

Platelet

Team Reference Material

凌皓煜 陈彤 顾逸

2018

Contents

1	Graph Theory	5
1.1	2-SAT	5
1.2	双连通分量	5
1.2.1	点双连通分量	5
1.2.2	边双连通分量	5
1.3	K 短路 (lhy)	5
1.4	最大团	9
1.5	一般图最大匹配	9
1.6	树	9
1.6.1	虚树	9
1.6.2	矩阵树定理	9
1.6.3	点分治	9
1.6.4	Prufer 编码	9
1.6.5	Link-Cut Tree (ct)	9
1.6.6	树上倍增	11
1.6.7	数链剖分	11
1.7	仙人掌	11
1.8	带花树	11
1.9	KM 算法	11
1.10	支配树	11
1.10.1	DAG	11
1.10.2	一般图	11
1.11	弦图	11
1.12	网络流	11
1.13	最小割	11
1.14	最大流	11
1.15	费用流	11
1.16	有上下界的网络流 (Durandal)	11
1.16.1	zkw 费用流	12
1.17	差分约束	12
2	Math	13
2.1	int64 相乘取模 (Durandal)	13
2.2	扩展欧几里得 (gy)	13
2.3	中国剩余定理 (Durandal)	14
2.4	线性同余不等式 (Durandal)	14
2.5	组合数	14
2.6	高斯消元	14
2.7	Miller Rabin & Pollard Rho (gy)	14
2.8	$O(m^2 \log n)$ 线性递推	16
2.9	Polynomial	16
2.9.1	FFT	16
2.9.2	NTT & 多项式求逆	16
2.10	拉格朗日插值	16

2.11	杜教筛	16
2.12	BSGS	16
2.12.1	BSGS	16
2.12.2	扩展 BSGS	16
2.13	直线下整点个数 (gy)	16
2.14	单纯形	17
2.15	辛普森积分	17
2.16	常用数列定理 (gy)	17
3	Geometry	18
3.1	点、直线、圆 (gy)	18
3.2	点到凸包切线	22
3.3	直线凸包交点	22
3.4	凸包游戏	22
3.5	半平面交	22
3.6	旋转卡壳	22
3.7	判断圆是否有交	22
3.8	最小圆覆盖	22
3.9	最小球覆盖	22
3.10	$O(n^2 \log n)$ 圆交面积和重心	22
3.11	圆与多边形交	22
3.12	$O(n \log n)$ 凸多边形内的最大圆	22
3.13	三角形的五心	22
3.14	三维凸包	22
3.15	三维绕轴旋转	22
3.16	几何公式	22
4	String	23
4.1	KMP	23
4.2	AC 自动机	23
4.3	后缀数组	23
4.4	后缀自动机	23
4.5	Manacher	23
4.6	回文自动机	23
4.7	最小表示法	23
5	Data Structure	24
5.1	莫队 (ct)	24
5.2	ST 表 (ct)	24
5.3	可并堆 (ct)	25
5.4	zkw 线段树 (ct)	25
5.5	主席树	26
5.6	Splay (ct)	26
5.7	Treap (ct)	30
5.8	可持久化平衡树 (ct)	32
5.9	CDQ 分治 (ct)	34
5.10	Bitset (ct)	35
6	Others	37
6.1	vimrc (gy)	37
6.2	STL 释放内存 (Durandal)	37
6.3	开栈 (Durandal)	37
6.4	Java Template (gy)	38
6.5	Big Fraction (gy)	40
6.6	模拟退火 (ct)	40
6.7	三分 (ct)	41
6.8	博弈论模型 (gy)	41

6.9 积分表	42
-------------------	----

Chapter 1

Graph Theory

1.1 2-SAT

1.2 双连通分量

1.2.1 点双连通分量

1.2.2 边双连通分量

1.3 K 短路 (lhy)

```
1  const int MAXNODE = MAXN + MAXM * 2;
2
3  bool used[MAXN];
4  int n, m, cnt, S, T, Kth, N, TT;
5  int rt[MAXN], seq[MAXN], adj[MAXN], from[MAXN], dep[MAXN];
6  LL dist[MAXN], w[MAXM], ans[MAXK];
7
8  struct GivenEdge{
9      int u, v, w;
10     GivenEdge() {}
11     GivenEdge(int _u, int _v, int _w) : u(_u), v(_v), w(_w){};
12 }edge[MAXM];
13
14 struct Edge{
15     int v, nxt, w;
16     Edge() {}
17     Edge(int _v, int _nxt, int _w) : v(_v), nxt(_nxt), w(_w) {};
18 }e[MAXM];
19
20 inline void addedge(int u, int v, int w)
21 {
22     e[++cnt] = Edge(v, adj[u], w); adj[u] = cnt;
23 }
24
25 void dij(int S)
26 {
27     for(int i = 1; i <= N; i++)
28     {
29         dist[i] = INF;
30         dep[i] = 0x3f3f3f3f;
31         used[i] = false;
32         from[i] = 0;
33     }
```

```

29 static priority_queue<pair<LL, int>, vector<pair<LL, int> >, greater<pair<LL, int> > > hp;
30 while(!hp.empty())hp.pop();
31 hp.push(make_pair(dist[S] = 0, S));
32 dep[S] = 1;
33 while(!hp.empty())
34 {
35     pair<LL, int> now = hp.top();
36     hp.pop();
37     int u = now.second;
38     if(used[u])continue;
39     else used[u] = true;
40     for(int p = adj[u]; p; p = e[p].nxt)
41     {
42         int v = e[p].v;
43         if(dist[u] + e[p].w < dist[v])
44         {
45             dist[v] = dist[u] + e[p].w;
46             dep[v] = dep[u] + 1;
47             from[v] = p;
48             hp.push(make_pair(dist[v], v));
49         }
50     }
51 }
52 for(int i = 1; i <= m; i++) w[i] = 0;
53 for(int i = 1; i <= N; i++)
54     if(from[i])w[from[i]] = -1;
55 for(int i = 1; i <= m; i++)
56 {
57     if(~w[i] && dist[edge[i].u] < INF && dist[edge[i].v] < INF)
58     {
59         w[i] = -dist[edge[i].u] + (dist[edge[i].v] + edge[i].w);
60     }
61     else
62     {
63         w[i] = -1;
64     }
65 }
66 }

67 inline bool cmp_dep(int p, int q)
68 {
69     return dep[p] < dep[q];
70 }

71 struct Heap{
72     LL key;
73     int id, lc, rc, dist;
74     Heap() {}
75     Heap(LL k, int i, int l, int r, int d) : key(k), id(i), lc(l), rc(r), dist(d) {};
76     inline void clear()
77     {
78         key = 0;
79         id = lc = rc = dist = 0;
80     }
81 }hp[MAXNODE];

82 inline int merge_simple(int u, int v)
83 {
84     if(!u)return v;
85     if(!v)return u;
86     if(hp[u].key > hp[v].key)

```

```

87     {
88         swap(u, v);
89     }
90     hp[u].rc = merge_simple(hp[u].rc, v);
91     if(hp[hp[u].lc].dist < hp[hp[u].rc].dist)
92     {
93         swap(hp[u].lc, hp[u].rc);
94     }
95     hp[u].dist = hp[hp[u].rc].dist + 1;
96     return u;
97 }

98 inline int merge_full(int u, int v)
99 {
100     if(!u) return v;
101     if(!v) return u;
102     if(hp[u].key > hp[v].key)
103     {
104         swap(u, v);
105     }
106     int nownode = ++cnt;
107     hp[nownode] = hp[u];
108     hp[nownode].rc = merge_full(hp[nownode].rc, v);
109     if(hp[hp[nownode].lc].dist < hp[hp[nownode].rc].dist)
110     {
111         swap(hp[nownode].lc, hp[nownode].rc);
112     }
113     hp[nownode].dist = hp[hp[nownode].rc].dist + 1;
114     return nownode;
115 }

116 priority_queue<pair<LL, int>, vector<pair<LL, int> >, greater<pair<LL, int> > > Q;

117 int main()
118 {
119     while(scanf("%d%d", &n, &m) != EOF)
120     {
121         scanf("%d%d%d", &S, &T, &Kth, &TT);
122         for(int i = 1; i <= m; i++)
123         {
124             int u, v, w;
125             scanf("%d%d%d", &u, &v, &w);
126             edge[i] = {u, v, w};
127         }
128         N = n;
129         memset(adj, 0, sizeof(*adj) * (N + 1));
130         cnt = 0;
131         for(int i = 1; i <= m; i++)
132             addedge(edge[i].v, edge[i].u, edge[i].w);
133         dijkstra(T);
134         if(dist[S] > TT)
135         {
136             puts("Whitesnake!");
137             continue;
138         }
139         for(int i = 1; i <= N; i++)
140             seq[i] = i;
141         sort(seq + 1, seq + N + 1, cmp_dep);

142         cnt = 0;
143         memset(adj, 0, sizeof(*adj) * (N + 1));

