

# Pira的模版库

Pira

November 13, 2013

## Contents

<b>1</b>	<b>String</b>	<b>2</b>
1.1	Extend KMP . . . . .	2
1.2	Manacher . . . . .	3
1.3	Aho-Corasick Automaton . . . . .	3
1.4	Suffix Array . . . . .	4
1.5	Suffix Tree . . . . .	6
1.6	Suffix Automaton . . . . .	8
<b>2</b>	<b>Data Structure</b>	<b>10</b>
2.1	Binary Index Tree . . . . .	10
2.2	Range Maximum Query . . . . .	11
2.3	Size Balanced Tree . . . . .	12
2.4	Splay Tree . . . . .	15
2.5	Path Decomposition . . . . .	18
2.6	Rectangle Tree . . . . .	22
2.7	Dancing Links Accurate Cover . . . . .	24
2.8	Dancing Links Complete Cover . . . . .	26
2.9	Persistent Data Structures . . . . .	28
	2.9.1 Interval k large . . . . .	28
	2.9.2 Interval k large on Tree . . . . .	30
2.10	Link Cut Tree . . . . .	33
2.11	Divide and Conquer on Tree . . . . .	37
<b>3</b>	<b>Dynamic Programming</b>	<b>40</b>
3.1	Multi Pack Optimization . . . . .	40
3.2	Matrix . . . . .	40
3.3	Slope Optimization . . . . .	41
3.4	Plug DP . . . . .	42

# 1 String

## 1.1 Extend KMP

```
1 #include <cstdio>
2 #include <cstring>
3 #include <iostream>
4 using namespace std;
5 #define REP(i,n) for(int i=0;i<n;i++)
6 #define FOR(i,l,r) for(int i=l;i<=r;i++)
7 #define N 1000010
8 int s[N], p[N];
9 int a[N], b[N];
10 int extend_kmp(int p[], int a[], int n, int s[], int b[], int m)
11 { // p 表示模式串
12     int k = 0;
13     FOR(i,1,n-1)
14     {
15         if( k>0 && a[i-k] < a[k] + k - i )
16             a[i] = a[i-k];
17         else
18         {
19             int j = k>0?a[k] + k - i : 0;
20             if(j<0)j = 0;
21             while( i+j<n && p[j] == p[i+j] ) j++;
22             a[ k = i ] = j;
23         }
24     }
25     k = 0;
26     REP(i,m)
27     {
28         if(k>0 && a[i-k] < b[k] + k - i )
29             b[i] = a[i-k];
30         else
31         {
32             int j = k>0?b[k]+k-i:0;
33             if(j<0)j=0;
34             while( i+j<m && p[j] == s[i+j] ) j++;
35             b[ k = i ] = j;
36             if(j>=n) return i+1;
37         }
38     }
39     return -1;
40 }
41 int main(int argc, char **argv)
42 {
43     int t, n, m;
44     scanf("%d",&t);
45     while(t--)
46     {
47         scanf("%d%d",&n,&m);
48         REP(i,n) scanf("%d",&s[i]);
49         REP(i,m) scanf("%d",&p[i]);
50         printf("%d\n",extend_kmp(p,a,m, s, b,n));
51     }
52     return 0;
53 }
```

## 1.2 Manacher

```
1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 using namespace std;
5 #define N 220010
6 char s[N], str[N];
7 int ans[N];
8 int Manacher(char src[], char tmp[], int len[])
9 {
10     int i, j, k, rightmost, n=strlen(src), ret=0;
11     tmp[0]='$';
12     tmp[1]='#';
13     for(i=0; i<n; ++i)
14     {
15         tmp[i*2+2]=src[i]; // i*2+2 为原串的 i 为中心的最长回文
16         tmp[i*2+3]='#';
17     }
18     n=n*2+2;
19     tmp[n]='\0';
20     for(i=0; i<n; ++i) len[i]=0;
21     rightmost=k=0;
22     for(i=1; i<n; ++i)
23     {
24         if(rightmost>=i) len[i]=min(len[2*k-i], rightmost-i);
25         else len[i]=0;
26         for(j=len[i]+1; tmp[i+j]==tmp[i-j]; ++j) ++len[i];
27         if(i+len[i]>rightmost)
28         {
29             rightmost=i+len[i];
30             k=i;
31         }
32         ret=max(ret, len[i]);
33     }
34     return ret;
35 }
36 int main()
37 {
38     while(~scanf("%s", str))
39         printf("%d\n", Manacher(str, s, ans));
40     return 0;
41 }
```

## 1.3 Aho-Corasick Automaton

```
1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <queue>
5 using namespace std;
6 #define REP(i, n) for(int i=0; i<n; i++)
7 #define FOR(i, l, r) for(int i=l; i<=r; i++)
8 #define DWN(i, r, l) for(int i=r; i>=l; i--)
9 #define N 1010
10 const int oo = 1e9;
```

```

11 struct node
12 {
13     node *fail , *go[4];
14     int val;
15 };
16 node mem[N], *root = mem, *cur;
17 int cg[333]; //字符转换规则, 记得初始化
18 node *new_node()
19 {
20     memset(cur->go, 0, sizeof cur->go);
21     cur->val = 0;
22     cur->fail = root;
23     return cur++;
24 }
25 void init()
26 {
27     cur = mem;
28     root = new_node();
29 }
30 void addword(char *s)
31 {
32     node *t = root;
33     for(; *s; s++)
34     {
35         int w = cg[(int)*s];
36         if(!t->go[w])
37             t->go[w] = new_node();
38         t = t->go[w];
39     }
40     t->val = 1;
41 }
42 queue<node*> q;
43 void build()
44 {
45     while(!q.empty()) q.pop();
46     REP(i, 4)
47         if(root->go[i]) q.push(root->go[i]);
48         else root->go[i] = root;
49     while(!q.empty())
50     {
51         node *t = q.front();
52         q.pop();
53         REP(i, 4)
54             if(t->go[i]) q.push(t->go[i]), t->go[i]->fail = t->fail->go[i];
55             else t->go[i] = t->fail->go[i];
56         t->val |= t->fail->val; //较短子串的值累加, 必须放在外面
57     }
58 }
59 int main(int argc, char *argv[])
60 {
61     return 0;
62 }

```

## 1.4 Suffix Array

```

1 #include <iostream>

```

```

2 #include <stdio>
3 #include <cstring>
4 using namespace std;
5 #define REP(i,n) for(int i=0;i<n;i++)
6 #define FOR(i,l,r) for(int i=l;i<=r;i++)
7 #define DWN(i,r,l) for(int i=r;i>=l;i--)
8 #define N 200010
9 struct suffixarray
10 {
11     int sa[N], rank[N], height[N];
12     int s[N], cnt[N];
13     int *x, *y, n, m;
14     void radixsort()
15     {
16         REP(i,n)s[i]=x[y[i]];
17         REP(i,m)cnt[i]=0;
18         REP(i,n)cnt[s[i]]++;
19         REP(i,m)cnt[i+1]+=cnt[i];
20         DWN(i,n-1,0)sa[--cnt[s[i]]]=y[i];
21     }
22     bool cmp(int a, int b, int l)
23     {
24         return y[a]==y[b] && y[a+l]==y[b+l];
25     }
26     void da(int *str, int len, int cset)
27     {
28         x = rank, y = height;
29         n = len, m = cset;
30         REP(i,n)x[i]=str[i], y[i]=i;
31         radixsort();
32         int j=1,p=0;
33         for (;p<n;j*=2,m=p+1)
34         {
35             p=0;
36             FOR(i,n-j,n-1)y[p++]=i;
37             REP(i,n)if(sa[i]>=j)y[p++]=sa[i]-j;
38             radixsort();
39             swap(x,y);
40             x[sa[0]]=p=1;
41             FOR(i,1,n-1)x[sa[i]]=cmp(sa[i],sa[i-1],j)?p:++p;
42         }
43         REP(i,n)rank[sa[i]]=i;
44         p=0;
45         REP(i,n)
46         {
47             if(p)p--;
48             if(rank[i])for(j=sa[rank[i]-1];str[i+p]==str[j+p];p++);
49             else p=0;
50             height[rank[i]]=p;
51         }
52     }
53 };
54 suffixarray sa;
55 int main(int argc, char *argv[])
56 {
57     return 0;
58 }

