# ACM笔记 by tyx

# 目录

1. 字符串…………………………………………………P2
2. 图论……………………………………………………P6
3. Dp………………………………………………………P20
4. 树………………………………………………………P25
5. 数学……………………………………………………P38
6. 数据结构………………………………………………P50
7. 不方便归类的算法……………………………………P54
8. 各种优化程序…………………………………………P63

# 1.字符串：

**KMP:**

对于短的那个串（待匹配串），Ne[i]=j表示与从i结束的后缀 相同的 从1开始的以j结束的前缀；（不是官方的next数组的定义，官方的是next[i]=j，但是i和j不一样）

void Init(){

int i=2;ne[1]=0;

while(i<=len2){

int j=ne[i-1];

while(s2[i]!=s2[j+1] && j) j=ne[j];

if(!j && s2[i]!=s2[1]) ne[i]=0;

else ne[i]=j+1;

++i;

}

}

void KMP(){

int i=1,j=0;

while(i<=len1){

if(s1[i]==s2[j+1]) ++i,++j;

else{

j=ne[j];

if(!j && s1[i]!=s2[1]) ++i;

}d

if(j==len2) ans[++tot]=i-len2,j=ne[j];

}

}

优化：

Nextval[i] = j

对于s2[i]和s2[j]相等的情况，可以将nextval[i] = nextval[j]

**AC自动机：**

对多个待匹配串建立trie树，文本串在trie上匹配

Fail[i]=j指针的意思是：满足i后缀等于j的前缀的最大深度的点为j

这里某些前缀的子串里面可能包含某些前缀（比如baab包含aa），需要通过每个点都跳fail指针去看哪些地方是单词的终结点。

1. 这里采用了构建trie图得到fail的方式，真正用trie树得到fail，就需要这里不能赋值，用循环不断往回跳，直到存在儿子或者为根节点

2. 理论上来说，这个地方的复杂度会很高，每次暴力跳fail边，每个点可能会被访问很多次（如a,aa,aaa,aaaa, aaaaaa…的fail指针，越短的，被访问次数越多）

这里就考虑用怎么去只访问一次？一个比较好的做法就是构建fail树，然后拓扑排序的从下往上遍历（类似后缀自动机的parent树）

#include<bits/stdc++.h>

#define For(aa, bb, cc) for(int aa = (bb); aa <= (int)(cc); ++aa)

using namespace std;

const int maxn = 1e6 + 10;

int n;

int tr[maxn][26], fail[maxn], tot;

int pos[maxn], cnt[maxn];

void insert(char \*s, int id){

int len = strlen(s + 1), u = 0;

For(i, 1, len){

int j = s[i] - 'a';

if(!tr[u][j]) tr[u][j] = ++tot;

u = tr[u][j];

}

pos[u] = id;

}

void getfail(){

queue<int> q;

For(i, 0, 25) if(tr[0][i]) q.push(tr[0][i]), fail[tr[0][i]] = 0;

while(!q.empty()){

int u = q.front();

q.pop();

For(i, 0, 25){

if(tr[u][i]){

fail[tr[u][i]] = tr[fail[u]][i];

q.push(tr[u][i]);

}

else tr[u][i] = tr[fail[u]][i]; //1

}

}

}

void Clear(){

For(i, 0, tot){

For(j, 0, 25) tr[i][j] = 0;

fail[i] = pos[i] = 0;

}

For(i, 1, n) cnt[i]= 0;

tot = 0;

}

int main(){

#ifndef ONLINE\_JUDGE

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

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

#endif

while(~scanf("%d", &n) && n){

char s[maxn], t[160][100];

For(i, 1, n){

scanf("%s", t[i] + 1);

insert(t[i], i);

}

getfail();

scanf("%s", s + 1);

int len = strlen(s + 1);

int u = 0;

For(i, 1, len){

u = tr[u][s[i] - 'a'];

for(int j = u; j; j = fail[j]) ++cnt[pos[j]];//2

}

int Max = 0;

For(i, 1, n) if(cnt[i] > Max) Max = cnt[i];

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

For(i, 1, n) if(cnt[i] == Max) printf("%s\n", t[i] + 1);

Clear();

}

return 0;

}

Fail树上的拓扑排序：

For(i, 1, len){

u = tr[u][s[i] - 'a'];

++f[u];

}

For(i, 1, tot){

if(fail[i]) G[i].push\_back(fail[i]), ++deg[fail[i]];

//cout<<fail[i]<<" "<<i<<endl;

}

Topo();

For(i, 1, n) printf("%d\n", f[pos[i]]);//pos[i]为第i个字符串的位置

后缀数组：

后缀数组sa[i]就表示**排名为i的后缀**的起始位置的下标

而它的映射数组Rank[i]就表示**起始位置的下标为i的后缀**的排名

LCP(i,j)为suffix(sa[i])与suffix(sa[j])的最长公共前缀

height[i]为LCP(i,i-1), LCP(i,k)=min(height[j]) i+1<=j<=k

void suffix\_array(){

int X[maxn],Y[maxn],c[maxn];

For(i,1,m) c[i]=0;

For(i,1,n) ++c[X[i]=s[i]];

For(i,2,m) c[i]+=c[i-1];

Forr(i,n,1) sa[c[X[i]]--]=i;

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

int num=0;

For(i,n-k+1,n) Y[++num]=i;

For(i,1,n) if(sa[i]>k) Y[++num]=sa[i]-k;

For(i,1,m) c[i]=0;

For(i,1,n) ++c[X[i]];

For(i,2,m) c[i]+=c[i-1];

Forr(i,n,1) sa[c[X[Y[i]]]--]=Y[i];

swap(X,Y);

num=0;

X[sa[1]]=++num;

For(i,2,n) X[sa[i]]=(Y[sa[i]]==Y[sa[i-1]] && Y[sa[i]+k]==Y[sa[i-1]+k])?num:++num;

if(num==n) break;

m=num;

}

For(i,1,n) Rank[sa[i]]=i;

int k=0;

height[1]=0;

For(i,1,n){

if(Rank[i]==1) continue;

if(k) --k;

int j=sa[Rank[i]-1];

while(i+k<=n && j+k<=n && s[i+k]==s[j+k]) ++k;

height[Rank[i]]=k;

}

}

**后缀自动机：**

SAM的结构包含两部分，有向无环单词图（DAWG）和后缀树（prefix tree），SAM中的每个节点同时存在于这两个结构中。

节点：母串某个前缀的若干个长度相差1的后缀 or 从起点开始到这个点的所有路径组成的子串的集合（每个节点所代表的那些串的endpos集合相同）

endpos(v)：节点v代表子串集合在母串中的结束位置的集合（统计endpos的个数，每个点从当前点跳后缀链接到S的点数（需要除掉因maxlen原因拆分的点，这个点并不是因为加入字符而多出来的）。为什么？因为每次加点都是因为endpos不同，及出现了新的位置，这个位置会增加1个（每次只会增加一个字符），所以只要统计原始的增加就可以了）

