

JYCの算法竞赛模板

求个 [star](#) (つ・ω・)っ”(つ・ω・)っ”(つ・ω・)っ”

目录

JYCの算法竞赛模板

缺省源

数据结构

树状数组

线段树

树链剖分（重链剖分）（LCA模板）

平衡树

吉司机线段树

图论

Tarjan

字符串

字符串哈希（多重哈希）

Z函数（扩展KMP）

数学

高精度

快速幂&矩阵乘法

扩展欧几里得

线性筛欧拉函数

组合数学

阶乘相关的初始化

错位排列

卡特兰数

斯特林数

第二类斯特林数

第一类斯特林数

杂项

火车头

模拟退火

写在最后（来自Benq大神的几句话）

缺省源

模板正式开始

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 // #define ONLINE
5 #ifndef ONLINE
6 #define debug(...) fprintf(stderr,##__VA_ARGS__)
7 #else
8 #define debug(...) ;
9 #endif
```

```

10
11 using LL=long long;
12 using PII=pair<int,int>;
13
14 const int mod=998244353;
15 mt19937 rng(chrono::system_clock::now().time_since_epoch().count());
16
17 //double关键字比大小
18 #define eps (1e-8)
19 inline int sign(const double& x){
20     if(fabs(x)<eps) return 0;
21     return x>0.0?1:-1;
22 }
23 inline int dcmp(const double& x,const double& y){
24     return sign(x-y);
25 }
26
27 //快读
28 template<typename T>
29 inline T READ(){
30     T x=0; bool f=0; char c=getchar();
31     while(c<'0' || c>'9') f|=(c=='-'),c=getchar();
32     while(c>='0' && c<='9') x=x*10+c-'0',c=getchar();
33     return f?-x:x;
34 }
35 inline int read(){return READ<int>();}
36 inline LL readLL(){return READ<LL>();}

```

数据结构

树状数组

```

1  template<typename T>//T is int or LL
2  class FenTree{
3  private:
4      int n;
5      vector<T>c;
6      inline int lowbit(const int& x){
7          return x&(-x);
8      }
9  public:
10     FenTree(int n_){
11         c.assign(n=n_,T(0));
12     }
13     void init(int n_){
14         c.assign(n=n_,T(0));
15     }
16     void add(int i,int x){
17         for(;i<n;i+=lowbit(i)) c[i]+=x;
18     }
19     T query(int i){
20         T res=0;
21         for(;i-=lowbit(i)) res+=c[i];
22         return res;
23     }
24 };

```

线段树

封装的不是很好，具体 *Info* 的传递还是得自己修改源码，没法直接传个 *class* 进来

初始化时，传入一个一维 *vector a* 以及它的长度 *n*，数据存储在下标 $1 \sim n$

```
1  #define ls (id<<1)
2  #define rs (id<<1|1)
3  class SGT{//以线段树维护区间最大值为例，支持区间修改
4  private:
5      struct Node{
6          int l,r;
7          int mx; bool tag;
8          friend Node operator +(const Node& A,const Node& B){
9              Node ret;
10             ret.l=A.l,ret.r=B.r;
11
12             //update the main content
13             ret.mx=max(A.mx,B.mx);
14             ret.tag=0;
15
16             return ret;
17         }
18     };
19     vector<Node>q;
20     void spread(int id){
21         if(q[id].l==q[id].r) return;
22         //spread the lazy tag
23         if(q[id].tag){
24             q[ls].mx=q[id].mx;
25             q[rs].mx=q[id].mx;
26             q[ls].tag=1;
27             q[rs].tag=1;
28             q[id].tag=0;
29         }
30     }
31     void build(const vector<int>& a,int l,int r,int id=1){
32         if(l==r){
33             q[id].l=l,q[id].r=r;
34
35             //init value
36             q[id].mx=a[l],q[id].tag=0;
37
38             return;
39         }
40         int mid=l+r>>1;
41         build(a,l,mid,ls);
42         build(a,mid+1,r,rs);
43         q[id]=q[ls]+q[rs];
44     }
45 public:
46     SGT(const vector<int>& a,int n){
47         q.resize(n*5);
48         build(a,1,n);
49     }
50     void init(const vector<int>& a,int n){
51         q.resize(n*5);
```

```

52     build(a,1,n);
53 }
54 void modify(int l,int r,int val,int id=1){
55     if(q[id].l==l && q[id].r==r){
56         //modify the value
57         q[id].mx=max(q[id].mx,val),q[id].tag=1;
58         return;
59     }
60     spread(id);
61     int mid=q[id].l+q[id].r>>1;
62     if(r<=mid) modify(l,r,val,ls);
63     else if(l>mid) modify(l,r,val,rs);
64     else modify(l,mid,val,ls),modify(mid+1,r,val,rs);
65     q[id]=q[ls]+q[rs];
66 }
67 int query(int l,int r,int id=1){
68     if(q[id].l==l && q[id].r==r) return q[id].mx;
69     spread(id);
70     int mid=q[id].l+q[id].r>>1;
71     if(r<=mid) return query(l,r,ls);
72     else if(l>mid) return query(l,r,rs);
73     return max(query(l,mid,ls),query(mid+1,r,rs));
74 }
75 int get(int pos,int id=1){
76     if(q[id].l==q[id].r) return q[id].mx;
77     spread(id);
78     int mid=q[id].l+q[id].r>>1;
79     if(pos<=mid) return get(pos,ls);
80     return get(pos,rs);
81 }
82 };
83 #undef ls
84 #undef rs

```

树链剖分（重链剖分）（LCA模板）

直接把[P3384【模板】重链剖分/树链剖分](#)的代码复制了一遍

因为原题有取模操作，抄模板的时候记得删去取模，删去取模，删去取模！！

也可以把树链剖分作为 *LCA* 的模板来使用，比倍增的写法优秀很多