```

## 1.5 Suffix Tree

```
1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4 using namespace std;
5 typedef long long ll;
6 #define N 1010
7 #define oo 10000000000
8 struct node
9 {
10     node *parent, *link, *go[27];
11     int a, b, len;
12 };
13 node mem[N<<1], *root, *cur, *last;
14 node *newNode(int a = 0, int b = 0, int len = 0, node *parent = 0, node *link =
15     0)
16 {
17     memset( cur->go, 0, sizeof cur->go);
18     cur->a = a;
19     cur->b = b;
20     cur->len = len;
21     cur->parent = parent;
22     cur->link = link;
23     return cur++;
24 }
25 void init()
26 {
27     cur = mem;
28     last = root = newNode( 0, 0, 0, cur, cur);
29 }
30 node *find_lcp(node *p, int len, int pos, int type, char *s)
31 {
32     len -= p->len;
33     while( len>0 )
34     {
35         int c = s[ pos - len + 1 ] - 'a';
36         node *e = p->go[c];
37         if(!e) return p;
38         int l, i, j;
39         if( type )
40         {
41             i = e->a;
42             j = pos - len + 1;
43             l = 0;
44             while( s[i]== s[j] && i<= e->b) i++, j++, l++;
45         }
46         else l = min( e->b - e->a + 1, len);
47         if( e->a + l <= e->b )
48         {
49             node *q = newNode( e->a, e->a + l - 1, p->len + l, p);
50             p->go[c] = q;
51             q->go[ s[ e->a + l ] - 'a' ] = e;
52             e->parent = q;
53             e->a += l;
54             return q;
55         }
56     }
```

```

55         len -= 1, p = e;
56     }
57     return p;
58 }
59 void suffix_tree(char *s)
60 {
61     int n = strlen(s);
62     s[n] = 'a' + 26;
63     init();
64     for( int i = 0; i<=n; i++)
65     {
66         node *p = last->link;
67         last = p = find_lcp( p, n - i + 1, n, 1, s);
68         p->go[ s[ i + p->len ] - 'a' ] = newNode( i + p->len, n, n - i + 1, p);
69         while( !p->link )
70         {
71             p->link = find_lcp( p->parent->link, p->len - 1, p->b, 0, s);
72             p = p->link;
73         }
74     }
75 }
76 char s[N];
77 ll ans;
78 void dfs(node *rt, int &ma, int &mb)
79 {
80     int minl = oo, maxl = -oo, leaf = 1;
81     for(int i = 0; i<27; i++)
82         if( rt->go[i])
83         {
84             dfs( rt->go[i], minl, maxl);
85             leaf = 0;
86         }
87     ma = min( ma, minl);
88     mb = max( mb, maxl);
89     if( leaf )
90     {
91         ma = min( ma, rt->len);
92         mb = max( mb, rt->len);
93     }
94     else
95     {
96         int now = min( rt->len, maxl - minl) - rt->parent->len;
97         if( now>0 )ans += now;
98     }
99 }
100 int main(int argc, char *argv[])
101 {
102     freopen("a","r",stdin);
103     while( scanf("%s", s)!=EOF )
104     {
105         if( s[0]=='#' )break;
106         suffix_tree(s);
107         ans = 0;
108         int a, b;
109         dfs(root, a, b);
110         printf("%lld\n", ans);
111     }

```

```

112     return 0;
113 }

```

## 1.6 Suffix Automaton

```

1  #include <cstdio>
2  #include <cstring>
3  #include <iostream>
4  using namespace std;
5  #define N 250010
6  struct node
7  {
8      node *parent, *next, *go[26];
9      int len, sum;
10 };
11 node mem[N*2], *head[N], *root, *last, *cur;
12 node *new_node(int len=0)
13 {
14     memset(cur->go, 0, sizeof cur->go);
15     cur->len = len;
16     cur->next = head[len];
17     head[len] = cur;
18     return cur++;
19 }
20 void init()
21 {
22     memset(head, 0, sizeof head);
23     cur = mem;
24     root = last = new_node();
25 }
26 void extend(int w)
27 {
28     node *p = last;
29     node *np = new_node(p->len + 1);
30     while(p && !p->go[w])
31     {
32         p->go[w] = np;
33         p = p->parent;
34     }
35     if(!p) np->parent = root;
36     else
37     {
38         node *q = p->go[w];
39         if(p->len + 1 == q->len)
40             np->parent = q;
41         else
42         {
43             node *nq = new_node(p->len + 1);
44             memcpy(nq->go, q->go, sizeof q->go);
45             nq->parent = q->parent;
46             q->parent = nq;
47             np->parent = nq;
48             while(p && p->go[w] == q)
49             {
50                 p->go[w] = nq;
51                 p = p->parent;

```



```

52         }
53     }
54 }
55 last = np;
56 }
57 char s[N];
58 int ans[N];
59 int main(int argc , char **argv)
60 {
61     scanf("%s",s);
62     int len = strlen(s);
63     init();
64     for(char *c = s; *c; c++)
65         extend(*c - 'a');
66     node *t = root;
67     for(char *c = s;*c;c++)
68     {
69         t = t->go[*c - 'a'];
70         t->sum = 1;
71     }
72     memset(ans, 0, sizeof ans);
73     for(int i = len; i>0; i--)
74     {
75         for(t = head[i]; t; t = t->next)
76         {
77             ans[ t->len ] = max(ans[ t->len ], t->sum);
78             if(t->parent)
79                 t->parent->sum += t->sum;
80         }
81         ans[i] = max( ans[i] , ans[i+1]);
82     }
83     for(int i=1;i<=len;i++)
84         printf("%d\n",ans[i]);
85     return 0;
86 }

```

## 2 Data Structure

### 2.1 Binary Index Tree

```
1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 using namespace std;
5 #define N 1010
6 struct BinaryIndexTree
7 {
8     int w[N];
9     int n;
10    void init(int len)
11    {
12        n = len;
13        memset(w, 0, (n+1)*sizeof(int));
14    }
15    void insert(int x)
16    {
17        for(;x<=n; x += x&-x)w[x]++;
18    }
19    int query(int x)
20    {
21        int ret = 0;
22        for(;x>0; x-= x&-x)ret += w[x];
23        return ret;
24    }
25 };
26 struct BinaryIndexTree2D
27 {
28     int w[N][N];
29     int n,m;
30     void init(int lx, int ly)
31     {
32         n = lx, m = ly;
33         memset(w, 0, sizeof w);
34     }
35     void insert(int x, int y, int val)
36     {
37         for(int i = x; i<=n; i += i&-i)
38         {
39             for(int j = y; j<=m; j += j&-j)
40             {
41                 w[i][j] += val;
42             }
43         }
44     }
45     int query(int x, int y)
46     {
47         int ret = 0;
48         for(int i = x; i>0; i -= i&-i)
49         {
50             for(int j = y; j>0; j -= j&-j)
51             {
52                 ret += w[i][j];
53             }
54         }
55     }
56 }
```

```

54     }
55     return ret;
56 }
57 };
58 int main(int argc, char *argv[])
59 {
60     return 0;
61 }

```

## 2.2 Range Maximum Query

```

1 void ST()
2 {
3     int i, j, k;
4     for (j=1; (1<<j)<=n; ++j)
5         for (i=1; i+(1<<j)-1<=n; ++i)
6             {
7                 k=i+(1<<(j-1));
8                 f[i][j]=max(f[i][j-1], f[k][j-1]);
9             }
10 }
11 int Query(int l, int r)
12 {
13     int m=0;
14     while (l+(1<<m)<r-(1<<m)+1) ++m;
15     r=r-(1<<m)+1;
16     return max(f[l][m], f[r][m]);
17 }
18 // 二维RMQ
19 int log2(int x)
20 {
21     int k=0;
22     while ((1<<(k+1))<x) ++k;
23     return k;
24 }
25 void ST()
26 {
27     int i, j, u, v, logn=log2(n), logm=log2(m);
28     for (u=0; u<=logn; ++u)
29         for (v=0; v<=logm; ++v)
30             if (u+v) for (i=1; i+(1<<u)-1<=n; ++i)
31                 for (j=1; j+(1<<v)-1<=m; ++j)
32                     if (v==0) f[i][j][u][v]=max(f[i][j][u-1][v], f[i+(1<<(u-1))][j][u-1][v]);
33                     else f[i][j][u][v]=max(f[i][j][u][v-1], f[i][j+(1<<(v-1))][u][v-1]);
34 }
35 int get(int r1, int c1, int r2, int c2)
36 {
37     int k=log2(r2-r1+1), t=log2(c2-c1+1);
38     int a=max(f[r1][c1][k][t], f[r1][c2-(1<<t)+1][k][t]);
39     int b=max(f[r2-(1<<k)+1][c1][k][t], f[r2-(1<<k)+1][c2-(1<<t)+1][k][t]);
40     return max(a, b);
41 }

```

## 2.3 Size Balanced Tree

```
1  #include <stdio>
2  #include <iostream>
3  using namespace std;
4  #define N 200010
5  #define oo 1000000000
6
7  struct node
8  {
9      node *son[2];
10     int size, key;
11 };
12 node mem[N], *cur, *til=mem;
13 int pool[N], top;
14 node *newNode(int val)
15 {
16     node *ret = top? mem + pool[ --top ] : cur++;
17     ret->key = val;
18     ret->size = 1;
19     ret->son[0] = ret->son[1] = til;
20     return ret++;
21 }
22 void dump(node *x)
23 {
24     pool[ top++ ] = x->mem;
25 }
26 void initAll()
27 {
28     top = 0;
29     cur = mem;
30     til = newNode(0);
31     til->size = 0;
32 }
33
34
35 struct SizeBalancedTree
36 {
37     node *root;
38     SizeBalancedTree():root(til){}
39
40     void rotate(node *&x, int c)
41     {
42         node *y = x->son[!c];
43         x->son[!c] = y->son[c];
44         y->son[c] = x;
45         y->size = x->size;
46         x->size = x->son[0]->size + x->son[1]->size + 1;
47         x=y;
48     }
49     void maintain(node *&x, int c)
50     {
51         node *&y = x->son[c];
52         if( y->son[c]->size > x->son[!c]->size )
53         {
54             rotate( x , !c );
55         }
```

```

56     else if( y->son[!c]->size > x->son[!c]->size )
57     {
58         rotate( y , c );
59         rotate( x , !c );
60     }
61     else return;
62
63     maintain( x->son[0] , 0 );
64     maintain( x->son[1] , 1 );
65     maintain( x , 0 );
66     maintain( x , 1 );
67 }
68 void insert(int val)
69 {
70     insert( root , val );
71 }
72 void insert(node *&x, int val)
73 {
74     if( x == til )
75     {
76         x = newNode( val );
77     }
78     else
79     {
80         x->size++;
81         insert( x->son[ val >= x->key ] , val );
82         maintain( x , val >= x->key );
83     }
84 }
85 bool find(int val)
86 {
87     node *x = root;
88     do
89     {
90         if( val == x->key )
91         {
92             return true;
93         }
94         x = x->son[ val > x->key ];
95     }
96     while( x!= til );
97     return false;
98 }
99 void erase(int val)
100 {
101     if( find( val ) )
102     {
103         erase( root , val );
104     }
105 }
106 int erase(node *&x, int val)
107 {
108     x->size--;
109     node *&y = x->son[ val > x->key ];
110     if( val == x->key || y == til )
111     {
112         int ret = x->key;

