1. 可持久化Treap
2. SA
3. Manacher
4. SBT
5. Dinic
6. 点分治
7. CDQ分治 最大最小曼哈顿距离
8. Euler筛
9. AC自动机
10. 扩展kmp
11. 左偏树
12. 费用流
13. 补图最短路 set优化
14. KD tree
15. KM算法 二分图最大（小）权匹配
16. 最小圆覆盖
17. KMP
18. 一类割意义下的网络流建模

**1、Treap\_durable:**

#define S(x) ((x)?((x)->size):0)

#define RAND(x) (double(rand())/RAND\_MAX <= (x))

struct Node

{

int size; Node \*l,\*r;

char c; bool rev;

Node(){}

Node(char \_c){c = \_c,size = 1,l = r = 0,rev = 0;}

Node(int \_size,Node \*\_l,Node \*\_r,char \_c,bool \_rev){size = \_size,c = \_c,l = \_l,r = \_r,rev = \_rev;}

}pool[maxn];

Node\* root;

inline void PutRev(Node \*x)

{

swap(x->l,x->r);

x->rev = 0;

if (x->l)

{

pool[++tail] = Node(x->l->size,x->l->l,x->l->r,x->l->c,(x->l->rev)^1);

x->l = &pool[tail];

}

if (x->r)

{

pool[++tail] = Node(x->r->size,x->r->l,x->r->r,x->r->c,(x->r->rev)^1);

x->r = &pool[tail];

}

}

inline void PushUp(Node \*x)

{

x->size = S(x->l) + S(x->r) + 1;

}

void Split(Node \*x,int p,Node \*&L,Node \*&R)

{

if (!x)

{

L = R = 0;

return;

}

if (x->rev) PutRev(x);

pool[++tail] = Node(x->c);

Node \*cur = &pool[tail];

if (S(x->l) >= p)

{

Split(x->l,p,L,R);

cur->l = R, cur->r = x->r;

PushUp(cur);

R = cur;

} else

{

Split(x->r,p-S(x->l)-1,L,R);

cur->l = x->l, cur->r = L;

PushUp(cur);

L = cur;

}

}

Node \*Merge(Node \*L,Node \*R)

{

if (!L) return R;

if (!R) return L;

if (L->rev) PutRev(L);

if (R->rev) PutRev(R);

Node \*cur;

if (RAND((double)L->size/(L->size+R->size)))

{

pool[++tail] = Node(L->c);

cur = &pool[tail];

cur->l = L->l, cur->r = Merge(L->r,R);

PushUp(cur);

} else

{

pool[++tail] = Node(R->c);

cur = &pool[tail];

cur->l = Merge(L,R->l), cur->r = R->r;

PushUp(cur);

}

return cur;

}

char Select(Node \*x,int k)

{

if (x->rev) PutRev(x);

int size = S(x->l);

if (k == size+1) return x->c;

return k<=size ? Select(x->l,k):Select(x->r,k-size-1);

}

Node \*A,\*B,\*C,\*D;

void Insert()

{

int x;char c;

scanf("%d %c",&x,&c);

Split(root,x,A,B);

pool[++tail] = Node(c);

root = Merge(A,Merge(&pool[tail],B));

}

void Delete()

{

int l,r;

scanf("%d%d",&l,&r);

Split(root,r,A,B);

Split(A,l-1,C,D);

root = Merge(C,B);

}

void Copy()

{

int l,r,x;

scanf("%d%d%d",&l,&r,&x);

Split(root,r,D,C);

Split(D,l-1,A,B);

Split(root,x,A,C);

root = Merge(A,Merge(B,C));

}

void Reverse()

{

int l,r;

scanf("%d%d",&l,&r);

Split(root,r,D,C);

Split(D,l-1,A,B);

if (B) B->rev ^= 1;

root = Merge(A,Merge(B,C));

}

void Query()

{

int k;

scanf("%d",&k);

printf("%c",Select(root,k));

}

**2、SA:**

int main(void)

{

cin >> (s+1);

n = strlen(s+1);

memset(tank,0,sizeof tank);

fo(i,1,n) tank[s[i]] ++;

fo(i,1,255) tank[i] += tank[i-1];

fo(i,1,n) sa[tank[s[i]]--] = i;

int j = 0;

fo(i,1,n)

