, R2);
180
           merge(root, root, R3);
181
182
183
       //output functions
       int poscnt=0;
void output_all(node *root)
184
185
186
      {
187
           if(!root)
188
                return;
189
           update(root);
190
           push (root);
           output_all(root->L);
191
           printf("[%d]_%d\n", poscnt++, root->x);
output_all(root->R);
192
193
194
195
196
      int output_kth(node *root, int x)
197
198
           if (!root)
199
                return \ -1;
200
           update(root);
201
           push(root);
202
            int Lc=get_cnt(root->L);
203
           if(x-Lc-1==0)
204
                return root->x;
205
            if(x>Lc)
206
                return output_kth(root->R, x-Lc-1);
207
208
                return output_kth(root->L, x);
209
```

1.4 Merge Sort & Swap Count

1.4.1 Merge Sort & Vector

```
#define INF 0x3F3F3F3F
 2
3
4
5
      int mergesort(vector<int>&data)
            if (data.size()==1)
                 return 0;
 6
7
           vector < int > L, R;
int t = data.size();
 8
            for(int i=0; i<t/2; i++)
           L.push_back(data[i]);

for(int i=t/2; i<t; i++)

R.push_back(data[i]);
 9
10
11
            int ret=mergesort(L)+mergesort(R);
12
            for(int i=0, j=0, k=0; j<L.size() | | k<R.size(); <math>i++
13
14
15
                 int x=j<L.size()?L[j]:INF;</pre>
                 int y=k<R.size()?R[k]:INF;
if(x<y)</pre>
16
17
18
19
                      data[i]=x;
20
                      j++;
21
22
23
24
                 else
                      data[i]=y;
25
26
                      k++;
                      ret+=(L. size()-j);
27
28
29
            return ret;
30
```

1.4.2 Merge Sort

```
1
2
3
      #define INF 0x3F3F3F3F
     int temp[100000];
int mergesort(int data[], int 1, int r)
 4
5
      {
           if (abs(l-r)<=1)
    return 0;
int mid=(l+r)/2;</pre>
 6
7
 8
            int ret=mergesort(data, 1, mid)+mergesort(data, mid, r);
            for(int i=1; i<r; i++)
    temp[i]=data[i];</pre>
 9
10
            for(int i=1, j=1, k=mid; j<mid || k<r; i++)</pre>
11
12
13
                 int x=j<mid?temp[j]:INF;</pre>
                 int y=k<r?temp[k]:INF;</pre>
14
15
                 if(x < y)//x <= y
16
                 {
17
                      data[i]=x;
18
                      j++;
19
20
                 else
21
22
23
                      data[i]=y;
                      k++;
24
                      ret+=(mid-j);
25
26
27
            return ret;
28
```

1.5 Sparse Table

```
class sparsetable
 1
             #define lbit(x) 63-__builtin_clzll(x);
const static int N=100000, LN=20;
int data[N][LN], n, ln;
 3
4
5
6
7
8
9
        public:
              sparsetable(){};
              void clear()
10
                    memset(data, 0, sizeof(data));
11
12
             void build(vector<int>&foo)
13
14
                    n=foo.size();
                    ln=lbit(n);
15
16
                    for(int i=0; i<n; i++)
                          data[i][0]=foo[i];
17
                    for(int j=1; j<=ln; j++)
for(int i=0; i<n-(1<<j)+1; i++)
18
19
20
                                data[i][j]=max(data[i][j-1], data[i+(1<<(j-1))][j-1]);
21
22
              int query(int 1, int r)
23
24
                    int i=abs(1-r)+1;
25
                    int j=lbit(i);
26
                    \textbf{return} \hspace{0.2cm} \max(\hspace{0.1cm} \texttt{data}\hspace{0.1cm} \texttt{[l][j]}, \hspace{0.2cm} \texttt{data}\hspace{0.1cm} \texttt{[l-(1<< j)+1][j])}\hspace{0.1cm};
27
28
        };
```

1.6 SQRT Decomposition

1.6.1 Array

```
const int N=100000;
 1
      int SN=sqrt(N);
 3
      class mo
 5
 6
7
      public:
          int 1, r, i;
 8
          mo(){};
 9
          mo(int _l, int _r, int _i)
10
           {
11
               l=_1, r=_r, i=_i;
12
13
           bool operator <(const mo &foo) const
14
                if((r/SN)!=(foo.r/SN))
15
16
                    return (r/SN)<(foo.r/SN);</pre>
17
                if(1!=foo.1)
18
                    return 1<foo.1;</pre>
19
               return i < foo.i;</pre>
20
           }
21
22
23
      int data[N], freq[N], ans[N];
24
      int cnt=0;
25
      void update(int p, int s)
26
27
           int x=data[p];
28
           if(s==1)
29
30
               if(freq[x]==0)
31
                    cnt++;
32
33
           else
34
           {
35
               if(freq[x]==1)
36
                    cnt--;
37
38
           freq[x]+=s;
39
40
41
    int main()
42
      {
43
           int n;
          fit i,
scanf("%d", &n);
for(int i=1; i<=n; i++)
    scanf("%d", &data[i]);</pre>
44
45
46
47
           int q;
scanf("%d", &q);
48
49
50
           vector < mo>querys;
51
           for (int i=0; i < q; i++)
52
               int 1, r;
scanf("%d_%d", &l, &r);
53
54
55
               querys.push_back(mo(1, r, i));
56
57
           sort(querys.begin(), querys.end());
58
59
           int l=1, r=1;
60
           cnt=0;
           memset(freq, 0, sizeof(freq));
61
62
           update(1, 1);
           for(int i=0; i < q; i++)
63
64
           {
65
               int li=querys[i].1;
66
               int ri=querys[i].r;
               int ii=querys[i].i;
while(l>li)
67
68
69
                    update(--1, 1);
70
               while (r<ri)
71
                    update(++r, 1);
```

1.6.2 Tree

```
#define pb push back
      #define ALL(x) x.begin(),x.end()
2.
3
      const int N=1e+5+35;
5
      const int M=20;
      const int SN=sqrt(2*N)+1;
8
      class mo
9
     {
      public:
10
11
          int 1, r, i, lc;
12
          mo(){};
          mo(\,\textbf{int}\,\,\_l\,,\,\,\,\textbf{int}\,\,\_r\,,\,\,\,\textbf{int}\,\,\_lc\,,\,\,\,\textbf{int}\,\,\_i\,)
13
14
15
               l=_l, r=_r, lc=_lc, i=_i;
16
          bool operator <(const mo &foo) const
17
18
          {
                if((r/SN)!=(foo.r/SN))
19
20
                    return (r/SN)<(foo.r/SN);</pre>
21
                if (1!=foo.1)
22
                    return 1<foo.1;</pre>
23
               return i < foo.i;</pre>
24
          }
25
     };
26
     int n, q;
int h[N], lca[N][M];
vector<int>g[N];
27
28
29
30
      int dl[N], dr[N], di[2*N], cur;
31
32
      void dfs(int u, int p)
33
34
           d1[u]=++cur;
35
          di[cur]=u;
36
           lca[u][0]=p;
          for(int i=1; i<M; i++)
lca[u][i]=lca[ lca[u][i-1] ][i-1];
37
38
39
          for(int i=0; i<g[u].size(); i++)</pre>
40
          {
41
               int v=g[u][i];
42
               if(v==p)
43
                    continue;
44
               h[v]=h[u]+1;
45
               dfs(v, u);
46
47
          dr[u]=++cur;
48
          di[cur]=u;
49
50
51
    inline int getLca(int u, int v)
53
           if(h[u]>h[v])
54
               swap(u, v);
          for(int i=M-1; i>=0; i--)
55
56
               if(h[v]-(1 << i)>=h[u])
57
                   v=lca[v][i];
58
          if(u==v)
59
               return u;
60
          for(int i=M-1; i>=0; i---)
61
62
               if (lca[u][i]!=lca[v][i])
63
               {
64
                    u=lca[u][i];
65
                    v=lca[v][i];
66
               }
67
68
          return lca[u][0];
69
70
71
     map<string , int>remap;
      int data[N], ans[N], vis[N], freq[N], cnt;
72
73
      inline void update(int u)
74
      {
```

