

# 上海理工大学 ICPC 校队模板

HiedanoAkyuu、Oneman233、KR12138

2019 年 10 月 18 日

目录

1	字符串	1
1.1	KMP	1
1.2	EX-KMP	1
1.3	Manacher	1
1.4	串的最小表示	1
1.5	后缀数组	1
1.5.1	倍增 SA	1
1.5.2	DC3	1
1.6	回文自动机	1
1.7	AC 自动机	1
1.7.1	多模匹配	1
1.7.2	自动机上 DP	2
1.8	后缀自动机	2
2	计算几何	2
2.1	二维几何	2
2.2	三维几何	4
3	图论	4
3.1	最短路	4
3.1.1	Dijkstra	4
3.1.2	SPFA	4
3.1.3	Floyd	4
3.1.4	负环	4
3.1.5	差分约束	4
3.2	最小生成树	4
3.2.1	Prim	4
3.2.2	Kruskal	4
3.2.3	最小生成树计数	4
3.2.4	次小生成树	4
3.2.5	最小乘积生成树	4
3.3	树的直径	4
3.4	LCA	4
3.4.1	Tarjan 离线	4
3.4.2	倍增 LCA	4
3.5	无向图与有向图联通性	4
3.5.1	割点	4
3.5.2	桥	4
3.5.3	e-DCC	4
3.5.4	v-DCC	4
3.5.5	SCC	4
3.5.6	2-SAT	4
3.5.7	支配树	4
3.6	二分图	4
3.6.1	最大匹配-匈牙利	4
3.6.2	带权匹配-KM	4
3.7	网络流	4
3.7.1	最大流-Dinic	4
3.7.2	最小费用最大流-Dij+Dinic	5
3.7.3	上下界流	5
3.8	欧拉路	5
3.9	Prufer 序列	5
4	数据结构	5
4.1	树状数组	5
4.2	线段树	5
4.2.1	多操作线段树	5
4.2.2	吉司机线段树	5
4.2.3	扫描线	6
4.3	RMQ	6
4.3.1	一维	6
4.3.2	二维	6
4.4	树链剖分	6
4.4.1	点剖分	6
4.4.2	边剖分	6
4.5	平衡树	6
4.5.1	Treap	6
4.5.2	Splay	8
4.6	动态树	8
4.7	主席树	8
4.8	树套树	8
4.8.1	线段树套 Treap	8
4.8.2	线段树套树状数组	8
4.9	K-D Tree	8
4.10	分治	8
4.10.1	CDQ	8
4.10.2	点分治	8
4.10.3	dsu on tree	8
4.10.4	整体二分	8
4.11	分块	8
4.11.1	普通分块	8
4.11.2	莫队	8
4.12	线性基	8
4.13	珂朵莉树	8
4.14	跳舞链	8
5	动态规划	8
5.1	SOS	8
5.2	动态 DP	8
5.3	插头 DP	8
6	数学	8
6.1	矩阵类	8
6.2	质数筛	8
6.2.1	埃筛	8
6.2.2	线筛	8
6.3	质数判定	8
6.3.1	Miller Rabin	8
6.4	质因数分解	8
6.4.1	Pollard-Rho	8
6.5	逆元	8
6.5.1	EX-GCD 求逆元	8
6.5.2	线性筛逆元	8
6.5.3	阶乘逆元	8
6.6	欧拉函数	8
6.6.1	欧拉线筛	8
6.6.2	求单个数的欧拉函数	8
6.6.3	欧拉降幂	8
6.6.4	一般积性函数求法	8
6.7	EX-GCD	8
6.8	CRT	8
6.9	N 次剩余	8
6.10	数论分块	8
6.11	高斯消元	8
6.11.1	普通消元	8
6.11.2	异或方程组消元	8
6.12	莫比乌斯反演	8
6.12.1	莫比乌斯函数	8
6.12.2	杜教筛	8
6.12.3	洲阁筛	8
6.12.4	min25 筛	8
6.13	BSGS	8
6.14	FFT	8
6.15	FWT	8
6.16	NTT	8
6.17	数值计算	8
6.17.1	辛普森	8
6.17.2	自适应辛普森	8
6.18	康拓展开	8
6.19	卢卡斯定理	8