```

1 void heavy_path_decomposition(){
2     int n=read(),T=read(),root=read();// mod=read();
3     vector<int>a(n+1);
4     for(int i=1;i<=n;i++) a[i]=read()%mod;//origin value
5     vector<vector<int>>e(n+1);
6     for(int i=1,u,v;i<=n;i++){
7         u=read(),v=read();
8         e[u].push_back(v);
9         e[v].push_back(u);
10    }
11    vector<int>dep(n+1),fa(n+1),siz(n+1),son(n+1);
12    auto dfs1=[&](auto self,int u,int pre)->void{
13        dep[u]=dep[pre]+1,fa[u]=pre,siz[u]=1;
14        for(int v:e[u]){
15            if(v==pre) continue;

```

```

16         self(self,v,u);
17         siz[u]+=siz[v];
18         if(siz[v]>siz[son[u]]) son[u]=v;
19     }
20 };
21 dfs1(dfs1,root,0);
22 vector<int>id(n+1),nw(n+1),top(n+1);
23 int timStamp=0;
24 auto dfs2=[&](auto self,int u,int t)->void{
25     id[u]++timStamp,nw[timStamp]=u,top[u]=t;
26     if(!son[u]) return;
27     self(self,son[u],t);
28     for(int v:e[u]){
29         if(v==fa[u] || v==son[u]) continue;
30         self(self,v,v);
31     }
32 };
33 dfs2(dfs2,root,root);
34 auto LCA=[&](int u,int v)->int{
35     while(top[u]!=top[v]){
36         if(dep[top[u]]<dep[top[v]]) swap(u,v);
37         u=fa[top[u]];
38     }
39     return dep[u]<dep[v]?u:v;
40 };
41 vector<int>sgt_init(n+1);
42 for(int i=1;i<=n;i++) sgt_init[i]=a[nw[i]];
43 SGT sgt(sgt_init,n);//sgt needs to support seg add, seg query
44 auto modify_path=[&](int u,int v,LL val)->void{
45     while(top[u]!=top[v]){
46         if(dep[top[u]]<dep[top[v]]) swap(u,v);
47         sgt.modify(id[top[u]],id[u],val);
48         u=fa[top[u]];
49     }
50     if(dep[u]<dep[v]) swap(u,v);
51     sgt.modify(id[v],id[u],val);
52 };
53 auto query_path=[&](int u,int v)->LL{
54     LL cnt=0ll;
55     while(top[u]!=top[v]){
56         if(dep[top[u]]<dep[top[v]]) swap(u,v);
57         cnt+=sgt.query(id[top[u]],id[u]);
58         u=fa[top[u]];
59     }
60     if(dep[u]<dep[v]) swap(u,v);
61     cnt+=sgt.query(id[v],id[u]);
62     return cnt%mod;
63 };
64 auto modify_tree=[&](int u,int val)->void{
65     sgt.modify(id[u],id[u]+siz[u]-1,val);
66 };
67 auto query_tree=[&](int u)->LL{
68     return sgt.query(id[u],id[u]+siz[u]-1);
69 };
70 for(int op,u,v,w;T--;){
71     op=read(),u=read();
72     if(op==1){
73         v=read(),w=read();

```

```

74         modify_path(u,v,w);
75     }
76     else if(op==2){
77         v=read();
78         printf("%lld\n",query_path(u,v));
79     }
80     else if(op==3){
81         w=read();
82         modify_tree(u,w);
83     }
84     else{//op==4
85         printf("%lld\n",query_tree(u));
86     }
87 }
88 }

```

平衡树

非常朴素的 *fhqTreap*, 用 *vector* 实现了一下动态开点

```

1  template<typename T>
2  class fhqTreap{
3  private:
4      struct Node{
5          int l,r,siz; LL rnd;
6          T val; LL sum;
7          Node(){
8              l=r=siz=0; rnd=0ll;
9              val=T(0); sum=0ll;
10         }
11         Node(int l_,int r_,int siz_,LL rnd_,T val_,LL sum_){
12             l=l_,r=r_,siz=siz_; rnd=rnd_;
13             val=val_; sum=sum_;
14         }
15     };
16     vector<Node>q;
17     int root,rootX,rootY,rootZ;
18     int New(T val){
19         Node new_node=Node(0,0,1,rng(),val,val);
20         q.push_back(new_node);
21         return q.size()-1;
22     }
23     void Update(int id){
24         q[id].siz=q[q[id].l].siz+q[q[id].r].siz+1;
25         q[id].sum=1ll*q[q[id].l].val+q[q[id].r].val+q[id].val;
26     }
27     void Split(int id,T key,int& idX,int& idY){
28         if(id==0){
29             idX=idY=0;
30             return;
31         }
32         if(q[id].val<=key){
33             idX=id;
34             Split(q[id].r,key,q[id].r,idY);
35         }
36         else{
37             idY=id;

```