```

```

144     memset(rt, 0, sizeof(*rt) * (N + 1));
145     for(int i = 1; i <= m; i++)
146         addedge(edge[i].u, edge[i].v, edge[i].w);
147     rt[T] = cnt = 0;
148     hp[0].dist = -1;
149     for(int i = 1; i <= N; i++)
150     {
151         int u = seq[i], v = edge[from[u]].v;
152         rt[u] = 0;
153         for(int p = adj[u]; p; p = e[p].nxt)
154         {
155             if(-w[p])
156             {
157                 hp[++cnt] = Heap(w[p], p, 0, 0, 0);
158                 rt[u] = merge_simple(rt[u], cnt);
159             }
160         }
161         if(i == 1) continue;
162         rt[u] = merge_full(rt[u], rt[v]);
163     }
164     while(!Q.empty()) Q.pop();
165     Q.push(make_pair(dist[S], 0));
166     edge[0].v = S;
167     for(int kth = 1; kth <= Kth; kth++)
168     {
169         if(Q.empty())
170         {
171             ans[kth] = -1;
172             continue;
173         }
174         pair<LL, int> now = Q.top(); Q.pop();
175         ans[kth] = now.first;
176         int p = now.second;
177         if(hp[p].lc)
178         {
179             Q.push(make_pair(+hp[hp[p].lc].key + now.first - hp[p].key, hp[p].lc));
180         }
181         if(hp[p].rc)
182         {
183             Q.push(make_pair(+hp[hp[p].rc].key + now.first - hp[p].key, hp[p].rc));
184         }
185         if(rt[edge[hp[p].id].v])
186         {
187             Q.push(make_pair(hp[rt[edge[hp[p].id].v]].key + now.first, rt[edge[hp[p].id].v]));
188         }
189     }
190     if(ans[Kth] == -1 || ans[Kth] > TT)
191     {
192         puts("Whitesnake!");
193     }
194     else
195     {
196         puts("Yareyaredawa");
197     }
198 }
199 }

```


1.4 最大团

1.5 一般图最大匹配

1.6 树

1.6.1 虚树

1.6.2 矩阵树定理

1.6.3 点分治

1.6.4 Prufer 编码

1.6.5 Link-Cut Tree (ct)

```

1 struct Node *null;
2 struct Node {
3     Node *ch[2], *fa, *pos;
4     int val, mn, l, len; bool rev;
5     // min_val in chain
6     inline bool type()
7     {
8         return fa -> ch[1] == this;
9     }
10    inline bool check()
11    {
12        return fa -> ch[type()] == this;
13    }
14    inline void pushup()
15    {
16        pos = this; mn = val;
17        ch[0] -> mn < mn ? mn = ch[0] -> mn, pos = ch[0] -> pos : 0;
18        ch[1] -> mn < mn ? mn = ch[1] -> mn, pos = ch[1] -> pos : 0;
19        len = ch[0] -> len + ch[1] -> len + 1;
20    }
21    inline void pushdown()
22    {
23        if (rev)
24        {
25            ch[0] -> rev ^= 1;
26            ch[1] -> rev ^= 1;
27            std::swap(ch[0], ch[1]);
28            rev ^= 1;
29        }
30    }
31    inline void pushdownall()
32    {
33        if (check()) fa -> pushdownall();
34        pushdown();
35    }
36    inline void rotate()
37    {
38        bool d = type(); Node *f = fa, *gf = f -> fa;
39        (fa = gf, f -> check()) ? fa -> ch[f -> type()] = this : 0;
40        (f -> ch[d] = ch[!d]) != null ? ch[!d] -> fa = f : 0;
41        (ch[!d] = f) -> fa = this;
42        f -> pushup();
43    }
44    inline void splay(bool need = 1)

```

```

45 {
46     if (need) pushdownall();
47     for (; check(); rotate())
48         if (fa -> check())
49             (type() == fa -> type() ? fa : this) -> rotate();
50     pushup();
51 }
52 inline Node *access()
53 {
54     Node *i = this, *j = null;
55     for (; i != null; i = (j = i) -> fa)
56     {
57         i -> splay();
58         i -> ch[1] = j;
59         i -> pushup();
60     }
61     return j;
62 }
63 inline void make_root()
64 {
65     access();
66     splay();
67     rev ^= 1;
68 }
69 inline void link(Node *that)
70 {
71     make_root();
72     fa = that;
73     splay(0);
74 }
75 inline void cut(Node *that)
76 {
77     make_root();
78     that -> access();
79     that -> splay(0);
80     that -> ch[0] = fa = null;
81     that -> pushup();
82 }
83 } mem[maxn];
84 inline Node *query(Node *a, Node *b)
85 {
86     a -> make_root(); b -> access(); b -> splay(0);
87     return b -> pos;
88 }
89 inline int dist(Node *a, Node *b)
90 {
91     a -> make_root(); b -> access(); b -> splay(0);
92     return b -> len;
93 }

```

1.6.6 树上倍增

1.6.7 数链剖分

1.7 仙人掌

1.8 带花树

1.9 KM 算法

1.10 支配树

1.10.1 DAG

1.10.2 一般图

1.11 弦图

1.12 网络流

1.13 最小割

1.14 最大流

1.15 费用流

1.16 有上下界的网络流 (Durandal)

$B(u, v)$ 表示边 (u, v) 流量的下界, $C(u, v)$ 表示边 (u, v) 流量的上界, 设 $F(u, v)$ 表示边 (u, v) 的实际流量
 设 $G(u, v) = F(u, v) - B(u, v)$, 则 $0 \leq G(u, v) \leq C(u, v) - B(u, v)$

- 无源汇的上下界可行流
 建立超级源点 S^* 和超级汇点 T^* , 对于原图每一条边 (u, v) 在新网络中连如下三条边: $S^* \rightarrow v$, 容量为 $B(u, v)$; $u \rightarrow T^*$, 容量为 $B(u, v)$; $u \rightarrow v$, 容量为 $C(u, v) - B(u, v)$ 。最后求新网络的最大流, 判断从超级源点 S^* 出发的边是否都满流即可, 边 (u, v) 的最终解中的实际流量为 $G(u, v) + B(u, v)$ 。
- 有源汇的上下界可行流
 从汇点 T 到源点 S 连一条上界为 ∞ , 下界为 0 的边。按照无源汇的上下界可行流一样做即可, 流量即为 $T \rightarrow S$ 边上的流量。
- 有源汇的上下界最大流
 - 在有源汇的上下界可行流中, 从汇点 T 到源点 S 的边改为连一条上界为 ∞ , 下界为 x 的边。 x 满足二分性质, 找到最大的 x 使得新网络存在有源汇的上下界可行流即为原图的最大流。
 - 从汇点 T 到源点 S 连一条上界为 ∞ , 下界为 0 的边, 变成无源汇的网络。按照无源汇的上下界可行流的方法, 建立超级源点 S^* 与超级汇点 T^* , 求一遍 $S^* \rightarrow T^*$ 的最大流, 再将从汇点 T 到源点 S 的这条边拆掉, 求一次 $S \rightarrow T$ 的最大流即可。
- 有源汇的上下界最小流
 - 在有源汇的上下界可行流中, 从汇点 T 到源点 S 的边改为连一条上界为 x , 下界为 0 的边。 x 满足二分性质, 找到最小的 x 使得新网络存在有源汇的上下界可行流即为原图的最大流。
 - 按照无源汇的上下界可行流的方法, 建立超级源点 S^* 与超级汇点 T^* , 求一遍 $S^* \rightarrow T^*$ 的最大流, 但是注意不加上汇点 T 到源点 S 的这条边, 即不使之改为无源汇的网络去求解。求完后, 再加上那条汇点 T 到源点 S 的边, 上界为 ∞ 的边。因为这条边的下界为 0, 所以 S^*, T^* 无影响, 再求

一次 $S^* \rightarrow T^*$ 的最大流。若超级源点 S^* 出发的边全部满流，则 $T \rightarrow S$ 边上的流量即为原图的最小流，否则无解。

1.16.1 zkw 费用流

1.17 差分约束

Chapter 2

Math

2.1 int64 相乘取模 (Durandal)

```
1 int64_t mul(int64_t x, int64_t y, int64_t p) {
2     int64_t t = (x * y - (int64_t) ((long double) x / p * y + 1e-3) * p) % p;
3     return t < 0 ? t + p : t;
4 }
```