```

```

113         if( y == til )
114         {
115             dump( x );
116             x = x->son[ val <= x->key ];
117         }
118         else
119         {
120             x->key = erase( y , val );
121         }
122         return ret;
123     }
124     else
125     {
126         return erase( y , val );
127     }
128 }
129 void pre_suc( int &pre , int &suc , int val )
130 {
131     node *x = root;
132     pre = -oo , suc = oo;
133     do
134     {
135         if( val == x->key )
136         {
137             pre = suc = val;
138             return ;
139         }
140         if( val > x->key )
141         {
142             pre = max( pre , x->key );
143         }
144         else
145         {
146             suc = min( suc , x->key );
147         }
148         x = x->son[ val >= x->key ];
149     }
150     while( x != til );
151 }
152 int below( int val )
153 {
154     node *x = root;
155     int ret = 0;
156     do
157     {
158         if( val >= x->key )
159         {
160             ret += x->son[0]->size + 1;
161         }
162         x = x->son[ val >= x->key ];
163     }
164     while( x != til );
165     return ret;
166 }
167 };
168 int main()
169 {

```

```

170     return 0;
171 }

```

## 2.4 Splay Tree

```

1  #include <cstdio>
2  #include <iostream>
3  using namespace std;
4  #define N 600010
5  #define oo 1000000000
6
7  struct mark
8  {
9      int mul, add;
10     mark(int m=1, int a=0): mul(m), add(a) {}
11     mark operator*(const mark &m)
12     {
13         return mark(mul*m.mul, add*m.mul+m.add);
14     }
15     bool ok() { return mul==1&&add==0; }
16 };
17 struct node
18 {
19     node *s[2], *p;
20     mark mk;
21     int rev;
22     int val, sum, sz;
23     int ms, lms, rms;
24     bool w() { return this==p->s[1]; }
25     void addIt(const mark &m)
26     {
27         mk = mk * m;
28         val = val*m.mul + m.add;
29         sum = sum*m.mul + m.add*sz;
30         ms = lms = rms = max(val, sum);
31     }
32     void revIt()
33     {
34         rev ^= 1;
35         swap(s[0], s[1]);
36         swap(lms, rms);
37     }
38     void up()
39     {
40         sz = s[0]->sz + s[1]->sz + 1;
41         sum = s[0]->sum + s[1]->sum + val;
42         ms = max(max(s[0]->ms, s[1]->ms), max(s[0]->rms, 0) + val + max(s[1]->lms, 0));
43         lms = max(s[0]->lms, s[0]->sum + val + max(s[1]->lms, 0));
44         rms = max(s[1]->rms, s[1]->sum + val + max(s[0]->rms, 0));
45     }
46     void down();
47     void sets(node *t, int w) { s[w]=t, t->p=this; }
48 };
49 node mem[N], *q[N], *root, *cur, *til = mem;
50 int a[N], top;
51 void node::down()

```

```

52 {
53     if (!mk.ok())
54     {
55         for (int i=0; i<2; i++) if (s[i] != til) s[i] -> addIt(mk);
56         mk = mark();
57     }
58     if (rev)
59     {
60         for (int i=0; i<2; i++) if (s[i] != til) s[i] -> revIt();
61         rev = 0;
62     }
63 }
64 node *new_node(int val=0)
65 {
66     node *c = top? q[--top] : cur++;
67     c->mk = mark();
68     c->rev = 0;
69     c->val = val;
70     c->p = c->s[0] = c->s[1] = til;
71     return c;
72 }
73 void erase(node *t)
74 {
75     if (t==til) return;
76     q[top++] = t;
77     erase(t->s[0]);
78     erase(t->s[1]);
79 }
80 void zig(node *t)
81 {
82     node *p = t->p;
83     p->down();
84     t->down();
85     int w = t->w();
86     p->p->sets(t, p->w());
87     p->sets(t->s[!w], w);
88     t->sets(p, !w);
89     p->up();
90 }
91 void splay(node *t, node *f=til)
92 {
93     for (t->down(); t->p != f;)
94     {
95         if (t->p->p == f) zig(t);
96         else if (t->w() == t->p->w()) zig(t->p), zig(t);
97         else zig(t), zig(t);
98     }
99     t->up();
100     if (f==til) root = t;
101 }
102 node *select(int k, node *f=til)
103 {
104     node *t = root;
105     int s;
106     t->down();
107     while ((s=t->s[0]->sz) != k)
108     {

```



```

109         if(k<s)t = t->s[0];
110         else k=s+1, t = t->s[1];
111         t->down();
112     }
113     splay(t,f);
114     return t;
115 }
116 node *build(int l, int r)
117 {
118     if(l>r)return til;
119     int m = (l+r)>>1;
120     node *t = new_node(a[m]);
121     t->sets(build(l,m-1),0);
122     t->sets(build(m+1,r),1);
123     t->up();
124     return t;
125 }
126 node *get(int l, int r)
127 {
128     if(l<1)l=1;
129     if(r>root->sz-2)r = root->sz-2;
130     return select(r+1,select(l-1))->s[0];
131 }
132 void getint(int &a)
133 {
134     char c;
135     while(((c=getchar())<'0' || c>'9')&& c!='-');
136     int flag = (c=='-')?-1:1;
137     if(c=='-')c=getchar();
138     for(a=0;c>='0' && c<='9';c=getchar())a = a*10 + c - '0';
139     a = a*flag;
140 }
141 void init(int n)
142 {
143     for(int i=1;i<=n;i++)getint(a[i]);
144     cur = mem;
145     top = 0;
146     til = new_node();
147     til->sz = til->sum = 0;
148     til->ms = til->lms = til->rms = -oo;
149     root = new_node();
150     root->sets(new_node(),1);
151     root->s[1]->sets(build(1,n),0);
152     splay(root->s[1]);
153 }
154 void Insert(int p, int n)
155 {
156     for(int i=1;i<=n;i++)getint(a[i]);
157     select(p+1,select(p))->sets(build(1,n),0);
158     splay(root->s[1]);
159 }
160 void Delete(int p, int n)
161 {
162     erase(select(p+n,select(p-1))->s[0]);
163     root->s[1]->s[0] = til;
164     splay(root->s[1]);
165 }

```

```

166 void Same(int p, int n)
167 {
168     int c;
169     scanf("%d",&c);
170     get(p,p+n-1)->addIt(mark(0,c));
171     splay(root->s[1]);
172 }
173 void Reverse(int p, int n)
174 {
175     get(p,p+n-1)->revIt();
176     splay(root->s[1]);
177 }
178 int Sum(int p, int n)
179 {
180     return get(p,p+n-1)->sum;
181 }
182 int Msum()
183 {
184     return get(1,root->sz-2)->ms;
185 }
186 int main()
187 {
188     int n,m,p,len;
189     char op[99];
190     getint(n),getint(m);
191     init(n);
192     while (m--)
193     {
194         scanf("%s",op);
195         if(op[2]!='X')getint(p),getint(len);
196         switch(op[2])
197         {
198             case 'S':
199                 Insert(p,len);
200                 break;
201             case 'L':
202                 Delete(p,len);
203                 break;
204             case 'K':
205                 Same(p,len);
206                 break;
207             case 'V':
208                 Reverse(p,len);
209                 break;
210             case 'T':
211                 printf("%d\n",Sum(p,len));
212                 break;
213             default:
214                 printf("%d\n",Msum());
215                 break;
216         }
217     }
218     return 0;
219 }

```

## 2.5 Path Decomposition

```

1  #include    <cstdio>
2  #include    <iostream>
3  #include    <vector>
4  using namespace std;
5  #define N 200010
6  #define oo 1000000000
7  /*****tree code*****/
8  vector<int> edge[N];
9  struct tree
10 {
11     int root , pos , deep , fa , top , mson , val , size ;
12 };
13 tree p[N];
14 int temp[N];
15 /*****segment tree*****/
16 struct node
17 {
18     node *ls , *rs ;
19     int ans , mark , lc , rc ;
20     node() {}
21     node(const node &a , const node &b)
22     {
23         ans=a.ans + b.ans - (a.rc == b.lc);
24         lc=a.lc ;
25         rc=b.rc ;
26     }
27     void up()
28     {
29         ans=ls->ans + rs->ans - (ls->rc == rs->lc);
30         lc=ls->lc ;
31         rc=rs->rc ;
32     }
33     void change(int c)
34     {
35         lc=rc=mark=c ;
36         ans=1;
37     }
38     void down()
39     {
40         if (mark>=0)
41         {
42             ls->change(mark);
43             rs->change(mark);
44             mark=-1;
45         }
46     }
47 };
48 node memPool[N<<2],*root[N],*cur;
49 int len[N],id;
50 void init()
51 {
52     id=0;
53     cur=memPool;
54 }
55 node *newNode(int c)
56 {
57     cur->change(c);