后缀链接link/nex：link[u] = v满足|min[u]| = |max[v]| + 1 （长->短）

后缀连接为边组成后缀树，DAWG起始节点为根（trans[][],map<>ch[]）

只需要存maxlen，minlen[u] = maxlen[fa [u]]+1

#include<cstdio>

#include<iostream>

#include<cstring>

#define For(aa, bb, cc) for(int aa = (bb); aa <= (int)(cc); ++aa)

#define Forr(aa, bb, cc) for(int aa = (bb); aa >= (int)(cc); --aa)

#define Cpy(aa, bb) memcpy(aa, bb, sizeof bb)

using namespace std;

const int maxn = 1e6 + 10;

int len;

char s[maxn];

int val[maxn << 1];

int num[maxn << 1], id[maxn << 1], cnt[maxn << 1], ans[maxn];

//sam

int maxlen[maxn << 1];

int trans[maxn << 1][26];

int link[maxn << 1];

int lst, tot;

void init(){

lst = tot = 1;

}

void extend(int c){

int now = ++tot, p;

val[now] = 1;//求endpos的大小

maxlen[now] = maxlen[lst] + 1;

for(p = lst; p && !trans[p][c]; p = link[p]) trans[p][c] = now;

if(!p) link[now] = 1;

else{

int q = trans[p][c];

if(maxlen[q] == maxlen[p] + 1) link[now] = q;

else{

int sp = ++tot;

maxlen[sp] = maxlen[p] + 1;

Cpy(trans[sp], trans[q]);

link[sp] = link[q];

for(; p && trans[p][c] == q; p = link[p]) trans[p][c] = sp;

link[q] = link[now] = sp;

}

}

lst = now;

}

void topsort(){

For(i, 1, tot) ++num[maxlen[i]];

For(i, 1, len) num[i] += num[i - 1];

For(i, 1, tot) id[num[maxlen[i]]--] = i;

Forr(i, tot, 1) val[link[id[i]]] += val[id[i]];

}

int main(){

#ifndef ONLINE\_JUDGE

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

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

#endif

ios::sync\_with\_stdio(false);

cin.tie(NULL);

init();

cin >> (s + 1);

len = strlen(s + 1);

For(i, 1, len) extend(s[i] - 'a');

topsort();

For(i, 1, tot) ans[maxlen[i]] = max(ans[maxlen[i]], val[i]);

Forr(i, len - 1, 1) ans[i] = max(ans[i], ans[i + 1]);

For(i, 1, len) printf("%d\n", ans[i]);

return 0;

}

大字符集：

struct SAM{

struct node{

map<int, node \*> ch;

node \*nex;

int maxlen;

node(int len = 0) : nex(NULL), maxlen(len) {}

} \*st, \*lst;

void init(){

st = lst = new node;

}

node \*extend(int c){

node \*u = new node(lst->maxlen + 1), \*v = lst;

for(; v && !(v->ch[c]); v = v->nex) v->ch[c] = u;

if(!v) u->nex = st;

else if(v->ch[c]->maxlen == v->maxlen + 1) u->nex = v->ch[c];

else{

node \*p = v->ch[c], \*pp = new node(v->maxlen + 1);

pp->ch = p->ch;;

pp->nex = p->nex;

p->nex = u->nex = pp;

for(; v && v->ch[c] == p; v = v->nex) v->ch[c] = pp;

}

lst = u;

return u;

}

}sam;

模块化SAM：

template<int N, int sigma> struct Suffix\_Automation{

int maxlen[N << 1], trans[N << 1][sigma],link[N << 1], lst, tot;

int sum[N], pos[N << 1]; //pos: terminal point

int num[N], id[N << 1], f[N << 1]; //id:toposort seq; f:endpos size

bool is\_clone[N << 1]; //clone point

vector<int> ch[N << 1];

void init(int Size){

tot = lst = 1;

link[1] = 0;

memset(trans, 0, (Size \* 2 + 5) \* sigma \* sizeof (int));

memset(pos, 0, (Size \* 2 + 5) \* sizeof (int));

For(i, 1, Size \* 2 + 5) ch[i].clear();

}

void extend(int c){

int now = ++tot, p = lst;

f[now] = 1; //endpos Size

maxlen[now] = maxlen[p] + 1;

for(; p && !trans[p][c]; p = link[p]) trans[p][c] = now;

if(!p) link[now] = 1;

else{

int q = trans[p][c];

if(maxlen[q] == maxlen[p] + 1) link[now] = q;

else{

int nq = ++tot;

is\_clone[nq] = 1; //clone point

maxlen[nq] = maxlen[p] + 1;

memcpy(trans[nq], trans[q], sigma \* sizeof (int));

link[nq] = link[q];

for(; p && trans[p][c] == q; p = link[p]) trans[p][c] = nq;

link[q] = link[now] = nq;

}

}

lst = now;

}

void toposort(){

For(i, 1, tot) ++num[maxlen[i]];

For(i, 1, n) num[i] += num[i - 1];

For(i, 1, tot) id[num[maxlen[i]]--] = i;

Forr(i, tot, 1) f[link[id[i]]] += f[id[i]];

}

void dfs(int u){

for(int v: ch[u]) dfs(v);

if(!pos[u]) pos[u] = pos[ch[u][0]];

}

void prepare(){

For(i, 1, n){

extend(s[i] - 'a');

sum[i] = sum[i - 1] + val[s[i] - 'a'];

pos[lst] = i;

}

For(i, 2, tot) ch[link[i]].push\_back(i);

dfs(1);

}

bool check(int x){

LL cnt = 0;

For(i, 1, tot){

int P = pos[i];

int L = P - maxlen[i] + 1, R = P - maxlen[link[i]];

while(L < R){ //二分每个点表示的后缀长度

int mid = (L + R) >> 1;

if(sum[P] - sum[mid - 1] <= x) R = mid;

else L = mid + 1;

}

if(sum[P] - sum[L - 1] <= x) cnt += P - maxlen[link[i]] - L + 1;

}

return cnt >= k;

}

};

Suffix\_Automation<maxn, 26> sam;

Manacher：

(Manacher其实就是一个DP)

对奇偶回文串，可以采用中间插入’#’的方法来转化为奇回文

定义1-i的最长回文串的中心位置pos和最右端R,Rds[i]为第i个位置的回文半径.