```
75
           int x=data[u];
 76
           if(vis[u] && (--freq[ data[u] ]==0))
 77
 78
           else if(!vis[u] && (freq[ data[u] ]++==0))
 79
               cnt++;
 80
           vis[u]^=1;
 81
 82
 83
     int main()
 84
 85
           scanf("%d_%d", &n, &q);
           for (int i=1; i <= n; i++)
 86
 87
 88
               char temp[25];
               scanf("%s", temp);
 89
 90
               string temp2=string(temp);
 91
               if (!remap.count(temp2))
 92
                   remap[temp2]=remap.size();
 93
               data[i]=remap[temp2];
 94
 95
           for(int i=1; i < n; i++)
 96
           {
 97
               int u, v;
scanf("%d_%d", &u, &v);
 98
 99
               g[u].pb(v);
100
               g[v].pb(u);
101
102
           dfs(1, 0);
103
104
           vector < mo>query;
105
           for (int i=0; i < q; i++)
106
               int u, v;
scanf("%d_%d", &u, &v);
107
108
109
               int lc=getLca(u, v);
110
               if(dl[u]>dl[v])
                   swap(u, v);
111
112
               query.p\bar{b}(mo(u==lc?dl[u]:dr[u], dl[v], lc, i));
113
114
           sort(ALL(query));
115
116
           int l=query[0].1, r=query[0].1-1;
117
           cnt=0;
118
           for (int i=0; i < q; i++)
119
120
               int li=query[i].l;
121
               int ri=query[i].r;
122
               int lc=query[i].lc;
               int ii=query[i].i;
123
               while(l>li)
124
125
                   update(di[--1]);
126
               while (r<ri)
127
                   update(di[++r]);
128
               while(l<li)
129
                   update(di[1++]);
130
               while (r>ri)
131
                   update(di[r--]);
132
               int u=di[1], v=di[r];
133
134
               if(lc!=u \&\& lc!=v)
                   update(lc);
135
136
               ans[ii]=cnt;
               if (lc!=u && lc!=v)
137
138
                   update(lc);
139
           for(int i=0; i<q; i++)
140
141
               printf("%d\n", ans[i]);
142
           return 0;
143
```

Chapter 2

Graph

- 2.1 Components
- 2.1.1 Articulations, Bridges & Cycles

2.1.2 Strongly Connected Components

Tarjan

```
class graph
1
2
3
          const static int MN=1e+5;
4
     public:
5
          vector<int>data[MN], aux;
6
7
          bool vis[MN];
          int grp[MN];
8
          int dfs_num[MN], dfs_low[MN];
9
          int dfs_cnt, numSCC;
10
11
          graph(){};
          void clear()
12
13
14
              for (int i=0; i \triangleleft MN; i++)
15
16
                   data[i].clear();
17
                   dfs_num[i]=-1;
                   dfs_low[i]=0;
18
19
                   vis[i]=false;
20
21
              aux.clear();
              dfs_cnt=numSCC=0;
23
24
          void add_edge(int u, int v)
25
26
              data[u].push_back(v);
27
28
          void tarjanSCC(int u)
29
30
              dfs_num[u] = dfs_low[u] = dfs_cnt++;
31
              aux.push_back(u);
32
              vis[u]=true;
33
              for(int i=0; i<data[u].size(); i++)</pre>
34
35
36
                   int v=data[u][i];
37
                   if (dfs_num[v] = = -1)
                       tarjanSCC(v);
39
                   if(vis[v])
                       dfs_low[v]=min(dfs_low[v], dfs_low[u]);
40
41
42
43
               if (dfs_num[u]==dfs_low[u])
44
              {
                   while(1)
45
46
47
                       int v=aux.back();
48
                       aux.pop_back();
49
                       vis[v] = \overline{false};
50
                       grp[v]=numSCC;
51
                        if(u==v)
52
                            break;
53
                   numSCC++;
55
56
```

2.1.3 Semi-Strongly Connected Components

2.2 Single Source Shortest Path

2.2.1 Dijkstra

2.2.2 Bellmanford

```
class node
1
3
    public:
4
         int x, y, d;
node(){};
5
6
7
          node(int _x, int _y, int _d)
8
              x=_x, y=_y, d=_d;
10
     };
11
12
     int n, v;
13
     vector<node>graph;
14
     int dist[1035];
15
     bool bellmanford(int s)
16
     {
17
          memset(dist, INF, sizeof(dist));
18
          dist[s]=0;
19
          for(int i=0; i< n-1; i++)
20
21
              for (int j=0; j < graph.size(); j++)
23
                   int x=graph[j].x;
24
                   int y=graph[j].y;
25
                   int d=graph[j].d;
26
                  if(dist[y]>dist[x]+d)
27
                       dist[y] = dist[x] + d;
28
              }
29
30
          for(int i=0; i<graph.size(); i++)
31
32
              int x=graph[i].x;
33
34
              int y=graph[i].y;
              int d=graph[i].d;
35
36
              if(dist[x]<INF \&\& dist[y]>dist[x]+d)
37
                  return true;
38
39
          return false;
40
```

2.3 All Pairs Shortest Path

2.3.1 Floyd Warshall

2.4 Minimum Spannig Tree

- 2.4.1 Kruskal
- 2.4.2 Prim
- **2.5** Flow

2.5.1 Maximum Bipartite Matching

```
const int MN=1e+3;
     vector<int>g[MN];
3
     int match[MN], rmatch[MN], vis[MN];
     int findmatch(int u)
5
6
         if(vis[u])
7
             return 0;
8
         vis[u]=true;
9
         for(int v:g[u])
10
             if(match[v]==-1 || findmatch(match[v]))
11
12
```

```
match[v]=u;
match[u]=v;
return 1;

return 0;

return 0;

int maxMatch(int n)

int ret=0;
memset(match, -1, sizeof(match));
for(int i=0; i<n; i++)

memset(vis, false, sizeof(vis));
ret+=findmatch(i);

return ret;

return ret;

return ret;
</pre>
```

2.5.2 Maximum Flow

Dinic

```
1
     class graph
2
3
          const static int N=100000;
 4
     public:
5
         vector< pair<int, int> >edge;
 6
         vector < int > adj[N];
7
         int ptr[N];
8
         int dist[N];
9
10
         graph(){};
11
         void clear()
12
         {
              for(int i=0; i< N; i++)
13
14
                  adj[i].clear();
15
              edge.clear();
16
17
         void add_edge(int u, int v, int c)
18
19
              adj[u].push_back(edge.size());
              edge.push_back(mp(v, c));
20
21
              adj[v].push_back(edge.size());
22
              edge.push_back(mp(u, 0)); //(u, c) if is non-directed
23
24
         bool dinic_bfs(int s, int t)
25
26
              memset(dist, -1, sizeof(dist));
27
              dist[s]=0;
28
29
              queue<int>bfs;
30
              bfs.push(s);
              while (! bfs.empty() && dist[t]==-1)
31
32
              {
                   int u=bfs.front();
33
34
                  bfs.pop();
35
                  for(int i=0; i<adj[u].size(); i++)</pre>
36
                  {
37
                       int idx=adj[u][i];
38
                       int v=edge[idx].F;
39
                       if(dist[v]==-1 \&\& edge[idx].S>0)
40
41
42
                           dist[v] = dist[u] + 1;
43
                           bfs.push(v);
44
                       }
45
46
47
              return dist[t]!=-1;
48
49
          int dinic_dfs(int u, int t, int flow)
50
51
              if(u==t)
52
                  return flow;
              for(int &i=ptr[u]; i<adj[u].size(); i++)</pre>
53
54
                   int idx=adj[u][i];
55
56
                  int v=edge[idx].F;
57
                  if(dist[v]==dist[u]+1 \&\& edge[idx].S>0)
58
                  {
                       int cf=dinic_dfs(v, t, min(flow, edge[idx].S));
59
60
                       if(cf>0)
61
62
                           edge[idx].S-=cf;
                           edge[idx^1].S+=cf;
63
                           return cf;
64
65
66
                  }
67
68
              return 0;
69
70
         int maxflow(int s, int t)
71
72
              int ret = 0;
```

2.5.3 Minimum Cost Maximum Flow