<b>7 其他</b>	<b>8</b>
7.1 快读快写 . . . . .	8
7.2 约瑟夫环 . . . . .	8
7.3 悬线法 . . . . .	8
7.4 蔡勒公式 . . . . .	8
7.5 三角公式 . . . . .	8
7.6 海伦公式 . . . . .	8
7.7 匹克定理 . . . . .	8
7.8 组合计数 . . . . .	8
7.8.1 计数原理 . . . . .	8
7.8.2 卡特兰数 . . . . .	8
7.8.3 Polya . . . . .	8
7.8.4 二项式反演公式 . . . . .	8
7.8.5 斯特林反演公式 . . . . .	8
7.8.6 组合数恒等式 . . . . .	8

# 1 字符串

## 1.1 KMP

## 1.2 EX-KMP

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*
5  ex[i]: s1[i..l1-1]与s2的最大公共前缀长度
6  exnext[i]: s2[i..l2-1]与s2的最大公共前缀长度
7  get_exnext(s2) 求exnext[]
8  exkmp(s1,s2) 求ex[]
9 */
10
11 const int N=50005;
12 char s1[N],s2[N];
13 int ex[N],exnext[N];
14
15 void get_exnext(char s[N])
16 {
17     int n=strlen(s),p=1,j,i;
18     exnext[0]=n;
19     for(i=0;i<n-1&&s[i]==s[i+1];i++);
20     exnext[1]=i;
21     for(i=2;i<n;i++)
22         if(exnext[i-p]+i<p+exnext[p])
23             exnext[i]=exnext[i-p];
24         else
25         {
26             j=exnext[p]+p-i;
27             if(j<0) j=0;
28             while(i+j<n&&s[j]==s[i+j]) j++;
29             exnext[i]=j;
30             p=i;
31         }
32 }
33
34 void exkmp(char s1[N],char s2[N])
35 {
36     int l1=strlen(s1),l2=strlen(s2),p=0,i,j;
37     get_exnext(s2);
38     for(i=0;i<l1&&i<l2&&s1[i]==s2[i];i++);
39     ex[0]=i;
40     for(int i=1;i<l1;i++)
41     {
42         if(exnext[i-p]+i<p+ex[p])
43             ex[i]=exnext[i-p];
44         else
45         {
46             j=ex[p]+p-i;
47             if(j<0) j=0;
48             while(i+j<l1&&s1[i+j]==s2[j]) j++;
49             ex[i]=j;
50             p=i;
51         }
52     }
53 }

```

## 1.3 Manacher

## 1.4 串的最小表示

## 1.5 后缀数组

### 1.5.1 倍增 SA

### 1.5.2 DC3

## 1.6 回文自动机

## 1.7 AC 自动机

### 1.7.1 多模匹配

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*
5  trie静态开点+trie图优化
6 */
7
8 int sz,hd=1,nxt[1000005][26],fail[1000005],id
   [1000005],n;
9 char s[1000005];
10
11 void trie_clean()
12 {
13     sz=1;
14     memset(nxt,0,sizeof(nxt));
15     memset(fail,0,sizeof(fail));
16     memset(id,0,sizeof(id));
17 }
18
19 void trie_insert(int head,char s[],int len,int idx)
20 {
21     int p=head;
22     for(int i=0;i<len;i++)
23     {
24         int c=s[i]-'a';
25         if(!nxt[p][c]) nxt[p][c]=++sz;
26         p=nxt[p][c];
27     }
28     id[p]+=idx;
29 }
30
31 void acatm_build(int head)
32 {
33     int p,tp;
34     queue<int> q;
35     q.push(head);
36     fail[head]=0;
37     while(!q.empty())
38     {
39         p=q.front();
40         q.pop();
41         for(int i=0;i<26;i++)
42             if(nxt[p][i])
43             {
44                 fail[nxt[p][i]]=p==head?head:nxt[fail[p]
   ][i];
45                 q.push(nxt[p][i]);
46             }
47         else
48             nxt[p][i]=p==head?head:nxt[fail[p]][i];
49     }
50 }