{

if (i == 1 || s[sa[i]] != s[sa[i-1]]) j ++;

rank[sa[i]] = j;

}

for (int k = 1;k <= n;k <<= 1)

{

j = 0;

fo(i,n+1,n+k) tmp[++j] = i - k;

fo(i,1,n) if (sa[i] > k) tmp[++j] = sa[i] - k;

memset(tank,0,sizeof tank);

fo(i,1,n) tank[rank[tmp[i]]] ++;

fo(i,1,n) tank[i] += tank[i-1];

fd(i,n,1) sa[tank[rank[tmp[i]]]--] = tmp[i];

copy(rank+1,rank+1+n,rk+1);

j = 0;

fo(i,1,n)

{

if ((i == 1) || (rk[sa[i]] != rk[sa[i-1]]) || (rk[sa[i]] == rk[sa[i-1]] && rk[sa[i]+k] != rk[sa[i-1]+k])) j ++;

rank[sa[i]] = j;

}

}

fo(i,1,n)

{

height[rank[i]] = max(0,height[rank[i-1]]-1);

while(s[i+height[rank[i]]]==s[sa[rank[i]-1]+height[rank[i]]])height[rank[i]] ++;

}

int ans = 0;

fo(i,1,n) if (height[i] > height[i-1]) ans += height[i] - height[i-1];

cout<<ans<<endl;

return 0;

}

**3、Manacher:**

void Manacher()

{

int id = 0;

fo(i,1,N)

{

if (id+r[id] > i) r[i] = min(id+r[id]-i,r[2\*id-i]);

else r[i] = 0;

while (i+r[i]+1 <= N && i-r[i]-1 >= 1 && s[i+r[i]+1] == s[i-r[i]-1]) r[i] ++;

if (i+r[i] > id+r[id]) id = i;

}

}

**4、SBT:**

inline void L\_rotate(int &t)

{

int k = right[t];

right[t] = left[k];

left[k] = t;

size[k] = size[t];

size[t] = size[left[t]] + size[right[t]] + 1;

t = k;

}

inline void R\_rotate(int &t)

{

int k = left[t];

left[t] = right[k];

right[k] = t;

size[k] = size[t];

size[t] = size[left[t]] + size[right[t]] + 1;

t = k;

}

inline void maintain(int &t,bool flag)

{

if (!flag)

{

if (size[left[left[t]]] > size[right[t]]) R\_rotate(t);

else if (size[right[left[t]]] > size[right[t]]) L\_rotate(left[t]), R\_rotate(t);

else return;

} else

{

if (size[right[right[t]]] > size[left[t]]) L\_rotate(t);

else if (size[left[right[t]]] > size[left[t]]) R\_rotate(right[t]), L\_rotate(t);

else return;

}

maintain(left[t],0);

maintain(right[t],1);

maintain(t,0);

maintain(t,1);

}

inline void Insert(int &t,int v)

{

if (!t)

{

t = ++ tot;

size[t] = 1;

left[t] = right[t] = 0;

key[t] = v;

} else

{

size[t] ++;

if (v < key[t]) Insert(left[t],v);

else Insert(right[t],v);

maintain(t,v >= key[t]);

}

}

inline int Delete(int &t,int v)

{

size[t] --;

if ((key[t] == v) || (v < key[t] && !left[t]) || (v >= key[t] && !right[t]))

{

int ret = key[t];

if (!left[t] || !right[t]) t = left[t] + right[t];

else key[t] = Delete(left[t],key[t]+1);

return ret;

}

if (v < key[t]) return Delete(left[t],v);

else return Delete(right[t],v);

}

inline int Rank(int t,int v)

{

if (!t) return 0;

if (v <= key[t]) return Rank(left[t],v);

else return size[left[t]] + 1 + Rank(right[t],v);

}

inline int Select(int t,int v)

{

int temp = size[left[t]] + 1;

if (temp == v) return key[t];

if (v <= temp) return Select(left[t],v);

else return Select(right[t],v-temp);

}

inline int Pred(int t,int v)