2.2 扩展欧几里得 (gy)

```
1 // return gcd(a, b)
2 // ax+by=gcd(a,b)
3 int extend_gcd(int a, int b, int &x, int &y) {
4     if (b == 0) {
5         x = 1, y = 0;
6         return a;
7     }
8     int res = extend_gcd(b, a % b, x, y);
9     int t = y;
10    y = x - a / b * y;
11    x = t;
12    return res;
13 }
14 // return minimal positive integer x so that ax+by=c
15 // or -1 if such x does not exist
16 int solve_equ(int a, int b, int c) {
17     int x, y, d;
18     d = extend_gcd(a, b, x, y);
19     if (c % d)
20         return -1;
21     int t = c / d;
22     x *= t;
23     y *= t;
24     int k = b / d;
25     x = (x % k + k) % k;
26     return x;
27 }
28 // return minimal positive integer x so that ax==b(mod p)
29 // or -1 if such x does not exist
30 int solve(int a, int b, int p) {
31     a = (a % p + p) % p;
32     b = (b % p + p) % p;
```

```

33     return solve_equ(a, p, b);
34 }

```

2.3 中国剩余定理 (Durandal)

返回是否可行, 余数和模数结果为 r_1, m_1

```

1 bool CRT(int &r1, int &m1, int r2, int m2) {
2     int x, y, g = extend_gcd(m1, m2, x, y);
3     if ((r2 - r1) % g != 0) return false;
4     x = 1ll * (r2 - r1) * x % m2;
5     if (x < 0) x += m2;
6     x /= g;
7     r1 += m1 * x;
8     m1 *= m2 / g;
9     return true;
10 }

```

2.4 线性同余不等式 (Durandal)

必须满足 $0 \leq d < m, 0 \leq l \leq r < m$, 返回 $\min\{x \geq 0 \mid l \leq x \cdot d \bmod m \leq r\}$, 无解返回 -1

```

1 int64_t calc(int64_t d, int64_t m, int64_t l, int64_t r) {
2     if (l == 0) return 0;
3     if (d == 0) return -1;
4     if (d * 2 > m) return calc(m - d, m, m - r, m - l);
5     if ((l - 1) / d < r / d) return (l - 1) / d + 1;
6     int64_t k = calc((-m % d + d) % d, d, l % d, r % d);
7     if (k == -1) return -1;
8     return (k * m + l - 1) / d + 1;
9 }

```

2.5 组合数

2.6 高斯消元

2.7 Miller Rabin & Pollard Rho (gy)

```

1 /*
2  * In Java, use BigInteger.isProbablePrime(int certainty) to replace miller_rabin(BigInteger
   ↪ number)
3  * Test Set / First Wrong Answer
4  * 2 / 2,047
5  * 2, 3 / 1,373,653
6  * 31, 73 / 9,080,191
7  * 2, 3, 5 / 25,326,001
8  * 2, 3, 5, 7 / 3,215,031,751 (> Int.MAX_VALUE)
9  * 2, 7, 61 / 4,759,123,141
10 * 2, 13, 23, 1662803 / 1,122,004,669,633
11 * 2, 3, 5, 7, 11 / 2,152,302,898,747
12 * 2, 3, 5, 7, 11, 13 / 3,474,749,660,383
13 * 2, 3, 5, 7, 11, 13, 17 / 341,550,071,728,321
14 * 2, 3, 5, 7, 11, 13, 17, 19, 23 / 3,825,123,056,546,413,051
15 * 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 / 318,665,857,834,031,151,167,461 (> Long.MAX_VALUE)
16 * 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41 / 3,317,044,064,679,887,385,961,981

```

```

17  */
18  const int test_case_size = 12;
19  const int test_cases[test_case_size] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

20  int64_t multiply_mod(int64_t x, int64_t y, int64_t p) {
21      int64_t t = (x * y - (int64_t) ((long double) x / p * y + 1e-3) * p) % p;
22      return t < 0 ? t + p : t;
23  }

24  int64_t add_mod(int64_t x, int64_t y, int64_t p) {
25      return (0ull + x + y) % p;
26  }

27  int64_t power_mod(int64_t x, int64_t exp, int64_t p) {
28      int64_t ans = 1;
29      while (exp) {
30          if (exp & 1)
31              ans = multiply_mod(ans, x, p);
32          x = multiply_mod(x, x, p);
33          exp >>= 1;
34      }
35      return ans;
36  }

37  bool miller_rabin_check(int64_t prime, int64_t base) {
38      int64_t number = prime - 1;
39      for (; ~number & 1; number >>= 1)
40          continue;
41      int64_t result = power_mod(base, number, prime);
42      for (; number != prime - 1 && result != 1 && result != prime - 1; number <<= 1)
43          result = multiply_mod(result, result, prime);
44      return result == prime - 1 || (number & 1) == 1;
45  }

46  bool miller_rabin(int64_t number) {
47      if (number < 2)
48          return false;
49      if (number < 4)
50          return true;
51      if (~number & 1)
52          return false;
53      for (int i = 0; i < test_case_size && test_cases[i] < number; i++)
54          if (!miller_rabin_check(number, test_cases[i]))
55              return false;
56      return true;
57  }

58  int64_t gcd(int64_t x, int64_t y) {
59      return y == 0 ? x : gcd(y, x % y);
60  }

61  int64_t pollard_rho_test(int64_t number, int64_t seed) {
62      int64_t x = rand() % (number - 1) + 1, y = x;
63      int head = 1, tail = 2;
64      while (true) {
65          x = multiply_mod(x, x, number);
66          x = add_mod(x, seed, number);
67          if (x == y)
68              return number;
69          int64_t answer = gcd(std::abs(x - y), number);
70          if (answer > 1 && answer < number)

```

```

71         return answer;
72     if (++head == tail) {
73         y = x;
74         tail <<= 1;
75     }
76 }
77 }

78 void factorize(int64_t number, std::vector<int64_t> &divisor) {
79     if (number > 1) {
80         if (miller_rabin(number)) {
81             divisor.push_back(number);
82         } else {
83             int64_t factor = number;
84             while (factor >= number)
85                 factor = pollard_rho_test(number, rand() % (number - 1) + 1);
86             factorize(number / factor, divisor);
87             factorize(factor, divisor);
88         }
89     }
90 }

```

2.8 $O(m^2 \log n)$ 线性递推

2.9 Polynomial

2.9.1 FFT

2.9.2 NTT & 多项式求逆

2.10 拉格朗日插值

2.11 杜教筛

2.12 BSGS

2.12.1 BSGS

2.12.2 扩展 BSGS

2.13 直线下整点个数 (gy)

必须满足 $a \geq 0, b \geq 0, m > 0$, 返回 $\sum_{i=0}^{n-1} \frac{a+bi}{m}$

```

1 int64_t count(int64_t n, int64_t a, int64_t b, int64_t m) {
2     if (b == 0)
3         return n * (a / m);
4     if (a >= m)
5         return n * (a / m) + count(n, a % m, b, m);
6     if (b >= m)
7         return (n - 1) * n / 2 * (b / m) + count(n, a, b % m, m);
8     return count((a + b * n) / m, (a + b * n) % m, m, b);
9 }

```


2.14 单纯形

2.15 辛普森积分

2.16 常用数列定理 (gy)

- 第一类 Stirling number
用 $s(n, k) = (-1)^{n-k} \begin{bmatrix} n \\ k \end{bmatrix}$ 表示第一类 Stirling number
 $\begin{bmatrix} n+1 \\ k \end{bmatrix} = n \begin{bmatrix} n \\ k \end{bmatrix} + \begin{bmatrix} n \\ k-1 \end{bmatrix}, k > 0$
 $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1, \begin{bmatrix} n \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ n \end{bmatrix} = 0, n > 0$
 $\begin{bmatrix} n \\ k \end{bmatrix}$ 为将 n 个元素分成 k 个环的方案数
- 第二类 Stirling number
用 $S(n, k) = \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$ 表示第二类 Stirling number
 $\left\{ \begin{smallmatrix} n+1 \\ k \end{smallmatrix} \right\} = k \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} + \left\{ \begin{smallmatrix} n \\ k-1 \end{smallmatrix} \right\}, k > 0$
 $\left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \right\} = 1, \left\{ \begin{smallmatrix} n \\ 0 \end{smallmatrix} \right\} = \left\{ \begin{smallmatrix} 0 \\ n \end{smallmatrix} \right\} = 0, n > 0$
 $\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$
 $\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$ 为将 n 个元素划分成 k 个非空集合的方案数
- Catalan Number
 c_n 表示长度为 $2n$ 的合法括号序的数量
 $c_1 = 1, c_{n+1} = \sum_{i=1}^n c_i \times c_{n+1-i}$
 $c_n = \frac{(2n)!}{n+1}$
- Bell number
 B_n 表示基数为 n 的集合的划分方案数
$$B_i = \begin{cases} 1 & i = 0 \\ \sum_{k=0}^n \binom{n}{k} B_k & i > 0 \end{cases}$$

$$B_n = \sum_{k=0}^n \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$$
- Bernoulli number
 $\sum_{j=0}^m \binom{m+1}{j} B_j = 0, m > 0$
$$B_i = \begin{cases} 1 & i = 0 \\ -\frac{\sum_{j=0}^{i-1} \binom{i+1}{j}}{i+1} & i > 0 \end{cases}$$

$$\sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k}$$
- Lagrange polynomial
给定次数为 n 的多项式函数 $L(x)$ 上的 $n+1$ 个点 $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$
则
$$L(x) = \sum_{j=0}^n y_j \prod_{0 \leq m \leq n, m \neq j} \frac{x - x_m}{x_j - x_m}$$