对于i<R，考虑关于pos的对称位置2\*pos-i，考虑rds[2\*pos-i]（已经有的对称）和R-i+1（边界）大小，在pos的rds[pos]范围内，可以保证i的对称一定成立，然后暴力去扩展边沿就可以了。

int len;

char s[maxn], t[maxn];

int rds[maxn];

int Manacher(){

int R = 0, pos, maxlen = 0;

For(i, 1, len){

if(i < R) rds[i] = min(rds[2 \* pos - i], R - i + 1);

else rds[i] = 1;

while(s[i - rds[i]] == s[i + rds[i]]) ++rds[i];

if(R < i + rds[i]) pos = i, R = i + rds[i] - 1;

maxlen = max(maxlen, rds[i] - 1);

}

return maxlen;

}

int main(){

ios::sync\_with\_stdio(false);

cin.tie(NULL);

cin >> (t + 1);

len = strlen(t + 1);

int n = 0;

s[n] = '@';

s[++n] = '#';

For(i, 1, len){

s[++n] = t[i];

s[++n] = '#';

}

s[++n] = '\0';

len = n - 1;

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

}

回文自动机：O(|S| log Σ)

回文自动机的思想和AC自动机的思想很类似，也是构建fail指针。PAM中每个节点仅代表一个状态—某个串的最长的回文后缀（和SAM不同），解决题目的做法一般是在fail树上进行DP操作。

回文自动机有两个根--一个是奇回文的根，一个是偶回文的根

num[u]：以u代表的回文串结尾的回文后缀的个数

Size[u]：u代表的回文串的数量，进行从叶子到根的累加后就是整个串出现的次数（每次都只加了最长的）

L[pos]：以pos结尾的回文串长度

一些结论：

1.不同回文子串个的个数为tot – 2（回文树中总结点个数-2）

2.个位置结尾的回文子串的个数，相当于fail链的深度

3.子串出现的次数为每个结尾位置出现次数加起来，可以通过标记每个最长的回文后缀，然后拓扑排序，从叶子到根扫一遍就可以得到。

#include<bits/stdc++.h>

#define For(aa, bb, cc) for(int aa = (bb); aa <= (int)(cc); ++aa)

#define Forr(aa, bb, cc) for(int aa = (bb); aa >= (int)(cc); --aa)

using namespace std;

const int maxn = 5e5 + 10;

char s[maxn];

struct PAM{

int maxlen[maxn], trans[maxn][26], fail[maxn];

int num[maxn], Size[maxn], L[maxn];

int tot, lst;

void init(){

maxlen[0] = 0; maxlen[1] = -1;

fail[0] = 1; fail[1] = 1;

tot = 1; lst = 0;

memset(trans[0], 0, sizeof trans[0]);

memset(trans[1], 0, sizeof trans[1]);

}

int newnode(int len){

int now = ++tot;

memset(trans[now], 0, sizeof trans[now]);

maxlen[now] = len;

return tot;

}

void extend(int c, int pos){

int u = lst;

while(s[pos - maxlen[u] - 1] != s[pos]) u = fail[u];//find self

if(!trans[u][c]){

int now = newnode(maxlen[u] + 2);

int v = fail[u];

while(s[pos - maxlen[v] - 1] != s[pos]) v = fail[v];//find second self

fail[now] = trans[v][c];

trans[u][c] = now;

num[now] = num[fail[now]] + 1;

}

lst = trans[u][c];

L[pos] = maxlen[lst];

++Size[lst];

}

void calc(){

Forr(i, tot, 0) Size[fail[i]] += Size[i];

}

int query(){

return num[lst];

}

}pam;

int main(){

ios::sync\_with\_stdio(false);

cin.tie(NULL);

pam.init();

cin >> (s + 1);

int len = strlen(s + 1), pre;

For(i, 1, len){

int c = s[i] - 'a';

if(i > 1) c = (c + pre) % 26;

s[i] = c + 'a';

pam.extend(c , i);

printf("%d ", pre = pam.query());

}

return 0;

}

支持修改的PAM：

暴力修改：每次通过暴力跳对fail指针进行重新更新

优化操作：每次修改其实只是修改一个状态后续字符的跳转，其他字符可以通过保存状态来进行直接跳转

这里trans[][]是本来的链接，fail[]是当前的字符串的跳转指针，fa[][]每个节点后面跟不同字符跳转到的节点

#include<bits/stdc++.h>

#define For(aa, bb, cc) for(int aa = (bb); aa <= (int)(cc); ++aa)

using namespace std;

const int maxn = 1e6 + 10;

typedef long long LL;

const LL mod = 1e9 + 7;

int n, m;

char s[maxn];

int tot, lst;

int trans[maxn][26], fail[maxn], fa[maxn][26], len[maxn];

int de[maxn];

LL sum[maxn], val[maxn], mul[maxn], ans;

//val[i]：以该点结尾的回文子串的贡献

void init(){

For(i, 0, tot){

fail[i] = 0;

memset(trans[i], 0, sizeof trans[i]);

memset(fa[i], -1, sizeof fa[i]);

}

tot = 0;

fail[0] = fail[1] = 1;

sum[0] = sum[1] = 0;

tot = lst = 1;

len[0] = 0;

len[1] = -1;

}

int newnode(int L){

++tot;

len[tot] = L;

return tot;

}

int get\_pos(int u, int c, int pos){

int v = u;

while(s[pos - len[v] - 1] != s[pos]){

if(fa[v][c] != -1) v = fa[v][c];

else v = fail[v];

}

for(int i = u; i != v; i = fail[i]){

if(fa[i][c] == -1) fa[i][c] = v;//如果一个节点存在跳转指针了，那么父亲对于该节点都存在了（从根往下面扩展）

else break;

}

return v;

}

void extend(int c, int pos){

int u = lst;

u = get\_pos(u, c, pos);

if(!trans[u][c]){

int v = newnode(len[u] + 2), p = fail[u];

p = get\_pos(p, c, pos);

fail[v] = trans[p][c];

trans[u][c] = v;

if(len[v] <= n) sum[v] = 1ll \* mul[n - len[v]] \* (n - len[v] + 1) % mod;

else sum[v] = 0;

sum[v] = (sum[v] + sum[fail[v]]) % mod;

}

lst = trans[u][c];

}

int main(){

#ifndef ONLINE\_JUDGE

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

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

#endif

ios::sync\_with\_stdio(false);

cin.tie(NULL);

int \_;

cin >> \_;

memset(fa, -1, sizeof fa);

mul[0] = 1;

For(i, 1, 1e6) mul[i] = mul[i - 1] \* 26 % mod;

while(\_--){

init();

cin >> n >> m;

cin >> (s + 1);

int L = strlen(s + 1);

ans = 0;

For(i, 1, L){

extend(s[i] - 'a', i);

de[i] = lst;

val[i] = sum[lst];

ans = (ans + val[i]) % mod;