{

if (!t) return INF;

if (v <= key[t]) return Pred(left[t],v);

else

{

int temp = Pred(right[t],v);

if (temp == INF) return key[t];

else return temp;

}

}

inline int Succ(int t,int v)

{

if (!t) return INF;

if (v >= key[t]) return Succ(right[t],v);

else

{

int temp = Succ(left[t],v);

if (temp == INF) return key[t];

else return temp;

}

}

**5、Dinic:**

inline bool bfs()

{

static int list[maxn\*maxn];

int st,en;

list[st=en=1] = source;

memset(lab,255,sizeof lab);

lab[source] = 0;

while (st <= en)

{

int x = list[st];

for(int i = a[x];i;i = c[i])

if (d[i] && lab[b[i]] == -1)

{

lab[b[i]] = lab[x] + 1;

list[++en] = b[i];

}

st ++;

}

return lab[sink] > -1;

}

inline int dfs(int x,int flow)

{

if (x == sink) return flow;

int ret = 0;

for (int i = a[x];i;i = c[i])

if (d[i] && lab[b[i]] == lab[x] + 1)

{

int now = dfs(b[i],min(flow,d[i]));

d[i] -= now;

d[i^1] += now;

flow -= now;

ret += now;

if (!flow) return ret;

}

lab[x] = -1;

return ret;

}

**6、点分治：**

void Get\_siz(int u,int fa = 0)

{

siz[u] = 1;

for (Edge \*e = H[u];e;e = e->next)

{

int v = e->v;

if (v != fa && vis[v] != Time)

{

Get\_siz(v,u);

siz[u] += siz[v];

}

}

}

void Get\_root(int u,int fa = 0)

{

num[u] = 0;

for (Edge \*e = H[u];e;e = e->next)

{

int v = e->v;

if (v != fa && vis[v] != Time)

{

Get\_root(v,u);

num[u] = max(num[u],siz[v]);

}

}

num[u] = max(num[u],size-siz[u]);

if (num[u] < num[root]) root = u;

}

void Get\_dis(int u,int val,int fa = 0)

{

dis[u] = val;

S[++cnt] = node[u] - val;

for (Edge \*e = H[u];e;e = e->next)

{

int v = e->v;

if (v != fa && vis[v] != Time) Get\_dis(v,dis[u]+e->c,u);

}

}

int Search(const Node &x)

{

int l = 0, r = cnt;

while (l < r)

{

int m = (l + r + 1) >> 1;

if (S[m] <= x) l = m;

else r = m - 1;

}

return r;

}

void Get\_ans(int u,int sign,int fa = 0)

{

if (!mart[u])

{

int tmp = Search(Node(dis[u],u));

ans[u] += (cnt-tmp) \* sign;

}

for (Edge \*e = H[u];e;e = e->next)

{

int v = e->v;

if (v != fa && vis[v] != Time) Get\_ans(v,sign,u);

}

}

void Deal(int u,int val,int sign)

{

cnt = 0;

Get\_dis(u,val);

sort(S+1,S+cnt+1);

Get\_ans(u,sign);

}

void Divide(int u)

{

Get\_siz(u);

size = siz[u];

root = 0;

Get\_root(u);

vis[root] = Time;

Deal(root,0,1);

for (Edge \*e = H[root];e;e = e->next) if (vis[e->v] != Time) Deal(e->v,e->c,-1);

for (Edge \*e = H[root];e;e = e->next) if (vis[e->v] != Time) Divide(e->v);

}

**7、cdq分治：**

【题目大意】为了写论文，Alex经常要整理大量的数据。 这一次，Alex面临一个严峻的考验：他需要实现一个数据结构来维护一个点集。

现在，二维平面上有N个点。Alex 需要实现以下三种操作：

1. 在点集里添加一个点；

2. 给出一个点，查询它到点集里所有点的曼哈顿距离的最小值；

3. 给出一个点，查询它到点集里所有点的曼哈顿距离的最大值。

两个点的曼哈顿距离定义为它们的横坐标差的绝对值与纵坐标差的绝对值的和。这么困难的问题，Alex当然不会做，只好再次请你帮忙了。

【题解】

这题的最大值是比较好做的，维护已经插入过的点的四种状态的最小值。

即x+y,x-y,-x+y,-x-y，求答案时用询问的点对应的状态减去出现过的最小值，再取个max即可。

最小值的维护：

这就用到CDQ分治了。

我们把所有操作和询问读进来，原本的点也当作插入操作。

然后我们得到了一个按照操作时间顺序的序列，有插入操作和询问操作。

注意到所有的询问操作只会被它前面的插入操作影响，我们从这里入手。

假设当前要处理的操作序列为[l,r]。

我们先二分出中间点mid。

先递归进去子问题[l,mid],[mid+1,r]