Chapter 3

Geometry

3.1 点、直线、圆 (gy)

```
1 using number = long double;
2 const number eps = 1e-8;

3 number _sqrt(number x) {
4     return std::sqrt(std::max(x, (number) 0));
5 }
6 number _asin(number x) {
7     x = std::min(x, (number) 1), x = std::max(x, (number) -1);
8     return std::asin(x);
9 }
10 number _acos(number x) {
11     x = std::min(x, (number) 1), x = std::max(x, (number) -1);
12     return std::acos(x);
13 }

14 int sgn(number x) {
15     return (x > eps) - (x < -eps);
16 }
17 int cmp(number x, number y) {
18     return sgn(x - y);
19 }

20 struct point {
21     number x, y;
22     point() {}
23     point(number x, number y) : x(x), y(y) {}

24     number len2() const {
25         return x * x + y * y;
26     }
27     number len() const {
28         return _sqrt(len2());
29     }
30     point unit() const {
31         return point(x / len(), y / len());
32     }
33     point rotate90() const {
34         return point(-y, x);
35     }

36     friend point operator+(const point &a, const point &b) {
37         return point(a.x + b.x, a.y + b.y);
```

```

38     }
39     friend point operator-(const point &a, const point &b) {
40         return point(a.x - b.x, a.y - b.y);
41     }
42     friend point operator*(const point &a, number b) {
43         return point(a.x * b, a.y * b);
44     }
45     friend point operator/(const point &a, number b) {
46         return point(a.x / b, a.y / b);
47     }
48     friend number dot(const point &a, const point &b) {
49         return a.x * b.x + a.y * b.y;
50     }
51     friend number det(const point &a, const point &b) {
52         return a.x * b.y - a.y * b.x;
53     }
54     friend number operator==(const point &a, const point &b) {
55         return cmp(a.x, b.x) == 0 && cmp(a.y, b.y) == 0;
56     }
57 };

58 number dis2(const point &a, const point &b) {
59     return (a - b).len2();
60 }
61 number dis(const point &a, const point &b) {
62     return (a - b).len();
63 }

64 struct line {
65     point a, b;
66     line() {}
67     line(point a, point b) : a(a), b(b) {}
68     point value() const {
69         return b - a;
70     }
71 };

72 bool point_on_line(const point &p, const line &l) {
73     return sgn(det(p - l.a, p - l.b)) == 0;
74 }
75 // including endpoint
76 bool point_on_ray(const point &p, const line &l) {
77     return sgn(det(p - l.a, p - l.b)) == 0 &&
78         sgn(dot(p - l.a, l.b - l.a)) >= 0;
79 }
80 // including endpoints
81 bool point_on_seg(const point &p, const line &l) {
82     return sgn(det(p - l.a, p - l.b)) == 0 &&
83         sgn(dot(p - l.a, l.b - l.a)) >= 0 &&
84         sgn(dot(p - l.b, l.a - l.b)) >= 0;
85 }
86 bool seg_has_intersection(const line &a, const line &b) {
87     if (point_on_seg(a.a, b) || point_on_seg(a.b, b) ||
88         point_on_seg(b.a, a) || point_on_seg(b.b, a))
89         return /* including endpoints */ true;
90     return sgn(det(a.a - b.a, b.b - b.a)) * sgn(det(a.b - b.a, b.b - b.a)) < 0
91         && sgn(det(b.a - a.a, a.b - a.a)) * sgn(det(b.b - a.a, a.b - a.a)) < 0;
92 }
93 point intersect(const line &a, const line &b) {
94     number s1 = det(a.b - a.a, b.a - a.a);
95     number s2 = det(a.b - a.a, b.b - a.a);

```

```

96     return (b.a * s2 - b.b * s1) / (s2 - s1);
97 }
98 point projection(const point &p, const line &l) {
99     return l.a + (l.b - l.a) * dot(p - l.a, l.b - l.a) / (l.b - l.a).len2();
100 }
101 number dis(const point &p, const line &l) {
102     return std::abs(dot(p - l.a, l.b - l.a)) / (l.b - l.a).len();
103 }
104 point symmetry_point(const point &a, const point &o) {
105     return o + o - a;
106 }
107 point reflection(const point &p, const line &l) {
108     return symmetry_point(p, projection(p, l));
109 }
110 struct circle {
111     point o;
112     number r;
113     circle() {}
114     circle(point o, number r) : o(o), r(r) {}
115 };
116 bool intersect(const line &l, const circle &a, point &p1, point &p2) {
117     number x = dot(l.a - a.o, l.b - l.a);
118     number y = (l.b - l.a).len2();
119     number d = x * x - y * ((l.a - a.o).len2() - a.r * a.r);
120     if (sgn(d) < 0) return false;
121     point p = l.a - (l.b - l.a) * (x / y), delta = (l.b - l.a) * (_sqrt(d) / y);
122     p1 = p + delta, p2 = p - delta;
123     return true;
124 }
125 bool intersect(const circle &a, const circle &b, point &p1, point &p2) {
126     if (a.o == b.o && cmp(a.r, b.r) == 0)
127         return /* value for coincident circles */ false;
128     number s1 = (b.o - a.o).len();
129     if (cmp(s1, a.r + b.r) > 0 || cmp(s1, std::abs(a.r - b.r)) < 0)
130         return false;
131     number s2 = (a.r * a.r - b.r * b.r) / s1;
132     number aa = (s1 + s2) / 2, bb = (s1 - s2) / 2;
133     point p = (b.o - a.o) * (aa / (aa + bb)) + a.o;
134     point delta = (b.o - a.o).unit().rotate90() * _sqrt(a.r * a.r - aa * aa);
135     p1 = p + delta, p2 = p - delta;
136     return true;
137 }
138 bool tangent(const point &p0, const circle &c, point &p1, point &p2) {
139     number x = (p0 - c.o).len2();
140     number d = x - c.r * c.r;
141     if (sgn(d) < 0) return false;
142     if (sgn(d) == 0)
143         return /* value for point_on_line */ false;
144     point p = (p0 - c.o) * (c.r * c.r / x);
145     point delta = ((p0 - c.o) * (-c.r * _sqrt(d) / x)).rotate90();
146     p1 = c.o + p + delta;
147     p2 = c.o + p - delta;
148     return true;
149 }
150 bool ex_tangent(const circle &a, const circle &b, line &l1, line &l2) {
151     if (cmp(std::abs(a.r - b.r), (b.o - a.o).len()) == 0) {
152         point p1, p2;
153         intersect(a, b, p1, p2);
154         l1 = l2 = line(p1, p1 + (a.o - p1).rotate90());

```

```

155     return true;
156 } else if (cmp(a.r, b.r) == 0) {
157     point dir = b.o - a.o;
158     dir = (dir * (a.r / dir.len())).rotate90();
159     l1 = line(a.o + dir, b.o + dir);
160     l2 = line(a.o - dir, b.o - dir);
161     return true;
162 } else {
163     point p = (b.o * a.r - a.o * b.r) / (a.r - b.r);
164     point p1, p2, q1, q2;
165     if (tangent(p, a, p1, p2) && tangent(p, b, q1, q2)) {
166         l1 = line(p1, q1);
167         l2 = line(p2, q2);
168         return true;
169     } else {
170         return false;
171     }
172 }
173 }
174 bool in_tangent(const circle &a, const circle &b, line &l1, line &l2) {
175     if (cmp(a.r + b.r, (b.o - a.o).len()) == 0) {
176         point p1, p2;
177         intersect(a, b, p1, p2);
178         l1 = l2 = line(p1, p1 + (a.o - p1).rotate90());
179         return true;
180     } else {
181         point p = (b.o * a.r + a.o * b.r) / (a.r + b.r);
182         point p1, p2, q1, q2;
183         if (tangent(p, a, p1, p2) && tangent(p, b, q1, q2)) {
184             l1 = line(p1, q1);
185             l2 = line(p2, q2);
186             return true;
187         } else {
188             return false;
189         }
190     }
191 }

```

3.2 点到凸包切线**3.3 直线凸包交点****3.4 凸包游戏****3.5 半平面交****3.6 旋转卡壳****3.7 判断圆是否有交****3.8 最小圆覆盖****3.9 最小球覆盖****3.10 $O(n^2 \log n)$ 圆交面积和重心****3.11 圆与多边形交****3.12 $O(n \log n)$ 凸多边形内的最大圆****3.13 三角形的五心****3.14 三维凸包****3.15 三维绕轴旋转****3.16 几何公式**