```

```

58     cur->mark=-1;
59     return cur++;
60 }
61 void build(int l, int r, node* &now)
62 {
63     if(l+1>=r)
64     {
65         now=newNode(temp[l]);
66         return;
67     }
68     now=newNode(0);
69     int m=(l+r)>>1;
70     build(l,m,now->ls);
71     build(m,r,now->rs);
72     now->up();
73 }
74 void update(int a, int b, int c, int l, int r, node *now)
75 {
76     if(a<=l && b+1>=r)
77     {
78         now->change(c);
79         return ;
80     }
81     int m=(l+r)>>1;
82     now->down();
83     if(a<m) update(a,b,c,l,m,now->ls);
84     if(b>=m) update(a,b,c,m,r,now->rs);
85     now->up();
86 }
87 node query(int a, int b, int l, int r, node *now)
88 {
89     if(a<=l && b+1>=r)
90     {
91         return *now;
92     }
93     int m=(l+r)>>1;
94     now->down();
95     if(b<m) return query(a,b,l,m,now->ls);
96     if(a>=m) return query(a,b,m,r,now->rs);
97     return node( query(a,b,l,m,now->ls), query(a,b,m,r,now->rs) );
98 }
99 /*****main part*****/
100 void chain(int x)
101 {
102     int &n=len[id];
103     int top=x;
104     while(x)
105     {
106         temp[n]=p[x].val;
107         p[x].pos=n++;
108         p[x].root=id;
109         p[x].top=top;
110         x=p[x].mson;
111     }
112     build(0,n,root[id++]);
113 }
114 void dfs(int u, int deep)

```

```

115 {
116     p[u].size=1;
117     p[u].deep=deep;
118     p[u].mson=0;
119     int n=edge[u].size();
120     int v,msize=0;
121     for(int i=0; i<n; i++)
122     {
123         if((v=edge[u][i])!=p[u].fa)
124         {
125             p[v].fa=u;
126             dfs(v,deep+1);
127             p[u].size+=p[v].size;
128             if(p[v].size>msize)
129             {
130                 if(msize)chain(p[u].mson);
131                 p[u].mson=v;
132                 msize=p[v].size;
133             }
134             else
135             {
136                 chain(v);
137             }
138         }
139     }
140 }
141 void change(int u, int v, int c)
142 {
143     while(p[u].top!=p[v].top)
144     {
145         if(p[p[u].top].deep<p[p[v].top].deep)
146         {
147             swap(u,v);
148         }
149         update(0,p[u].pos,c,0,len[p[u].root],root[p[u].root]);
150         u=p[p[u].top].fa;
151     }
152     if(p[u].pos>p[v].pos)
153     {
154         swap(u,v);
155     }
156     update(p[u].pos,p[v].pos,c,0,len[p[u].root],root[p[u].root]);
157 }
158 int answer(int u, int v)
159 {
160     int ret=0,a=-1,b=-1;
161     while(p[u].top!=p[v].top)
162     {
163         if(p[p[u].top].deep<p[p[v].top].deep)
164         {
165             swap(u,v);
166             swap(a,b);
167         }
168         node now=query(0,p[u].pos,0,len[p[u].root],root[p[u].root]);
169         ret+=now.ans - (now.rc==a);
170         a=now.lc;
171         u=p[p[u].top].fa;

```

```

172     }
173     if(p[u].pos>p[v].pos)
174     {
175         swap(u,v);
176         swap(a,b);
177     }
178     node now=query(p[u].pos,p[v].pos,0,len[ p[u].root ], root[ p[u].root ]);
179     ret+=now.ans - (now.lc==a) - (now.rc==b);
180     return ret;
181 }
182 int main(int argc , char *argv [])
183 {
184     int n,m,a,b,c;
185     char op[99];
186     scanf("%d%d",&n,&m);
187     for(int i = 1; i <= n; i++)
188     {
189         edge[i].clear();
190         scanf("%d",&p[i].val);
191     }
192     for(int i = 1; i < n; i++)
193     {
194         scanf("%d%d",&a,&b);
195         edge[a].push_back(b);
196         edge[b].push_back(a);
197     }
198     init();
199     dfs(1,1);
200     chain(1);
201     while(m--)
202     {
203         scanf("%s%d%d",op,&a,&b);
204         if(op[0]=='C')
205         {
206             scanf("%d",&c);
207             change(a,b,c);
208         }
209         else
210         {
211             printf("%d\r\n",answer(a,b));
212         }
213     }
214     return 0;
215 }

```

## 2.6 Rectangle Tree

```

1 //询问大平面里维护少数点，询问矩形内信息
2 struct retange
3 {
4     int x1,y1,x2,y2;
5     retange(){}
6     retange(int x1, int y1, int x2, int y2):x1(x1),y1(y1),x2(x2),y2(y2){}
7     bool cover(const retange &a)const
8     {
9         if(x1>a.x1||x2<a.x2||y1>a.y1||y2<a.y2)return 0;

```

```

10         return 1;
11     }
12     bool interset(const retange &a) const
13     {
14         if (x1>a.x2 || y1>a.y2 || a.x1>x2 || a.y1>y2) return 0;
15         return 1;
16     }
17     bool ok()
18     {
19         return x1>=x2&& y1>=y2;
20     }
21     int mx()
22     {
23         return (x1+x2)>>1;
24     }
25     int my()
26     {
27         return (y1+y2)>>1;
28     }
29     void rd(char *s)
30     {
31         sscanf(s,"%d%d%d%d",&x1,&x2,&y1,&y2);
32     }
33 };
34 struct Node
35 {
36     retange r;
37     Node *son[4];
38     LL sum;
39 } TNode,* nil=&TNode;
40 Node mem[N],* C=mem;
41 Node* Create(retange r)
42 {
43     C->r=r;
44     C->sum=0;
45     REP(i,4) C->son[i]=nil;
46     return C++;
47 }
48 void Update(int x, int y, int n, Node *now)
49 {
50     now->sum+=n;
51     while(1)
52     {
53         int mx=now->r.mx();
54         int my=now->r.my();
55         if(x<=mx)
56         {
57             if(y<=my)
58             {
59                 if(now->son[0]==nil)
60                     now->son[0]=Create(retange(now->r.x1,now->r.y1,mx,my));
61                 now=now->son[0];
62             }
63             else
64             {
65                 if(now->son[1]==nil)
66                     now->son[1]=Create(retange(now->r.x1,my+1,mx,now->r.y2));

```

```

67         now=now->son[1];
68     }
69 }
70 else
71 {
72     if(y<=my)
73     {
74         if(now->son[2]==nil)
75             now->son[2]=Create(retange(mx+1,now->r.y1,now->r.x2,my));
76         now=now->son[2];
77     }
78     else
79     {
80         if(now->son[3]==nil)
81             now->son[3]=Create(retange(mx+1,my+1,now->r.x2,now->r.y2));
82         now=now->son[3];
83     }
84 }
85 now->sum+=n;
86 if(now->r.ok()) break;
87 }
88 }
89 LL Query(retange r, Node *now)
90 {
91     if(r.cover(now->r)) return now->sum;
92     LL ret=0;
93     REP(i,4)
94         if(now->son[i]!=nil&&now->son[i]->r.interset(r))
95             ret+=Query(r,now->son[i]);
96     return ret;
97 }

```

## 2.7 Dancing Links Accurate Cover

```

1  #include<stdio>
2  #define mm 555555
3  #define mn 999
4  int U[mm],D[mm],L[mm],R[mm],C[mm];
5  int H[mm],S[mm];
6  int id[33][33];
7  int n,m,p,size,T,ans;
8  void remove(int c)
9  {
10     L[R[c]]=L[c],R[L[c]]=R[c];
11     for(int i=D[c];i!=c;i=D[i])
12         for(int j=R[i];j!=i;j=R[j])
13             U[D[j]]=U[j],D[U[j]]=D[j],--S[C[j]];
14 }
15 void resume(int c)
16 {
17     L[R[c]]=R[L[c]]=c;
18     for(int i=U[c];i!=c;i=U[i])
19         for(int j=L[i];j!=i;j=L[j])
20             ++S[C[U[D[j]]=D[U[j]]=j]];
21 }
22 void Dance(int k)