```

38         Split(q[id].l, key, idx, q[id].l);
39     }
40     Update(id);
41 }
42 int Merge(int l, int r){
43     if(l==0 || r==0) return l+r;
44     if(q[l].rnd<=q[r].rnd){
45         q[r].l=Merge(l, q[r].l);
46         Update(r);
47         return r;
48     }
49     else{
50         q[l].r=Merge(q[l].r, r);
51         Update(l);
52         return l;
53     }
54 }
55 public:
56     fhqTreap(){
57         init();
58     }
59     void init(){
60         root=0; q.clear();
61         Node empty_node=Node();
62         q.push_back(empty_node);
63     }
64     void insert(T val){
65         Split(root, val, rootX, rootY);
66         root=Merge(Merge(rootX, New(val)), rootY);
67     }
68     void erase(T val){//actually, 'extract' may be more precise
69         Split(root, val, rootX, rootZ);
70         Split(rootX, val-1, rootX, rootY);
71         rootY=Merge(q[rootY].l, q[rootY].r);
72         root=Merge(Merge(rootX, rootY), rootZ);
73     }
74     T prev(T val){
75         Split(root, val-1, rootX, rootY);
76         int tmp=rootX;
77         while(q[tmp].r) tmp=q[tmp].r;
78         root=Merge(rootX, rootY);
79         return q[tmp].val;
80     }
81     T next(T val){
82         Split(root, val, rootX, rootY);
83         int tmp=rootY;
84         while(q[tmp].l) tmp=q[tmp].l;
85         root=Merge(rootX, rootY);
86         return q[tmp].val;
87     }
88     int rank(T val){//val's rank
89         Split(root, val-1, rootX, rootY);
90         int ans=q[rootX].siz+1;
91         root=Merge(rootX, rootY);
92         return ans;
93     }
94     T get(int rank){
95         int id=root;

```

```

96         while(1){
97             //          printf("id:%d val:%d l:%d lsiz:%d r:%d
rsiz:%d|rank:%d\n",id,q[id].val,q[id].l,q[q[id].l].siz,q[id].r,q[q[id].r].s
iz,rank);
98                 if(q[q[id].l].siz>=rank) id=q[id].l;
99                 else if(q[q[id].l].siz+1==rank) return q[id].val;
100                else{
101                    rank--(q[q[id].l].siz+1);
102                    id=q[id].r;
103                }
104            }
105        }
106        // void output(){
107        //     auto dfs=[&](auto self,int u)->void{
108        //         if(!u) return;
109        //         self(self,q[u].l);
110        //         debug("%d|val=%d ls=%d rs=%d\n",u,q[u].val,q[u].l,q[u].r);
111        //         self(self,q[u].r);
112        //     };
113        //     debug("output fhqTreap\n");
114        //     dfs(dfs,root);
115        //     debug("\n");
116        // }
117    };

```

吉司机线段树

把 wiki 里的代码改了改

```

1  const int inf=INT_MAX;
2
3  #define ls (id<<1)
4  #define rs (id<<1|1)
5  #define SPREAD_ID spread(id,q[id].l,q[id].r)
6
7  #define MODIFY(FUNC) \
8  SPREAD_ID; \
9  int mid=q[id].l+q[id].r>>1; \
10 if(r<=mid) FUNC(l,r,val,ls); \
11 else if(l>mid) FUNC(l,r,val,rs); \
12 else FUNC(l,mid,val,ls),FUNC(mid+1,r,val,rs); \
13 q[id]=q[ls]+q[rs]
14
15 #define ADD(A,B) A+B
16 #define QUERY(FUNC,QRY_FUNC) \
17 SPREAD_ID;\
18 int mid=q[id].l+q[id].r>>1; \
19 if(r<=mid) return FUNC(l,r,ls); \
20 else if(l>mid) return FUNC(l,r,rs); \
21 else return QRY_FUNC(FUNC(l,mid,ls),FUNC(mid+1,r,rs))
22
23 class JisGT{
24 private:
25     struct Node{
26         int l,r;
27         //count_max, count_min, tag_max, tag_min, tag_add
28         int mx,mx2,mn,mn2,cmx,cmn,tmx,tmn,tad;

```



```

29     LL sum;
30     friend Node operator +(const Node& A, const Node& B){
31         Node ret;
32         ret.l=A.l, ret.r=B.r;
33         ret.sum = A.sum + B.sum;
34         if (A.mx == B.mx) {
35             ret.mx = A.mx, ret.cmx = A.cmx + B.cmx;
36             ret.mx2 = max(A.mx2, B.mx2);
37         } else if (A.mx > B.mx) {
38             ret.mx = A.mx, ret.cmx = A.cmx;
39             ret.mx2 = max(A.mx2, B.mx);
40         } else {
41             ret.mx = B.mx, ret.cmx = B.cmx;
42             ret.mx2 = max(A.mx, B.mx2);
43         }
44         if (A.mn == B.mn) {
45             ret.mn = A.mn, ret.cmn = A.cmn + B.cmn;
46             ret.mn2 = min(A.mn2, B.mn2);
47         } else if (A.mn < B.mn) {
48             ret.mn = A.mn, ret.cmn = A.cmn;
49             ret.mn2 = min(A.mn2, B.mn);
50         } else {
51             ret.mn = B.mn, ret.cmn = B.cmn;
52             ret.mn2 = min(A.mn, B.mn2);
53         }
54         ret.tmx=-inf, ret.tmn=inf, ret.tad=0;
55         return ret;
56     }
57 };
58 vector<Node>q;
59 void spread_add(int id, int l, int r, int v) {
60     // 更新加法标记的同时, 更新 $\min$ 和 $\max$ 标记
61     q[id].sum += (r - l + 1ll) * v;
62     q[id].mx += v, q[id].mn += v;
63     if (q[id].mx2 != -inf) q[id].mx2 += v;
64     if (q[id].mn2 != inf) q[id].mn2 += v;
65     if (q[id].tmx != -inf) q[id].tmx += v;
66     if (q[id].tmn != inf) q[id].tmn += v;
67     q[id].tad += v;
68 }
69 void spread_min(int id, int tg) {
70     // 注意比较 $\max$ 标记
71     if (q[id].mx <= tg) return;
72     q[id].sum += (tg * 1ll - q[id].mx) * q[id].cmx;
73     if (q[id].mn2 == q[id].mx) q[id].mn2 = tg; // !!!
74     if (q[id].mn == q[id].mx) q[id].mn = tg; // !!!!
75     if (q[id].tmx > tg) q[id].tmx = tg; // 更新取 $\max$ 标记
76     q[id].mx = tg, q[id].tmn = tg;
77 }
78 void spread_max(int id, int tg) {
79     if (q[id].mn > tg) return;
80     q[id].sum += (tg * 1ll - q[id].mn) * q[id].cmn;
81     if (q[id].mx2 == q[id].mn) q[id].mx2 = tg;
82     if (q[id].mx == q[id].mn) q[id].mx = tg;
83     if (q[id].tmn < tg) q[id].tmn = tg;
84     q[id].mn = tg, q[id].tmx = tg;
85 }
86 void spread(int id, int l, int r){

```