Chapter 4

String

4.1 KMP

4.2 AC 自动机

4.3 后缀数组

4.4 后缀自动机

4.5 Manacher

4.6 回文自动机

4.7 最小表示法

Chapter 5

Data Structure

5.1 莫队 (ct)

```
1 int size;
2 struct Query {
3     int l, r, id;
4     inline bool operator < (const Query &that) const {return l / size != that.l / size ? l < that.l
5         ↪ : ((l / size) & 1 ? r < that.r : r > that.r);}
6 } q[maxn];
7 int main()
8 {
9     size = (int) sqrt(n * 1.0);
10    std::sort(q + 1, q + m + 1);
11    int l = 1, r = 0;
12    for (int i = 1; i <= m; ++i)
13    {
14        for (; r < q[i].r; ) add(++r);
15        for (; r > q[i].r; ) del(r--);
16        for (; l < q[i].l; ) del(l++);
17        for (; l > q[i].l; ) add(--l);
18        /*
19         * write your code here.
20         */
21    }
22    return 0;
23 }
```

5.2 ST 表 (ct)

```
1 int a[maxn], f[20][maxn], n;
2 int Log[maxn];
3 void build()
4 {
5     for (int i = 1; i <= n; ++i) f[0][i] = a[i];
6
7     int lim = Log[n];
8     for (int j = 1; j <= lim; ++j)
9     {
10        int *fj = f[j], *fj1 = f[j - 1];
11        for (int i = 1; i <= n - (1 << j) + 1; ++i)
12            fj[i] = dmax(fj1[i], fj1[i + (1 << (j - 1))]);
13    }
14 }
```



```

14 int Query(int l, int r)
15 {
16     int k = Log[r - l + 1];
17     return dmax(f[k][l], f[k][r - (1 << k) + 1]);
18 }
19 int main()
20 {
21     scanf("%d", &n);
22     Log[0] = -1;
23     for (int i = 1; i <= n; ++i)
24     {
25         scanf("%d", &a[i]);
26         Log[i] = Log[i >> 1] + 1;
27     }
28     build();
29     int q;
30     scanf("%d", &q);
31     for (; q; --q)
32     {
33         int l, r; scanf("%d%d", &l, &r);
34         printf("%d\n", Query(l, r));
35     }
36 }

```

5.3 可并堆 (ct)

```

1 struct Node {
2     Node *ch[2];
3     ll val; int size;
4     inline void update()
5     {
6         size = ch[0] -> size + ch[1] -> size + 1;
7     }
8 } mem[maxn], *rt[maxn];
9 Node *merge(Node *a, Node *b)
10 {
11     if (a == mem) return b;
12     if (b == mem) return a;
13     if (a -> val < b -> val) std::swap(a, b);
14     // a -> pushdown();
15     std::swap(a -> ch[0], a -> ch[1]);
16     a -> ch[1] = merge(a -> ch[1], b);
17     a -> update();
18     return a;
19 }

```

5.4 zkw 线段树 (ct)

```

1 // must be 0-based !
2 inline void build()
3 {
4     for (int i = M - 1; i; --i) tr[i] = dmax(tr[i << 1], tr[i << 1 | 1]);
5 }
6 inline void Change(int x, int v)
7 {
8     x += M; tr[x] = v; x >>= 1;
9     for (; x; x >>= 1) tr[x] = dmax(tr[x << 1], tr[x << 1 | 1]);
10 }

```

```

11 inline int Query(int s, int t)
12 {
13     int ret = -0x7fffffff;
14     for (s = s + M - 1, t = t + M + 1; s ^ t ^ 1; s >>= 1, t >>= 1)
15     {
16         if (~s & 1) cmax(ret, tr[s ^ 1]);
17         if (t & 1) cmax(ret, tr[t ^ 1]);
18     }
19     return ret;
20 }
21 int main()
22 {
23     int n; scanf("%d", &n);
24     for (M = 1; M < n; M <= 1) ;
25     for (int i = 0; i < n; ++i)
26         scanf("%d", &tr[i + M]);
27     for (int i = n; i < M; ++i) tr[i + M] = -0x7fffffff;
28     build();
29     int q; scanf("%d", &q);
30     for (; q; --q)
31     {
32         int l, r; scanf("%d%d", &l, &r); --l, --r;
33         printf("%d\n", Query(l, r));
34     }
35     return 0;
36 }

```

5.5 主席树

5.6 Splay (ct)

指针版

```

1 struct Node *null;
2 struct Node {
3     Node *ch[2], *fa;
4     int val; bool rev;
5     inline bool type()
6     {
7         return fa -> ch[1] == this;
8     }
9     inline void pushup()
10    {
11    }
12    inline void pushdown()
13    {
14        if (rev)
15        {
16            ch[0] -> rev ^= 1;
17            ch[1] -> rev ^= 1;
18            std::swap(ch[0], ch[1]);
19            rev ^= 1;
20        }
21    }
22    inline void rotate()
23    {
24        bool d = type(); Node *f = fa, *gf = f -> fa;
25        (fa = gf, f -> fa != null) ? fa -> ch[f -> type()] = this : 0;
26        (f -> ch[d] = ch[!d]) != null ? ch[!d] -> fa = f : 0;

```

```

27     (ch[!d] = f) -> fa = this;
28     f -> pushup();
29 }
30 inline void splay()
31 {
32     for (; fa != null; rotate())
33         if (fa -> fa != null)
34             (type() == fa -> type() ? fa : this) -> rotate();
35     pushup();
36 }
37 } mem[maxn];

```

数组版

```

1 // BZOJ - 1500 维修数列
2 int fa[maxn], ch[maxn][2], a[maxn], size[maxn], cnt;
3 int sum[maxn], lmx[maxn], rmx[maxn], mx[maxn], v[maxn], id[maxn], root;
4 bool rev[maxn], tag[maxn];
5 inline void update(R int x)
6 {
7     R int ls = ch[x][0], rs = ch[x][1];
8     size[x] = size[ls] + size[rs] + 1;
9     sum[x] = sum[ls] + sum[rs] + v[x];
10    mx[x] = gmax(mx[ls], mx[rs]);
11    cmax(mx[x], lmx[rs] + rmx[ls] + v[x]);
12    lmx[x] = gmax(lmx[ls], sum[ls] + v[x] + lmx[rs]);
13    rmx[x] = gmax(rmx[rs], sum[rs] + v[x] + rmx[ls]);
14 }
15 inline void pushdown(R int x)
16 {
17     R int ls = ch[x][0], rs = ch[x][1];
18     if (tag[x])
19     {
20         rev[x] = tag[x] = 0;
21         if (ls) tag[ls] = 1, v[ls] = v[x], sum[ls] = size[ls] * v[x];
22         if (rs) tag[rs] = 1, v[rs] = v[x], sum[rs] = size[rs] * v[x];
23         if (v[x] >= 0)
24         {
25             if (ls) lmx[ls] = rmx[ls] = mx[ls] = sum[ls];
26             if (rs) lmx[rs] = rmx[rs] = mx[rs] = sum[rs];
27         }
28         else
29         {
30             if (ls) lmx[ls] = rmx[ls] = 0, mx[ls] = v[x];
31             if (rs) lmx[rs] = rmx[rs] = 0, mx[rs] = v[x];
32         }
33     }
34     if (rev[x])
35     {
36         rev[x] ^= 1; rev[ls] ^= 1; rev[rs] ^= 1;
37         swap(lmx[ls], rmx[ls]); swap(lmx[rs], rmx[rs]);
38         swap(ch[ls][0], ch[ls][1]); swap(ch[rs][0], ch[rs][1]);
39     }
40 }
41 inline void rotate(R int x)
42 {
43     R int f = fa[x], gf = fa[f], d = ch[f][1] == x;
44     if (f == root) root = x;
45     (ch[f][d] = ch[x][d ^ 1]) > 0 ? fa[ch[f][d]] = f : 0;
46     (fa[x] = gf) > 0 ? ch[gf][ch[gf][1] == f] = x : 0;
47     fa[ch[x][d ^ 1] = f] = x;

