

小小青蛙听风就是雨

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	树状数组套线段树	10
4.9	K-D Tree	11
4.10	分治	11
4.10.1	CDQ	11
4.10.2	点分治	11
4.10.3	dsu on tree	11
4.10.4	整体二分	11
4.11	分块	11
4.11.1	普通分块	11
4.11.2	莫队	11
4.12	线性基	11
4.13	珂朵莉树	11
4.14	跳舞链	11
5	动态规划	11
5.1	SOS	11
5.2	动态 DP	11
5.3	插头 DP	11
6	数学	11
6.1	矩阵类	11
6.2	质数筛	11
6.2.1	埃筛	11
6.2.2	线筛	11
6.3	质数判定	11
6.3.1	Miller Rabin	11
6.4	质因数分解	11
6.4.1	Pollard-Rho	11
6.5	逆元	11
6.5.1	EX-GCD 求逆元	11
6.5.2	线性筛逆元	11
6.5.3	阶乘逆元	11
6.6	欧拉函数	11
6.6.1	欧拉线筛	11
6.6.2	求单个数的欧拉函数	11
6.6.3	欧拉降幂	11
6.6.4	一般积性函数求法	11
6.7	EX-GCD	11
6.8	CRT	11
6.9	N 次剩余	11
6.10	数论分块	11
6.11	高斯消元	11
6.11.1	普通消元	11
6.11.2	异或方程组消元	11
6.12	莫比乌斯反演	11
6.12.1	莫比乌斯函数	11
6.12.2	杜教筛	11
6.12.3	洲阁筛	11
6.12.4	min25 筛	11
6.13	BSGS	11
6.14	FFT	11
6.15	FWT	11
6.16	NTT	11
6.17	数值计算	11
6.17.1	辛普森	11
6.17.2	自适应辛普森	11
6.18	康拓展开	11
6.19	卢卡斯定理	11

7 其他11

7.1 快读快写 11

7.2 约瑟夫环 11

7.3 悬线法 11

7.4 蔡勒公式 11

7.5 三角公式 11

7.6 海伦公式 11

7.7 匹克定理 11

7.8 组合计数 11

7.8.1 计数原理 11

7.8.2 卡特兰数 11

7.8.3 Polya 11

7.8.4 二项式反演公式 11

7.8.5 斯特林反演公式 11

7.8.6 组合数恒等式 11

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
9   [1000005],n;
10 char s[1000005];
11
12 void trie_clean()
13 {
14     sz=1;
15     memset(nxt,0,sizeof(nxt));
16     memset(fail,0,sizeof(fail));
17     memset(id,0,sizeof(id));
18 }
19
20 void trie_insert(int head,char s[],int len,int idx)
21 {
22     int p=head;
23     for(int i=0;i<len;i++)
24     {
25         int c=s[i]-'a';
26         if(!nxt[p][c]) nxt[p][c]=++sz;
27         p=nxt[p][c];
28     }
29     id[p]=idx;
30 }
31
32 void acatm_build(int head)
33 {
34     int p,tp;
35     queue<int> q;
36     q.push(head);
37     fail[head]=0;
38     while(!q.empty())
39     {
40         p=q.front();
41         q.pop();
42         for(int i=0;i<26;i++)
43             if(nxt[p][i])
44             {
45                 fail[nxt[p][i]]=p==head?head:nxt[fail[p]
46                     ][i];
47                 q.push(nxt[p][i]);
48             }
49         else
50             nxt[p][i]=p==head?head:nxt[fail[p]][i];
51     }
52 }

```

```

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         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 两维

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     _delete(root,x) 删除一个x
11     _insert(root,x) 插入一个x
12     getRank(root,x) 返回x的排名+1(包含了无穷小)
13     getVal(root,x+1) 返回排名为x的数
14     getPrev(x) x的前驱
15     getNext(x) x的后继
16 */
17
18 namespace Treap
19 {
20     int tot,root;
21     struct node
22     {
23         int cnt,val,dat,siz,lc,rc;
24     }bst[MAXN];
25
26     inline void pushup(int rt)
27     {
28         bst[rt].siz=bst[rt].cnt;

```