然后我们处理前半部分插入对后半部分询问的影响。

我们先标记前半部分的询问和后半部分的询问，然后把它们按x坐标排序。

然后便按x坐标顺序小到大做。

按照离散化后的y坐标建立两个树状数组，维护做到当前询问之前，插入的某个状态的最小值。

[设当前询问的点的坐标为(x0,y0)，我们分别考虑之前插入的点的y坐标大于y0的和小于y0的（计算曼哈顿距离的方式不同）]

（此处较难表达，请看代码或画图）

遇到插入的操作则插入相应的状态值到相应的树状数组，遇到询问则更新该询问的答案。

最后还要反过来按x坐标从大到小做，同样要用树状数组维护。

状态为（x+y,x-y,-x+y,-x-y），即为询问点的四个方向上计算曼哈顿距离所需状态。

void Solve(int l,int r)

{

if (l == r) return;

int mid = (l+r) >> 1;

Solve(l,mid);

Solve(mid+1,r);

// Prepare

fo(i,l,r) A[i].flag = -1;

fo(i,l,mid) if (A[i].op == 0) A[i].flag = 0;

fo(i,mid+1,r) if (A[i].op == 1) A[i].flag = 1;

sort(A+l,A+r+1,cmpx);

int tot = 0;

fo(i,l,r)

if (A[i].flag == 0 || A[i].flag == 1)

B[++tot] = OPER(A[i].y,i);

// Work

int lim = Discretize(1,tot);

fill(up+1,up+lim+1,-INF);

fill(down+1,down+lim+1,-INF);

fo(i,l,r)

if (A[i].flag == 0)

{

Ins\_up(ord[i],A[i].x-A[i].y);

Ins\_down(ord[i],A[i].x+A[i].y,lim);

} else

if (A[i].flag == 1)

{

ans[A[i].id] = min(ans[A[i].id],A[i].x-A[i].y-Query\_up(ord[i],lim));

ans[A[i].id] = min(ans[A[i].id],A[i].x+A[i].y-Query\_down(ord[i]));

}

fill(up+1,up+lim+1,-INF);

fill(down+1,down+lim+1,-INF);

fd(i,r,l)

if (A[i].flag == 0)

{

Ins\_up(ord[i],-A[i].x-A[i].y);

Ins\_down(ord[i],-A[i].x+A[i].y,lim);

} else

if (A[i].flag == 1)

{

ans[A[i].id] = min(ans[A[i].id],-A[i].x-A[i].y-Query\_up(ord[i],lim));

ans[A[i].id] = min(ans[A[i].id],-A[i].x+A[i].y-Query\_down(ord[i]));

}

}

LL xy,x\_y,\_xy,\_x\_y;

inline void Update(LL x,LL y)

{

xy = min(xy,x+y);

x\_y = min(x\_y,x-y);

\_xy = min(\_xy,-x+y);

\_x\_y = min(\_x\_y,-x-y);

}

void Task\_max()

{

xy = INF, x\_y = INF, \_xy = INF, \_x\_y = INF;

fo(i,1,N)

{

if (A[i].op == 0) Update(A[i].x,A[i].y);

if (A[i].op == 2)

{

ans[A[i].id] = max(A[i].x+A[i].y-xy,A[i].x-A[i].y-x\_y);

ans[A[i].id] = max(ans[A[i].id],max(-A[i].x+A[i].y-\_xy,-A[i].x-A[i].y-\_x\_y));

}

}

}

**8、Euler筛：**

void EulerSelect()

{

static int prime[maxn],p[maxn],s[maxn];

static bool notprime[maxn];

int tot = 0;

miu[1] = 1;

fo(i,2,mx)