```

```

48     update(f);
49 }
50 inline void splay(R int x, R int rt)
51 {
52     while (fa[x] != rt)
53     {
54         R int f = fa[x], gf = fa[f];
55         if (gf != rt) rotate((ch[gf][1] == f) ^ (ch[f][1] == x) ? x : f);
56         rotate(x);
57     }
58     update(x);
59 }
60 void build(R int l, R int r, R int rt)
61 {
62     if (l > r) return ;
63     R int mid = l + r >> 1, now = id[mid], last = id[rt];
64     if (l == r)
65     {
66         sum[now] = a[l];
67         size[now] = 1;
68         tag[now] = rev[now] = 0;
69         if (a[l] >= 0) lmx[now] = rmx[now] = mx[now] = a[l];
70         else lmx[now] = rmx[now] = 0, mx[now] = a[l];
71     }
72     else
73     {
74         build(l, mid - 1, mid);
75         build(mid + 1, r, mid);
76     }
77     v[now] = a[mid];
78     fa[now] = last;
79     update(now);
80     ch[last][mid >= rt] = now;
81 }
82 int find(R int x, R int rank)
83 {
84     if (tag[x] || rev[x]) pushdown(x);
85     R int ls = ch[x][0], rs = ch[x][1], lsize = size[ls];
86     if (lsize + 1 == rank) return x;
87     if (lsize >= rank)
88         return find(ls, rank);
89     else
90         return find(rs, rank - lsize - 1);
91 }
92 inline int prepare(R int l, R int tot)
93 {
94     R int x = find(root, l - 1), y = find(root, l + tot);
95     splay(x, 0);
96     splay(y, x);
97     return ch[y][0];
98 }
99 std::queue <int> q;
100 inline void Insert(R int left, R int tot)
101 {
102     for (R int i = 1; i <= tot; ++i) a[i] = FastIn();
103     for (R int i = 1; i <= tot; ++i)
104         if (!q.empty()) id[i] = q.front(), q.pop();
105         else id[i] = ++cnt;
106     build(1, tot, 0);
107     R int z = id[(1 + tot) >> 1];
108     R int x = find(root, left), y = find(root, left + 1);

```

```

109     splay(x, 0);
110     splay(y, x);
111     fa[z] = y;
112     ch[y][0] = z;
113     update(y);
114     update(x);
115 }
116 void rec(R int x)
117 {
118     if (!x) return ;
119     R int ls = ch[x][0], rs = ch[x][1];
120     rec(ls); rec(rs); q.push(x);
121     fa[x] = ch[x][0] = ch[x][1] = 0;
122     tag[x] = rev[x] = 0;
123 }
124 inline void Delete(R int l, R int tot)
125 {
126     R int x = prepare(l, tot), f = fa[x];
127     rec(x); ch[f][0] = 0;
128     update(f); update(fa[f]);
129 }
130 inline void Makesame(R int l, R int tot, R int val)
131 {
132     R int x = prepare(l, tot), y = fa[x];
133     v[x] = val; tag[x] = 1; sum[x] = size[x] * val;
134     if (val >= 0) lmx[x] = rmx[x] = mx[x] = sum[x];
135     else lmx[x] = rmx[x] = 0, mx[x] = val;
136     update(y); update(fa[y]);
137 }
138 inline void Reverse(R int l, R int tot)
139 {
140     R int x = prepare(l, tot), y = fa[x];
141     if (!tag[x])
142     {
143         rev[x] ^= 1;
144         swap(ch[x][0], ch[x][1]);
145         swap(lmx[x], rmx[x]);
146         update(y); update(fa[y]);
147     }
148 }
149 inline void Query(R int l, R int tot)
150 {
151     R int x = prepare(l, tot);
152     printf("%d\n", sum[x] );
153 }
154 #define inf ((1 << 30))
155 int main()
156 {
157     R int n = FastIn(), m = FastIn(), l, tot, val;
158     R char op, op2;
159     mx[0] = a[1] = a[n + 2] = -inf;
160     for (R int i = 2; i <= n + 1; i++)
161     {
162         a[i] = FastIn();
163     }
164     for (R int i = 1; i <= n + 2; ++i) id[i] = i;
165     n += 2; cnt = n; root = (n + 1) >> 1;
166     build(1, n, 0);
167     for (R int i = 1; i <= m; i++)
168     {
169         op = getc();

```

```

170     while (op < 'A' || op > 'Z') op = getc();
171     getc(); op2 = getc();getc();getc();getc();getc();
172     if (op == 'M' && op2 == 'X')
173     {
174         printf("%d\n",mx[root] );
175     }
176     else
177     {
178         l = FastIn() + 1; tot = FastIn();
179         if (op == 'I') Insert(l, tot);
180         if (op == 'D') Delete(l, tot);
181         if (op == 'M') val = FastIn(), Makesame(l, tot, val);
182         if (op == 'R')
183             Reverse(l, tot);
184         if (op == 'G')
185             Query(l, tot);
186     }
187 }
188 return 0;
189 }

```

5.7 Treap (ct)

```

1  struct Treap {
2      Treap *ls, *rs;
3      int size;
4      bool rev;
5      inline void update()
6      {
7          size = ls -> size + rs -> size + 1;
8      }
9      inline void set_rev()
10     {
11         rev ^= 1;
12         std::swap(ls, rs);
13     }
14     inline void pushdown()
15     {
16         if (rev)
17         {
18             ls -> set_rev();
19             rs -> set_rev();
20             rev = 0;
21         }
22     }
23 } mem[maxn], *root, *null = mem;
24 struct Pair {
25     Treap *fir, *sec;
26 };
27 Treap *build(R int l, R int r)
28 {
29     if (l > r) return null;
30     R int mid = l + r >> 1;
31     R Treap *now = mem + mid;
32     now -> rev = 0;
33     now -> ls = build(l, mid - 1);
34     now -> rs = build(mid + 1, r);
35     now -> update();

```

```

36     return now;
37 }
38 inline Treap *Find_kth(R Treap *now, R int k)
39 {
40     if (!k) return mem;
41     if (now -> ls -> size >= k) return Find_kth(now -> ls, k);
42     else if (now -> ls -> size + 1 == k) return now;
43     else return Find_kth(now -> rs, k - now -> ls -> size - 1);
44 }
45 Treap *merge(R Treap *a, R Treap *b)
46 {
47     if (a == null) return b;
48     if (b == null) return a;
49     if (rand() % (a -> size + b -> size) < a -> size)
50     {
51         a -> pushdown();
52         a -> rs = merge(a -> rs, b);
53         a -> update();
54         return a;
55     }
56     else
57     {
58         b -> pushdown();
59         b -> ls = merge(a, b -> ls);
60         b -> update();
61         return b;
62     }
63 }
64 Pair split(R Treap *now, R int k)
65 {
66     if (now == null) return (Pair) {null, null};
67     R Pair t = (Pair) {null, null};
68     now -> pushdown();
69     if (k <= now -> ls -> size)
70     {
71         t = split(now -> ls, k);
72         now -> ls = t.sec;
73         now -> update();
74         t.sec = now;
75     }
76     else
77     {
78         t = split(now -> rs, k - now -> ls -> size - 1);
79         now -> rs = t.fir;
80         now -> update();
81         t.fir = now;
82     }
83     return t;
84 }
85 inline void set_rev(int l, int r)
86 {
87     R Pair x = split(root, l - 1);
88     R Pair y = split(x.sec, r - l + 1);
89     y.fir -> set_rev();
90     root = merge(x.fir, merge(y.fir, y.sec));
91 }

```

5.8 可持久化平衡树 (ct)

```

1  char str[maxn];
2  struct Treap
3  {
4      Treap *ls, *rs;
5      char data; int size;
6      inline void update()
7      {
8          size = ls -> size + rs -> size + 1;
9      }
10 } *root[maxn], mem[maxcnt], *tot = mem, *last = mem, *null = mem;
11 inline Treap* new_node(char ch)
12 {
13     *++tot = (Treap) {null, null, ch, 1};
14     return tot;
15 }
16 struct Pair
17 {
18     Treap *fir, *sec;
19 };
20 inline Treap *copy(Treap *x)
21 {
22     if (x == null) return null;
23     if (x > last) return x;
24     *++tot = *x;
25     return tot;
26 }
27 Pair Split(Treap *x, int k)
28 {
29     if (x == null) return (Pair) {null, null};
30     Pair y;
31     Treap *nw = copy(x);
32     if (nw -> ls -> size >= k)
33     {
34         y = Split(nw -> ls, k);
35         nw -> ls = y.sec;
36         nw -> update();
37         y.sec = nw;
38     }
39     else
40     {
41         y = Split(nw -> rs, k - nw -> ls -> size - 1);
42         nw -> rs = y.fir;
43         nw -> update();
44         y.fir = nw;
45     }
46     return y;
47 }
48 Treap *Merge(Treap *a, Treap *b)
49 {
50     if (a == null) return b;
51     if (b == null) return a;
52     Treap *nw;
53     if (rand() % (a -> size + b -> size) < a -> size)
54     {
55         nw = copy(a);
56         nw -> rs = Merge(nw -> rs, b);
57     }
58     else