```
Undirected graph:

u \rightarrow uu(flow, 0)

uu \rightarrow vv(flow, cost)

vv \rightarrow v(flow, 0)

v \rightarrow uu(flow, 0)

vv \rightarrow u(flow, 0)
```

Dijkstra

```
typedef int FTYPE; //type of flow
typedef int CTYPE; //type of cost
typedef pair <FTYPE, CTYPE>pfc;
 1
 3
      const CTYPE CINF=INF;
 5
     const FTYPE FINF=INF;
 6
 7
     void operator+=(pfc &p1, pfc &p2)
 8
 9
          p1.F+=p2.F;
10
          p1.S+=p2.S;
11
12
13
     class graph
14
          const static int MN=1e+4;
15
      public:
16
17
          int n;
18
          FTYPE flow [MN];
          CTYPE dist[MN], pot[MN];
19
          int prev[MN], eidx[MN];
20
21
22
          struct Edge
23
24
               int to;
               FTYPE cap;
25
26
               CTYPE cost;
27
               Edge(){};
28
               Edge(int _to, FTYPE _cap, CTYPE _cost)
29
30
                    to=_to;
31
                    cap=_cap;
32
                    cost=_cost;
33
34
35
          struct node
36
               int u;
37
               CTYPE d;
38
39
               node(){};
40
               node(int _u, CTYPE _d)
41
42
                    u=_u;
                    d=_d;
43
44
45
               bool operator <(const node &foo) const
46
               {
47
                    return d>foo.d;
48
               }
49
          graph(){};
50
          vector < int > adj [MN];
51
52
          vector<Edge>edge;
53
           inline void set(int n)
54
          {
55
               n=_n ;
56
57
          inline void reset()
58
          {
59
               for (int i=0; i < MN; i++)
                    adj[i].clear();
60
61
               edge.clear();
62
          }
```

```
inline void add_edge(int u, int v, FTYPE c, FTYPE cst)
 63
 64
 65
               adj[u].push_back(edge.size());
               edge.push_back(Edge(v, c, cst));
 66
 67
               adj[v].push_back(edge.size());
 68
               edge.push_back(Edge(u, 0, -cst));
 69
 70
 71
           pfc dijkstra(int s, int t)
 72
 73
               for (register int i=0; i< n; i++)
                    dist[i]=CINF;
 74
 75
               dist[s]=0;
 76
               flow[s]=FINF;
 77
               priority_queue < node > heap;
 78
               heap.push(node(s, 0));
 79
               while (! heap.empty())
 80
 81
                    int u=heap.top().u;
                   CTYPE d=heap.top().d;
 82
 83
                   heap.pop();
 84
                   if (d>dist[u])
 85
                        continue;
 86
                    for(int i=0; i<adj[u].size(); i++)</pre>
 87
 88
                        int idx=adj[u][i];
 89
                        int v=edge[idx].to;
                        CTYPE w=edge[idx].cost;
 90
 91
                        if(!edge[idx].cap \mid \mid dist[v] \le d+w+pot[u]-pot[v])
 92
                            continue;
                        if (d+w<dist[v])
 93
 94
                        {
 95
                            dist[v]=d+w;
 96
                            prev[v]=u;
 97
                            eidx[v]=idx;
                            flow[v]=min(flow[u], edge[idx].cap);
 98
 99
                            heap.push(node(v, d+w));
100
                        }
101
102
               if ( dist[t]==CINF)
103
104
               return mp(FINF, CINF);
pfc ret=mp(flow[t], 0);
105
               for(int u=t; u!=s; u=prev[u])
106
107
108
                    int idx=eidx[u];
                   edge[idx].cap—=flow[t];
109
110
                   edge[idx^1].cap+=flow[t];
111
                    ret.second+=flow[t]*edge[idx].cost;
112
113
               return ret;
114
          }
115
116
           inline pfc mfmc(int s, int t)
117
118
               pfc ret=mp(0, 0);
119
               pfc got;
120
               while((got=dijkstra(s, t)).first!=FINF)
121
                    ret+=got;
122
               return ret;
123
124
      };
```

Bellmanford

2.5.4 Minimum Cut

Stoer Wagner

```
1
    int stoer_wagner(int n)
     {
 3
          int ret=INF;
          for(int i=0; i<n; i++)
v[i]=i;</pre>
 4
5
 6
7
          while (n>1)
 8
          {
 9
               a[ v[0] ]=true;
               for (int i=1; i < n; i++)
10
11
12
                   a[v[i]] = false;
13
                   na[i-1]=i;
14
                   w[i]=graph[v[0]][v[i]];
15
16
17
               int prev=v[0];
18
               for (int i=1; i< n; i++)
19
20
                   int zj = -1;
21
                   for (int j=1; j < n; j++)
22
                   {
23
                        if(!a[ v[j] ] && (zj<0 || w[j]>w[zj]))
24
                            zj=j;
25
26
27
                   a[v[zj]]=true;
28
29
                   if(i==n-1)
30
31
                       ret=min(ret, w[zj]);
32
33
                        for(int j=0; j< n; j++)
                            graph[ v[j] ][prev]=graph[prev][ v[j] ]+=graph[ v[zj] ][ v[j] ];
35
                        v[zj]=v[--n];
36
                       break;
37
38
                   prev=v[zj];
39
                   for(int j=1; j<n; j++)
    if(!a[ v[j] ])</pre>
40
41
42
                           w[j]+=graph[ v[zj] ][ v[j] ];
43
44
45
          return ret;
46
```

2.6 Tree

2.6.1 Lowest Common Ancestor

```
const int MN=1e+5+35;
 1
     const int LMN=1+log2(MN);
     vector<int>graph[MN];
 3
     int LVL[MN];
    int T[MN];
6
     int dp[MN][LMN];
     bool vis[MN];
 8
9
    void dfs (int u, int f, int d)
10
         vis[u]=true;
11
12
         LVL[x]=d;
13
         dp[x][0] = f;
         for (int i=1; i < LMN; i++)
14
             dp[x][i]=dp[dp[x][i-1]][i-1];
15
16
17
         vis[x]=true;
         for(int i=0; i<graph[x].size(); i++)</pre>
18
19
20
              int v=graph[x][u];
21
              if (! vis [v])
22
                  dfs(v, x, d+1);
23
24
     }
25
26
     inline int lca(int u, int v)
27
28
          if(LVL[u]>LVL[v])
29
              swap(u, v);
         for(int i=LMN-1; i>=0; i---)
30
              if(LVL[v]-(1 << i)>=LVL[u])
31
32
                  v=dp[v][i];
33
          if(u==v)
34
              return u;
35
         for (int i=LMN-1; i>=0; i---)
36
37
              if(dp[u][i]!=dp[v][i])
38
              {
39
                  u=dp[u][i];
40
                  v=dp[v][i];
41
42
43
         return dp[u][0];
44
```

2.6.2 Centroid Decomposition

```
class graph
2
3
          const static int N=1e+5;
          const static int LN=log 2(N)+1;
5
     public:
 6
          vector<int>g[N];
7
          int h[N], lca[N][LN];
8
9
          int sz[N];
10
          int cg[N], gsz, dlt[N];
11
          graph(){};
12
          inline void addEdge(int u, int v)
13
14
              g[u].pb(v);
15
              g[v].pb(u);
16
17
          void buildLca(int u, int f)
18
              lca[u][0] = f;
19
20
              for (int i=1; i < LN; i++)
21
                  lca[u][i]=lca[ lca[u][i-1] ][i-1];
22
              for(int v:g[u])
23
                  if(v==f)
24
25
                      continue;
                  h[v]=h[u]+1;
26
27
                  buildLca(v, u);
28
29
30
          inline int getLca(int u, int v)
31
32
              if(h[u]>h[v])
33
                  swap(u, v);
              for(int i=LN-1; i>=0; i--)
if(h[v]-(1<<i)>=h[u])
34
35
                       v=lca[v][i];
36
37
              if(u==v)
38
                  return u;
39
              for (int i=LN-1; i>=0; i---)
40
41
                   if (lca[u][i]!=lca[v][i])
42
                   {
43
                       u=lca[u][i];
44
                       v=lca[v][i];
45
46
47
              return lca[u][0];
48
49
          inline int getDist(int u, int v)
50
          {
51
              return h[u]+h[v]-2*h[getLca(u, v)];
53
          void buildSz(int u, int f)
54
          {
55
              gsz++;
56
              sz[u]=1;
57
              for(int v:g[u])
58
59
                   if (v==f || dlt[v])
60
                       continue;
                   buildSz(v, u);
61
62
                  sz[u]+=sz[v];
63
64
65
          int findCentroid(int u, int f)
66
67
              for(int v:g[u])
68
              {
                   if (v==f || dlt [v])
69
70
                       continue;
71
                   if(sz[v]*2>=gsz)
72
                       return findCentroid(v, u);
73
74
              return u;
```

2.6.3 Heavy Light Decomposition on Edges