```

```

50 }
51
52 int acatm_match(int head,char s[],int len)
53 {
54     int p=head,ret=0;
55     for(int i=0;i<len;i++)
56     {
57         int c=(int)s[i]-'a';
58         p=nxt[p][c];
59         for(int tp=p;tp;tp=fail[tp])
60             if(id[tp]) ret++;
61     }
62     return ret;
63 }

```

### 1.7.2 自动机上 DP

## 1.8 后缀自动机

# 2 计算几何

## 2.1 二维几何

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 #define db double
5 const db EPS=1e-9;
6 inline int sign(db a){return a<-EPS?-1:a>EPS;}
7 inline int cmp(db a,db b){return sign(a-b);}
8 struct P
9 {
10     db x,y;
11     P(){}
12     P(db x,db y):x(x),y(y){}
13     P operator+(P p){return {x+p.x,y+p.y};}
14     P operator-(P p){return {x-p.x,y-p.y};}
15     P operator*(db d){return {x*d,y*d};}
16     P operator/(db d){return {x/d,y/d};}
17     bool operator<(P p) const
18     {
19         int c=cmp(x,p.x);
20         if(c) return c==1;
21         return cmp(y,p.y)==1;
22     }
23     bool operator==(P o) const
24     {
25         return cmp(x,o.x)==0&&cmp(y,o.y)==0;
26     }
27     db distTo(P p){return (*this-p).abs();}
28     db alpha(){return atan2(y,x);}
29     void read(){scanf("%lf%lf",&x,&y);}
30     void write(){printf("%.10f,%.10f\n",x,y);}
31     db abs(){return sqrt(abs2());}
32     db abs2(){return x*x+y*y;}
33     P rot90(){return P(-y,x);}
34     P unit(){return *this/abs();}
35     int quad() const {return sign(y)==1||((sign(y)==0&&
36         sign(x)>=0);}
37     db dot(P p){return x*p.x+y*p.y;}
38     db det(P p){return x*p.y-y*p.x;}
39     P rot(db an){return {x*cos(an)-y*sin(an),x*sin(an)
40         +y*cos(an)}};

```

```

40 //For segment
41 #define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.
42     x-p1.x)*(p2.y-p1.y))
43 #define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
44
45 bool chkLL(P p1,P p2,P q1,P q2) //0:parallel
46 {
47     db a1=cross(q1,q2,p1),a2=-cross(q1,q2,p2);
48     return sign(a1+a2)!=0;
49 }
50
51 P isLL(P p1,P p2,P q1,P q2) //crossover point if
52     chkLL()
53 {
54     db a1=cross(q1,q2,p1),a2=-cross(q1,q2,p2);
55     return (p1*a2+p2*a1)/(a1+a2);
56 }
57
58 bool intersect(db l1,db r1,db l2,db r2)
59 {
60     if(l1>r1) swap(l1,r1);if(l2>r2) swap(l2,r2);
61     return !(cmp(r1,l2)==-1|cmp(r2,l1)==-1);
62 }
63
64 bool isSS(P p1,P p2,P q1,P q2)
65 {
66     return intersect(p1.x,p2.x,q1.x,q2.x)&&intersect(
67         p1.y,p2.y,q1.y,q2.y)&&
68         crossOp(p1,p2,q1)*crossOp(p1,p2,q2)<=0&&crossOp(q1
69             ,q2,p1)*crossOp(q1,q2,p2)<=0;
70 }
71
72 bool isSS_strict(P p1,P p2,P q1,P q2)
73 {
74     return crossOp(p1,p2,q1)*crossOp(p1,p2,q2)<0
75         &&crossOp(q1,q2,p1)*crossOp(q1,q2,p2)<0;
76 }
77
78 bool isMiddle(db a,db m,db b)
79 {
80     return sign(a-m)==0||sign(b-m)==0||((a<m!=b<m);
81 }
82
83 bool isMiddle(P a,P m,P b)
84 {
85     return isMiddle(a.x,m.x,b.x)&&isMiddle(a.y,m.y,b.y
86         );
87 }
88
89 bool onSeg(P p1,P p2,P q)
90 {
91     return crossOp(p1,p2,q)==0&&isMiddle(p1,q,p2);
92 }
93
94 bool onSeg_strict(P p1,P p2,P q)
95 {
96     return crossOp(p1,p2,q)==0&&sign((q-p1).dot(p1-p2)
97         )*sign((q-p2).dot(p1-p2))<0;
98 }
99
100 P proj(P p1,P p2,P q)
101 {
102     P dir=p2-p1;
103     return p1+dir*(dir.dot(q-p1)/dir.abs2());

```