```

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

```

```

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

```

```

153         while(bst[rt].lc) rt=bst[rt].lc;
154         ret=rt;
155     }
156     break;
157 }
158 if(bst[rt].val>val&&bst[rt].val<bst[ret].
159     val) ret=rt;
160 if(val<bst[rt].val) rt=bst[rt].lc;
161 else rt=bst[rt].rc;
162 }
163 return bst[ret].val;
164 }
165 int getRank(int rt,int val)
166 {
167     if(rt==0) return 0;
168     if(val==bst[rt].val) return bst[rt].lc.
169         siz+1;
170     if(val<bst[rt].val) return getRank(bst[rt].lc,
171         val);
172     else return bst[rt].lc.siz+bst[rt].cnt+
173         getRank(bst[rt].rc,val);
174 }
175 int getVal(int rt,int k)
176 {
177     if(rt==0) return inf;
178     if(bst[rt].lc.siz>=k) return getVal(bst[
179         rt].lc,k);
180     if(bst[rt].lc.siz+bst[rt].cnt>=k) return
181         bst[rt].val;
182     return getVal(bst[rt].rc,k-bst[rt].lc.siz
183         -bst[rt].cnt);
184 }
185 }
186 int main()
187 {
188     using namespace Treap;
189     srand(time(0));
190     build();
191     scanf("%d",&n);
192     while(n--)
193     {
194         scanf("%d",&op,&x);
195         if(op==1) _insert(root,x);
196         else if(op==2) _delete(root,x);
197         else if(op==3) printf("%d\n",getRank(root,x)
198             -1);
199         else if(op==4) printf("%d\n",getVal(root,x+1))
200             ;
201         else if(op==5) printf("%d\n",getPrev(x));
202         else if(op==6) printf("%d\n",getNext(x));
203     }
204     return 0;
205 }

```

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

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  /*
5   空间 O(nlogn)
6   单点修改, 区间rank, 前驱后继(不存在则为±2147483647) 单
7   次 O(log^2(n))
8   区间排名为k的值 单次 O(log^3(n))
9  */
10 const int inf=2147483647;
11 const int MAXN=50005;
12 int root[MAXN<<2],n,m,a[MAXN];
13 struct Treap
14 {
15     int tot;
16     struct node
17     {
18         int lc,rc,dat,val,cnt,siz;
19     }bst[MAXN*4*20];
20
21     int newNode(int v)
22     {
23         bst[++tot].val=v;
24         bst[tot].dat=rand();
25         bst[tot].siz=bst[tot].cnt=1;
26         bst[tot].lc=bst[tot].rc=0;
27         return tot;
28     }
29
30     void zig(int &rt)
31     {
32         int p=bst[rt].lc;
33         bst[rt].lc=bst[p].rc;
34         bst[p].rc=rt;
35         rt=p;
36         pushup(bst[rt].rc);
37         pushup(rt);
38     }
39
40     void zag(int &rt)
41     {
42         int p=bst[rt].rc;
43         bst[rt].rc=bst[p].lc;
44         bst[p].lc=rt;
45         rt=p;
46         pushup(bst[rt].lc);
47         pushup(rt);
48     }
49
50     void pushup(int rt)
51     {
52         bst[rt].siz=bst[rt].cnt;
53         if(bst[rt].lc) bst[rt].siz+=bst[bst[rt].lc].
54             siz;
55         if(bst[rt].rc) bst[rt].siz+=bst[bst[rt].rc].

```

siz;

}

int build()

{

int rt=newnode(-inf);

bst[rt].rc=newnode(inf);

pushup(rt);

return rt;

}

void _delete(**int** &rt,**int** x)

{

if(bst[rt].val==x)

{

if(bst[rt].cnt>1)

{

bst[rt].cnt--;

pushup(rt);

return;

}

if(bst[rt].lc||bst[rt].rc)

{

if(bst[rt].rc==0||bst[bst[rt].rc].dat<
bst[bst[rt].lc].dat)

zig(rt),_delete(bst[rt].rc,x);

else

zag(rt),_delete(bst[rt].lc,x);

pushup(rt);

}

else rt=0;**return**;