}

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

while(m--){

int x;

char t[3];

cin >> x;

if(x == 1){

cin >> (t + 1);

s[++L] = t[1];

extend(s[L] - 'a', L);

de[L] = lst;

val[L] = sum[lst];

ans = (ans + val[L]) % mod;

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

}

else{

ans = (ans - val[L] + mod) % mod;

lst = de[--L];

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

}

}

}

return 0;

}

字符串哈希：

一般用的函数: (x \* mod1 + mod2) % mod，根据得到的数字是否相同来进行判断

String的使用：

1. 初始化：

String s(str), string str = s

1. 存取：

At(n),[]

Length, empty

Iterator / unsigned int I 遍历

Assign(str,{start},{n})

1. 转换char:

s.c\_str();s.copy(buf,n,pos=0)

4.比较

s.compare(string/char\*) 1, 0, -1

5.连接

+=; append

# 2.图论：

最大流Dinic：

int bfs(){

For(i,1,n+m+2) dis[i]=-1;//修改点数

int q[maxn],l=0,r=0;

q[++r]=s,dis[s]=0;

while(l<r){

int k=q[++l];

for(int i=be[k];i;i=ne[i]){

int u=to[i];

if(flow[i] && dis[u]==-1){

dis[u]=dis[k]+1;

q[++r]=u;

}

}

}

return dis[t]!=-1;

}

int dfs(int node,int f){

if(t==node) return f;

if(!f) return 0;

int res,sum=0;

for(int &i=cur[node];i;i=ne[i]){

int u=to[i];

if(flow[i] && dis[u]==dis[node]+1){

if((res=dfs(u,min(f,flow[i])))){

flow[i]-=res;

flow[i^1]+=res;

f-=res;

sum+=res;

}

if(!f) return sum;

}

}

return sum;

}

int Dinic(){

int ans=0,tmp;

while(bfs()){

memcpy(cur,be,sizeof be);

while((tmp=dfs(s,inf))) ans+=tmp;

}

return ans;

}

E=1;//重点

最大流Isap:

int isap(int node,int flow){

if(node==t) return flow;

int res=flow;

for(int i=be[node];i;i=ne[i]){

int u=to[i];

if(d[node]==d[u]+1){

int minflow=isap(u,min(w[i],res));

w[node]-=minflow;

w[node^1]+=minflow;

res-=minflow;

if(!res) return flow;

}

}

if(!(--gap[d[node]])) d[s]=t;

++gap[++d[node]];

return flow-res;

}

for(gap[0]=t;d[s]<t;) maxflow+=isap(s,inf);

最小费用最大流：在最大流的前提下，求最小费用的流

spfa费用流：

inline void add\_edge(int x,int y,int f,int c){

to[++e]=y,ne[e]=be[x],be[x]=e,flow[e]=f,w[e]=c,st[e]=x;

to[++e]=x,ne[e]=be[y],be[y]=e,flow[e]=0,w[e]=-c,st[e]=y;

}

int spfa(){

queue<int>q;

Set(vis,0),Set(pre,0);

For(i,1,t) dis[i]=inf;

dis[s]=0;

q.push(s);

while(!q.empty()){

int k=q.front();q.pop();vis[k]=0;

for(int i=be[k];i;i=ne[i]){

int u=to[i];

if(flow[i] && dis[u]>dis[k]+w[i]){

dis[u]=dis[k]+w[i];

pre[u]=i;

if(!vis[u]){

vis[u]=1;

q.push(u);

}

}

}

}

return dis[t]<inf;

}

int spfa\_flow(){

int ans=0,f=0;

while(spfa()){

int now=t,minf=inf;

while(now!=s){

minf=min(minf,flow[pre[now]]);

now=st[pre[now]];

}

ans+=dis[t]\*minf;

f+=minf;

now=t;

while(now!=s){

flow[pre[now]]-=minf;

flow[pre[now]^1]+=minf;

now=st[pre[now]];

}

}

return ans;

}

Zkw费用流：

Zkw最原始的版本：通过类似KM的顶标修改方式，进行修改

适用于：最终流量较大, 而费用取值范围不大的图, 或者是增广路径比较短的图 (如二分图),

inline void add\_edge(int x,int y,int f,int c){

to[++e]=y,ne[e]=be[x],be[x]=e,flow[e]=f,cost[e]=c;

to[++e]=x,ne[e]=be[y],be[y]=e,flow[e]=0,cost[e]=-c;

}

int dfs(int node,int f){

vis[node]=1;

if(node==t) return allcost+=-dis[s]\*f,allflow+=f,f;

int sum=0;

for(int &i=cur[node];i;i=ne[i]){

int u=to[i];

if(!vis[u] && flow[i] && dis[u]==dis[node]+cost[i]){

int tmp=dfs(u,min(flow[i],f));

if(tmp){

flow[i]-=tmp;

flow[i^1]+=tmp;

f-=tmp;

sum+=tmp;

}

if(!f) return sum;

}

}

return sum;

}

bool update(){

if(vis[t]) return 1;

int tmp=inf;

For(k,1,n){

if(vis[k]){

for(int i=be[k];i;i=ne[i]){

int u=to[i];

if(flow[i] && !vis[u])

tmp=min(tmp,dis[k]-dis[u]+cost[i]);

}

}

}

if(tmp==inf) return 0;

For(i,1,n)

if(vis[i])

dis[i]-=tmp;

return 1;

}

void minflow(){

allcost=allflow=0;

do{

memset(vis,0,sizeof vis);

memcpy(cur,be,sizeof be);

dfs(s,inf);

}while(update());

}

Spfa的SLF优化版本：（更加普适）

#include<bits/stdc++.h>

#define For(aa, bb, cc) for(int aa = (bb); aa <= (int)(cc); ++aa)

using namespace std;

const int inf = 0x3f3f3f3f;

const int maxn = 5e3 + 10, maxm = 1e6 + 10;

int n, s, t, allflow, allcost;

int a[maxn];

int vis[maxn], cur[maxn], dis[maxn];

int be[maxn], ne[maxm], to[maxm], flow[maxm], cost[maxm], e;

void add\_edge(int x, int y, int z, int w){

to[++e] = y, ne[e] = be[x], be[x] = e, flow[e] = z, cost[e] = w;

to[++e] = x, ne[e] = be[y], be[y] = e, flow[e] = 0, cost[e] = -w;

}

int dfs(int u, int f){

vis[u] = 1;

if(u == t) return allcost += dis[s] \* f, allflow += f, f;

int sum = 0;

for(int &i = cur[u]; i; i = ne[i]){

int v = to[i];

if(!vis[v] && flow[i] > 0 && dis[v] == dis[u] - cost[i]){

int tmp = dfs(v, min(flow[i], f));

if(tmp){

flow[i] -= tmp;

flow[i ^ 1] += tmp;

f -= tmp;

sum += tmp;