```

99 }
100
101 P reflect(P p1,P p2,P q)
102 {
103     return proj(p1,p2,q)*2-q;
104 }
105
106 db nearest(P p1,P p2,P q)
107 {
108     P h=proj(p1,p2,q);
109     if(isMiddle(p1,h,p2))
110         return q.distTo(h);
111     return min(p1.distTo(q),p2.distTo(q));
112 }
113
114 db disSS(P p1,P p2,P q1,P q2) //dist of 2 segments
115 {
116     if(isSS(p1,p2,q1,q2)) return 0;
117     return min(min(nearest(p1,p2,q1),nearest(p1,p2,q2)),
118                min(nearest(q1,q2,p1),nearest(q1,q2,p2)));
119 }
120
121 db rad(P p1,P p2)
122 {
123     return atan2l(p1.det(p2),p1.dot(p2));
124 }
125
126 db area(vector<P> ps)
127 {
128     db ret=0;
129     for(int i=0;i<ps.size();i++)
130         ret+=ps[i].det(ps[(i+1)%ps.size()]);
131     return ret/2;
132 }
133
134 int contain(vector<P> ps,P p) //2:inside,1:on_seg,0:
135     outside
136 {
137     int n=ps.size(),ret=0;
138     for(int i=0;i<n;i++)
139     {
140         P u=ps[i],v=ps[(i+1)%n];
141         if(onSeg(u,v,p)) return 1;
142         if(cmp(u.y,v.y)<=0) swap(u,v);
143         if(cmp(p.y,u.y)>0||cmp(p.y,v.y)<=0) continue;
144         ret^=crossOp(p,u,v)>0;
145     }
146     return ret*2;
147 }
148
149 vector<P> convexHull(vector<P> ps)
150 {
151     int n=ps.size();if(n<=1) return ps;
152     sort(ps.begin(),ps.end());
153     vector<P> qs(n*2);int k=0;
154     for(int i=0;i<n;qs[k++]=ps[i++])
155         while(k>1&&crossOp(qs[k-2],qs[k-1],ps[i])<=0)
156             --k;
157     for(int i=n-2,t=k;i>=0;qs[k++]=ps[i--])
158         while(k>t&&crossOp(qs[k-2],qs[k-1],ps[i])<=0)
159             --k;
160     qs.resize(k-1);
161     return qs;
162 }

```

```

160 db convexDiameter(vector<P> ps)
161 {
162     int n=ps.size();if(n<=1) return 0;
163     int is=0,js=0;
164     for(int k=1;k<n;k++) is=ps[k]<ps[is]?k:is,js=ps[js]
165         <ps[k]?js:k;
166     int i=is,j=js;
167     db ret=ps[i].distTo(ps[j]);
168     do{
169         if((ps[(i+1)%n]-ps[i]).det(ps[(j+1)%n]-ps[j])
170             >=0) (++j)%=n;
171         else (++i)%=n;
172         ret=max(ret,ps[i].distTo(ps[j]));
173     }while(i!=is||j!=js);
174     return ret;
175 }

```

## 2.2 三维几何

## 3 图论

### 3.1 最短路

#### 3.1.1 Dijkstra

#### 3.1.2 SPFA

#### 3.1.3 Floyd

#### 3.1.4 负环

#### 3.1.5 差分约束

### 3.2 最小生成树

#### 3.2.1 Prim

#### 3.2.2 Kruskal

#### 3.2.3 最小生成树计数

#### 3.2.4 次小生成树

#### 3.2.5 最小乘积生成树

### 3.3 树的直径

### 3.4 LCA

#### 3.4.1 Tarjan 离线

#### 3.4.2 倍增 LCA

### 3.5 无向图与有向图联通性

#### 3.5.1 割点

#### 3.5.2 桥

#### 3.5.3 e-DCC

#### 3.5.4 v-DCC

#### 3.5.5 SCC

#### 3.5.6 2-SAT

#### 3.5.7 支配树

### 3.6 二分图

#### 3.6.1 最大匹配-匈牙利

#### 3.6.2 带权匹配-KM

### 3.7 网络流

#### 3.7.1 最大流-Dinic

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 /*
6  s,t 超级源、超级汇
7  cur[] 当前弧优化
8  时间复杂度 O(n^2*m)
9 */
10
11 const int MAXN=10005;
12 const ll inf=0x3f3f3f3f3f3f3f3f;
13 int n,m,s,t,tot,dis[MAXN],cur[MAXN];