}

if(x<bst[rt].val) _delete(bst[rt].lc,x);**else** _delete(bst[rt].rc,x);

pushup(rt);

}

void _insert(**int** &rt,**int** x)

{

if(rt==0)

{

rt=newnode(x);

return;

}

if(bst[rt].val==x) bst[rt].cnt++;**else if**(x<bst[rt].val)

{

_insert(bst[rt].lc,x);

if(bst[bst[rt].lc].dat>bst[rt].dat) zig(rt)

;

}

else

{

_insert(bst[rt].rc,x);

if(bst[bst[rt].rc].dat>bst[rt].dat) zag(rt)

;

}

pushup(rt);

}

int get_rank(**int** rt,**int** x)

{

if(!rt) **return** 1;**if**(bst[rt].val==x) **return** bst[bst[rt].lc].siz

+1;

if(x<bst[rt].val) **return** get_rank(bst[rt].lc,x

);

else **return** get_rank(bst[rt].rc,x)+bst[bst[rt]

].lc].siz+bst[rt].cnt;

}

int get_num(**int** rt,**int** x)

{

if(!rt) **return** 0;**if**(bst[rt].val==x) **return** bst[bst[rt].lc].siz+
bst[rt].cnt;**if**(x<bst[rt].val) **return** get_num(bst[rt].lc,x)

;

else **return** get_num(bst[rt].rc,x)+bst[bst[rt].

lc].siz+bst[rt].cnt;

}

int get_prev(**int** rt,**int** x)

{

int ret=-inf;**while**(rt)

{

if(bst[rt].val==x)

{

if(bst[rt].lc)

{

rt=bst[rt].lc;

while(bst[rt].rc) rt=bst[rt].rc;

ret=bst[rt].val;

}

break;

}

if(bst[rt].val<x&&bst[rt].val>ret) ret=bst[
rt].val;**if**(x<bst[rt].val) rt=bst[rt].lc;**else** rt=bst[rt].rc;

}

return ret;

}

int get_nxt(**int** rt,**int** x)

{

int ret=inf;**while**(rt)

{

if(bst[rt].val==x)

{

if(bst[rt].rc)

{

rt=bst[rt].rc;

while(bst[rt].lc) rt=bst[rt].lc;

ret=bst[rt].val;

}

break;

}

if(bst[rt].val>x&&bst[rt].val<ret) ret=bst[
rt].val;**if**(x<bst[rt].val) rt=bst[rt].lc;**else** rt=bst[rt].rc;

}

return ret;

}

}treap;

```

173 void build(int rt,int l,int r)
174 {
175     root[rt]=treap.build();
176     if(l==r) return;
177     int m=l+r>>1;
178     build(rt<<1,l,m);
179     build(rt<<1|1,m+1,r);
180 }
181
182 void modify(int rt,int l,int r,int x,int v,int y)
183 {
184     if(y==-1) treap._delete(root[rt],v);
185     else treap._insert(root[rt],v);
186     if(l==r) return;
187     int m=l+r>>1;
188     if(x<=m) modify(rt<<1,l,m,x,v,y);
189     else modify(rt<<1|1,m+1,r,x,v,y);
190 }
191
192 int query(int rt,int l,int r,int op,int L,int R,int x
193 )
194 {
195     if(L<=l&&r<=R)
196     {
197         if(op==1) return treap.get_rank(root[rt],x)-2;
198         if(op==2) return treap.get_num(root[rt],x)-1;
199         if(op==4) return treap.get_prev(root[rt],x);
200         if(op==5) return treap.get_nxt(root[rt],x);
201     }
202     int m=l+r>>1,ret;
203     if(op==1||op==2)
204     {
205         ret=0;
206         if(L<=m) ret+=query(rt<<1,l,m,op,L,R,x);
207         if(m<R) ret+=query(rt<<1|1,m+1,r,op,L,R,x);
208     }
209     if(op==4)
210     {
211         ret=-inf;
212         if(L<=m) ret=max(ret,query(rt<<1,l,m,op,L,R,x));
213         if(m<R) ret=max(ret,query(rt<<1|1,m+1,r,op,L,R,x));
214     }
215     if(op==5)
216     {
217         ret=inf;
218         if(L<=m) ret=min(ret,query(rt<<1,l,m,op,L,R,x));
219         if(m<R) ret=min(ret,query(rt<<1|1,m+1,r,op,L,R,x));
220     }
221     return ret;
222 }
223
224 int main()
225 {
226     srand(time(0));
227     scanf("%d%d",&n,&m);
228     build(1,1,n);
229     for(int i=1;i<=n;i++)
230     {
231         scanf("%d",a+i);
232         modify(1,1,n,i,a[i],1);
233     }