```
class segtree
 2
 3
          const static int N=1e+5;
     public:
 5
          int tr[4*N];
 6
          segtree(){};
 7
          void reset()
 8
 9
              memset(tr, 0, sizeof(tr));
10
          void update(int no, int 1, int r, int i, int val)
11
12
          {
13
              if (r<i || l>i)
14
                  return;
15
              if (1>=i && r<=i)
16
              {
17
                  tr[no]=val;
18
                  return;
19
20
              int nxt = (no << 1);
21
              int mid=(1+r)>>1;
22
              update(nxt, 1, mid, i, val);
              update(nxt+1, mid+1, r, i, val);
23
24
              tr[no] = tr[nxt] + tr[nxt+1];
25
          int query(int no, int 1, int r, int i, int j)
26
27
28
              if (r<i || l>j)
29
                  return 0;
30
              if(l>=i && r<=j)
31
                  return tr[no];
32
              int nxt = (no << 1);
33
              int mid=(1+r)>>1;
34
              return query(nxt, 1, mid, i, j)+query(nxt+1, mid+1, r, i, j);
35
36
     };
37
38
     const int N=1e+5;
     const int M=log2(N)+1;
39
40
     int n;
41
     segtree tr;
42
     vector< pair<int,int> >g[N];
43
     int lca[N][M];
44
     int h[N], trSz[N];
45
46
     //in - use X[], Y[] in case
     //of edge weights
47
     int X[N], Y[N], W[N];
48
49
50
     //hld
51
     int chainInd[N], chainSize[N], chainHead[N], chainPos[N], chainNo, posInBase[N];
52
     int ptr;
53
54
     void dfs(int u, int 1)
55
56
          trSz[u]=1;
57
          lca[u][0]=1;
          for (int i=1; i \triangleleft M; i++)
58
59
              lca[u][i]=lca[lca[u][i-1]][i-1];
60
          for(int i=0; i<g[u].size(); i++)</pre>
61
62
              int v=g[u][i].first;
              if(v==1)
63
64
                  continue;
65
              h[v]=h[u]+1;
              dfs(v, u);
66
67
              trSz[u]+=trSz[v];
68
         }
69
70
71
    inline int getLca(int u, int v)
72
     {
73
          if(h[u]>h[v])
74
              swap(u, v);
```

```
75
          for(int i=M-1; i>=0; i---)
 76
               if(h[v]-(1<< i)>=h[u])
 77
                   v=lca[v][i];
 78
          if(u==v)
 79
               return u;
 80
          for (int i=M-1; i>=0; i---)
 81
 82
               if (lca[u][i]!=lca[v][i])
 83
 84
                   u=lca[u][i];
 85
                   v=lca[v][i];
 86
               }
 87
 88
          return lca[u][0];
 89
 90
 91
     //dont use 'c' if the weight is on the vertex
      //instead of the edge
 92
     inline void hld(int u, int l, int c)
 93
 94
 95
           if (chainHead[chainNo]==-1)
 96
               chainHead[chainNo]=u;
 97
          chainInd[u]=chainNo;
 98
          chainPos[u]=chainSize[chainNo]++;
 99
          tr.update(1, 0, n, ptr, c);
100
          posInBase[u]=ptr++;
101
102
          int msf, idx;
103
          msf=idx=-1;
104
          for(int i=0; i<g[u].size(); i++)</pre>
105
106
               int v=g[u][i].first;
107
               if(v==1)
108
                   continue;
109
               if(trSz[v]>msf)
110
               {
                   msf=trSz[v];
111
112
                   idx=i;
113
114
          if(idx>=0)
115
               hld(g[u][idx].first, u, g[u][idx].second);
116
117
          for(int i=0; i<g[u].size(); i++)</pre>
118
119
               if(i==idx)
120
                   continue;
121
               int v=g[u][i].first;
122
               int w=g[u][i].second;
123
               if(v==1)
                   continue;
124
125
               chainNo++;
126
               hld(v, u, w);
127
128
      }
129
130
      inline int query_up(int u, int v)
131
          int uchain=chainInd[u];
132
133
          int vchain=chainInd[v];
          int ret = 0;
134
135
          while (true)
136
               uchain=chainInd[u];
137
138
               if (uchain==vchain)
139
                   ret+=tr.query(1, 0, n, posInBase[v]+1, posInBase[u]);
140
141
                   break;
142
143
               int head=chainHead[uchain];
144
               ret+=tr.query(1, 0, n, posInBase[head],posInBase[u]);
145
               u=head;
146
               u=lca[u][0];
147
148
          return ret;
149
150
    //returns sum of all edges weights
151
```

```
//from 'u' to 'v'
152
      inline int query(int u, int v)
153
154
      {
155
           if(u==v)
156
               return 0;
           int l=getLca(u, v);
157
158
           return query_up(u, 1)+query_up(v, 1);
159
160
       //set and edge to value 'val'
161
162
      inline void update(int u, int val)
163
           int x=X[u], y=Y[u];
if(lca[x][0]==y)
164
165
166
               tr.update(1, 0, n, posInBase[x], val);
167
168
               tr.update(1, 0, n, posInBase[y], val);
169
170
171
      void clearHld()
172
173
           //tr.reset();
174
           for(int i=0; i \le n; i++)
175
176
               g[i].clear();
177
               chainHead[i]=-1;
178
               chainSize[i]=0;
179
180
           ptr=1;
181
           chainNo=0;
182
      }
183
184
      int main()
185
186
           scanf("%d", &n);
           clearHld();
187
           for(int i=1; i < n; i++)
188
189
190
               scanf("\%d\_\%d\_\%d", \&X[i], \&Y[i], \&W[i]);\\
               g[ X[i] ].push_back({Y[i], W[i]});
191
192
               g[ Y[i] ].push_back({X[i], W[i]});
193
194
           dfs(1, 0);
195
           hld(1, 0, 0);
           int q;
scanf("%d", &q);
196
197
198
           while (q---)
199
               int o, x, y;
scanf("%d_%d_%d", &o, &x, &y);
200
201
202
               if(o==1)
                    printf("%d\n", query(x, y));
203
204
               else
205
                    update(x, y);
206
207
           return 0;
208
```

- 2.6.4 Heavy Light Decomposition on Vertex
- 2.6.5 All-Pairs Distance Sum
- 2.6.6 All-Pairs Distance & FFT
- **2.7 MISC**
- 2.7.1 2-SAT

Dynamic Programming

3.1 Optimizations

3.1.1 Divide and Counquer

```
/// David Mateus Batista <david.batista3010@gmail.com>
     /// Computer Science - Federal University of Itajuba - Brazil
    /// Uri Online Judge - 2475
     #include <bits/stdc++.h>
     using namespace std;
8
      typedef long long 11;
     typedef unsigned long long ull;
10
     typedef long double ld;
11
     typedef pair < int , int > pii;
      typedef pair<ll, ll> pll;
12
13
14
     #define INF 0x3F3F3F3F
     #define LINF 0x3F3F3F3F3F3F3F3FLL
15
     #define DINF (double)1e+30
16
17
     #define EPS (double)1e-9
     #define PI (double) acos(-1.0)
18
19
     #define RAD(x) (double)(x*PI)/180.0
     #define PCT(x,y) (double)x*100.0/y
     #define pb push_back
21
22
     #define mp make_pair
     #define pq priority_queue
#define F first
23
     #define S second
     #define D(x) x&(-x)
27
     #define ALL(x) x.begin(),x.end()
     #define SET(a,b) memset(a, b, sizeof(a))
29
     \#define DEBUG(x,y) cout << x << y << endl
     #define gcd(x,y) __gcd(x, y)
#define lcm(x,y) (x/gcd(x,y))*y
30
31
     #define bitcnt(x) __builtin_popcountll(x)
#define lbit(x) 63-_builtin_clzll(x)
#define zerosbitl(x) __builtin_ctzll(x)
32
33
     #define zerosbit(x) __builtin_ctz(x)
35
36
     \textbf{enum} \ \{ North \, , \ East \, , \ South \, , \ West \};
37
38
     //{0, 1, 2, 3}
39
     //{Up, Right, Down, Left}
40
41
      int mi[] = \{-1, 0, 1, 0, -1, 1, 1, -1\};
     int mj[] = \{0, 1, 0, -1, 1, 1, -1, -1\};
42
43
      const int MN=1e+4+35;
45
     const int MN2=535;
46
      int p, a;
      11 data[MN];
47
48
49
     inline 11 getValue(int 1, int r)
50
```