```

```

14 struct edge
15 {
16     int to,cap,rev;
17     edge(){}
18     edge(int to,int cap,int rev):to(to),cap(cap),rev(
19         rev){}
20 };
21 vector<edge> E[MAXN];
22
23 inline void add_edge(int x,int y,int f)
24 {
25     E[x].emplace_back(y,f,E[y].size());
26     E[y].emplace_back(x,0,E[x].size()-1);
27 }
28
29 int bfs()
30 {
31     for(int i=1;i<=n;i++) dis[i]=0x3f3f3f3f;
32     dis[s]=0;
33     queue<int> q;
34     q.push(s);
35     while(!q.empty())
36     {
37         int now=q.front();q.pop();
38         for(int i=0;i<E[now].size();i++)
39         {
40             edge &e=E[now][i];
41             if(dis[e.to]>dis[now]+1&&e.cap)
42             {
43                 dis[e.to]=dis[now]+1;
44                 if(e.to==t) return 1;
45                 q.push(e.to);
46             }
47         }
48     }
49     return 0;
50 }
51
52 ll dfs(int now,ll flow)
53 {
54     if(now==t) return flow;
55     ll rest=flow,k;
56     for(int i=cur[now];i<E[now].size();i++)
57     {
58         edge &e=E[now][i];
59         if(e.cap&&dis[e.to]==dis[now]+1)
60         {
61             cur[now]=i;
62             k=dfs(e.to,min(rest,(long long)e.cap));
63             e.cap-=k;
64             E[e.to][e.rev].cap+=k;
65             rest-=k;
66         }
67     }
68     return flow-rest;
69 }
70
71 ll dinic()
72 {
73     ll ret=0,delta;
74     while(bfs())
75     {
76         for(int i=1;i<=n;i++) cur[i]=0;
77         while(delta=dfs(s,inf)) ret+=delta;
78     }
79 }

```

```

78     return ret;
79 }

```

### 3.7.2 最小费用最大流-Dij+Dinic

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef pair<int,int> pii;
4
5  /*
6   第一遍跑的spfa,然后是加上势函数的dij,玄学
7   h[] 势函数
8   cur[] 当前弧优化
9   msmf 最大流时的最小费用
10  s,t 超级源、超级汇
11  时间复杂度 O(n^2*m)
12  */
13
14  const int MAXN=2005;
15  const int inf=0x3f3f3f3f;
16  int msmf,s,t,cur[MAXN],dis[MAXN],vis[MAXN],h[MAXN];
17  struct edge
18  {
19      int to,val,cap,rev;
20      edge(){}
21      edge(int to,int cap,int val,int rev):to(to),cap(
22          cap),val(val),rev(rev){}
23  };
24  vector<edge> E[MAXN];
25
26  inline void add_edge(int x,int y,int f,int cost)
27  {
28      E[x].emplace_back(y,f,cost,E[y].size());
29      E[y].emplace_back(x,0,-cost,E[x].size()-1);
30  }
31
32  int dij()
33  {
34      fill(dis,dis+t+1,inf);
35      priority_queue<pii,vector<pii>,greater<pii>> q;
36      q.emplace(0,s);dis[s]=0;
37      while(!q.empty())
38      {
39          pii p=q.top();q.pop();
40          int now=p.second;
41          if(dis[now]<p.first) continue;
42          for(int i=0;i<E[now].size();i++)
43          {
44              edge &e=E[now][i];
45              if(e.cap>0&&dis[e.to]>p.first+e.val+h[now]-
46                  h[e.to])
47              {
48                  dis[e.to]=p.first+e.val+h[now]-h[e.to];
49                  q.emplace(dis[e.to],e.to);
50              }
51          }
52          return dis[t]!=inf;
53      }
54
55  int dfs(int now,int flow)
56  {
57      if(now==t) return flow;
58      int rest=flow,k;

```