{

if (!notprime[i]) prime[++tot] = i, s[i] = i, p[i] = i, miu[i] = -1;

fo(j,1,tot)

{

if (i \* prime[j] > mx) break;

notprime[i\*prime[j]] = 1;

s[i\*prime[j]] = prime[j];

if (i % prime[j] == 0)

{

p[i\*prime[j]] = p[i] \* prime[j];

miu[i\*prime[j]] = 0;

break;

}

p[i\*prime[j]] = prime[j];

miu[i\*prime[j]] = miu[i] \* miu[prime[j]];

}

}

a[1] = PI(f[1]=1,1);

fo(i,2,mx)

{

if (i == p[i]) f[i] = ((LL)p[i]\*s[i]-1)/(s[i]-1);

else f[i] = f[i/p[i]] \* f[p[i]];

a[i]= PI(f[i],i);

}

}

**9、AC自动机：**

struct Node

{

int son[10],fail;

bool flag;

}t[maxn];

void Insert(int \*s)

{

int x = 0;

fo(i,1,s[0])

{

if (!t[x].son[s[i]]) t[x].son[s[i]] = ++ tot;

x = t[x].son[s[i]];

}

t[x].flag = 1;

}

void Construct\_fail()

{

static int d[maxn];

int l = 0,r = 1;

while (l < r)

{

int x = d[++l];

fo(i,0,9)

if (t[x].son[i])

{

int v = t[x].son[i];

if (x == 0) t[v].fail = 0;

else t[v].fail = t[t[x].fail].son[i];

t[v].flag |= t[t[v].fail].flag;

d[++r] = v;

} else t[x].son[i] = t[t[x].fail].son[i];

}

}

**10、Extended\_kmp:**

void Calc\_next(char \*s,int \*next)

{

next[0] = N;

int pos,prev,j=-1;

for (int i = 1;i < N;i ++,j --)

if (j < 0 || i+next[i-prev] >= pos)

{

if (j < 0) j = 0, pos = i;

while (pos < N && s[pos] == s[j]) pos ++,j ++;

next[i] = j, prev = i;

} else next[i] = next[i-prev];

}

void Extended\_KMP(char \*T,char \*s,int \*next,int \*ext)

{

Calc\_next(T,next);

int pos,prev,j=-1;

for (int i = 0;i < N;i ++,j --)

if (j < 0 || i+next[i-prev] >= pos)

{

if (j < 0) j = 0, pos = i;

while (pos < N && j < N && s[pos] == T[j]) pos ++,j ++;

ext[i] = j, prev = i;

} else ext[i] = next[i-prev];

}

**11、左偏树：**

int Merge(int a,int b)

{

if (!a) return b;

if (!b) return a;

if (t[a].key < t[b].key) swap(a,b);

t[a].r = Merge(t[a].r,b);

if (t[t[a].l].dis < t[t[a].r].dis) swap(t[a].l,t[a].r);

t[a].dis = t[t[a].r].dis + 1;

t[a].size = t[t[a].l].size + t[t[a].r].size + 1;

t[a].sum = t[t[a].l].sum + t[t[a].r].sum + t[a].key;

return a;

}

void Work()

{

static int q[maxn];

int l = 0, r = 1;

q[1] = 1;

while (l < r)

{

int x = q[++l];

for (int i = a[x];i;i = c[i]) q[++r] = b[i];

}

LL ans = 0;

for (;r;r --)

{

int x = q[r];

t[++n] = Node(C[x]);

f[x] = n;

for (int i = a[x];i;i = c[i]) f[x] = Merge(f[x],f[b[i]]);

while (t[f[x]].sum > M) f[x] = Merge(t[f[x]].l,t[f[x]].r);

ans = max(ans,(LL)t[f[x]].size\*L[x]);

}

printf("%lld\n",ans);

}

**12、费用流**

bool SPFA()

{

int l = 0, r = 1;

fo(i,0,T) dist[i] = INF;

memset(flag,0,(T+1));

q[1] = S; flag[S] = 1;

dist[S] = 0; MinD[S] = INF;

while (l < r)