```
51
           return (r-l+1)*(data[r]-data[l-1]);
52
      }
53
54
      11 \ dp [MN2] [MN];
      inline void solve(int k, int 1, int r, int L, int R)
55
56
      {
57
            if(l>r)
58
                 return;
59
           int m=(1+r)/2;
60
           int s=L;
61
           dp[k][m]=LINF;
62
           for(int i=max(m, L); i \le R; i++)
63
64
                 if(dp[k][m]>dp[k-1][i+1]+getValue(m+1, i+1))
65
66
                     dp[k][m]=dp[k-1][i+1]+getValue(m+1, i+1);
67
                     s=i;
68
69
70
71
           solve(k, l, m-1, L, s);
solve(k, m+1, r, s, R);
72
      }
73
74
      int main()
75
76
77
           scanf("%d_%d", &p, &a);
for(int i=1; i<=p; i++)
78
           {
79
                 11 x;
                scanf("%lld", &x);
data[i]=data[i-1]+x;
80
81
82
           }
83
           for(int i=0; i \le p; i++)
84
85
                dp[0][i]=LINF;
86
           for (int i=0; i \le a; i++)
87
                dp[i][p]=0;
           for(int i=1; i<=a; i++)
solve(i, 0, p-1, 0, p-1);
printf("%lld\n", dp[a][0]);
88
89
90
91
           return 0;
92
```

3.1.2 Convex Hull I

```
Original recurrence: dp[i] = min(dp[j] + b[j] * a[i]) \text{ for } j < i Conditions: b[j] >= b[j+1] a[i] <= a[i+1] Solution: Hull \text{ cht} = Hull() \text{ or DynamicHull cht}; \text{cht.insertLine}(b[0], dp[0]) \text{for}(\text{int } i = 1; i < n; i + +) \{ dp[i] = \text{cht.query}(a[i]); \text{cht.insertLine}(b[i], dp[i]); \} answer is dp[n-1];
```

Linear

```
class Hull
 1
 2
3
4
5
6
7
          const static int CN=1e+5+35;
      public:
          long long a[CN], b[CN];
          double x[CN];
          int head, tail;
          Hull():head(1), tail(0){};
 8
 9
10
          long long query(long long xx)
11
               if (head>tail)
12
13
                   return 0;
               while(head<tail && x[head+1]<=xx)</pre>
14
15
                    head++;
16
               x[head]=xx;
               return a[head]*xx+b[head];
17
18
          }
19
          void insertLine(long long aa, long long bb)
20
21
22
               double xx=-1e18;
23
               while (head <= tail)
24
               {
25
26
                    if (aa==a[tail])
                        return;
                    xx=1.0*(b[tail]-bb)/(aa-a[tail]);
if(head==tail || xx>=x[tail])
27
28
29
                        break;
30
                    tail --;
31
32
               a[++tail]=aa;
               b[tail]=bb;
33
34
               x[tail]=xx;
35
36
      };
```

Dynamic

```
const long long is_query=-(1LL<<62);</pre>
     class Line
3
     public:
5
         long long m, b;
         mutable function < const Line *() > succ;
 6
 7
         bool operator < (const Line &rhs) const
8
9
              if (rhs.b!=is_query)
10
                  return m<rhs.m;
              const Line *s=succ();
11
12
              if (!s)
13
                 return 0;
              long long x=rhs.m;
14
15
              return (b-s->b)<((s->m-m)*x);
         }
16
17
18
     class HullDynamic: public multiset<Line>
19
20
     public:
21
22
         void clear()
23
24
              clear();
25
26
         bool bad(iterator y)
27
28
              auto z=next(y);
29
              if (y==begin())
30
              {
31
                  if (z==end())
32
                      return 0;
33
                  return (y->m==z->m \&\& y->b<=z->b);
35
              auto x=prev(y);
36
              if (z==end())
37
                 return (y->m == x->m \&\& y->b<=x->b);
              return ((x-b-y-b)*(z-m-y-m) >= (y-b-z-b)*(y-m-x-m));
38
39
40
         void insertLine(11 m, 11 b)
41
42
              auto y=insert({m, b});
43
              y->succ=[=]
44
45
                  return next(y)==end()?0:&*next(y);
46
47
              if (bad(y))
48
              {
49
                  erase(y);
50
                  return;
51
52
              while(next(y)!=end() && bad(next(y)))
              erase(next(y));
while(y!=begin() && bad(prev(y)))
53
54
55
                  erase(prev(y));
56
57
         long long query(long long x)
              auto ret=*lower_bound((Line){x, is_query});
59
60
              return ret.m*x+ret.b;
61
62
     };
```

3.1.3 Convex Hull II

3.1.4 Knuth Optimization

3.2 Matrix Exponentiation

```
typedef long long 11;
      typedef vector<vector<ll>> matrix;
      const 11 MOD=303700049;
      int n, t;
     11 k;
 6
      ll val[101];
 8
      11 modmul(11 a, 11 b)
 9
10
          return ((a%MOD)*(b%MOD))%MOD;
11
12
13
      11 modsum(11 a, 11 b)
14
     {
          return ((a%MOD)+(b%MOD))%MOD;
15
16
     }
17
18
      matrix basem;
19
      matrix mat_mul(matrix A, matrix B)
2.0
21
          int t=A.size();
          matrix ret=basem;
for(int i=0; i<t; i++)
22
23
24
25
               for(int j=0; j< t; j++)
26
27
                   for(int k=0; k< t; k++)
28
                   {
29
                        ret[i][j]=(ret[i][j]+A[i][k]*B[k][j]);
30
31
                   ret[i][j]%=MOD;
32
33
34
          return ret;
35
36
37
    matrix mat_pow(matrix &A, 11 k)
38
39
          if(k==1)
40
               return A;
41
          if (k&1)
42
               return mat_mul(A, mat_pow(A,k-1));
43
          matrix ret=mat_pow(A, k >> 1);
44
          return mat_mul(ret, ret);
45
46
47
    //o build pode variar, sendo ele a base do fibonacci
48
      matrix build()
49
          matrix ret(t, vector<ll>(t));
for(int i=0; i<n; i++)</pre>
50
51
               ret[0][i]=i+1;
52
53
          for (int i=1; i < n; i++)
               for(int j=0; j<n; j++)
ret[i][j]=(j+1==i);
54
55
          for(int i=0; i< n; i++)
57
               ret[t-1][i]=i+1;
58
          ret[t-1][t-1]=1;
59
          return ret;
60
     }
61
62
      pair<ll, ll>calc(ll k)
63
64
          if(n>=k)
               return mp(val[k-1], 0);
65
66
          matrix base=build();
67
          matrix fib=mat_pow(base, k-n);
68
69
          11 ret=0;
          reverse(val, val+n);
for(int i=0; i<n; i++)
70
71
72
               ret=modsum(ret, modmul(fib[0][i], val[i]));
73
          11 sum=0;
74
```

```
 \begin{array}{ll} \mbox{for(int $i=0$; $i<n$; $i++)$} \\ \mbox{sum=modsum(sum, modmul(fib[n][i], val[i]));} \end{array} 
 75
 76
 77
             return mp(ret, sum);
 78
79
       }
 80
       void solve()
 81
             //First = f(n-x), Second = somatoaria de f(0) ate f(n-x) pair < ll, ll > ans = calc(k);
 82
 83
 84
             if (k>n)
 85
                   for(int i=0; i < n; i++)
 86
 87
 88
                        ans.S=ans.S+val[i];
                        if (ans.S>MOD)
 89
 90
                             ans.S%≔MOD;
 91
 92
 93
             else
 94
             {
 95
                   for(int i=0; i< k; i++)
 96
                   {
 97
                        ans.S=ans.S+val[i];
 98
                        if (ans.S>MOD)
 99
                             ans.S%≔MOD;
100
                  }
101
102
             printf("%lld_%lld\n", ans.F, ans.S);
103
104
105
       int main()
106
107
             \label{eq:while} while (\, \text{scanf} \, (\, \text{"%d\_\%lld} \, \text{"} \, , \, \, \&n \, , \, \, \&k \, ) \, ! = EOF)
108
109
                   t=n+1;
                  basem.clear();
110
111
                   basem.resize(t, vector<11>(t));
112
                   //val[i] = valores iniciais conhecidos da recorrencia
113
                   for(int i=0; i<n; i++)
114
115
                        scanf("%lld", &val[i]);
116
                   }
117
                   solve();
118
119
             return 0;
120
```

3.3 Digits