```



```

23 {
24     if (k >= ans) return;
25     if (!R[0])
26     {
27         ans = k;
28         return;
29     }
30     int i, j, tmp, c;
31     for (tmp = mm, i = R[0]; i; i = R[i])
32         if (S[i] < tmp) tmp = S[c = i];
33     remove(c);
34     for (i = D[c]; i != c; i = D[i])
35     {
36         for (j = R[i]; j != i; j = R[j]) remove(C[j]);
37         Dance(k + 1);
38         for (j = L[i]; j != i; j = L[j]) resume(C[j]);
39     }
40     resume(c);
41 }
42 void Link(int r, int c)
43 {
44     ++S[C[++size] = c];
45     D[size] = D[c];
46     U[D[c]] = size;
47     U[size] = c;
48     D[c] = size;
49     if (H[r] < 0) H[r] = L[size] = R[size] = size;
50     else
51     {
52         R[size] = R[H[r]];
53         L[R[H[r]]] = size;
54         L[size] = H[r];
55         R[H[r]] = size;
56     }
57 }
58 void prepare(int r, int c)
59 {
60     for (int i = 0; i <= c; ++i)
61     {
62         S[i] = 0;
63         U[i] = D[i] = i;
64         R[i] = i + 1;
65         L[i + 1] = i;
66     }
67     R[c] = 0;
68     while (r) H[r--] = -1;
69 }
70 int main()
71 {
72     int i, j, k, x1, y1, x2, y2;
73     scanf("%d", &T);
74     while (T--)
75     {
76         scanf("%d%d%d", &n, &m, &p);
77         for (size = i = 0; i < n; ++i)
78             for (j = 0; j < m; ++j) id[i][j] = ++size;
79         prepare(p, size);

```

```

80         for (k=1;k<=p;++k)
81         {
82             scanf ("%d%d%d%d", &x1, &y1, &x2, &y2);
83             for (i=x1; i<x2; ++i)
84                 for (j=y1; j<y2; ++j) Link(k, id[i][j]);
85         }
86         ans=mm;
87         Dance(0);
88         if (ans<mm) printf ("%d\n", ans);
89         else puts ("-1");
90     }
91     return 0;
92 }

```

## 2.8 Dancing Links Complete Cover

```

1  #include <stdio>
2  #define mm 55555
3  #define mn 333
4  int U[mm], D[mm], L[mm], R[mm], C[mm];
5  int H[mm], S[mm], id[22][22];
6  bool v[mm];
7  int n, m, size, ans;
8  void prepare(int r, int c)
9  {
10     for (int i=0; i<=c; ++i)
11     {
12         S[i]=0;
13         D[i]=U[i]=i;
14         L[i+1]=i;
15         R[i]=i+1;
16     }
17     R[c]=0;
18     while (r) H[r--]=-1;
19 }
20 int f()
21 {
22     int i, j, c, ret=0;
23     for (c=R[0]; c; c=R[c]) v[c]=1;
24     for (c=R[0]; c; c=R[c])
25         if (v[c]) for (v[c]=0, ++ret, i=D[c]; i!=c; i=D[i])
26             for (j=R[i]; j!=i; j=R[j]) v[C[j]]=0;
27     return ret;
28 }
29 void remove(int c)
30 {
31     for (int i=D[c]; i!=c; i=D[i])
32         R[L[i]]=R[i], L[R[i]]=L[i];
33 }
34 void resume(int c)
35 {
36     for (int i=U[c]; i!=c; i=U[i])
37         R[L[i]]=L[R[i]]=i;
38 }
39 void Dance(int k)
40 {

```

```

41     if (k+f()>=ans) return ;
42     if (!R[0])
43     {
44         ans=k;
45         return ;
46     }
47     int i , j , c , tmp=mm;
48     for ( i=R[0] ; i ; i=R[ i ])
49         if (S[ i ]<tmp) tmp=S[ c=i ] ;
50     for ( i=D[ c ] ; i!=c ; i=D[ i ])
51     {
52         remove( i ) ;
53         for ( j=R[ i ] ; j!=i ; j=R[ j ]) remove( j ) ;
54         Dance( k+1 ) ;
55         for ( j=L[ i ] ; j!=i ; j=L[ j ]) resume( j ) ;
56         resume( i ) ;
57     }
58 }
59 void Link( int r , int c )
60 {
61     ++S[ C[++size]=c ] ;
62     D[ size]=D[ c ] ;
63     U[ D[ c ] ] = size ;
64     U[ size]=c ;
65     D[ c]=size ;
66     if (H[r]<0) H[r]=L[ size]=R[ size]=size ;
67     else
68     {
69         R[ size]=R[ H[ r ] ] ;
70         L[ R[ H[ r ] ] ] = size ;
71         L[ size]=H[ r ] ;
72         R[ H[ r ] ] = size ;
73     }
74 }
75 int main()
76 {
77     int i , j , k , l , mr , mc , r ;
78     while ( scanf ( "%d%d" , &n , &m ) != -1 )
79     {
80         for ( size=0 , i=1 ; i<=n ; ++i )
81             for ( j=1 ; j<=m ; ++j )
82             {
83                 scanf ( "%d" , &k ) ;
84                 id [ i ] [ j ] =(k)?++size : 0 ;
85             }
86         scanf ( "%d%d" , &mr , &mc ) ;
87         prepare ( n*m , size ) ;
88         for ( i=r=1 ; i<=n-mr+1 ; ++i )
89             for ( j=1 ; j<=m-mc+1 ; ++j , ++r )
90                 for ( k=i ; k<i+mr ; ++k )
91                     for ( l=j ; l<j+mc ; ++l )
92                         if ( id [ k ] [ l ] ) Link ( r , id [ k ] [ l ] ) ;
93         ans=n*m ;
94         Dance ( 0 ) ;
95         printf ( "%d\n" , ans ) ;
96     }
97     return 0 ;

```

## 2.9 Persistent Data Structures

### 2.9.1 Interval k large

```

1 struct node
2 {
3     node *ls , *rs;
4     int num;
5 };
6 node mem[N*11] , *root [N] , *cur , *til = mem;
7 node *newNode()
8 {
9     cur->num = 0;
10    cur->ls = cur->rs = til;
11    return cur++;
12 }
13 void init ()
14 {
15     cur = mem;
16     til = newNode();
17 }
18 void insert(node *x, node *&y, int l, int r, int p)
19 {
20     y = newNode();
21     y->num = x->num + 1;
22     int m = (l+r)>>1;
23     if( l+1>=r )return;
24     if( p<m )
25     {
26         y->rs = x->rs;
27         insert(x->ls , y->ls , l , m, p);
28     }
29     else
30     {
31         y->ls = x->ls;
32         insert(x->rs , y->rs , m , r , p);
33     }
34 }
35 void update(node *&rt , int l, int r, int p, int val)
36 {
37     if( rt==til )rt = newNode();
38     rt->num += val;
39     int m = (l+r)>>1;
40     if( l+1>=r )return;
41     if( p<m )
42         update( rt->ls , l , m, p, val);
43     else
44         update( rt->rs , m , r , p, val);
45 }
46 struct answer
47 {
48     node *t [N];
49     int n;
50     answer() :n(0) {}
51     void push(node *now){ t [n++] = now; }
```

```

52     void left() { REP(i,n) t[i] = t[i]->ls; }
53     void right() { REP(i,n) t[i] = t[i]->rs; }
54     int value() { int ret = 0; REP(i,n) ret += t[i]->ls->num; return ret; }
55 };
56 struct command
57 {
58     int o, l, r, k;
59     void read()
60     {
61         char op[9];
62         scanf("%s%d%d", op, &l, &r);
63         o = op[0] == 'Q';
64         if(o) scanf("%d", &k);
65     }
66 };
67 struct BinaryIndexTree
68 {
69     node *t[N];
70     int n, m;
71     void init(int la, int lb)
72     {
73         n = la, m = lb;
74         REP(i, n+1) t[i] = newNode();
75     }
76     void insert(int x, int p, int val)
77     {
78         for(; x <= n; x += x & -x) update(t[x], 0, m, p, val);
79     }
80     void query(int x, answer &ans)
81     {
82         for(; x > 0; x -= x & -x) ans.push(t[x]);
83     }
84 };
85 BinaryIndexTree bit;
86 int query(int l, int r, int k, int n)
87 {
88     answer x, y;
89     bit.query(l-1, x);
90     bit.query(r, y);
91     x.push(root[l-1]);
92     y.push(root[r]);
93     l = 0, r = n;
94     while( l+1 < r )
95     {
96         int m = (l+r) >> 1;
97         int sum = y.value() - x.value();
98         if( k <= sum ) r = m, x.left(), y.left();
99         else k -= sum, l = m, x.right(), y.right();
100     }
101     return l;
102 }
103 command cmd[N];
104 int a[N], b[N];
105 int main(int argc, char *argv[])
106 {
107     int t, n, m, w;
108     scanf("%d", &t);

```

```

109     while( t— )
110     {
111         scanf("%d%d",&n,&m);
112         w = 0;
113         FOR(i,1,n)
114         {
115             scanf("%d",&a[i]);
116             b[w++] = a[i];
117         }
118         REP(i,m)
119         {
120             cmd[i].read();
121             if(!cmd[i].o)b[w++] = cmd[i].r;
122         }
123         sort( b, b+w);
124         w = unique( b, b+w) - b;
125         init();
126         bit.init( n, w);
127         root[0] = newNode();
128         FOR(i,1,n)
129         {
130             a[i] = lower_bound( b, b+w, a[i]) - b;
131             insert( root[i-1], root[i], 0, w, a[i]);
132         }
133         REP(i,m)
134         {
135             if( cmd[i].o )
136                 printf("%d\n",b[query( cmd[i].l, cmd[i].r, cmd[i].k, w)]);
137             else
138             {
139                 bit.insert( cmd[i].l, a[ cmd[i].l ], -1);
140                 a[ cmd[i].l ] = lower_bound( b, b+w, cmd[i].r) - b;
141                 bit.insert( cmd[i].l, a[ cmd[i].l ], 1);
142             }
143         }
144     }
145     return 0;
146 }