```



```

59     {
60         nw = copy(b);
61         nw -> ls = Merge(a, nw -> ls);
62     }
63     nw -> update();
64     return nw;
65 }
66 Treap *Build(int l, int r)
67 {
68     if (l > r) return null;
69     R int mid = l + r >> 1;
70     Treap *nw = new_node(str[mid]);
71     nw -> ls = Build(l, mid - 1);
72     nw -> rs = Build(mid + 1, r);
73     nw -> update();
74     return nw;
75 }
76 int now;
77 inline void Insert(int k, char ch)
78 {
79     Pair x = Split(root[now], k);
80     Treap *nw = new_node(ch);
81     root[++now] = Merge(Merge(x.fir, nw), x.sec);
82 }
83 inline void Del(int l, int r)
84 {
85     Pair x = Split(root[now], l - 1);
86     Pair y = Split(x.sec, r - l + 1);
87     root[++now] = Merge(x.fir, y.sec);
88 }
89 inline void Copy(int l, int r, int ll)
90 {
91     Pair x = Split(root[now], l - 1);
92     Pair y = Split(x.sec, r - l + 1);
93     Pair z = Split(root[now], ll);
94     Treap *ans = y.fir;
95     root[++now] = Merge(Merge(z.fir, ans), z.sec);
96 }
97 void Print(Treap *x, int l, int r)
98 {
99     if (!x) return ;
100    if (l > r) return;
101    R int mid = x -> ls -> size + 1;
102    if (r < mid)
103    {
104        Print(x -> ls, l, r);
105        return ;
106    }
107    if (l > mid)
108    {
109        Print(x -> rs, l - mid, r - mid);
110        return ;
111    }
112    Print(x -> ls, l, mid - 1);
113    printf("%c", x -> data );
114    Print(x -> rs, 1, r - mid);
115 }
116 void Printtree(Treap *x)
117 {
118     if (!x) return;
119     Printtree(x -> ls);

```

```

120     printf("%c", x -> data );
121     Printtree(x -> rs);
122 }
123 int main()
124 {
125     srand(time(0) + clock());
126     null -> ls = null -> rs = null; null -> size = 0; null -> data = 0;
127     int n = F();
128     gets(str + 1);
129     int len = strlen(str + 1);
130     root[0] = Build(1, len);
131     while (1)
132     {
133         last = tot;
134         R char opt = getc();
135         while (opt < 'A' || opt > 'Z')
136         {
137             if (opt == EOF) return 0;
138             opt = getc();
139         }
140         if (opt == 'I')
141         {
142             R int x = F();
143             R char ch = getc();
144             Insert(x, ch);
145         }
146         else if (opt == 'D')
147         {
148             R int l = F(), r = F();
149             Del(l, r);
150         }
151         else if (opt == 'C')
152         {
153             R int x = F(), y = F(), z = F();
154             Copy(x, y, z);
155         }
156         else if (opt == 'P')
157         {
158             R int x = F(), y = F(), z = F();
159             Print(root[now - x], y, z);
160             puts("");
161         }
162     }
163     return 0;
164 }

```

5.9 CDQ 分治 (ct)

```

1 struct event
2 {
3     int x, y, id, opt, ans;
4 } t[maxn], q[maxn];
5 void cdq(int left, int right)
6 {
7     if (left == right) return ;
8     R int mid = left + right >> 1;
9     cdq(left, mid);
10    cdq(mid + 1, right);
11    //分成若干个子问题

```

```

12 ++now;
13 for (int i = left, j = mid + 1; j <= right; ++j)
14 {
15     for (; i <= mid && q[i].x <= q[j].x; ++i)
16         if (!q[i].opt)
17             add(q[i].y, q[i].ans);
18     //考虑前面的修改操作对后面的询问的影响
19     if (q[j].opt)
20         q[j].ans += query(q[j].y);
21 }
22 R int i, j, k = 0;
23 //以下相当于归并排序
24 for (i = left, j = mid + 1; i <= mid && j <= right; )
25 {
26     if (q[i].x <= q[j].x)
27         t[k++] = q[i++];
28     else
29         t[k++] = q[j++];
30 }
31 for (; i <= mid; )
32     t[k++] = q[i++];
33 for (; j <= right; )
34     t[k++] = q[j++];
35 for (int i = 0; i < k; ++i)
36     q[left + i] = t[i];
37 }

```

5.10 Bitset (ct)

```

1 namespace Game {
2     #define maxn 300010
3     #define maxs 30010
4     uint b1[32][maxs], b2[32][maxs];
5     int popcnt[256];
6     inline void set(R uint *s, R int pos)
7     {
8         s[pos >> 5] |= 1u << (pos & 31);
9     }
10    inline int popcount(R uint x)
11    {
12        return popcnt[x >> 24 & 255]
13            + popcnt[x >> 16 & 255]
14            + popcnt[x >> 8 & 255]
15            + popcnt[x & 255];
16    }
17    void main() {
18        int n, q;
19        scanf("%d%d", &n, &q);
20
21        char *s1 = new char[n + 1];
22        char *s2 = new char[n + 1];
23        scanf("%s%s", s1, s2);
24
25        uint *anss = new uint[q];
26
27        for (R int i = 1; i < 256; ++i) popcnt[i] = popcnt[i >> 1] + (i & 1);
28
29        #define modify(x, _p)\
30        {\

```

```

27     for (R int j = 0; j < 32 && j <= _p; ++j)\
28         set(b##x[j], _p - j);\
29 }
30 for (R int i = 0; i < n; ++i)
31     if (s1[i] == '0') modify(1, 3 * i)
32     else if (s1[i] == '1') modify(1, 3 * i + 1)
33     else modify(1, 3 * i + 2)
34
35 for (R int i = 0; i < n; ++i)
36     if (s2[i] == '1') modify(2, 3 * i)
37     else if (s2[i] == '2') modify(2, 3 * i + 1)
38     else modify(2, 3 * i + 2)
39
40 for (int Q = 0; Q < q; ++Q) {
41     R int x, y, l;
42     scanf("%d%d%d", &x, &y, &l); x *= 3; y *= 3; l *= 3;
43     uint *f1 = b1[x & 31], *f2 = b2[y & 31], ans = 0;
44     R int i = x >> 5, j = y >> 5, p, lim;
45     for (p = 0, lim = l >> 5; p + 8 < lim; p += 8, i += 8, j += 8)
46     {
47         ans += popcount(f1[i + 0] & f2[j + 0]);
48         ans += popcount(f1[i + 1] & f2[j + 1]);
49         ans += popcount(f1[i + 2] & f2[j + 2]);
50         ans += popcount(f1[i + 3] & f2[j + 3]);
51         ans += popcount(f1[i + 4] & f2[j + 4]);
52         ans += popcount(f1[i + 5] & f2[j + 5]);
53         ans += popcount(f1[i + 6] & f2[j + 6]);
54         ans += popcount(f1[i + 7] & f2[j + 7]);
55     }
56     for (; p < lim; ++p, ++i, ++j) ans += popcount(f1[i] & f2[j]);
57     R uint S = (1u << (l & 31)) - 1;
58     ans += popcount(f1[i] & f2[j] & S);
59     anss[Q] = ans;
60 }
61
62 output_arr(anss, q * sizeof(uint));
63 }
64 }

```

Chapter 6

Others

6.1 vimrc (gy)

```
1 se et ts=4 sw=4 sts=4 nu sc sm lbr is hls mouse=a
2 sy on
3 ino <tab> <c-n>
4 ino <s-tab> <tab>
5 au winnew * winc L

6 nm <f6> ggVG"+y
7 nm <f7> :w<cr>:make<cr>
8 nm <f8> :!@<cr>
9 nm <f9> :!@< in<cr>
10 nm <s-f9> :!(time @@ < in &>> out) &>> out<cr>:sp out<cr>

11 au filetype cpp cm @@ ./a.out | se cin fdm=syntax mp=g++\ \% -std=c++11\ -Wall\ -Wextra\ -O2

12 map <c-p> :ha<cr>
13 se pheader=%n\ %f

14 au filetype java cm @@ java %< | se cin fdm=syntax mp=javac\ \%
15 au filetype python cm @@ python % | se si fdm=indent
16 au bufenter *.kt setf kotlin
17 au filetype kotlin cm @@ kotlin _%<Kt | se si mp=kotlinc\ %
```

6.2 STL 释放内存 (Durandal)

```
1 template <typename T>
2 __inline void clear(T &container) {
3     container.clear();
4     T(container).swap(container);
5 }
```

6.3 开栈 (Durandal)

```
1 register char *_sp __asm__("rsp");
2 int main() {
3     const int size = 400 << 20; // 400 MB
4     static char *sys, *mine(new char[size] + size - 4096);
5     sys = _sp; _sp = mine;
6     _main(); // main method
7     _sp = sys;
```

```

8     return 0;
9 }