```
char str[100];
 1
 2
     int dp[100][300][2];
 3
     bool memo[100][300][2];
     int n, k;
      //numeros de 0 a x, tal que a soma dos digitos eh igual a k
 6
     int solve(int i, int s, int t)
 8
     {
 9
         if(i==n)
10
11
              if(!t \&\& s==k)
12
                  return 1;
13
              return 0;
14
         if(s>k)
15
16
              return 0;
17
          if (memo[i][s][t])
18
              return dp[i][s][t];
19
          int &ret=dp[i][s][t]=0;
20
         if(t)
21
22
              for(int j=0; j<=str[i]-'0'; j++)
23
24
                  if (j==str[i]-'0')
25
                      ret+=solve(i+1, s+j, 1);
26
                  else
27
                      ret+=solve(i+1, s+j, 0);
28
              }
29
30
         else
31
32
              for(int j=0; j<10; j++)
33
34
                  ret+=solve(i+1, s+j, 0);
35
36
37
         memo[i][s][t]=true;
38
         return ret;
39
40
41
     //quantos bits ativos existem entre 0 e x
42
     string str2;
43
     int n2;
     int dp2[100][300][2];
44
45
     bool memo2[100][300][2];
     int solve2(int i, int s, int t)
46
47
48
          if(i==n2)
49
             return s;
50
          if (memo2[i][s][t])
51
              return dp2[i][s][t];
52
         int &ret=dp2[i][s][t]=0;
53
         if(t)
54
         {
              for(int j=0; j<=str2[i]-'0'; j++)
55
56
                  if (j==str2[i]-'0')
57
58
                      ret+=solve2(i+1, s+(j==1), 1);
59
60
                      ret += solve2(i+1, s+(j==1), 0);
61
62
         }
63
         else
64
         {
65
              for (int j=0; j<2; j++)
66
              {
67
                  ret+=solve2(i+1, s+(j==1), 0);
68
              }
69
70
         memo2[i][s][t]=true;
71
         return ret;
72
     }
73
    //numeros de 1 a x, tal que a soma dos digitos eh multiplo de k
```

```
75
      char str3[100];
 76
      int n3;
 77
      int dp3[100][300][2];
      bool memo3[100][300][2];
 78
      int solve3(int i, int s, int t)
 79
 80
      {
 81
           if(i==n3)
 82
               return !s;
           if (memo3[i][s][t])
 83
 84
               return dp3[i][s][t];
 85
           int &ret=dp3[i][s][t]=0;
 86
          if(t)
 87
 88
               for(int j=0; j<=str3[i]-'0'; j++)
 89
 90
                   if (j==str3[i]-'0')
 91
                       ret+=solve3(i+1, (s+j)%k, 1);
 92
                   else
 93
                       ret += solve3(i+1, (s+j)\%k, 0);
 94
               }
 95
 96
          else
 97
 98
               for(int j=0; j<10; j++)
 99
100
                   ret+=solve3(i+1, (s+j)%k, 0);
101
102
103
          memo3[i][s][t]=true;
104
          return ret;
105
106
      //numeros de 1 a x, tal que o xor dos digitos eh igual a k
107
      char str4[100];
108
109
      int n4;
      int dp4[100][300][2];
110
      bool memo4[100][300][2];
111
112
      int solve4(int i, int s, int t)
113
           if(i==n4)
114
115
               return s==k;
          if (memo4[i][s][t])
116
117
              return dp4[i][s][t];
118
          int &ret=dp4[i][s][t]=0;
119
          if(t)
120
          {
121
               for(int j=0; j<=str4[i]-'0'; j++)
122
               {
123
                   if (j==str4[i]-'0')
124
                       ret+=solve4(i+1, (s^j), 1);
125
126
                       ret+=solve4(i+1, (s^j), 0);
127
128
          }
129
          else
130
131
               for (int j=0; j<10; j++)
132
               {
133
                   ret+=solve4(i+1, (s^j), 0);
134
               }
135
136
          memo4[i][s][t]=true;
137
          return ret;
138
```

3.4 Grundy Numbers

Positions have the following properties:

- All terminal positions are losing.
- If a player is able to move to a losing position then he is in a winning position.
- If a player is able to move only to the winning positions then he is in a losing position.

```
const int MN=1e+5;
 1
     bool memo[MN];
2
3
4
5
     int dp[MN];
     int grundy(int x)
6
          if(x==0)
              return 0;
          if (memo[x])
8
9
              return dp[x];
10
          set < int>mex;
11
          for (;;) // moves
              mex.insert(grundy(x-(moves)));
12
13
          int &ret=dp[x]=0;
14
          while (\verb"mex.count(ret)")
15
              ret++;
16
         memo[x] = true;
17
          return ret;
18
```

String

4.1 Hash

```
typedef unsigned long long ull;
     class hashc
3
     public:
5
         vector<ull>prefix;
         vector<ull>power;
7
         int k=37;
8
          int t;
         hashc(){};
         hashc(vector<int>&data)
10
11
12
              t=data.size();
13
              prefix.resize(t+1, 0);
14
              power.resize(t+1, 0);
              prefix[0]=0;
15
16
              power[0]=1;
17
              for (int i=0; i < t; i++)
18
                  prefix[i+1]=prefix[i]*k+data[i];
19
20
                  power[i+1]=power[i]*k;
21
22
         }
23
24
         hashc build(string &str)
26
              vector < int > data (str.size());
27
              for(int i=0; i<str.size(); i++)</pre>
28
                  data[i]=(str[i]-'a'+1);
29
              return hashc(data);
30
         }
31
          ull get()
32
33
34
              return prefix[t];
35
36
          ull calc(int 1, int r)
37
              return prefix[r]-(prefix[l-1]*power[r-l+1]);
38
39
         bool same(int xl, int xr, int yl, int yr)
40
41
42
              return this->calc(xl, xr)==this->calc(yl, yr);
43
44
         int find(hashc &pattern)
45
46
              int pt=pattern.t;
47
              ull val=pattern.calc(1, pt);
48
              for(int i=1; i \le t-pt+1; i++)
49
50
                  if(this \rightarrow calc(i, i+pt-1)==val)
                      return i-1;
51
52
53
              return -1;
```

```
54 }
55 };
```

4.2 KMP

```
int lps[1000000];
1
2
     void lps_calc(string &str)
3
    {
4
          lps[0]=0;
5
          for(int i=1, j=0, f=0; i < str.size(); i+=f, f=0)
6
7
              if(str[i]==str[j])
8
9
                   lps[i]=j;
10
                   j++;
                   f=1;
11
12
              }
13
              else
14
              {
                   if(j>0)
15
16
                   {
17
                        j=lps[j-1];
18
19
                   else
20
                   {
21
                        lps[i]=0;
                        f=1;
23
24
              }
25
26
27
28
     //finding str in pat
29
    void kmp(string &str, string &pat)
30
31
          lps_calc(pat);
32
          int i = 0, j = 0;
33
          while(i<str.size())
34
35
              if(str[i]==pat[j])
36
              {
37
                   i++;
38
                   j++;
39
40
              if (j==pat.size())
41
                   printf("Padrao\_encontrado\_em: \_[\%d, \_\%d]", \ i-j \,, \ (i-j)+pat.\,size\,()-1);
42
43
                   j=lps[j-1];
44
              }
45
              else if(i<str.size() && str[i]!=pat[j])</pre>
46
              {
47
                   if(j!=0)
48
                       j=lps[j-1];
49
                   else
                       i++;
50
51
52
          }
53
```

4.3 Aho Corasick

```
class aho_corasick
1
    private:
3
         static const int MNT=1e+6;
5
         static const int MNC=26;
6
     public:
7
         int trie[MNT][MNC];
8
         int term[MNT];
9
         int link[MNT];
10
         int sum[MNT];
11
         int cnt=1;
         aho_corasick(){};
12
```