```

87         if(q[id].l==q[id].r) return;
88         int mid=l+r>>1;
89         if (q[id].tad){
90             spread_add(ls, l, mid, q[id].tad);
91             spread_add(rs, mid + 1, r, q[id].tad);
92         }
93         if (q[id].tmx != -inf){
94             spread_max(ls, q[id].tmx);
95             spread_max(rs, q[id].tmx);
96         }
97         if (q[id].tmn != inf){
98             spread_min(ls, q[id].tmn);
99             spread_min(rs, q[id].tmn);
100        }
101        q[id].tmx = -inf, q[id].tmn = inf, q[id].tad = 0;
102    }
103    void build(const vector<int>& a,int l,int r,int id=1){
104        if(l==r){
105            q[id].l=l,q[id].r=r;
106            q[id].sum=q[id].mx = q[id].mn = a[l];
107            q[id].mx2=-inf, q[id].mn2 = inf;
108            q[id].cmx=q[id].cmn = 1;
109            q[id].tmx=-inf, q[id].tmn = inf, q[id].tad=0;
110            return;
111        }
112        int mid=l+r>>1;
113        build(a,l,mid,ls);
114        build(a,mid+1,r,rs);
115        q[id]=q[ls]+q[rs];
116    }
117    public:
118    JISGT(const vector<int>& a,int n){
119        q.resize(n*4);
120        build(a,1,n);
121    }
122    void init(const vector<int>& a,int n){
123        q.resize(n*4);
124        build(a,1,n);
125    }
126    void add(int l,int r,int val,int id=1){
127        if(q[id].l==l && q[id].r==r){
128            return spread_add(id,l,r,val);
129        }
130        MODIFY(add);
131    }
132    void tomin(int l,int r,int val,int id=1){
133        if(r<q[id].l || l>q[id].r || q[id].mx<=val) return;
134        if(l<=q[id].l && q[id].r<=r && q[id].mx2<val){
135            return spread_min(id,val);
136        }
137        MODIFY(tomin);
138    }
139    void tomax(int l,int r,int val,int id=1){
140        if(r<q[id].l || l>q[id].r || q[id].mn>=val) return;
141        if(l<=q[id].l && q[id].r<=r && q[id].mn2>val){
142            return spread_max(id,val);
143        }
144        MODIFY(tomax);

```

```

145     }
146     LL qsum(int l,int r,int id=1){
147         if(q[id].l==l && q[id].r==r) return q[id].sum;
148         QUERY(qsum,ADD);
149     }
150     LL qmax(int l,int r,int id=1){
151         if(q[id].l==l && q[id].r==r) return q[id].mx;
152         QUERY(qmax,max);
153     }
154     LL qmin(int l,int r,int id=1){
155         if(q[id].l==l && q[id].r==r) return q[id].mn;
156         QUERY(qmin,min);
157     }
158 };
159 #undef ls
160 #undef rs
161 #undef SPREAD_ID
162 #undef MODIFY
163 #undef ADD
164 #undef QUERY

```

图论

Tarjan

```

1  namespace Tarjan{
2      vector<int>dfn(n+1),low(n+1),inStack(n+1),scc(n+1),siz(n+1);
3      int timeStamp=0,col=0;
4      stack<int>stk;
5      auto tarjan=[&](auto self,int u)->void{
6          low[u]=dfn[u]=++timeStamp;
7          stk.push(u),inStack[u]=1;
8          for(int v:e[u]){
9              if(!dfn[v]){
10                 self(self,v);
11                 low[u]=min(low[u],low[v]);
12             }
13             else if(inStack[v]){
14                 low[u]=min(low[u],low[v]);
15             }
16         }
17         if(dfn[u]==low[u]){
18             ++col;
19             while(stk.top()!=u){
20                 siz[scc[stk.top()]=col]++;
21                 inStack[stk.top()]=0; stk.pop();
22             }
23             siz[scc[stk.top()]=col]++;
24             inStack[stk.top()]=0; stk.pop();
25         }
26     };
27 }

```

字符串

字符串哈希（多重哈希）

修改 *HL* (*hash_layer*) 以决定使用几重哈希

init 时，传入字符数组首地址，字符串长度（以及是否倒着求哈希，0和缺省为正着求，1为倒着求）

注意字符数组内数据的下标是 $1 \sim n$ ， $1 \sim n$ ， $1 \sim n$!!!