```

### 2.9.2 Interval k large on Tree

```

1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
4  #include <vector>
5  #include <algorithm>
6  using namespace std;
7  #define REP(i,n) for(int i=0;i<n;i++)
8  #define FOR(i,l,r) for(int i=l;i<=r;i++)
9  #define DWN(i,r,l) for(int i=r;i>=l;i--)
10 #define N 200010
11 struct node
12 {
13     node *ls, *rs;
14     int sz;
15 };

```

```

16 node mem[N*20], *rt[N], *cur;
17 node *new_node()
18 {
19     cur->ls = cur->rs = mem;
20     cur->sz = 0;
21     return cur++;
22 }
23 void init()
24 {
25     cur = mem;
26     new_node();
27     rt[0] = new_node();
28 }
29 void update(node *x, node *&y, int l, int r, int p)
30 {
31     y = new_node();
32     y->sz = x->sz + 1;
33     if(l+l>=r) return;
34     int m = (l+r)>>1;
35     if(p<m)
36     {
37         y->rs = x->rs;
38         update(x->ls, y->ls, l, m, p);
39     }
40     else
41     {
42         y->ls = x->ls;
43         update(x->rs, y->rs, m, r, p);
44     }
45 }
46 int query(node *x, node *y, node *a, node *b, int l, int r, int k)
47 {
48     if(l+l>=r) return l;
49     int m = (l+r)>>1, now = y->ls->sz + x->ls->sz - a->ls->sz - b->ls->sz;
50     if(now>=k)
51         return query(x->ls, y->ls, a->ls, b->ls, l, m, k);
52     return query(x->rs, y->rs, a->rs, b->rs, m, r, k-now);
53 }
54 vector<int> e[N];
55 int q[N], d[N], dep[N], pos[N], fa[N], a[N], b[N], id[N*2], nds;
56 int f[N][20];
57 void Simulation_dfs(int u, int n, int w)
58 {
59     int v, r = 0;
60     init();
61     update(rt[0], rt[u], 0, w, a[u]);
62     FOR(i, 1, n) d[i] = e[i].size() - 1;
63     fa[q[++r] = u] = 0;
64     dep[u] = 1;
65     nds = 0;
66     while(r)
67     {
68         u = q[r];
69         id[pos[u] = nds++] = u;
70         while(d[u]>=0)
71             if((v = e[u][d[u]--]) != fa[u])
72                 {

```

```

73         fa[ q[++r] = v ] =u;
74         dep[v] = dep[u] + 1;
75         update( rt[u], rt[v], 0, w, a[v] );
76         goto end;
77     }
78     r--;
79     end::;
80 }
81 }
82 void init_rmq()
83 {
84     REP(i, nds) f[i][0] = id[i];
85     for(int j = 1; (1<<j)<nds; ++j)
86         REP(i, nds-(1<<j)+1)
87         {
88             int k = i + (1<<(j-1));
89             if(dep[ f[i][j-1] ]<dep[ f[k][j-1] ])
90                 f[i][j] = f[i][j-1];
91             else f[i][j] = f[k][j-1];
92         }
93 }
94 int rmq(int l, int r)
95 {
96     if(l>r) swap(l, r);
97     int m = 0;
98     while( l+(1<<m)<r-(1<<m)+1 ) m++;
99     r = r-(1<<m)+1;
100    if( dep[ f[l][m] ]<dep[ f[r][m] ] ) return f[l][m];
101    return f[r][m];
102 }
103 int main(int argc, char **argv)
104 {
105     int n, m, w, u, v, k;
106     scanf("%d%d", &n, &m);
107     FOR(i, 1, n)
108     {
109         scanf("%d", &a[i]);
110         b[i-1] = a[i];
111         e[i].clear();
112     }
113     REP(i, n-1)
114     {
115         scanf("%d%d", &u, &v);
116         e[u].push_back(v);
117         e[v].push_back(u);
118     }
119     sort(b, b+n);
120     w = unique(b, b+n) - b;
121     FOR(i, 1, n) a[i] = lower_bound(b, b+w, a[i]) - b;
122     Simulation_dfs(1, n, w);
123     init_rmq();
124     REP(i, m)
125     {
126         scanf("%d%d%d", &u, &v, &k);
127         int lca = rmq(pos[u], pos[v]);
128         printf("%d\n", b[query(rt[u], rt[v], rt[lca], rt[fa[lca]], 0, w, k)]);
129     }

```



```

130     return 0;
131 }

```

## 2.10 Link Cut Tree

```

1  #include<cstdio>
2  #include<algorithm>
3  #include<cstring>
4  #include<vector>
5  #define REP(i,n) for(int i=0;i<n;++i)
6  using namespace std;
7
8  //Dynamic Tree
9
10 typedef long long int64;
11 const int MOD = 51061;
12 const int MAXN = int(1e5) + 10;
13
14 struct Mark {
15     int64 add, mul; //x*mul+add
16     Mark(int64 add, int64 mul) {
17         this->add = add;
18         this->mul = mul;
19     }
20     Mark() {
21         mul = 1;
22         add = 0;
23     }
24     bool isId() {
25         return mul == 1 && add == 0;
26     }
27 };
28
29 Mark operator*(Mark a, Mark b) {
30     return Mark((a.add * b.mul + b.add) % MOD, a.mul * b.mul % MOD);
31 }
32
33 struct Node {
34     Node*p, *ch[2];
35     bool rev;
36     Mark m;
37     int64 sum, val;
38     int size;
39     bool isRoot;
40     Node*fa;
41     Node() {
42         sum = 0;
43         isRoot = 0;
44         size = 0;
45     }
46     void sc(Node*c, int d) {
47         ch[d] = c;
48         c->p = this;
49     }
50     bool d() {
51         return this == p->ch[1];

```

```

52     }
53     void upd() {
54         sum = (val + ch[0]->sum + ch[1]->sum) % MOD;
55         size = 1 + ch[0]->size + ch[1]->size;
56     }
57     void apply(Mark a) {
58         m = m * a;
59         sum = (sum * a.mul + a.add * size) % MOD;
60         val = (val * a.mul + a.add) % MOD;
61     }
62     void revIt() {
63         rev ^= 1;
64         swap(ch[0], ch[1]);
65     }
66     void relax();
67     void setRoot(Node*f);
68 } Tnull, *null = &Tnull;
69
70 void Node::setRoot(Node*f) {
71     fa = f;
72     isRoot = true;
73     p = null;
74 }
75
76 void Node::relax() {
77     if (!m.isId()) {
78         REP(i, 2)
79             if (ch[i] != null)
80                 ch[i]->apply(m);
81         m = Mark();
82     }
83     if (rev) {
84         REP(i, 2)
85             if (ch[i] != null)
86                 ch[i]->revIt();
87         rev = 0;
88     }
89 }
90
91 Node mem[MAXN], *C = mem;
92
93 Node*make(int v) {
94     C->sum = C->val = v;
95     C->rev = 0;
96     C->m = Mark();
97     C->ch[0] = C->ch[1] = null;
98     C->isRoot = true;
99     C->p = null;
100    C->fa = null;
101    return C++;
102 }
103
104 void rot(Node*t) {
105     Node*p = t->p;
106     p->relax();
107     t->relax();
108     bool d = t->d();

```

```

109     p->p->sc(t, p->d());
110     p->sc(t->ch[!d], d);
111     t->sc(p, !d);
112     p->upd();
113     if (p->isRoot) {
114         p->isRoot = false;
115         t->isRoot = true;
116         t->fa = p->fa;
117     }
118 }
119
120 void pushTo(Node*t) {
121     static Node*stk[MAXN];
122     int top = 0;
123     while (t != null) {
124         stk[top++] = t;
125         t = t->p;
126     }
127     for (int i = top - 1; i >= 0; --i)
128         stk[i]->relax();
129 }
130
131 void splay(Node*u, Node*f = null) {
132     pushTo(u);
133     while (u->p != f) {
134         if (u->p->p == f)
135             rot(u);
136         else
137             u->d() == u->p->d() ? (rot(u->p), rot(u)) : (rot(u), rot(u));
138     }
139     u->upd();
140 }
141
142 Node*v[MAXN];
143 vector<int> E[MAXN];
144 int n, nQ;
145
146 int que[MAXN], fa[MAXN], qh = 0, qt = 0;
147
148 void bfs() {
149     que[qt++] = 0;
150     fa[0] = -1;
151     while (qh < qt) {
152         int u = que[qh++];
153         for (vector<int>::iterator e = E[u].begin(); e != E[u].end(); ++e)
154             if (*e != fa[u])
155                 fa[*e] = u, v[*e]->fa = v[u], que[qt++] = *e;
156     }
157 }
158
159 Node* expose(Node*u) {
160     Node*v;
161     for (v = null; u != null; v = u, u = u->fa) {
162         splay(u);
163         u->ch[1]->setRoot(u);
164         u->sc(v, 1);
165         v->fa = u;