```
13
          void clear()
14
15
              RESET(trie, 0);
              RESET(term, 0);
RESET(link, 0);
16
17
18
              RESET(sum, 0);
19
              cnt = 1;
20
21
          int node(int x, int j)
22
          {
23
              return trie[x][j];
24
25
          int end(int x, int j)
26
          {
27
              return term[ node(x,j) ];
28
29
          void insert(char *str)
30
31
              int sz=strlen(str);
32
              int no=0;
33
              for(int i=0; i < sz; i++)
34
              {
35
                   int x=str[i]-'a';
36
                   if (! trie [no][x])
37
                       trie[no][x]=cnt++;
38
                  sum[ trie[no][x] ]++;
39
                  no=trie[no][x];
40
41
              term[no]++;
42
          bool find (char *str)
43
44
45
              int sz=strlen(str);
46
              int no=0;
47
              for (int i=0; i < sz; i++)
48
              {
                   int x=str[i]-'a';
49
50
                   if(!sum[ trie[no][x] ])
51
                       return false;
52
                  no=trie[no][x];
53
54
              return true;
55
56
          void erase(char *str)
57
58
              int sz=strlen(str);
59
              int no=0;
60
              for(int i=0; i < sz; i++)
61
                  int x=str[i]-'a';
62
63
                  sum[ trie[no][x] ]--;
64
                  no=trie[no][x];
65
66
              term[no]--;
67
68
          void update_link()
69
70
              queue<int>aho;
71
              aho.push(0);
72
              while (! aho.empty())
73
74
                   int x=aho.front();
75
                  aho.pop();
76
                  term[x] = term[link[x]];
77
                   for (int i=0; i < MNC; i++)
78
79
                       if(trie[x][i])
80
                       {
                            int y=trie[x][i];
81
82
                           aho.push(y);
83
                           link[y]=x?trie[link[x]][i]:0;
84
85
                       else
86
                       {
                            trie[x][i]=trie[ link[x] ][i];
87
88
                       }
89
```

```
90 }
91 }
92 };
```

4.4 Manacher

4.5 Z-Algorithm

4.6 Suffix Array & LCP

```
1
     const int MN=1e+6+35;
     int data[MN], sa[MN], lcp[MN], lcp_rank[MN];
      // lexicographic order for pairs
 5
     inline bool leq(int a1, int a2, int b1, int b2)
 6
 7
          return(a1 < b1 || a1 == b1 && a2 <= b2);
 8
     }
10
     // and triples
     inline bool leq(int a1, int a2, int a3, int b1, int b2, int b3)
11
12
     {
13
          return(a1 < b1 || a1 == b1 && leq(a2,a3, b2,b3));
14
     } // and triples
15
     // stably sort a[0..n-1] to b[0..n-1] with keys in 0..K from r
16
17
     static void radixPass(int* a, int* b, int* r, int n, int K)
18
         count occurrences
          int* c = new int[K + 1]; // counter array
19
20
         for (int i = 0; i \le K; i++)
         c[i] = 0; // reset counters
for (int i = 0; i < n; i++)
21
22
23
              c[r[a[i]]]++; // count occurrences
24
         for (int i = 0, sum = 0; i \le K; i++) // exclusive prefix sums
25
              int t = c[i];
27
              c[i] = sum;
28
              sum += t;
30
         for (int i = 0; i < n; i++)
31
              b[c[r[a[i]]]++] = a[i]; // sort
32
     }
33
34
     // find the suffix array SA of s[0..n-1] in {1..K}EEn
35
     // require s[n]=s[n+1]=s[n+2]=0, n>=2
36
     void suffixArray(int* s, int* SA, int n, int K)
37
38
          int n0 = (n+2)/3, n1 = (n+1)/3, n2 = n/3, n02 = n0+n2;
         int* s12 = new int[n02+3]; s12[n02] = s12[n02+1] = s12[n02+2] = 0;
39
40
          int* SA12 = new int[n02+3]; SA12[n02] = SA12[n02+1] = SA12[n02+2] = 0;
41
         int* s0 = new int[n0];
42
          int* SA0 = new int[n0];
43
         // generate positions of mod 1 and mod 2 suffixes
         // the "+(n0-n1)" adds a dummy mod 1 suffix if n%3 == 1 for (int i=0, j=0; i < n + (n0-n1); i++)
44
45
              if (i\%3 != 0) s12[j++] = i;
47
         // lsb radix sort the mod 1 and mod 2 triples
         radixPass(s12, SA12, s+2, n02, K);
radixPass(SA12, s12, s+1, n02, K);
48
49
50
         radixPass(s12 , SA12, s , n02, K);
51
          // find lexicographic names of triples
         int name = 0, c0 = -1, c1 = -1, c2 = -1;
52
53
         for (int i = 0; i < n02; i++)
55
              if (s[SA12[i]] != c0 || s[SA12[i]+1] != c1 || s[SA12[i]+2] != c2)
56
57
                  name++:
58
                  c0 = s[SA12[i]];
59
                  c1 = s[SA12[i]+1];
60
                  c2 = s[SA12[i]+2];
61
```

```
62
               if (SA12[i]%3 == 1) s12[SA12[i]/3] = name; // left half
               else s12[SA12[i]/3 + n0] = name; // right half
 63
 64
           // recurse if names are not yet unique
 65
 66
           if (name < n02)
 67
 68
               suffixArray(s12, SA12, n02, name);
 69
               // store unique names in s12 using the suffix array
 70
               for (int i=0; i< n02; i++)
 71
                    s12[SA12[i]] = i + 1;
 72
 73
           else // generate the suffix array of s12 directly
 74
 75
               for(int i = 0; i < n02; i++)
 76
                    SA12[s12[i] - 1] = i;
 77
 78
              stably sort the mod 0 suffixes from SA12 by their first character
 79
           for (int i=0, j=0; i< n02; i++)
 80
               if (SA12[i] < n0) s0[j++] = 3*SA12[i];
           radixPass(s0, SA0, s, n0, K);
 81
 82
           // merge sorted SAO suffixes and sorted SA12 suffixes
 83
           for (int p = 0, t = n0-n1, k = 0; k < n; k++)
 84
               #define GetI() (SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2)
 85
 86
               int i = GetI(); // pos of current offset 12 suffix
               int j = SAO[p]; // pos of current offset 0 suffix
if (SA12[t] < n0 ? // different compares for mod 1 and mod 2 suffixes
    leq(s[i], s12[SA12[t] + n0], s[j], s12[j/3]) :</pre>
 87
 88
 89
 90
                    leq(s[i], s[i+1], s12[SA12[t]-n0+1], s[j], s[j+1], s12[j/3+n0]))
 91
               {// suffix from SA12 is smaller
                    SA[k] = i; t++;
 92
 93
                    if (t == n02) // done --- only SAO suffixes left
 94
                    for (k++; p < n0; p++, k++) SA[k] = SA0[p];
95
 96
               else
               {// suffix from SAO is smaller
 97
 98
                    SA[k] = j; p++;
 99
                    if (p == n0) // done -
                                               only SA12 suffixes left
100
                    for (k++; t < n02; t++, k++) SA[k] = GetI();
101
102
           }
103
104
105
      void buildlcp(int n)
106
107
           int k=0;
108
           for(int i=0; i< n; i++)
109
               lcp_rank[ sa[i] ]=i;
110
           for(int i=0; i< n; i++, k?k--:0)
111
112
               if(lcp_rank[i]==n-1)
113
114
                    k=0:
115
                    continue;
116
               int j=sa[ lcp_rank[i]+1 ];
117
118
               while (i+k \le n \& j+k \le n \& data[i+k] == data[k+j])
119
                   k++;
120
               lcp[ lcp_rank[i] ]=k;
121
122
      }
123
124
      int main()
125
126
           int n;
           scanf("%d", &n);
127
128
           for(int i=0; i < n; i++)
129
               char x;
scanf("_%c", &x);
130
131
132
               data[i]=(int)x;
133
134
           //data[i]>=1
135
           data[n]=data[n+1]=data[n+2]=data[n+3]=0;
136
137
           //suffixArray(string, ponteiro para suffix array, numero de elementos da string, number of
               elementos do alfabeto);
```

4.7 Suffix Tree

Mathematic

5.1 Prime Numbers

5.1.1 Erastotenes Sieve

5.1.2 Linear Sieve

```
//prime(x):(lp[x]==x)
    const int MN=1e+6;
    long long lp [MN+1];
    vector<long long>pr;
6
7
    void sieve()
8
       for (long long i=2; i \le MN; i++)
9
          if(lp[i]==0)
10
11
             lp[i]=i;
12
13
              pr.push_back(i);
14
          15
16
              lp[i*pr[j]]=pr[j];
17
18
```

5.1.3 Miller Rabin