```
1  const int HL=2;//hash layer
2  namespace HC{//Hash Const
3      const int P[4]={13331,233,131,19260817};
4      const int MOD[4]={(int)(1e9+7),998244353,1004535809,754974721};
5      LL ksm[N][4];
6      void init(int use=HL){
7          for(int j=0;j<use;j++){
8              ksm[0][j]=1;
9              for(int i=1;i<N;i++){
10                 ksm[i][j]=(ksm[i-1][j]*P[j])%MOD[j];
11             }
12         }
13     }
14 }
15 template<int T>//T must be a constant
16 class Hash{
17 private:
18     vector<array<LL,T>>h;
19     bool sign;//0:normal 1:reverse
20 public:
21     Hash(char *s,int n,bool sign_=0){
22         init(s,n,sign_);
23     }
24     void init(char *s,int n,bool sign_=0){//s stores at pos [1,n], and just
give para s (NOT s+1)
25         h.resize(n+2);
26         sign=sign_;
27         if(!sign){
28             for(int j=0;j<T;j++){
29                 h[0][j]=0;
30                 for(int i=1;i<=n;i++){
31                     h[i][j]=(h[i-1][j]*HC::P[j]+s[i]-'a'+1)%HC::MOD[j];
32                 }
33             }
34         }
35         else{
36             for(int j=0;j<T;j++){
37                 h[n+1][j]=0;
38                 for(int i=n;i>0;i--){
39                     h[i][j]=(h[i+1][j]*HC::P[j]+s[i]-'a'+1)%HC::MOD[j];
40                 }
41             }
42         }
43     }
44     array<LL,T>calc(const int& l,const int& r){
45         array<LL,T>ret;
46         if(!sign){
47             for(int j=0;j<T;j++){
48                 ret[j]=h[r][j]-h[l-1][j]*HC::ksm[r-l+1][j];
49             }
50         }
51         else{
52             for(int j=0;j<T;j++){
53                 ret[j]=h[l][j]-h[r+1][j]*HC::ksm[r-l+1][j];
54             }
55         }
56     }
57 }
```

```

50     }
51     else{
52         for(int j=0;j<T;j++){
53             ret[j]=h[l][j]-h[r+1][j]*HC::ksm[r-l+1][j];
54         }
55     }
56     for(int j=0;j<T;j++){
57         ret[j]=(ret[j]%HC::MOD[j]+HC::MOD[j])%HC::MOD[j];
58     }
59     return ret;
60 }
61 static bool check(const array<LL,T>& a,const array<LL,T>& b){
62     for(int i=0;i<T;i++){
63         if(a[i]!=b[i]) return 0;
64     }
65     return 1;
66 }
67 };

```

Z函数 (扩展KMP)

约定：字符串下标以 0 为起点

定义函数 $z[i]$ 表示 s 和 $s[i, n - 1]$ （即以 $s[i]$ 开头的后缀）的最长公共前缀（LCP）的长度，则 z 被成为 s 的 **Z函数**。特别的， $z[0] = 0$

```

1  vector<int>z_function(string s){
2      int n=(int)s.length();
3      vector<int>z(n);
4      for (int i=1,l=0,r=0;i<n;i++){
5          if(i<=r&& z[i-l]<r-i+1){
6              z[i]=z[i-l];
7          }
8          else{
9              z[i]=max(0,r-i+1);
10             while(i+z[i]<n && s[z[i]]==s[i+z[i]]) z[i]++;
11         }
12         if(i+z[i]-1>r) l=i,r=i+z[i]-1;
13     }
14     return z;
15 }

```

数学

高精度

[BigIntTiny Template Link](#)，顺便从大佬的 README 里放个 Features preview，不支持的功能就现写吧。

operators	BigIntHex	BigIntDec	BigIntMini	BigIntTiny
constructor Bigint	✓	✓	✓	✓
constructor int	✓	✓	✓	✓
constructor char*	✓	✓	✗	✗
constructor string	✓	✓	✓	✓
=Bigint	✓	✓	✓	✓
=int	✓	✓	✓	✓
=string	✓	✓	✓	✓
=char*	✓	✓	✓	✗
<, ==, >, <=, >=, != Bigint	✓	✓	✓	✓
+, -, *, /, % int	✗	✗	✗	✓
+=, -=, *=, /=, %= int	✗	✗	✗	✗
+, -, *, /, % Bigint	✓	✓	✓	✓
+=, -=, *=, /=, %= Bigint	✓	✓	✗	✗
Base conversion	✓	✓	✗	✗
Efficiency	★★★★	★★★	★★	★

```

1  struct BigIntTiny {
2      int sign;
3      std::vector<int> v;
4
5      BigIntTiny() : sign(1) {}
6      BigIntTiny(const std::string &s) { *this = s; }
7      BigIntTiny(int v) {
8          char buf[21];
9          sprintf(buf, "%d", v);
10         *this = buf;
11     }
12     void zip(int unzip) {
13         if (unzip == 0) {
14             for (int i = 0; i < (int)v.size(); i++)
15                 v[i] = get_pos(i * 4) + get_pos(i * 4 + 1) * 10 + get_pos(i
16 * 4 + 2) * 100 + get_pos(i * 4 + 3) * 1000;
17         } else
18             for (int i = (v.resize(v.size() * 4), (int)v.size() - 1), a; i
19 >= 0; i--)
20                 a = (i % 4 >= 2) ? v[i / 4] / 100 : v[i / 4] % 100, v[i] =
21 (i & 1) ? a / 10 : a % 10;
22         setsign(1, 1);
23     }
24     int get_pos(unsigned pos) const { return pos >= v.size() ? 0 : v[pos]; }
25 }
26
27 BigIntTiny &setsign(int newsign, int rev) {
28     for (int i = (int)v.size() - 1; i > 0 && v[i] == 0; i--)

```