}

if(!f) return sum;

}

}

return sum;

}

bool spfa(){

For(i, 1, n + 2) dis[i] = inf, vis[i] = 0;

deque<int> q;

dis[t] = 0;

q.push\_back(t);

while(!q.empty()){

int u = q.front();

q.pop\_front();

vis[u] = 0;

for(int i = be[u]; i; i = ne[i]){

int v = to[i];

if(flow[i ^ 1] > 0 && dis[v] > dis[u] - cost[i]){

dis[v] = dis[u] - cost[i];

if(!vis[v]){

vis[v] = 1;

if(!q.empty() && dis[v] < dis[q.front()]) q.push\_front(v);

else q.push\_back(v);

}

}

}

}

return dis[s] < inf;

}

void minflow(){

allflow = allcost = 0;

while(spfa()){

memset(vis, 0, sizeof vis);

memcpy(cur, be, sizeof be);

dfs(s, inf);

}

}

int main(){

#ifndef ONLINE\_JUDGE

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

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

#endif

ios::sync\_with\_stdio(false);

cin.tie(NULL);

cin >> n;

For(i, 1, n) cin >> a[i];

e = 1;

s = n + 1, t = n + 2;

For(i, 1, n){

if(a[i]) add\_edge(s, i, 1, 0);

else add\_edge(i, t, 1, 0);

}

For(i, 1, n - 1) add\_edge(i, i + 1, inf, 1), add\_edge(i + 1, i, inf, 1);

minflow();

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

return 0;

}

**上下界网络流：**（待补）

**最小割：**

最小割等于最大流（对偶问题）

网络流24题

最大权闭合子图：

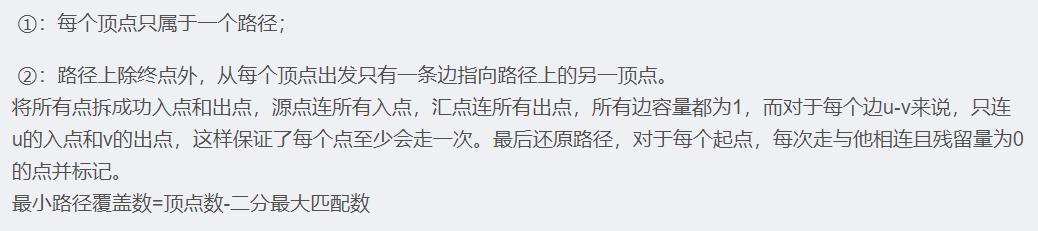
选一个物品Xi，权值为a，需要附带选择一些物品Yj，权值为-b；求最大权值；

建图：add（S, Xi, a）,add(Xi, Yj, inf), add(Yj, T, b)；首先先加上所有正权值，然后跑最小割，减去最小割就是答案。（最小割中要么舍弃Xi的权值，要么支出Yj的权值）

最大权独立集：

和上面思想很接近，转为总权值-最小点权覆盖，然后最小割处理；

最小路径覆盖、最大独立集、最小点覆盖、最长反链：



最大独立集=n-最小点覆盖

最小点覆盖=最大匹配数

最长反链=最小链覆盖（Dilworth定理）

链：该集合中每个元素都可以比较大小（全序集）

反链是一个点的集合，这个集合中任意两点谁也不能走到谁。

最小链覆盖：Floyd传递闭包，转为最小路径覆盖

**图的度数序列：**

给定每个点的读数，判断是否存在树

N个带标号的顶点的树的个数是n^(n-2)

priority\_queue<int>q;

queue<int>l;

bool solve(){

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

For(i,1,n) q.push(a[i]);

int k,ls;

while(!q.empty()){//一次处理一个点

k=q.top();q.pop();

if(k>(int)q.size()) return 0;

while(k--){//每次取度数最大的点

ls=q.top()-1;q.pop();

if(ls) l.push(ls);

}

while(!l.empty()) q.push(l.front()),l.pop();

}

return 1;

}

Void work(){

For(i,1,n) scanf("%d",&a[i]),sum+=a[i];

if(sum&1){//度数和一定是偶数

puts("no");

continue;

}

if(solve()) puts("yes");

else puts("no");

}

**2-sat：**

满足条件x|y，建边!*x*−>*y*，! y − > x ，剩下的跑tarjan就可以了。（一个不成立则另一个必须成立）

Tarjan求缩点：

void tarjan(int u){

low[u]=dfn[u]=++cnt;

st[++top]=u;

for(int i=be[u];i;i=ne[i]){

int v=to[i];

if(!dfn[v]){

tarjan(v);

low[u]=min(low[u],low[v]);

}

else if(!ring[v]) low[u]=min(low[u],dfn[v]);

}

if(dfn[u]==low[u]){

++tot;

int v;

do{

v=st[top--];

ring[v]=tot;

}while(v!=u);

}

}

Tarjan求割点与桥：

割点：删除该点后，联通分量个数变多；自己一个点的点集也算

桥：删除边~

void tarjan(int u,int f){

dfn[u]=low[u]=++cnt;

int tot=0;

For(i,0,G[u].size()-1){

int v=G[u][i];

if(!dfn[v]){

++tot;

tarjan(v,u);

low[u]=min(low[u],low[v]);

if(!f && tot>1) cut[u]=1;//根节点

if(f && dfn[u]<=low[v]) cut[u]=1;//割点

}

else if(v!=f) low[u]=min(low[u],dfn[v]);

}

}

For(I,1,n){

x=fa[i],y=i;

if(x && low[y]>dfn[x]) printf(“%d,%d\n”,x,y);//桥  
}

**二分图的等价定义：**

不含有「含奇数条边的环」的图。

**二分图匹配：**

匈牙利：O(n^2)

寻找未匹配的点或者可增广的路径

bool dfs(int x){

for(int i=head[x];i!=-1;i=e[i].next){

int j=e[i].to;

if(!vis[j]){

vis[j]=1;

if(match[j]==-1||dfs(match[j])){

match[j]=x;

return true;