```

```

233 while(m-->0)
234 {
235     int op,l,r,k,pos;
236     scanf("%d",&op);
237     if(op==1)
238     {
239         scanf("%d%d%d",&l,&r,&k);
240         printf("%d\n",query(1,1,n,op,l,r,k)+1);
241     }
242     else if(op==2)
243     {
244         scanf("%d%d%d",&l,&r,&k);
245         int L=-inf,R=inf,mid;
246         while(L<R)
247         {
248             mid=(L+R+1)>>1;
249             if(query(1,1,n,1,l,r,mid)+1>k) R=mid-1;
250             else L=mid;
251         }
252         printf("%d\n",L);
253     }
254     else if(op==3)
255     {
256         scanf("%d%d",&pos,&k);
257         modify(1,1,n,pos,a[pos],-1);
258         a[pos]=k;
259         modify(1,1,n,pos,k,1);
260     }
261     else
262     {
263         scanf("%d%d%d",&l,&r,&k);
264         printf("%d\n",query(1,1,n,op,l,r,k));
265     }
266 }
267 return 0;

```

4.8.2 树状数组套线段树

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*
5 带单点修区间k小
6 用的时候注意下空间 时空 O(nlog^2(n))
7 外层 add(pos,x,y) 空间上为pos的点且值域上为x的点,加上y
8 query(l,r,k) 询问区间[l,r]里k小
9 内层 modify 值域线段树动态开点
10 query 值域线段树区间k小
11 VAL 值域大小
12 */
13
14 const int MAXN=200005;
15 int n,a[MAXN],X[MAXN],Y[MAXN],c1,c2,VAL;
16 struct SEG
17 {
18     int root[MAXN],lc[MAXN*500],rc[MAXN*500],cnt[MAXN*500],tot;
19     void modify(int &rt,int l,int r,int x,int y)
20     {
21         if(rt==0) rt=++tot;
22         cnt[rt]+=y;
23         if(l==r) return;

```

```

24     int m=l+r>>1;
25     if(x<=m) modify(lc[rt],l,m,x,y);
26     else modify(rc[rt],m+1,r,x,y);
27 }
28 int query(int l,int r,int k)
29 {
30     if(l==r) return l;
31     int sum=0,m=l+r>>1;
32     for(int i=0;i<c1;i++) sum-=cnt[lc[X[i]]];
33     for(int i=0;i<c2;i++) sum+=cnt[lc[Y[i]]];
34     if(sum>=k)
35     {
36         for(int i=0;i<c1;i++) X[i]=lc[X[i]];
37         for(int i=0;i<c2;i++) Y[i]=lc[Y[i]];
38         return query(l,m,k);
39     }
40     else
41     {
42         for(int i=0;i<c1;i++) X[i]=rc[X[i]];
43         for(int i=0;i<c2;i++) Y[i]=rc[Y[i]];
44         return query(m+1,r,k-sum);
45     }
46 }
47 }seg;
48
49 void add(int pos,int x,int y)
50 {
51     for(;pos<=n;pos+=pos&-pos) seg.modify(seg.root[pos],1,VAL,x,y);
52 }
53
54 int query(int l,int r,int k)
55 {
56     c1=c2=0;
57     for(int i=l-1;i;i-=i&-i) X[c1++]=seg.root[i];
58     for(int i=r;i;i-=i&-i) Y[c2++]=seg.root[i];
59     return seg.query(1,VAL,k);
60 }

```

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

6.6.3 欧拉降幂

6.6.4 一般积性函数求法

6.7 EX-GCD

6.8 CRT

6.9 N 次剩余

6.10 数论分块

6.11 高斯消元