```

24         v.erase(v.begin() + i);
25         sign = (v.size() == 0 || (v.size() == 1 && v[0] == 0)) ? 1 : (rev ?
newsign * sign : newsign);
26         return *this;
27     }
28     std::string to_str() const {
29         BigIntTiny b = *this;
30         std::string s;
31         for (int i = (b.zip(1), 0); i < (int)b.v.size(); ++i)
32             s += char(*(b.v.rbegin() + i) + '0');
33         return (sign < 0 ? "-" : "") + (s.empty() ? std::string("0") : s);
34     }
35     bool absless(const BigIntTiny &b) const {
36         if (v.size() != b.v.size()) return v.size() < b.v.size();
37         for (int i = (int)v.size() - 1; i >= 0; i--)
38             if (v[i] != b.v[i]) return v[i] < b.v[i];
39         return false;
40     }
41     BigIntTiny operator-() const {
42         BigIntTiny c = *this;
43         c.sign = (v.size() > 1 || v[0]) ? -c.sign : 1;
44         return c;
45     }
46     BigIntTiny &operator=(const std::string &s) {
47         if (s[0] == '-')
48             *this = s.substr(1);
49         else {
50             for (int i = (v.clear(), 0); i < (int)s.size(); ++i)
51                 v.push_back(*(s.rbegin() + i) - '0');
52             zip(0);
53         }
54         return setsign(s[0] == '-' ? -1 : 1, sign = 1);
55     }
56     bool operator<(const BigIntTiny &b) const {
57         return sign != b.sign ? sign < b.sign : (sign == 1 ? absless(b) :
b.absless(*this));
58     }
59     bool operator==(const BigIntTiny &b) const { return v == b.v && sign ==
b.sign; }
60     BigIntTiny &operator+=(const BigIntTiny &b) {
61         if (sign != b.sign) return *this = (*this) - -b;
62         v.resize(std::max(v.size(), b.v.size()) + 1);
63         for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
64             carry += v[i] + b.get_pos(i);
65             v[i] = carry % 10000, carry /= 10000;
66         }
67         return setsign(sign, 0);
68     }
69     BigIntTiny operator+(const BigIntTiny &b) const {
70         BigIntTiny c = *this;
71         return c += b;
72     }
73     void add_mul(const BigIntTiny &b, int mul) {
74         v.resize(std::max(v.size(), b.v.size()) + 2);
75         for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
76             carry += v[i] + b.get_pos(i) * mul;
77             v[i] = carry % 10000, carry /= 10000;
78         }

```

```

79     }
80     BigIntTiny operator-(const BigIntTiny &b) const {
81         if (b.v.empty() || b.v.size() == 1 && b.v[0] == 0) return *this;
82         if (sign != b.sign) return (*this) + -b;
83         if (absless(b)) return -(b - *this);
84         BigIntTiny c;
85         for (int i = 0, borrow = 0; i < (int)v.size(); i++) {
86             borrow += v[i] - b.get_pos(i);
87             c.v.push_back(borrow);
88             c.v.back() -= 10000 * (borrow >= 31);
89         }
90         return c.setsign(sign, 0);
91     }
92     BigIntTiny operator*(const BigIntTiny &b) const {
93         if (b < *this) return b * *this;
94         BigIntTiny c, d = b;
95         for (int i = 0; i < (int)v.size(); i++, d.v.insert(d.v.begin(), 0))
96             c.add_mul(d, v[i]);
97         return c.setsign(sign * b.sign, 0);
98     }
99     BigIntTiny operator/(const BigIntTiny &b) const {
100         BigIntTiny c, d;
101         BigIntTiny e=b;
102         e.sign=1;
103
104         d.v.resize(v.size());
105         double db = 1.0 / (b.v.back() + (b.get_pos((unsigned)b.v.size() -
106 2) / 1e4) +
107                             (b.get_pos((unsigned)b.v.size() - 3) + 1) /
108                             1e8);
109         for (int i = (int)v.size() - 1; i >= 0; i--) {
110             c.v.insert(c.v.begin(), v[i]);
111             int m = (int)((c.get_pos((int)e.v.size()) * 10000 +
112 c.get_pos((int)e.v.size() - 1)) * db);
113             c = c - e * m, c.setsign(c.sign, 0), d.v[i] += m;
114             while (!(c < e))
115                 c = c - e, d.v[i] += 1;
116         }
117         return d.setsign(sign * b.sign, 0);
118     }
119     BigIntTiny operator%(const BigIntTiny &b) const { return *this - *this
120 / b * b; }
121     bool operator>(const BigIntTiny &b) const { return b < *this; }
122     bool operator<=(const BigIntTiny &b) const { return !(b < *this); }
123     bool operator>=(const BigIntTiny &b) const { return !(*this < b); }
124     bool operator!=(const BigIntTiny &b) const { return !(*this == b); }
125 };

```

快速幂&矩阵乘法

矩阵乘法中， f 为答案矩阵，是一维长为 n 的 *vector*； a 为转移矩阵，是二维 $n \cdot n$ 的 *vector*

传参数时，直接传入当前的 f 与 a 即可

```

1 LL ksm(LL a,LL b){
2     a%=mod; LL ret=1;
3     while(b){

```



```

4         if(b&1) ret=ret*a%mod;
5         a=a*a%mod;
6         b>>=1;
7     }
8     return ret;
9 }
10
11 void mul(const vector<vector<LL>>& a,vector<LL>& f,int n){//f=a*f
12     vector<LL>b(n,0);
13     for(int i=0;i<n;i++){
14         for(int j=0;j<n;j++){
15             b[i]=(b[i]+a[i][j]*f[j]%mod)%mod;
16         }
17     }
18     swap(f,b);
19 }
20 void self_mul(vector<vector<LL>>& a,int n){//a=a*a
21     vector b(n,vector<LL>(n,0));
22     for(int k=0;k<n;k++){
23         for(int i=0;i<n;i++){
24             for(int j=0;j<n;j++){
25                 b[i][j]=(b[i][j]+a[i][k]*a[k][j]%mod)%mod;
26             }
27         }
28     }
29     swap(a,b);
30 }

```

扩展欧几里得

最终会给出一组满足 $a \cdot x + b \cdot y = g$ 的解

- 迭代法实现

```

1 int exgcd(int a, int b, int &x, int &y) {
2     if (!b) {
3         x = 1;
4         y = 0;
5         return a;
6     }
7     int d = Exgcd(b, a % b, x, y);
8     int t = x;
9     x = y;
10    y = t - (a / b) * y;
11    return d;
12 }

```

- 非迭代法实现