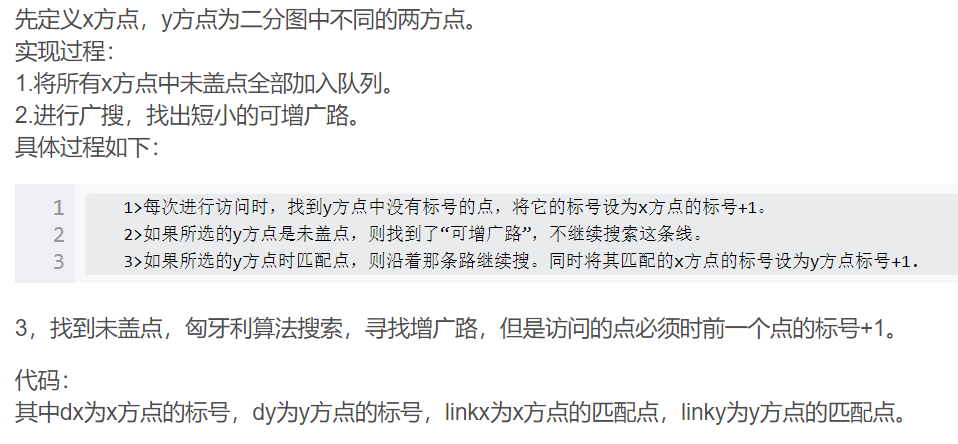
}

}

}

}

HK: O(n\*sqrt(n))



void add(int x,int y){

to[++e]=y,ne[e]=be[x],be[x]=e;

if(!linkx[x] && !linky[y]) linkx[linky[y]=x]=y,++ans;

}

bool bfs(){

bool flag=0;

int q[maxn],f=0,l=0;

Set(dx,0),Set(dy,0);

For(i,1,n){

if(!linkx[i]) q[++l]=i;

}

while(f<l){

int k=q[++f];

for(int i=be[k];i;i=ne[i]){

int u=to[i];

if(!dy[u]){

dy[u]=dx[k]+1;

if(!linky[u]) flag=1;

else dx[linky[u]]=dy[u]+1,q[++l]=linky[u];

}

}

}

return flag;

}

bool dfs(int node){

for(int i=be[node];i;i=ne[i]){

int u=to[i];

if(dy[u]==dx[node]+1){

dy[u]=0;

if(!linky[u] || dfs(linky[u])){

linkx[linky[u]=node]=u;

return 1;

}

}

}

return 0;

}

void work(){

scanf("%d%d%d",&n,&m,&s);

For(i,1,s){

int x,y;

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

add(x,y);

}

while(bfs()){

For(i,1,n){

if(!linkx[i] && dfs(i)) ++ans;

}

}

}

**二分图带权匹配KM：**

最大匹配的基础上，权值最大;

复杂度：O(n^3);

定理：若由二分图中所有满足A[i]+B[j]=w[i][j]的边C（i，j）构成的子图（即相等子图）有完备匹配，那么这个完备匹配就是二分图的最大权匹配

Lx,ly为顶标，满足lx[i]+ly[j]>=w[i][j]；

(1)初始化时为了使 LX[i] + LY[j] >= w [i,j]恒成立，将X的点的标号记为与其相连的最大边权值，Y的点标号记为0  
（2）用匈牙利算法在相等子图寻找完备匹配  
（3）若未找到完备匹配，则修改可行顶标的值，扩充相等子图  
（4）重复（2）（3）直到找到相等子图的完备匹配为止

给每个Y顶点一个“松弛量”函数 slack，每次开始找增广路时初始化为无穷大。在寻找增广路的过程中，检查边(i,j)时，如果它不在相等子图中，则让slack[j]变成原值与A [i]+B[j]-w[i,j]的较小值。这样，在修改顶标时，取所有不在交错树（匹配过程中标记的点）中的Y顶点的slack值中的最小值作为d值即可。（减小lx,增大ly）

int n;

LL delta;

int pre[N];

LL slack[N];

LL w[N][N]; //边权

LL la[N], lb[N]; //左、右部点的顶标

bool visb[N]; //访问标记，是否在交错树中

int match[N]; //右部点匹配的左部点（一个只能匹配一个嘛）

void bfs(int x){

int a, y = 0, y1 = 0;

for(int i = 1; i <= n; ++ i) pre[i] = 0, slack[i] = INF;

match[y] = x;

do{

a = match[y], delta = INF, visb[y] = true;

for(int b = 1; b <= n; ++ b){

if(!visb[b]){

if(slack[b] > la[a] + lb[b] - w[a][b])

slack[b] = la[a] + lb[b] - w[a][b], pre[b] = y;

if(slack[b] < delta) delta = slack[b], y1 = b; //还是取最小的

}

}

for(int b = 0; b <= n; ++ b){

if(visb[b]) la[match[b]] -= delta, lb[b] += delta;

else slack[b] -= delta;

}

y = y1;

}while(match[y]);

while(y) match[y] = match[pre[y]], y = pre[y];

}

LL KM(){

for(int i = 1; i <= n; ++ i) match[i] = la[i] = lb[i] = 0;

for(int i = 1; i <= n; ++ i){

for(int j = 1; j <= n; ++ j) visb[j] = false;

bfs(i);

}

LL res = 0;

for(int y = 1; y <= n; ++ y) res += w[match[y]][y];

return res;

}

**Flyod传递闭包：**

一个点是否能到另一个点

void Floyd(){

For(i,1,n){

For(j,1,n){

For(k,1,n){

map[j][k]=map[j][k]|(map[j][i]&map[i][k]);

}

}

}

}

Bitset优化：

For(I, 1, n) For(k, 1 ,n) if(map[i][k]) map[i] |= map[k];

**A\*算法：**

首先建反图，跑一次最短路算法，得到每个点到t的最短路的距离。然后用当前走的距离加上到终点的最短路的长度作为优先级进行A\*。当一个点第k次出队时，答案是它的优先级；当终点第k次出队时，答案是它已经走的路程；（可能被卡）

int be[maxn],ne[maxn],to[maxn],w[maxn],e;

int revbe[maxn],revne[maxn],revto[maxn],revw[maxn],reve;

int dis[maxm];

bool vis[maxn];

int n,m,s,t,k;

struct Astar{

int f,g,p;//第一关键字（当前最短路），第二关键字（f+到终点最短路），当前点

bool operator <(const Astar a)const{

return f==a.f?g>a.g:f>a.f;

}

};

void add(int x,int y,int z){

ne[++e]=be[x],be[x]=e,to[e]=y,w[e]=z;

revne[e]=revbe[y],revbe[y]=e,revto[e]=x,revw[e]=z;

}

void spfa(int node){

For(i,1,n) dis[i]=inf,vis[i]=0;

dis[node]=0;

int q[maxn],f=0,l=0;

q[++l]=node;vis[node]=1;

while(f<l){

int ls=q[++f];vis[ls]=0;

for(int i=revbe[ls];i;i=revne[i]){

int u=revto[i];

if(dis[u]>dis[ls]+revw[i]){

dis[u]=dis[ls]+revw[i];

if(!vis[u]){

vis[u]=1;

q[++l]=u;

}

}

}

}

}

int A\_star(int l,int r){

if(dis[l]==inf) return -1;

if(l==r) ++k;

priority\_queue<Astar>q;