```

58  vis[now]=1;
59  for(int i=cur[now];i<E[now].size();i++)
60  {
61      edge &e=E[now][i];
62      if(e.cap&&dis[now]+e.val+h[now]-h[e.to]==dis[
63          e.to]&&!vis[e.to])
64      {
65          cur[now]=i;
66          k=dfs(e.to,min(e.cap,rest));
67          e.cap-=k;
68          E[e.to][e.rev].cap+=k;
69          rest-=k;
70          msmf+=k*e.val;
71      }
72  }
73  vis[now]=0;
74  return flow-rest;
75
76  int dinic()
77  {
78      int ret=0,delta;
79      while(dij())
80      {
81          for(int i=s;i<=t;i++) cur[i]=0;
82          while(delta=dfs(s,inf)) ret+=delta;
83          for(int i=s;i<=t;i++) h[i]+=(dis[i]==inf)?0:
84              dis[i];
85      }
86      return ret;

```

### 3.7.3 上下界流

## 3.8 欧拉路

## 3.9 Prufer 序列

# 4 数据结构

## 4.1 树状数组

## 4.2 线段树

### 4.2.1 多操作线段树

### 4.2.2 吉司机线段树

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  /*
6   modify 将区间大于x的数变成x
7   query 询问区间和
8   单次复杂度 O(log^2(n))
9  */
10
11  const ll INF=0xc0c0c0c0c0c0c0c0ll;
12  const int MAXN=200005;
13  ll seg[MAXN<<2],m1[MAXN<<2],m2[MAXN<<2],cnt[MAXN<<2],
14      tag[MAXN<<2],a[MAXN];
15
16  int n,q;
17
18  void pushdown(int rt)

```



```

17 {
18     if(!tag[rt]) return;
19     ll y=m1[rt];
20     if(y<m1[rt<<1])
21     {
22         tag[rt<<1]=1;
23         seg[rt<<1]-=(m1[rt<<1]-y)*cnt[rt<<1];
24         m1[rt<<1]=y;
25     }
26     if(y<m1[rt<<1|1])
27     {
28         tag[rt<<1|1]=1;
29         seg[rt<<1|1]-=(m1[rt<<1|1]-y)*cnt[rt<<1|1];
30         m1[rt<<1|1]=y;
31     }
32     tag[rt]=0;
33 }
34
35 void pushup(int rt)
36 {
37     seg[rt]=seg[rt<<1]+seg[rt<<1|1];
38     if(m1[rt<<1]==m1[rt<<1|1])
39     {
40         m1[rt]=m1[rt<<1];
41         cnt[rt]=cnt[rt<<1]+cnt[rt<<1|1];
42         m2[rt]=max(m2[rt<<1],m2[rt<<1|1]);
43     }
44     else if(m1[rt<<1]>m1[rt<<1|1])
45     {
46         m1[rt]=m1[rt<<1];
47         cnt[rt]=cnt[rt<<1];
48         m2[rt]=max(m2[rt<<1],m1[rt<<1|1]);
49     }
50     else
51     {
52         m1[rt]=m1[rt<<1|1];
53         cnt[rt]=cnt[rt<<1|1];
54         m2[rt]=max(m2[rt<<1|1],m1[rt<<1]);
55     }
56 }
57
58 void build(int rt,int l,int r)
59 {
60     tag[rt]=0;
61     if(l==r)
62     {
63         seg[rt]=m1[rt]=a[l];
64         cnt[rt]=1;
65         m2[rt]=INF;
66         return;
67     }
68     int m=l+r>>1;
69     if(l<=m) build(rt<<1,l,m);
70     if(m<r) build(rt<<1|1,m+1,r);
71     pushup(rt);
72 }
73
74 void modify(int rt,int l,int r,int L,int R,ll y)
75 {
76     if(y>m1[rt]) return;
77     if(L<=l&&r<=R&&y>m2[rt])
78     {
79         tag[rt]=1;
80         seg[rt]-=(m1[rt]-y)*cnt[rt];
81         m1[rt]=y;