```

6.4 Java Template (gy)

```

1  import java.io.BufferedReader;
2  import java.io.IOException;
3  import java.io.InputStreamReader;
4  import java.math.BigDecimal;
5  import java.math.BigInteger;
6  import java.math.RoundingMode;
7  import java.util.ArrayDeque;
8  import java.util.ArrayList;
9  import java.util.Arrays;
10 import java.util.Comparator;
11 import java.util.Deque;
12 import java.util.LinkedList;
13 import java.util.List;
14 import java.util.Scanner;
15 import java.util.StringTokenizer;

16 public class Template {
17     // Input
18     private static BufferedReader reader;
19     private static StringTokenizer tokenizer;

20     private static String next() {
21         try {
22             while (tokenizer == null || !tokenizer.hasMoreTokens())
23                 tokenizer = new StringTokenizer(reader.readLine());
24         } catch (IOException e) {
25             // do nothing
26         }
27         return tokenizer.nextToken();
28     }

29     private static int nextInt() {
30         return Integer.parseInt(next());
31     }

32     private static double nextDouble() {
33         return Double.parseDouble(next());
34     }

35     private static BigInteger nextBigInteger() {
36         return new BigInteger(next());
37     }

38     public static void main(String[] args) {
39         reader = new BufferedReader(new InputStreamReader(System.in));
40         Scanner scanner = new Scanner(System.in);
41         while (scanner.hasNext())
42             scanner.next();
43     }

44     // BigInteger & BigDecimal
45     private static void bigDecimal() {
46         BigDecimal a = BigDecimal.valueOf(1.0);
47         BigDecimal b = a.setScale(50, RoundingMode.HALF_EVEN);

```

```

48     BigDecimal c = b.abs();
49     // if scale omitted, b.scale is used
50     BigDecimal d = c.divide(b, 50, RoundingMode.HALF_EVEN);
51     // since Java 9
52     BigDecimal e = d.sqrt(new MathContext(50, RoundingMode.HALF_EVEN));
53     BigDecimal x = new BigDecimal(BigInteger.ZERO);
54     BigInteger y = BigDecimal.ZERO.toBigInteger(); // RoundingMode.DOWN
55     y = BigDecimal.ZERO.setScale(0, RoundingMode.HALF_EVEN).unscaledValue();
56 }

57 // sqrt for Java 8
58 private static BigDecimal sqrt(BigDecimal a, int scale, RoundingMode mode) {
59     if (a.equals(BigDecimal.ZERO))
60         return BigDecimal.ZERO;
61     a = a.setScale(scale, mode);
62     BigDecimal ans = a;
63     BigDecimal TWO = BigDecimal.valueOf(2L);
64     for (int i = 1; i <= scale; i++)
65         ans = ans.add(a.divide(ans, scale, mode)).divide(TWO, scale, mode);
66     return ans;
67 }

68 private static BigInteger sqrt(BigInteger a) {
69     BigInteger about = BigInteger.ZERO.setBit(a.bitLength() / 2);
70     return sqrt(new BigDecimal(a.toString()), new BigDecimal(about.toString())).setScale(0,
71         ↪ RoundingMode.FLOOR).unscaledValue();
72 }

73 private static BigDecimal sqrt(BigDecimal a, BigDecimal initial) {
74     if (a.equals(BigDecimal.ZERO))
75         return BigDecimal.ZERO;
76     a = a.setScale(50, RoundingMode.HALF_EVEN);
77     BigDecimal ans = initial;
78     for (int i = 1; i <= 10; i++)
79         ans = ans.add(a.divide(ans, RoundingMode.HALF_EVEN)).divide(BigDecimal.valueOf(2),
80             ↪ RoundingMode.HALF_EVEN);
81     return ans;
82 }

83 // ArrayList
84 private static void arrayList() {
85     List<Integer> list = new ArrayList<>();
86     // Generic array is banned
87     List[] lists = new List[100];
88     lists[0] = new ArrayList<Integer>();
89     // for List<Integer>, remove(Integer) stands for element, while remove(int) stands for
90     ↪ index
91     list.remove(list.get(1));
92     list.remove(list.size() - 1);
93     list.clear();
94 }

95 // Queue
96 private static void queue() {
97     LinkedList<Integer> queue = new LinkedList<>();
98     // return the value without popping
99     queue.peek();
100    // pop and return the value
101    queue.poll();
102    Deque<Integer> deque = new ArrayDeque<>();
103    deque.peekFirst();

```

```

101     deque.peekLast();
102     deque.pollFirst();
103 }

104 // Others
105 private static void others() {
106     Arrays.sort(new int[10]);
107     Arrays.sort(new Integer[10], (a, b) -> {
108         if (a.equals(b)) return 0;
109         if (a > b) return -1;
110         return 1;
111     });
112     Arrays.sort(new Integer[10], Comparator.comparingInt((a) -> (int) a).reversed());
113     long a = 1_000_000_000_000_000L;
114     int b = Integer.MAX_VALUE;
115     int c = 'a';
116 }
117 }

```

6.5 Big Fraction (gy)

```

1 fun gcd(a: Long, b: Long): Long = if (b == 0L) a else gcd(b, a % b)

2 class Fraction(val a: BigInteger, val b: BigInteger) {
3     constructor(a: Long, b: Long) : this(BigInteger.valueOf(a / gcd(a, b)), BigInteger.valueOf(b /
4         ↪ gcd(a, b)))

5     operator fun plus(o: Fraction): Fraction {
6         var gcd = b.gcd(o.b)
7         val tempProduct = (b / gcd) * (o.b / gcd)
8         var ansA = a * (o.b / gcd) + o.a * (b / gcd)
9         val gcd2 = ansA.gcd(gcd)
10        ansA /= gcd2
11        gcd /= gcd2
12        return Fraction(ansA, gcd * tempProduct)
13    }

14    operator fun minus(o: Fraction): Fraction {
15        var gcd = b.gcd(o.b)
16        val tempProduct = (b / gcd) * (o.b / gcd)
17        var ansA = a * (o.b / gcd) - o.a * (b / gcd)
18        val gcd2 = ansA.gcd(gcd)
19        ansA /= gcd2
20        gcd /= gcd2
21        return Fraction(ansA, gcd * tempProduct)
22    }

23    operator fun times(o: Fraction): Fraction {
24        val gcd1 = a.gcd(o.b)
25        val gcd2 = b.gcd(o.a)
26        return Fraction((a / gcd1) * (o.a / gcd2), (b / gcd2) * (o.b / gcd1))
27    }
28 }

```

6.6 模拟退火 (ct)

```

1 db ans_x, fans;
2 inline double rand01() {return rand() / 2147483647.0;}

```



```

3 inline double randp() {return (rand() & 1 ? 1 : -1) * rand01();}
4 inline double f(double x)
5 {
6     /*
7      * write your function here.
8      */
9     if (maxx < fans) {fans = maxx; ans_x = x;}
10    return maxx;
11 }
12 int main()
13 {
14     srand(time(NULL) + clock());
15     db x = 0, fnow = f(x);
16     fans = 1e30;
17     for (db T = 1e4; T > 1e-4; T *= 0.997)
18     {
19         db nx = x + randp() * T, fnext = f(nx);
20         db delta = fnext - fnow;
21         if (delta < 1e-9 || exp(-delta / T) > rand01())
22         {
23             x = nx;
24             fnow = fnext;
25         }
26     }
27     return 0;
28 }

```

6.7 三分 (ct)

```

1 inline db cubic_search()
2 {
3     double l = -1e4, r = 1e4;
4     for (int i = 1; i <= 100; ++i)
5     {
6         double ll = (l + r) * 0.5;
7         double rr = (ll + r) * 0.5;
8         if (check(ll) < check(rr)) r = rr;
9         else l = ll;
10    }
11    return (l + r) * 0.5;
12 }

```

6.8 博弈论模型 (gy)

- Wythoff's game
 给定两堆石子，每次可以从任意一堆中取至少一个石子，或从两堆中取相同的至少一个石子，取走最后石子的胜
 先手胜当且仅当石子数满足：
 $\lfloor (b - a) \times \phi \rfloor = a, (a \leq b, \phi = \frac{\sqrt{5}+1}{2})$
 先手胜对应的石子数构成两个序列：
 Lower Wythoff sequence: $a_n = \lfloor n \times \phi \rfloor$
 Upper Wythoff sequence: $b_n = \lfloor n \times \phi^2 \rfloor$
- Fibonacci nim
 给定一堆石子，第一次可以取至少一个、少于石子总数数量的石子，之后每次可以取至少一个、不超过上次取石子数量两倍的石子，取走最后石子的胜
 先手胜当且仅当石子数为斐波那契数

6.9 积分表

- $a^x \rightarrow \frac{a^x}{\ln a}$
- $\sin x \rightarrow -\cos x$
- $\cos x \rightarrow \sin x$
- $\tan x \rightarrow -\ln \cos x$
- $\sec x \rightarrow \ln \tan(\frac{x}{2} + \frac{\pi}{4})$
- $\csc x \rightarrow \ln \tan \frac{x}{2}$
- $\sin^2 x \rightarrow \frac{x}{2} - \frac{1}{2} \sin x \cos x$
- $\cos^2 x \rightarrow \frac{x}{2} + \frac{1}{2} \sin x \cos x$
- $\tan^2 x \rightarrow \tan x - x$
- $\sec^2 x \rightarrow \tan x$
- $\csc^2 x \rightarrow -\tan x$
- $\arcsin x \rightarrow \frac{1}{\sqrt{1-x^2}}$
- $\arccos x \rightarrow -\frac{1}{\sqrt{1-x^2}}$
- $\arctan x \rightarrow \frac{1}{1+x^2}$