Astar now;

now.g=0,now.p=l,now.f=dis[l];

q.push(now);

int sum=0;

while(!q.empty()){

Astar ls=q.top();q.pop();

if(ls.p==r){

++sum;

if(sum==k) return ls.g;

}

for(int i=be[ls.p];i;i=ne[i]){

now.p=to[i];

now.g=ls.g+w[i];

now.f=now.g+dis[to[i]];

q.push(now);

}

}

return -1;

}

void work(){

scanf("%d%d",&n,&m);

For(i,1,m){

int x,y,z;

scanf("%d%d%d",&x,&y,&z);

add(x,y,z);

}

scanf("%d%d%d",&s,&t,&k);

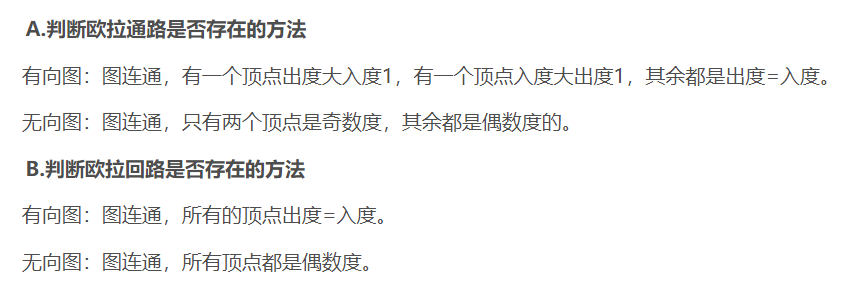
spfa(t);

printf("%d\n",A\_star(s,t));

}

**欧拉回路：**

（这里的是直接dfs，fleury是一个比较优秀的算法）



void fleury(int node){//一定能走出个回路来

For(i,1,n)

if(map[node][i])

--map[node][i],--map[i][node],fleury(i);

st.push(node);

}

最短路Dijkstra堆优化

#include<bits/stdc++.h>

#define For(aa, bb, cc) for(int aa = (bb); aa <= (int)(cc); ++aa)

#define Pair pair<LL, int>

#define mk(x, y) make\_pair(x, y)

using namespace std;

typedef long long LL;

const int maxn = 1e5 + 10;

int n, m, s;

int be[maxn], ne[maxn << 1], to[maxn << 1], w[maxn << 1], e;

LL dis[maxn];

priority\_queue<Pair, vector<Pair >, greater<Pair > >q;

inline void add\_edge(int x, int y, int z){

to[++e] = y, ne[e] = be[x], be[x] = e, w[e] = z;;

}

void Dijkstra(){

For(i, 1, n) dis[i] = -1;

dis[s] = 0;

q.push(mk(dis[s], s));

while(!q.empty()){

Pair res = q.top();

q.pop();

LL d = res.first;

int u = res.second;

if(d > dis[u]) continue;

for(int i = be[u]; i; i = ne[i]){

int v = to[i];

if(dis[v] == -1 || dis[v] > d + w[i]){

dis[v] = d + w[i];

q.push(mk(dis[v], v));

}

}

}

}

int main(){

#ifndef ONLINE\_JUDGE

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

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

#endif

std::ios::sync\_with\_stdio(false);

std::cin.tie(nullptr);

cin >> n >> m >> s;

int x, y, z;

For(i, 1, m) cin >> x >> y >> z, add\_edge(x, y, z);

Dijkstra();

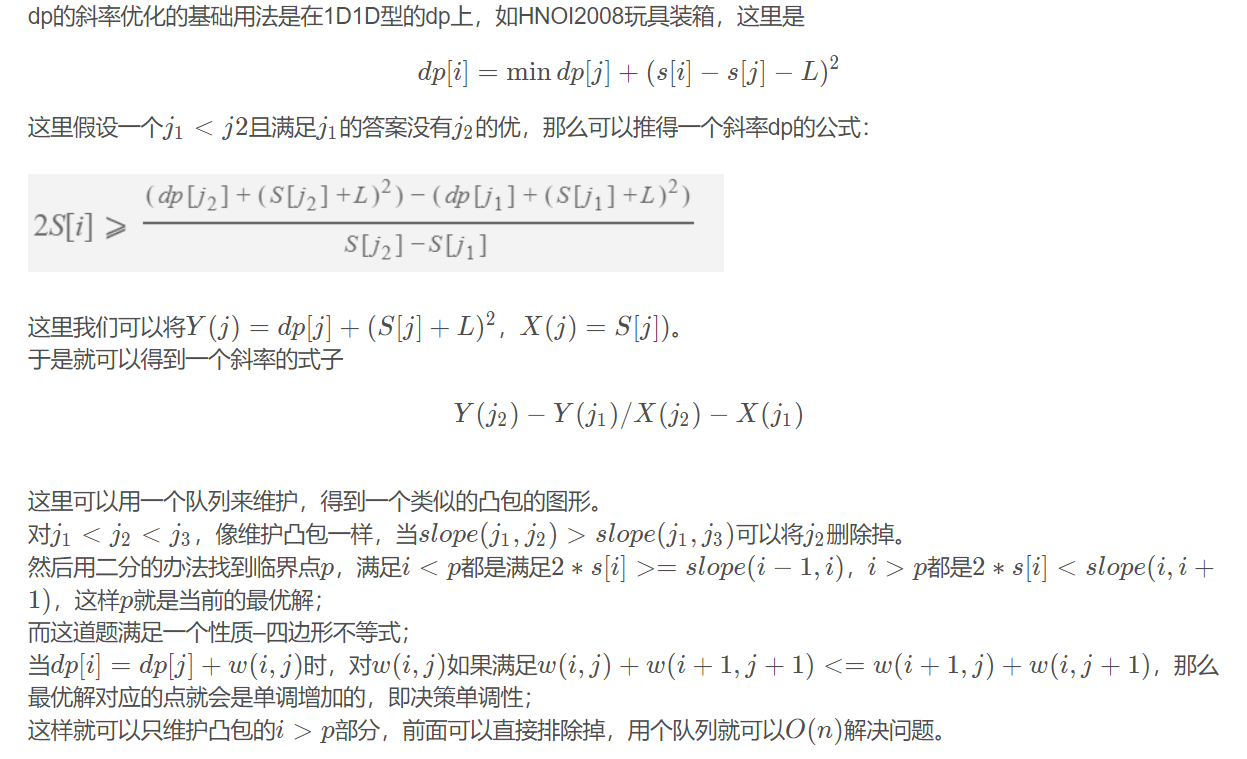
For(i, 1, n) printf("%lld ", dis[i]);

return 0;

}

# 3.dp

斜率优化：



维护凸包时，斜率的比较来删除j2点，满足无论slope(x,y)与2s[i]是什么大小关系，都有j1或者j3优于j2.