{

int x = q[++l];

for (int i = a[x];i;i = c[i])

if (d[i] && dist[x]+cost[i] < dist[b[i]])

{

dist[b[i]] = dist[x] + cost[i];

MinD[b[i]] = min(MinD[x],d[i]);

pre[b[i]] = i;

if (!flag[b[i]])

{

q[++r] = b[i];

flag[b[i]] = 1;

}

}

flag[x] = 0;

}

return dist[T] < INF;

}

void Work()

{

int ans = 0;

while (SPFA())

{

for (int x = T,i = pre[T];i;x = b[i^1],i = pre[x])

{

d[i] -= MinD[T];

d[i^1] += MinD[T];

}

ans += MinD[T] \* dist[T];

}

printf("%d\n",ans);

}

1. **补图最短路**

void BFS()

{

fo(i,0,n) dis[i] = INF;

link.clear(), tmp.clear();

fo(i,2,n) link.insert(i);

queue<int> Q;

while (!Q.empty()) Q.pop();

Q.push(1); dis[1] = 0;

while (!Q.empty())

{

int u = Q.front(); Q.pop();

for (int i = head[u];i != -1;i = edge[i].next)

{

int v = edge[i].v;

if (link.count(v) == 0) continue;

link.erase(v), tmp.insert(v);

}

for (set<int>::iterator it = link.begin();it != link.end();it ++)

Q.push(\*it), dis[\*it] = dis[u] + b;

link.swap(tmp), tmp.clear();

}

printf("%lld\n",min(dis[n],a));

}

1. **KD树**

int ansid,split;

long long ans;

struct KD\_Node

{

int x[2],id;

bool operator < (const KD\_Node &b)

{

fo(i,0,1)

if (x[i] != b.x[i]) return x[i] < b.x[i];

return 0;

}

}A[maxn],T[maxn],dest;

bool cmp(const KD\_Node &a,const KD\_Node &b)

{

return a.x[split] < b.x[split];

}

void Build(int l,int r,int d)

{

if (l >= r) return;

int mid = (l + r) >> 1;

split = d & 1;

nth\_element(T+l,T+mid,T+r+1,cmp);

Build(l,mid-1,d+1);

Build(mid+1,r,d+1);

}

void Query(int l,int r,int d)

{

if (l > r) return;

int mid = (l + r) >> 1;

long long dist = 0;

fo(i,0,1) dist += (long long)(T[mid].x[i]-dest.x[i])\*(T[mid].x[i]-dest.x[i]);

if (dist && (dist < ans || (dist == ans && T[mid] < A[ansid])))

{

ans = dist;

ansid = T[mid].id;

}

split = d & 1;

long long rad = (long long)(dest.x[split]-T[mid].x[split])\*(dest.x[split]-T[mid].x[split]);

if (dest.x[split] < T[mid].x[split])

{

Query(l,mid-1,d+1);

if (ans >= rad) Query(mid+1,r,d+1);

} else

{

Query(mid+1,r,d+1);

if (ans >= rad) Query(l,mid-1,d+1);

}

}

1. **KM**

**最大权匹配细节：**

L初始化：

Li=max{wi,j}(i∈x,j∈y)

Lj=0

找到一个改进量dx，dx=min{Li+Lj-wi,j}(i∈S,j不∈T)

Li=Li-dx (i∈S)

Li=Li+dx (i∈T)

**最小权匹配细节：**

L初始化：

Li=min{wi,j}(i∈x,j∈y)

Lj=0

dx=min{wi,j-Li-Lj}(i∈S,j不∈T)

Li=Li+dx (i∈S)

Li=Li-dx (i∈T)

bool dfs(int x)

{

if (qx[x]) return 0;

qx[x] = 1;

fo(i,1,N)

if (px[x] != i)

{

int d = lx[x] + ly[i] - C[x][i];

if (!d)

{

qy[i] = 1;

if (!py[i] || dfs(py[i]))

{

px[x] = i;

py[i] = x;

return 1;

}

} else slk[i] = min(slk[i],d);

}

return 0;

}

int KM()

{

fo(i,1,N) lx[i] = -INF, ly[i] = 0, px[i] = py[i] = 0;

fo(i,1,N) fo(j,1,N) lx[i] = max(lx[i],C[i][j]);

fo(i,1,N)