```

```

166     }
167     return v;
168 }
169
170 void makeRoot(Node*u) {
171     expose(u);
172     splay(u);
173     u->revIt();
174 }
175
176 void addEdge(Node*u, Node*v) {
177     makeRoot(v);
178     v->fa = u;
179 }
180
181 void delEdge(Node*u, Node*v) {
182     makeRoot(u);
183     expose(v);
184     splay(u);
185     u->sc(null, 1);
186     u->upd();
187     v->fa = null;
188     v->isRoot = true;
189     v->p = null;
190 }
191
192 void markPath(Node*u, Node*v, Mark m) {
193     makeRoot(u);
194     expose(v);
195     splay(v);
196     v->apply(m);
197 }
198
199 int queryPath(Node*u, Node*v) {
200     makeRoot(u);
201     expose(v);
202     splay(v);
203     return v->sum;
204 }
205
206 int main() {
207     scanf("%d%d", &n, &nQ);
208     REP(i, n-1) {
209         int u, v;
210         scanf("%d%d", &u, &v);
211         —u, —v;
212         E[u].push_back(v);
213         E[v].push_back(u);
214     }
215     REP(i, n)
216         v[i] = make(1);
217     bfs();
218     REP(i, nQ) {
219         char cmd;
220         scanf("%c", &cmd);
221         scanf("%c", &cmd);
222         int i, j;

```

```

223     scanf("%d%d", &i, &j);
224     Node*u = ::v[--i], *v = ::v[--j];
225     if (cmd == '+') {
226         int c;
227         scanf("%d", &c);
228         markPath(u, v, Mark(c, 1));
229     } else if (cmd == '*') {
230         int c;
231         scanf("%d", &c);
232         markPath(u, v, Mark(0, c));
233     } else if (cmd == '/') {
234         printf("%d\n", queryPath(u, v));
235     } else {
236         int k, l;
237         scanf("%d%d", &k, &l);
238         delEdge(u, v);
239         addEdge(::v[--k], ::v[--l]);
240     }
241 }
242 }

```

## 2.11 Divide and Conquer on Tree

```

1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
4  #include <algorithm>
5  using namespace std;
6  #define REP(i,n) for(int i=0;i<n;i++)
7  #define FOR(i,l,r) for(int i=l;i<=r;i++)
8  #define DWN(i,r,l) for(int i=r;i>=l;i--)
9  #define N 20010
10 int head[N], ver[N], cost[N], next[N];
11 int dis[N], del[N], sz[N], q[N], fa[N];
12 int K, ans, edge;
13 void init(int n)
14 {
15     REP(i,n+1)head[i] = 0, del[i] = 0;
16     ans = 0, edge = 2;
17 }
18 void addedge(int u, int v, int c)
19 {
20     ver[edge] = v, cost[edge]= c, next[edge] =head[u], head[u] = edge++;
21     ver[edge] = u, cost[edge]= c, next[edge] =head[v], head[v] = edge++;
22 }
23 int findroot(int u)
24 {
25     int v, r = 0, root = u;
26     fa[ q[r++] = u ] = 0;
27     REP(l,r)
28         for(int e = head[u=q[l]];e;e=next[e])
29             if(!del[v=ver[e]] && v != fa[u])
30                 fa[ q[r++] = v ] = u;
31     int msz = 1e9;
32     DWN(i,r-1,0)
33     {

```

```

34     sz[ u = q[i] ] = 1;
35     int now = 0;
36     for(int e = head[u]; e; e=next[e])
37         if(!del[v=ver[e]] && v != fa[u])
38             sz[u] += sz[v], now = max(now, sz[v]);
39     now = max(now, r - sz[u]);
40     if(now<msz)msz = now, root = u;
41 }
42 return root;
43 }
44 int counts(int s, int t)
45 {
46     int ret = 0;
47     while(s<t)
48     {
49         while(s<t && dis[s]+dis[t]>K)t--;
50         ret += t-s++;
51     }
52     return ret;
53 }
54 int recover(int u, int s, int d)
55 {
56     if(d>K)return 0;
57     int v, r = 0;
58     fa[ q[r++] = u ] = 0;
59     dis[s] = d;
60     REP(1, r)
61         for(int e = head[u=q[l]]; e; e=next[e])
62             if(!del[v=ver[e]] && v != fa[u])
63             {
64                 dis[s+r] = dis[s+l] + cost[e];
65                 if(dis[s+r]>K)continue;
66                 fa[ q[r++] = v ] = u;
67             }
68     sort(dis+s, dis+s+r);
69     return r;
70 }
71 int dfs(int u, int s, int d)
72 {
73     int v, root = findroot(u);
74     int n, tot = 1;
75     del[root] = 1;
76     for(int e = head[root]; e; e=next[e])
77         if(!del[v=ver[e]])
78         {
79             n = dfs(v, s+tot, cost[e]);
80             ans -= counts(s+tot, s+tot+n-1);
81             tot += n;
82         }
83     dis[s] = 0;
84
85     sort(dis+s, dis+s+tot);
86     ans += counts(s, s+tot-1);
87     del[root] = 0;
88     return recover(u, s, d);
89 }
90 void getint(int &a)

```

```

91 {
92     char c;
93     while(!isdigit(c=getchar()));
94     for(a=0;isdigit(c);c=getchar())a = a*10 + c-'0';
95 }
96 int main(int argc, char **argv)
97 {
98     freopen("a","r",stdin);
99     int n, a, b, c;
100    while( scanf("%d%d",&n,&K)!=EOF)
101    {
102        if(n==0&&K==0)break;
103        init(n);
104        REP(i,n-1)
105        {
106            getint(a);
107            getint(b);
108            getint(c);
109            addedge(a,b,c);
110        }
111        dfs(1,0,0);
112        printf("%d\n",ans);
113    }
114    return 0;
115 }

```

## 3 Dynamic Programming

### 3.1 Multi Pack Optimization

```
1 void multi_pack(int s, int v, int c, int n)
2 {
3     REP(i, c)
4     {
5         int l = 0, r = -1, m = (n+c-1)/c;
6         FOR(j, 0, m)
7         {
8             int p = j*c + i;
9             int now = f1[p] - j*v;
10            while(l<=r && w[r]<= now) r--;
11            w[++r] = now;
12            q[r] = j;
13            if(j - q[l]>s) l++;
14            f[p] = max(f[p], w[l] + j*v);
15        }
16    }
17 }
```

### 3.2 Matrix

```
1 点稀疏算法:
2
3 struct point
4 {
5     int x, y;
6 }map[maxn];
7 int i, j, k, n, L, W, high, low, best, maxl;
8 inline bool cmp(point a, point b)
9 {
10     return a.x<b.x || (a.x==b.x && a.y<b.y);
11 }
12 inline void max(int a)
13 {
14     if(a>best) best=a;
15 }
16 int main()
17 {
18     while(scanf("%d%d", &L, &W) != EOF)
19     {
20         scanf("%d", &n);
21         for(i=0; i<n; ++i) scanf("%d%d", &map[i].x, &map[i].y);
22         map[n].x=map[n].y=0;
23         map[++n].x=L, map[n].y=W;
24         map[++n].x=L, map[n].y=0;
25         map[++n].x=0, map[n].y=W;
26         sort(map, map+n, cmp);
27         for(best=i=0; i<n; ++i)
28         {
29             for(low=0, high=W, maxl=L-map[i].x, j=i+1; j<n; ++j)
30                 if(low<=map[j].y && map[j].y<=high)
31                 {
32                     if(maxl*(high-low)<=best) break;
```



```

33         max((map[j].x-map[i].x)*(high-low));
34         if(map[j].y==map[i].y) break;
35         if(map[j].y>map[i].y) high=map[j].y; else low=map[j].y;
36     }
37     for(low=0,high=W,maxl=map[i].x,j=i-1;j>=0;--j)
38         if(low<=map[j].y&&map[j].y<=high)
39         {
40             if(maxl*(high-low)<=best) break;
41             max((map[i].x-map[j].x)*(high-low));
42             if(map[j].y==map[i].y) break;
43             if(map[j].y>map[i].y) high=map[j].y; else low=map[j].y;
44         }
45     }
46     printf("%d/n",best);
47 }
48 }点密集格点算法:
49
50
51 int l[maxn],r[maxn],h[maxn],s[maxn][maxn]={0},now,ans,lm,rm,i,j,k,n,m;
52 int main()
53 {
54     scanf("%d%d",&n,&m);
55     for(i=0;i<=m;++i) h[i]=0,l[i]=1,r[i]=m;
56     for(ans=0,i=1;i<=n;++i)
57     {
58         for(k=lm=0,j=1;j<=m;++j)
59         {
60             scanf("%d",&now),k+=now,s[i][j]=s[i-1][j]+k;
61             if(now)
62             {
63                 h[j]=h[j]+1;
64                 if(lm>l[j]) l[j]=lm;
65             }
66             else h[j]=0,l[j]=1,r[j]=m,lm=j+1;
67         }
68         for(j=rm=m;j>0;--j)
69             if(h[j])
70             {
71                 if(r[j]>rm) r[j]=rm;
72                 now=s[i][r[j]]+s[i-h[j]][l[j]-1]-s[i-h[j]][r[j]]-s[i][l[j]-1];
73                 if(now>ans) ans=now;
74             }
75             else rm=j-1;
76     }
77     printf("%d/n",ans);
78 }