```
1
     //millerRabin(n) returns if n is prime
     //not accurate for all n
#define gcd(x, y) __gcd(x, y)
11 powmod(ll a, ll b, ll m)
 3
 5
           11 ret = 1;
 6
 7
           while(b)
 8
 9
                if (b&1)
10
                    ret = (ret*a)\%m, --b;
11
                else
12
                     a=(a*a)%m, b>>=1;
13
14
           return ret;
15
16
    bool millerRabin(11 n)
17
18
19
           for(ll g; (g=gcd(n, b))!=1; b++)
20
21
                if(n>g)
22
                    return false;
23
           11 p=0, q=n-1;
           while ((q&1)==0)
24
25
               p++, q>>=1;
```

```
26
          11 \text{ rem=powmod(b, q, n)};
2.7
          if(rem==1 | rem==n-1)
28
               return true;
29
          for(11 i=1; i < p; i++)
30
31
               rem=(rem*rem)%n;
32
               if(rem==n-1)
33
                   return true;
34
35
          return false;
36
```

5.1.4 BPSW

```
//bpsw(n) returns if n is prime
     #define gcd(x, y) __gcd(x, y)
11 jacobi(11 a, 11 b)
 3
 4
      {
 5
            if (a==0 || a==1)
 6
                 return a;
 7
            if (a<0)
 8
            {
 9
                  if((b\&2)==0)
10
                      return jacobi(-a, b);
11
                 return -jacobi(-a, b);
12
13
            11 a1=a, e=0;
14
            while ((a1&1)==0)
15
                a1>>=1, e++;
            11 s;
16
17
            if((e\&1)==0 | | (b\&7)==1 | | (b\&7)==7)
18
                 s=1;
            else
19
20
                 s = -1;
21
            if ((b&3)==3 && (a1&3)==3)
22
                 s=-s;
23
            if(a1==1)
24
                 return s:
25
            return s*jacobi(b%a1, a1);
26
      }
27
28
      bool bpsw(ll n)
29
30
            if((11) sqrt(n+0.0)*(11) sqrt(n+0.0)==n)
31
                 return false;
            11 dd=5;
32
33
            while (1)
34
            {
35
                 11 g=gcd(n, abs(dd));
36
                 if(1<g && g<n)
                      return false;
37
38
                 if(jacobi(dd, n) = = -1)
39
                      break;
40
                 dd = dd < 0? - dd + 2: - dd - 2;
41
            11 p=1, q=(p*p-dd)/4;
11 d=n+1, s=0;
42
43
44
            while ((d&1)==0)
                 s++, d>>=1;
45
            11\ u\!=\!1,\ v\!=\!p\,,\ u2m\!=\!1,\ v2m\!=\!p\,,\ qm\!=\!q\,,\ qm2\!=\!q\!*\!2\,,\ qkd\!=\!q\,;
46
47
            for(11 mask=2; mask<=d; mask<<=1)</pre>
48
49
                 u2m = (u2m * v2m)%n;
50
                 v2m = (v2m * v2m) %n;
51
                 while (v2m<qm2)
52
                      v2m+=n;
53
                 v2m-=qm2;
54
                 qm=(qm*qm)%n;
55
                 qm2=qm*2;
56
                 if (d&mask)
57
                       \begin{array}{lll} 11 & t1\!=\!(u2m\!*v)\!\%\!n\,, & t2\!=\!(v2m\!*u)\!\%\!n\,; \\ 11 & t3\!=\!(v2m\!*v)\!\%\!n\,, & t4\!=\!(((u2m\!*u)\!\%\!n)\!*dd)\!\%\!n\,; \end{array}
58
59
60
                       u=t1+t2;
61
                       if (u&1)
62
                            u+=n;
```

```
63
                  u=(u>>1)%n;
64
                   v = (t3 + t4);
65
                   if (v&1)
66
                       v+=n;
67
                   v = (v >> 1) \% n;
68
                  qkd=(qkd*qm)%n;
69
70
71
          if (u==0 || v==0)
72
              return true;
73
          11 qkd2=qkd*2;
          for(11 r=1; r<s; r++)
74
75
76
              v=(v*v)%n-qkd2;
              v+=v<0?n:0;
77
78
              v+=v<0?n:0;
79
              v=v>=n?n:0;
              v=v>=n?n:0;
80
81
              if(v==0)
82
                  return true;
83
              if(r < s-1)
              {
85
                  qkd=(qkd*1LL*qkd)%n;
86
                  qkd2=qkd*2;
87
88
          return false;
89
90
```

5.1.5 Primality Test

```
//call sieve() before isPrime(x)
 1
     //define k=50 as trivial limit
 3
     bool isPrime(11 x)
 4
5
     {
          if(x==1)
6
7
              return false;
          if(x==2)
 8
              return true;
 9
          if(x\%2==0)
10
              return false;
          for(int i=0; i<k && x>pr[i]; i++)
11
12
              if (x%pr[i]==0)
                  return false;
13
14
          if(pr[k-1]*pr[k-1]>=x)
15
              return true;
          //return only millerRabin(x) for fast process
//not accurate for all x
16
17
18
          return millerRabin(x)?bpsw(x):false;
19
```

5.2 Chinese Remainder Theorem

5.3 Fast Fourier Transformation

```
#define PI (double) acos (-1.0)
      typedef complex<double> base;
 3
     void fft(vector<base>&data, bool invert)
           int n=data.size();
 6
          for(int i=1, j=0; i < n; i++)
 8
               int bit=n>>1;
 9
               for (; j>=bit; bit>>=1)
10
                   j-=bit;
               j+=bit;
11
               if(i<j)
12
13
                   swap(data[i], data[j]);
14
          }
15
16
          for(int len=2; len<=n; len<<=1)</pre>
17
18
               double ang=2*PI/len*(invert?-1:1);
               base wlen(cos(ang), sin(ang));
for(int i=0; i<n; i+=len)</pre>
19
2.0
21
22
                   base w(1);
23
                    for(int j=0; j<len/2; j++)
24
                    {
25
                        base u=data[i+j], v=data[i+j+len/2]*w;
26
                        data[i+j]=u+v;
                        data[i+j+len/2]=u-v;
28
                        w*=wlen;
29
30
               }
31
32
           if(invert)
33
               for(int i=0; i < n; i++)
                   data[i]/=n;
34
35
36
37
    vector<int>fft_multiply(vector<int>&a, vector<int>&b)
38
39
          vector<base>fa(a.begin(), a.end());
40
          vector < base > fb (b.begin(), b.end());
41
          int n=1;
          while(n<max(a.size(), b.size()))</pre>
42
43
              n<<=1;
          n<<=1:
44
45
          fa.resize(n);
46
          fb.resize(n);
47
          fft(fa, false);
          fft(fb, false);
for(int i=0; i<n; i++)
48
49
50
               fa[i]*=fb[i];
51
          fft(fa, true);
52
53
          vector<int>ret(n);
          for(int i=0; i<n; i++)
ret[i]=(int)(fa[i].real()+0.5);</pre>
54
55
57
          int carry=0;
58
          for(int i=0; i < n; i++)
60
               ret[i]+=carry;
61
               carry=ret[i]/10;
62
               ret[i]%=10;
63
64
          return ret;
65
66
67
    int main()
68
69
          scanf("%d_%d", &n, &m);
70
          vector<int>a,b;
71
72
73
          for(int i=0; i < n; i++)
74
```

```
75
76
77
                  int x;
scanf("%d", &x);
                  a.pb(x);
 78
79
             for(int i=0; i \triangleleft m; i++)
 81
                  int x;
scanf("%d", &x);
 82
 83
                  b.pb(x);
 84
 85
            reverse(a.begin(), a.end());
reverse(b.begin(), b.end());
 86
 87
 88
             vector<int>ans=fft_multiply(a, b);
 89
             reverse(ans.begin(), ans.end());
 90
            bool flag=false;
for(int i=0; i<ans.size(); i++)</pre>
 91
 92
 93
 94
                  if(ans[i])
 95
                       flag=true;
                  if (flag)
 97
                       printf("%d", ans[i]);
 98
 99
            printf("\n");
100
            return 0;
101
```

5.4 Modular Math

- **5.4.1** Multiplicative Inverse
- 5.4.2 Linear All Multiplicative Inverse
- 5.5 Gaussian Elimination
- 5.6 Combinatorics

Geometry

- 6.1 2d Template
- 6.2 3d Template
- 6.3 Polygon Template
- 6.4 Convex Hull
- 6.4.1 Graham Scan
- **6.4.2** Monotone Chain
- 6.5 Rotating Calipers
- 6.6 KD Tree
- 6.7 Range Tree
- 6.8 Circle Sweep