```

1  int exgcd(int a, int b, int& x, int& y) {
2      x = 1, y = 0;
3      int x1 = 0, y1 = 1, a1 = a, b1 = b;
4      while (b1) {
5          int q = a1 / b1;
6          tie(x, x1) = make_tuple(x1, x - q * x1);
7          tie(y, y1) = make_tuple(y1, y - q * y1);
8          tie(a1, b1) = make_tuple(b1, a1 - q * b1);
9      }
10     return a1;
11 }

```

线性筛欧拉函数

P.S. 以下 N 均为据题意设定好的一个常数

```

1  bool notP[N]={}; int phi[N], mn_p[N]={1};
2  vector<int> prime;
3  void init_prime(const int& n=N){ //init [1,n-1]
4      phi[1]=1;
5      for(int i=2; i<n; i++){
6          if(!notP[i]){
7              prime.push_back(mn_p[i]=i);
8              phi[i]=i-1;
9          }
10         for(const int& p:prime){
11             if(i>(n-1)/p) break;
12             notP[i*p]=1, mn_p[i*p]=p;
13             if(i%p==0){
14                 phi[i*p]=phi[i]*p;
15                 break;
16             }
17             phi[i*p]=phi[i]*(p-1);
18         }
19     }
20 }

```

组合数学

阶乘相关的初始化

这个给模板只用来充数的qwq

```

1  namespace Fac{//factorial
2      LL fc[N]={1}, fc_inv[N]={1};
3      void init(){
4          for(int i=1; i<N; i++){
5              fc[i]=fc[i-1]*i%mod;
6              fc_inv[i]=fc_inv[i-1]*ksm(i, mod-2)%mod;
7          }
8      }
9      LL F(const int& x){
10         return fc[x];
11     }
12     LL P(const int& x, const int& y){

```

```

13         return fc[x]*fc_inv[x-y]%mod;
14     }
15     LL C(const int& x,const int& y){
16         return fc[x]*fc_inv[y]%mod*fc_inv[x-y]%mod;
17     }
18 }

```

错位排列

- 递推公式（两种任选即可）

$$D_n = (n-1) \cdot (D_{n-1} + D_{n-2})$$

$$D_n = n \cdot D_{n-1} + (-1)^n$$

其中 D 的前几项是 $D_1 = 0, D_2 = 1, D_3 = 9, D_4 = 44, D_5 = 265$ ([OEIS A000166](#))

- 其他关系

错位排列数有一个向下取整的简单表达式，增长速度与阶乘仅相差常数：

$$D_n = \left\lfloor \frac{n!}{e} \right\rfloor$$

随着元素数量的增加，形成错位排列的概率 P 接近：

$$P = \lim_{n \rightarrow \infty} \frac{D_n}{n!} = \frac{1}{e}$$

卡特兰数

H_n = 进栈序列为 $1, 2, 3, \dots, n$ 的栈的出栈序列个数，以下是一些常见公式

$$H_n = \frac{\binom{2n}{n}}{n+1} (n \geq 2, n \in \mathbb{N}_+)$$

$$H_n = \begin{cases} \sum_{i=1}^n H_{i-1} H_{n-i} & n \geq 2, n \in \mathbb{N}_+ \\ 1 & n = 0, 1 \end{cases}$$

$$H_n = \frac{H_{n-1}(4n-2)}{n+1}$$

$$H_n = \binom{2n}{n} - \binom{2n}{n-1}$$

斯特林数

第二类斯特林数

$\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}, S(n, k)$, 表示将 n 个两两不同的元素，划分为 k 个互不区分的非空子集的方案数

- 递推公式

$$\left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\} = \left\{ \begin{smallmatrix} n-1 \\ m-1 \end{smallmatrix} \right\} + k \left\{ \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right\}$$

边界是 $\left\{ \begin{smallmatrix} n \\ 0 \end{smallmatrix} \right\} = [n = 0]$

- 通项公式

$$\left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\} = \sum_{i=0}^m \frac{(-1)^{m-i} \cdot i^n}{i! \cdot (m-i)!}$$

待补充：同一行/列的第二类斯特林数的计算

第一类斯特林数

$\left[\begin{smallmatrix} n \\ k \end{smallmatrix} \right]$, $s(n, k)$, 表示将 n 个两两不同的元素, 划分为 k 个互不区分的非空轮换的方案数

一个轮换即一个首尾相接的环形排列, 如 $[A, B, C, D] = [B, C, D, A] = [C, D, A, B] = [D, A, B, C]$, 即两个可以通过旋转而相互得到的轮换是等价的。但是, 翻转不算旋转, 如 $[A, B, C, D] \neq [D, C, B, A]$

- 递推公式

$$\left[\begin{smallmatrix} n \\ m \end{smallmatrix} \right] = \left[\begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right] + (n-1) \left[\begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right]$$

边界是 $\left[\begin{smallmatrix} n \\ 0 \end{smallmatrix} \right] = [n = 0]$

- 通项公式

没有

待补充: 同一行/列的第一类斯特林数的计算

杂项

火车头

娱乐用, 用来平时做题凹卡常题