```

```

82         return;
83     }
84     pushdown(rt);
85     int m=l+r>>1;
86     if(L<=m) modify(rt<<1,l,m,L,R,y);
87     if(m<R) modify(rt<<1|1,m+1,r,L,R,y);
88     pushup(rt);
89 }
90
91 ll query(int rt,int l,int r,int L,int R)
92 {
93     if(L<=l&&r<=R) return seg[rt];
94     int m=l+r>>1;
95     pushdown(rt);
96     ll ret=0;
97     if(L<=m) ret+=query(rt<<1,l,m,L,R);
98     if(m<R) ret+=query(rt<<1|1,m+1,r,L,R);
99     pushup(rt);
100     return ret;
101 }

```

#### 4.2.3 扫描线

### 4.3 RMQ

#### 4.3.1 一维

#### 4.3.2 二维

### 4.4 树链剖分

#### 4.4.1 点剖分

#### 4.4.2 边剖分

### 4.5 平衡树

#### 4.5.1 Treap

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAXN=1e5+5;
5 const int inf=0x7fffffff;
6 int n,op,x;
7
8 /*
9     树内初始化时有无穷大和无穷小两个结点
10 */
11
12 namespace Treap
13 {
14     int tot,root;
15     struct node
16     {
17         int cnt,val,dat,siz,lc,rc;
18     }bst[MAXN];
19
20     inline void pushup(int rt)
21     {
22         bst[rt].siz=bst[rt].cnt;
23         if(bst[rt].lc) bst[rt].siz+=bst[bst[rt].lc].siz;
24         if(bst[rt].rc) bst[rt].siz+=bst[bst[rt].rc].siz;
25     }
26

```

```

27 inline void zig(int &rt)
28 {
29     int p=bst[rt].lc;
30     bst[rt].lc=bst[p].rc;
31     bst[p].rc=rt;
32     rt=p;
33     pushup(bst[rt].rc);pushup(rt);
34 }
35
36 inline void zag(int &rt)
37 {
38     int p=bst[rt].rc;
39     bst[rt].rc=bst[p].lc;
40     bst[p].lc=rt;
41     rt=p;
42     pushup(bst[rt].lc);pushup(rt);
43 }
44
45 int new_node(int val)
46 {
47     bst[++tot].val=val;
48     bst[tot].dat=rand();
49     bst[tot].siz=bst[tot].cnt=1;
50     bst[tot].lc=bst[tot].rc=0;
51     return tot;
52 }
53
54 void build()
55 {
56     new_node(-inf);new_node(inf);
57     root=1,bst[1].rc=2;
58     pushup(1);
59 }
60
61 void _insert(int &rt,int val)
62 {
63     if(rt==0)
64     {
65         rt=new_node(val);
66         return;
67     }
68     if(bst[rt].val==val)
69     {
70         bst[rt].cnt++;
71         pushup(rt);
72         return;
73     }
74     if(val<bst[rt].val)
75     {
76         _insert(bst[rt].lc,val);
77         if(bst[rt].dat<bst[bst[rt].lc].dat) zig(rt)
78         ;
79     }
80     else
81     {
82         _insert(bst[rt].rc,val);
83         if(bst[rt].dat<bst[bst[rt].rc].dat) zag(rt)
84         ;
85     }
86     pushup(rt);
87 }
88
89 void _delete(int &rt,int val)
90 {
91     if(rt==0) return;

```