{

while (1)

{

memset(qx,0,N+1);

memset(qy,0,N+1);

fo(j,1,N) slk[j] = INF;

if (dfs(i)) break;

int M = INF;

fo(j,1,N) if (!qy[j] && slk[j] < M) M = slk[j];

fo(j,1,N) if (qx[j]) lx[j] -= M;

fo(j,1,N) if (qy[j]) ly[j] += M;

}

}

int ret = 0;

fo(i,1,N) ret += C[i][px[i]];

return ret;

}

**16、最小圆覆盖**

template<class T> T sqr(T x){return x\*x;}

double dis(Point a,Point b){return sqr(a.x-b.x)+sqr(a.y-b.y);}

double dist(Point a,Point b){return sqrt(dis(a,b));}

void Update(int i,int j,int k)

{

double a1 = 2\*(a[j].x-a[i].x), b1 = 2\*(a[j].y-a[i].y);

double c1 = -((a[i].x+a[j].x)\*(a[j].x-a[i].x)+(a[i].y+a[j].y)\*(a[j].y-a[i].y));

double a2 = 2\*(a[k].x-a[i].x), b2 = 2\*(a[k].y-a[i].y);

double c2 = -((a[i].x+a[k].x)\*(a[k].x-a[i].x)+(a[i].y+a[k].y)\*(a[k].y-a[i].y));

double s = a1\*b2-a2\*b1;

if (fabs(s) > eps) C = Point((b1\*c2-b2\*c1)/s,(a2\*c1-a1\*c2)/s);

}

int main()

{

freopen("amplifier.in","r",stdin);

freopen("amplifier.out","w",stdout);

scanf("%d",&N);

fo(i,1,N) scanf("%lf%lf",&a[i].x,&a[i].y);

scanf("%lf%d",&r,&p);

r = r\*pi/180;

fo(i,1,N)

{

double x = a[i].x, y = a[i].y;

a[i].x = (x\*cos(r)+y\*sin(r))/p;

a[i].y = y\*cos(r)-x\*sin(r);

}

random\_shuffle(a+1,a+1+N);

if (N == 1) printf("0.000\n");

else

{

C = Point((a[1].x+a[2].x)/2,(a[1].y+a[2].y)/2);

R = dis(C,a[1]);

fo(i,3,N)

{

if (dis(C,a[i]) < R+eps) continue;

C = Point((a[1].x+a[i].x)/2,(a[1].y+a[i].y)/2);

R = dis(C,a[1]);

fo(j,2,i-1)

{

if (dis(C,a[j]) < R+eps) continue;

C = Point((a[i].x+a[j].x)/2,(a[i].y+a[j].y)/2);

R = dis(C,a[i]);

fo(k,1,j-1)

{

if (dis(C,a[k]) < R+eps) continue;

Update(i,j,k);

R = dis(C,a[i]);

}

}

}

printf("%.3f\n",sqrt(R));

}

return 0;

}

**17、KMP**

P[1]:=0;

j:=0;

for i:=2 to m do

begin

while (j>0) and (B[j+1]<>B[i]) do j:=P[j];

if B[j+1]=B[i] then j:=j+1;

P[i]:=j;

end;

j:=0;

for i:=1 to n do

begin

while (j>0) and (B[j+1]<>A[i]) do j:=P[j];

if B[j+1]=A[i] then j:=j+1;

if j=m then

begin

writeln('Pattern occurs with shift ',i-m);

j:=P[j];

end;

end;