```
1 #pragma GCC target("avx")
2 #pragma GCC optimize(1)
3 #pragma GCC optimize(2)
4 #pragma GCC optimize(3)
5 #pragma GCC optimize("ofast")
6 #pragma GCC optimize("inline")
7 #pragma GCC optimize("-fgcse")
8 #pragma GCC optimize("-fgcse-lm")
9 #pragma GCC optimize("-fipa-sra")
10 #pragma GCC optimize("-ftree-pre")
11 #pragma GCC optimize("-ftree-vrp")
12 #pragma GCC optimize("-fpeephole2")
13 #pragma GCC optimize("-ffast-math")
14 #pragma GCC optimize("-fsched-spec")
15 #pragma GCC optimize("unroll-loops")
16 #pragma GCC optimize("-falign-jumps")
17 #pragma GCC optimize("-falign-loops")
18 #pragma GCC optimize("-falign-labels")
19 #pragma GCC optimize("-fdevirtualize")
20 #pragma GCC optimize("-fcaller-saves")
21 #pragma GCC optimize("-fcrossjumping")
22 #pragma GCC optimize("-fthread-jumps")
23 #pragma GCC optimize("-funroll-loops")
24 #pragma GCC optimize("-fwhole-program")
25 #pragma GCC optimize("-freorder-blocks")
26 #pragma GCC optimize("-fschedule-insns")
27 #pragma GCC optimize("inline-functions")
28 #pragma GCC optimize("-ftree-tail-merge")
29 #pragma GCC optimize("-fschedule-insns2")
30 #pragma GCC optimize("-fstrict-aliasing")
31 #pragma GCC optimize("-fstrict-overflow")
32 #pragma GCC optimize("-falign-functions")
```

```

33 #pragma GCC optimize("-fcse-skip-blocks")
34 #pragma GCC optimize("-fcse-follow-jumps")
35 #pragma GCC optimize("-fsched-interblock")
36 #pragma GCC optimize("-fpartial-inlining")
37 #pragma GCC optimize("no-stack-protector")
38 #pragma GCC optimize("-freorder-functions")
39 #pragma GCC optimize("-findirect-inlining")
40 #pragma GCC optimize("-fhoist-adjacent-loads")
41 #pragma GCC optimize("-frerun-cse-after-loop")
42 #pragma GCC optimize("inline-small-functions")
43 #pragma GCC optimize("-finline-small-functions")
44 #pragma GCC optimize("-ftree-switch-conversion")
45 #pragma GCC optimize("-foptimize-sibling-calls")
46 #pragma GCC optimize("-fexpensive-optimizations")
47 #pragma GCC optimize("-funsafe-loop-optimizations")
48 #pragma GCC optimize("inline-functions-called-once")
49 #pragma GCC optimize("-fdelete-null-pointer-checks")

```

模拟退火

用一句话概括过程：如果新状态的解更优则修改答案，否则以一定的概率接收新状态

定义当前温度为 T ，新状态 S' 和已知状态 S （新状态由已知状态通过随机的方式得到）之间的能量（值）差为 ΔE ($\Delta E \geq 0$)，则发生状态转移（修改最优解）的概率为

$$P(\Delta E) = \begin{cases} 1 & S' \text{ is better than } S \\ \exp(-\Delta E/T) & \text{otherwise} \end{cases}$$

$P.S.$

1. 这里 E 越小代表越接近最优解，实际若 E 越大越接近最优解，可考虑把 $-E$ 作为能量值函数
2. 为了使得解更为精确，通常不直接取当前解作为答案，而是在退火过程中维护遇到的所有解的最优值
3. 有时为了使得到的解更有质量，会在模拟退火结束后，以当前温度在得到的解附近多次随机状态，尝试得到更优的解（其过程与模拟退火相似）

以下代码以[P1337 \[JSOI2004\] 平衡点 / 吊打XXX](#)（求 n 个点的带权费马点）为例（改了改 wiki 里的代码）

```

1  mt19937 rng(chrono::system_clock::now().time_since_epoch().count());
2  double Rand(){return (long double)rng()/UINT_MAX;} //生成一个[0,1]的小数
3  int Rand(int l,int r){return rng%(r-l+1)+l;} //生成一个[l,r]的整数
4
5  #define N 1010
6  int n,x[N],y[N],w[N];
7  double ansx,ansy,dis;
8
9  //calc返回能量值函数，在calc中更新答案
10 double calc(double xx,double yy){
11     double res=0;
12     for (int i=1;i<=n;i++){
13         double dx=x[i]-xx,dy=y[i]-yy;
14         res+=sqrt(dx*dx+dy*dy)*w[i];
15     }
16     if(res<dis) dis=res,ansx=xx,ansy=yy;
17     return res;
18 }

```

```

19
20 void SA(){//simulated_annealing
21     double nowx=ansx,nowy=ansy;
22     double eps_t=1e-4;
23     //模拟退火参数，起始温度，终止温度，每次乘以的系数
24     for(auto t=2e5;t>eps_t;t*=0.99){
25         double nextx=nowx+t*(Rand()*2-1);
26         double nexty=nowy+t*(Rand()*2-1);
27         double delta=calc(nextx,nexty)-calc(nowx,nowy);
28         if(exp(-delta/t)>Rand()) nowx=nextx,nowy=nexty;
29     }
30     //在得到的解附近多次随机状态，尝试得到更优的解
31     for (int i=1;i<=1000;i++){
32         double nextx=ansx+eps_t*(Rand()*2-1);
33         double nexty=ansy+eps_t*(Rand()*2-1);
34         calc(nextx,nexty);
35     }
36 }
37
38 int main() {
39     n=read();
40     for(int i=1;i<=n;i++){
41         x[i]=read(),y[i]=read(),w[i]=read();
42         ansx+=x[i],ansy+=y[i];
43     }
44     ansx/=n,ansy/=n,dis=calc(ansx,ansy);
45     //卡时技巧，填一个比题目时限略小的数
46     while((double)clock()/CLOCKS_PER_SEC<0.9) SA();
47     printf("%.31f %.31f\n", ansx, ansy);
48     return 0;
49 }

```

写在最后（来自Benq大神的几句话）

```

1  stuff you should look for
2  int overflow, array bounds
3  special cases (n=1?)
4  do smth instead of nothing and stay organized
5  WRITE STUFF DOWN
6  DON'T GET STUCK ON ONE APPROACH

```