```

90 if(bst[rt].val==val)
91 {
92     if(bst[rt].cnt>1)
93     {
94         bst[rt].cnt--;
95         pushup(rt);
96         return;
97     }
98     if(bst[rt].rc||bst[rt].lc)
99     {
100         if(bst[rt].rc==0||bst[bst[rt].rc].dat<
101             bst[bst[rt].lc].dat)
102             zig(rt),_delete(bst[rt].rc,val);
103         else
104             zag(rt),_delete(bst[rt].lc,val);
105         pushup(rt);
106     }
107     else rt=0;
108     return;
109 }
110 if(val<bst[rt].val) _delete(bst[rt].lc,val);
111 else _delete(bst[rt].rc,val);
112 pushup(rt);
113 }
114
115 int getPrev(int val)
116 {
117     int ret=1,rt=root;
118     while(rt)
119     {
120         if(bst[rt].val==val)
121         {
122             if(bst[rt].lc)
123             {
124                 rt=bst[rt].lc;
125                 while(bst[rt].rc) rt=bst[rt].rc;
126                 ret=rt;
127             }
128             break;
129         }
130         if(bst[rt].val<val&&bst[rt].val>bst[ret].
131             val) ret=rt;
132         if(val<bst[rt].val) rt=bst[rt].lc;
133         else rt=bst[rt].rc;
134     }
135     return bst[ret].val;
136 }
137
138 int getNext(int val)
139 {
140     int ret=2,rt=root;
141     while(rt)
142     {
143         if(bst[rt].val==val)
144         {
145             if(bst[rt].rc)
146             {
147                 rt=bst[rt].rc;
148                 while(bst[rt].lc) rt=bst[rt].lc;
149                 ret=rt;
150             }
151             break;
152         }
153         if(bst[rt].val>val&&bst[rt].val<bst[ret].

```

```

153         val) ret=rt;
154         if(val<bst[rt].val) rt=bst[rt].lc;
155         else rt=bst[rt].rc;
156     }
157     return bst[ret].val;
158 }
159 int getRank(int rt,int val)
160 {
161     if(rt==0) return 0;
162     if(val==bst[rt].val) return bst[bst[rt].lc].
        siz+1;
163     if(val<bst[rt].val) return getRank(bst[rt].lc,
        val);
164     else return bst[bst[rt].lc].siz+bst[rt].cnt+
        getRank(bst[rt].rc,val);
165 }
166
167 int getVal(int rt,int k)
168 {
169     if(rt==0) return inf;
170     if(bst[bst[rt].lc].siz>=k) return getVal(bst[
        rt].lc,k);
171     if(bst[bst[rt].lc].siz+bst[rt].cnt>=k) return
        bst[rt].val;
172     return getVal(bst[rt].rc,k-bst[bst[rt].lc].siz
        -bst[rt].cnt);
173 }
174 }
175
176 int main()
177 {
178     using namespace Treap;
179     srand(time(0));
180     build();
181     scanf("%d",&n);
182     while(n-->0)
183     {
184         scanf("%d",&op,&x);
185         if(op==1) _insert(root,x);
186         else if(op==2) _delete(root,x);
187         else if(op==3) printf("%d\n",getRank(root,x)
            -1);
188         else if(op==4) printf("%d\n",getVal(root,x+1))
            ;
189         else if(op==5) printf("%d\n",getPrev(x));
190         else if(op==6) printf("%d\n",getNext(x));
191     }
192     return 0;
193 }

```

#### 4.5.2 Splay

### 4.6 动态树

#### 4.7 主席树

#### 4.8 树套树

##### 4.8.1 线段树套 Treap

##### 4.8.2 线段树套树状数组

### 4.9 K-D Tree

#### 4.10 分治

##### 4.10.1 CDQ

##### 4.10.2 点分治

##### 4.10.3 dsu on tree

##### 4.10.4 整体二分

#### 4.11 分块

##### 4.11.1 普通分块

##### 4.11.2 莫队

#### 4.12 线性基

#### 4.13 珂朵莉树

#### 4.14 跳舞链

## 5 动态规划

### 5.1 SOS

### 5.2 动态 DP

### 5.3 插头 DP

## 6 数学

### 6.1 矩阵类

### 6.2 质数筛

#### 6.2.1 埃筛

#### 6.2.2 线筛

### 6.3 质数判定

#### 6.3.1 Miller Rabin

### 6.4 质因数分解

#### 6.4.1 Pollard-Rho

### 6.5 逆元

#### 6.5.1 EX-GCD 求逆元

#### 6.5.2 线性筛逆元

#### 6.5.3 阶乘逆元

### 6.6 欧拉函数

#### 6.6.1 欧拉线筛

#### 6.6.2 求单个数的欧拉函数

#### 6.6.2 欧拉降幂