```

### 3.3 Slope Optimization

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 using namespace std;
5 #define REP(i,n) for(int i=0;i<n;i++)
6 #define FOR(i,l,r) for(int i=l;i<=r;i++)
7 #define N 1010

```

```

8  int f[N][N];
9  int a[N], s[N], ss[N], q[N];
10 #define GX(i) (s[i-1]) //对应第 i 个点的 x 坐标
11 #define GY(i,j) (f[i-1][j-1]+ss[i-1]) //对应第 i 个点的 y 坐标
12 #define G(i,j,k) (GY(i,j) - (k)*GX(i)) //对应第 i 个点斜率为 k 的值
13 double slope(int p, int q, int j) // p q 两点的斜率
14 {
15     return (double)(GY(q,j)-GY(p,j))/(GX(q)-GX(p));
16 }
17 int main(int argc, char **argv)
18 {
19     int n, m;
20     while (scanf("%d%d", &n, &m) != EOF)
21     {
22         if (n==0 && m==0) break;
23         m++;
24         ss[0] = s[0] = 0;
25         FOR(i, 1, n)
26         {
27             scanf("%d", &a[i]);
28             s[i] = s[i-1] + a[i];
29             ss[i] = ss[i-1] + a[i]*s[i];
30         }
31         REP(i, n+1)
32             REP(j, m+1)
33                 f[i][j] = 1e9;
34         f[0][0] = 0;
35         FOR(j, 1, m)
36         {
37             int l = 0, r = -1;
38             FOR(i, 1, n)
39             {
40                 while (l < r && slope(q[r-1], q[r], j) > slope(q[r], i, j)) r--;
41                 q[++r] = i;
42                 while (l < r && G(q[l], j, s[i]) >= G(q[l+1], j, s[i])) l++;
43                 f[i][j] = G(q[l], j, s[i]) + s[i-1]*s[i] - ss[i-1];
44             }
45         }
46         printf("%d\n", f[n][m]);
47     }
48     return 0;
49 }

```

### 3.4 Plug DP

```

1  //一条回路问题
2  #include <cstdio>
3  #include <cstring>
4  using namespace std;
5  const int mm=15511;
6  typedef long long LL;
7  struct hashTable
8  {
9      int h[mm], s[mm], p[mm], t;
10     LL v[mm];
11     void insert(int w, LL val)

```

```

12     {
13         int i, id=w%mm;
14         for ( i=h[id]; i>=0; i=p[i] )
15             if ( s[i]==w )
16                 {
17                     v[i]+=val;
18                     return;
19                 }
20         v[t]=val, s[t]=w, p[t]=h[id], h[id]=t++;
21     }
22 void clear()
23 {
24     t=0, memset(h, -1, sizeof(h));
25 }
26 } f[2];
27 bool g[22][22];
28 int i, j, k, n, m, em, g1, g2;
29 bool ok(int s)
30 {
31     if (s==1) return g[i+1][j];
32     if (s==2) return g[i][j+1];
33     if (s==3) return g[i+1][j]&&g[i][j+1];
34 }
35 int Link(int s, bool flag)
36 {
37     int n=1, w, x=3<<(j<<1), a=(flag?1:2)<<(j<<1);
38     while(n)
39     {
40         if (flag) a<=2, x<=2;
41         else a>=2, x>=2;
42         w=s&x;
43         if (w) n+=(w==a)?1:-1;
44     }
45     return s^x;
46 }
47 void Work(int s, LL val)
48 {
49     int e, w=j<<1, ss=(s>>w)&15;
50     if (ss==9) return;
51     if (!ss)
52     {
53         if (ok(3)) f[g2].insert(s|(9<<w), val);
54     }
55     else if (!(ss&3) || !(ss&12))
56     {
57         if (ss&3) e=1, ss|=ss<<2;
58         else e=0, ss|=ss>>2;
59         if (ok(1+!e)) f[g2].insert(s, val);
60         if (ok(1+e)) f[g2].insert(s^(ss<<w), val);
61     }
62     else if (ss==6) f[g2].insert(s^(ss<<w), val);
63     else f[g2].insert(Link(s^(ss<<w), ss==5), val);
64 }
65 void end()
66 {
67     while(n--)
68         for (em=m-1; em>=0; --em)

```

```

69         if (g[n][em]) return;
70     }
71 LL PlugDP()
72 {
73     end();
74     f[0].clear();
75     f[0].insert(0,1);
76     for (g2=i=0;i<=n;++i)
77     {
78         for (k=0;k<f[g2].t;++k) f[g2].s[k]<=2;
79         for (j=0;j<m;++j)
80             if (g[i][j]) for (g1=g2, g2=!g2, f[g2].clear(), k=0;k<f[g1].t;++k)
81             {
82                 if (i==n&&j==em)
83                     if (((f[g1].s[k]>>(j<<1))&15)==9) return f[g1].v[k];
84                     else continue;
85                 Work(f[g1].s[k], f[g1].v[k]);
86             }
87     }
88     return 0;
89 }
90 char c;
91 int main()
92 {
93     scanf("%d%d",&n,&m);
94     memset(g,0,sizeof(g));
95     for (i=0;i<n;++i)
96         for (j=0;j<m;++j)
97             scanf("%c",&c), g[i][j]=(c==' ');
98     printf("%I64d\n",PlugDP());
99     return 0;
100 }
101 //一条路径问题
102 #include<stdio>
103 #include<cstring>
104 #define mm 100007
105 #define mm 11
106 #define getRP(a,b) ((a)<<((b)<<1))
107 struct hash
108 {
109     int h[mm], s[mm], p[mm], d[mm], t;
110     int push(int x, int v)
111     {
112         int i, c=x%mm;
113         for (i=h[c]; i>=0; i=p[i])
114             if (s[i]==x)
115             {
116                 if (v>d[i]) d[i]=v;
117                 return i;
118             }
119         d[t]=v, s[t]=x, p[t]=h[c], h[c]=t;
120         return t++;
121     }
122     void clear()
123     {
124         t=0;
125         memset(h,-1,sizeof(h));

```

```

126     }
127 } f[2];
128 int g[mm][mm];
129 int i, j, k, g1, g2, x, y, z, s, n, m, ans;
130 int eat(bool f, bool l)
131 {
132     int a=getRP(z, j+f), b=getRP(3^z, j+f), c=getRP(3, j+f), n=1;
133     s=s^getRP(x, j)^getRP(y, j+1);
134     while(n)
135     {
136         if(f) a<=&=2, b<=&=2, c<=&=2;
137         else a>=&=2, b>=&=2, c>=&=2;
138         x=s&c;
139         if(x==a) ++n;
140         if(x==b) --n;
141     }
142     return l?(s|c):((s^b)|a);
143 }
144 bool ok(int c)
145 {
146     if(c==1) return g[i+1][j];
147     if(c==2) return g[i][j+1];
148     if(c==3) return g[i+1][j]&&g[i][j+1];
149     return 0;
150 }
151 void move(int v)
152 {
153     int w=v+g[i][j];
154     if(!x&&!y)
155     {
156         if(ok(1)) f[g2].push(s|getRP(3, j), w);
157         if(ok(2)) f[g2].push(s|getRP(3, j+1), w);
158         if(ok(3)) f[g2].push(s|getRP(9, j), w);
159         f[g2].push(s, v);
160     }
161     else if(!x||!y)
162     {
163         z=x+y;
164         if(ok(x?1:2)) f[g2].push(s, w);
165         if(ok(x?2:1)) f[g2].push(s^getRP(z, j)^getRP(z, j+1), w);
166         if(z<3) f[g2].push(eat(z==1, 1), w);
167         else if((s^getRP(x, j)^getRP(y, j+1))==0&&w>ans) ans=w;
168     }
169     else if(x==y)
170     {
171         if((z=x)<3) f[g2].push(eat(z==1, 0), w);
172         else if((s^getRP(x, j)^getRP(y, j+1))==0&&w>ans) ans=w;
173     }
174     else if(x>2||y>2)
175     {
176         z=x<y?x:y;
177         f[g2].push(eat(z==1, 1), w);
178     }
179     else if(x==2&&y==1) f[g2].push(s^getRP(x, j)^getRP(y, j+1), w);
180 }
181 int PlugDP()
182 {

```

```

183     f[0].clear();
184     f[0].push(0,0);
185     for(g1=1,g2=i=0;i<n;++i)
186     {
187         for(j=0;j<f[g2].t;++j)f[g2].s[j]<=2;
188         for(j=0;j<m;++j)
189             if(g[i][j])for(g1!=g1,g2!=g2,f[g2].clear(),k=0;k<f[g1].t;++k)
190             {
191                 s=f[g1].s[k],x=(s>>(j<<1))&3,y=(s>>((j+1)<<1))&3;
192                 move(f[g1].d[k]);
193             }
194     }
195     return ans;
196 }
197 int main()
198 {
199     int t;
200     scanf("%d",&t);
201     while(t--)
202     {
203         scanf("%d%d",&n,&m);
204         memset(g,0,sizeof(g));
205         for(ans=i=0;i<n;++i)
206             for(j=0;j<m;++j)
207             {
208                 scanf("%d",&g[i][j]);
209                 if(g[i][j]>ans)ans=g[i][j];
210             }
211         printf("%d\n",PlugDP());
212     }
213     return 0;
214 }

```