**18、一类割意义下的网络流建模：**

先来看这么一个题。

给出一个无向图G=(V,E)，并有定义在V上的权a和b，以及定义在E上的权w。

（这里提及的所有权都默认为非负整数。）

选出一个点集S，最大化∑(x属于S)a[x]+∑(x不属于S)b[x]+∑(x和y同时属于或不同时属于S)w(x,y)。

这是一个可以直接用最小割解决的问题。

添加源S和汇T，对于每个x，连边(S,x,a[x])和(x,T,b[x])，而w(x,y)直接变为对应边的容量。

考察这个图的割的意义不难得出最终的答案=∑a[x]+∑b[x]+∑w(x,y)-最小割。

好..我们来改一改。

现在我们需要最大的化的式子变成了这样。

∑(x属于S)a[x]+∑(x不属于S)b[x]+∑(x属于S且y不属于S)w(x,y)。

....很可惜似乎直接做是不可行的，如果有靠谱的做法请不吝赐教。

我们需要一个重要的条件：图G是二分图。

对于X中的每个点x，连边(S,x,a[x])和(x,T,b[x])；

对于Y中的每个点y，连边(S,y,b[y])和(y,T,a[y])。

而边权w(x,y)仍然直接变为对应边的容量。

下面来看另外一种改法。

新增一个定义在E上的权r。

最大化∑(x属于S)a[x]+∑(x不属于S)b[x]+∑(x和y都属于S)w(x,y)+∑(x和y都不属于S)r(x,y)。

（请留意，二分图的条件已经吃粑粑去了。）

回顾前面的解题过程，在考虑使用割的意义建图的时候，我们的割边实际上是“损失”。

在这一题中点权似乎仍然是老样子，但是边权有些麻烦了。

1. 当x和y同时属于S的时候，我们损失了r(x,y)。

2. 当x和y同时不属于S的时候，我们损失了w(x,y)。

3. 当x属于S和y属于S的真假性不同的时候，我们损失了w(x,y)以及r(x,y)。

正确的表达这些损失要用到的技巧是权值的重新分配。

首先，添加源S和汇T，对于每个x，连边(S,x,a[x])和(x,T,b[x])

考虑第1条，可以视为，当x属于S的时候，损失r(x,y)/2，y同理；

此时即要将x，y向T连各连一条r(x,y)/2的边，因为割的边代表损失的利益，而S连向x，y的边保留代表取此种方案。

第2条类似。

而在第3条描述的情形下，根据前面已经做出的调整（根据前两条，已经割了S向x的边，y向T的边（或是S到y，x到T）），我们就会计算出(r(x,y)+w(x,y))/2的损失；

然而真正的损失是w(x,y)+r(x,y)，因此我们对x到y，y到x分别连上一条容量(r(x,y)+w(x,y))/2的边来补上这个损失。

这样一来所有的损失都被正确的计算了。

再来看一道有点像的题。

给定无重边无自环的无向图G=(V,E)，定义在V上的权a，E上的权w，求一个点集S，最大化：

2\*∑(x,y属于S)w(x,y)-∑(x属于S且y不属于S)w(x,y)-∑(x属于S)a[x]。

（为什么要乘2呢...∑枚举的x,y是无序的，但是根据题意w(x,y)和w(y,x)都要算进去）

令b[x]=∑(y)w(x,y)，则∑(x属于S且y不属于S)w(x,y)=∑(x属于S)b[x]-2\*∑(x,y属于S)w(x,y)，

进而要最大化的式子变为 ∑(x,y属于S)4\*w(x,y) - ∑(x属于S)(a[x]+b[x])。

很经典的最大权闭合图，这里略去。

但是这样搞总觉得有点脱离主题，于是我们再来看一个做法。

据∑w(x,y)=∑(x,y属于S)w(x,y)+∑(x,y不属于S)w(x,y)+∑(x属于S且y不属于S)w(x,y)，将目标式化为

2\*∑w(x,y) - 3\*∑(x属于S且y不属于S)w(x,y) - ∑(x属于S)a[x]。

从而只需要最小化 3\*∑(x属于S且y不属于S)w(x,y) + 2\*∑(x,y不属于S)w(x,y) + ∑(x属于S)a[x]。

而这个东西等于 ∑(y不属于S)w(x,y) + 2\*∑(x属于S且y不属于S)w(x,y) + ∑(x属于S)a[x]。

（恩，2\*∑(x,y不属于S)w(x,y)+∑(x属于S且y不属于S)w(x,y)=∑(y不属于S)w(x,y)看起来挺纠结的。）

不管怎么说弄成这样了以后就可以利用割的意义来构造了：

∑w(x,y)就是把y割出S的损失，a[x]就是把x割进S的损失，将x,y分开的损失则是2\*w(x,y)。

于是直接最小割就完了。