（代码不是玩具的代码，可能符号会相反）

   斜率为负,从小到大,维护U左半部分凸包,求最小y截距

   斜率为负,从大到小,维护倒U右半部分凸包,求最大y截距

   斜率为正,从小到大,维护U右半部分凸包,求最小y截距

   斜率为正,从大到小,维护倒U左半部分凸包,求最大y截距

LL X(int x,int y){ return w[x]-w[y]; }//x>y

LL Y(int x,int y){ return dp[x]+w[x]\*w[x]-dp[y]-w[y]\*w[y]; }//x>y

Void work(){

q[++r]=0;

For(i,1,n){

while(l<r && 2\*w[i]\*X(q[l+1],q[l])>=Y(q[l+1],q[l])) ++l;

dp[i]=dp[q[l]]+(w[i]-w[q[l]])\*(w[i]-w[q[l]])+m;

while(l<r && Y(q[r],q[r-1])\*X(i,q[r])>=Y(i,q[r])\*X(q[r],q[r-1])) --r;

q[++r]=i;

}

}

**单调队列：**

用单调栈或单调队列就可以取前一段的最值；(双端)

背包问题：

**0/1背包：**倒过来枚举，或者另外使用数组保存当前层更新的状态；

**完全背包：**正向枚举

倒过来枚举和正向枚举：两种枚举方式是考虑了当前的更新会不会立刻作为后面更新的一部分，也就是能够购买几次物品；如果能够无限次使用，就可以正向更新；如果只能一次，那么就得反向更新；

**单调队列优化多重背包：**单调队列的应用

**树形依赖背包：**当选一个物品的前提是选另一件物品；

有三种种解决的办法:

1.强制选择当前节点，用当前的状态去更新子树信息，然后再用子树信息更新父亲。

2.对于这一类问题，只有选择了父亲节点，才能选择子树。所以我们用dfs序来进行dp，对于子树，再dfs序上是连续的一段，可以考虑前i个dfs序带来的最大收益，正常的背包即可，稍微注意整个子树都不选的更新即可。

3.第三种解决方法比较巧妙，将多叉树转为了二叉树。对于正常的树而言，我们的左右儿子都是儿子节点，但是在转化中，我们将当前节点的左儿子设置为它的任意一个儿子，将它的右儿子连接为与它所有深度相同的节点。这样对于选择当前节点就可以沿着左儿子走，否则只能沿着右儿子走，二维DP即可。

dp[i][j]表示选第i个点，从子树中选j个点出来的代价;

int dfs(int node){

if(node>n-m){

dp[node][1]=c[node];

return 1;

}//题目中的叶子节点

int res=0,sum=0;

for(int i=be[node];i;i=ne[i]){

int u=to[i];

res=dfs(u);sum+=res;

Forr(j,sum,1) For(k,0,res)

If(j>=k) dp[node][j]=max(dp[node][j], dp[u][k]+dp[node][j-k]-w[i]);

}

return sum;

}

状压dp:

枚举子集：for(int i=x;i>0;i=x&(i-1)) a=i,b=x^i;//这里需要单独考虑空集

取x第i位：y = (x >> (I - 1)) & 1;

将x第i位取反： x ^= 1 << (I – 1);

将x第i位变为1： x |= 1 << (I – 1);

将x第i位变为0： x &= ~(1 << (I – 1));

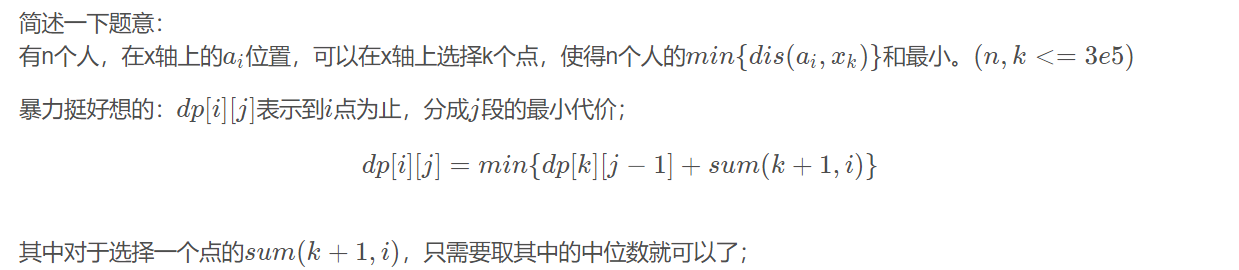
将x最右边的1变为0： x = x & (x – 1);

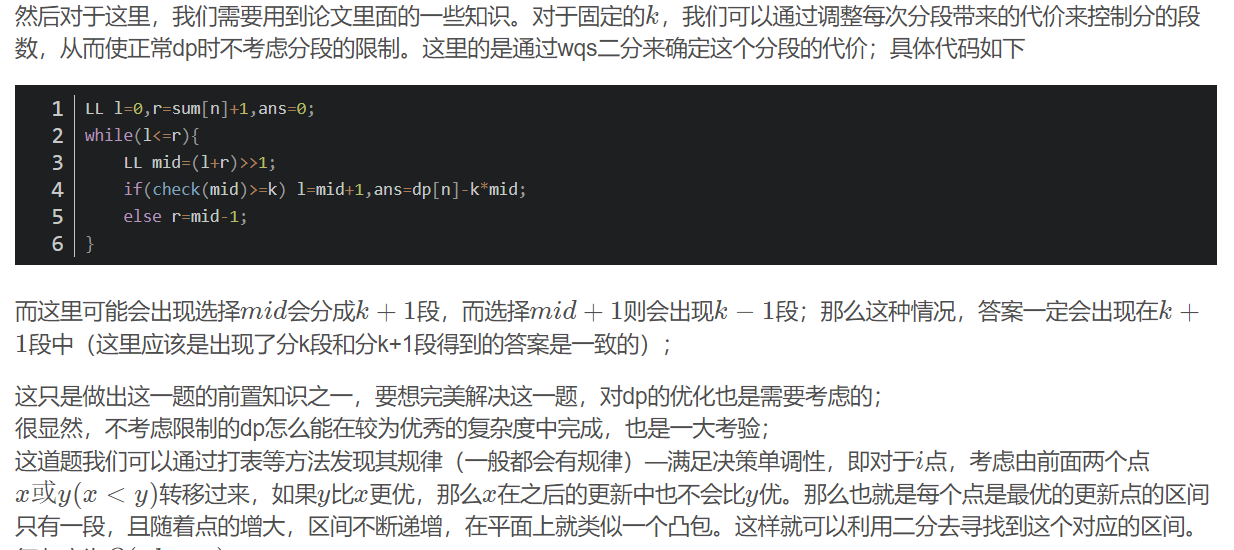
取出x最右边的1： y = x & (x – 1);

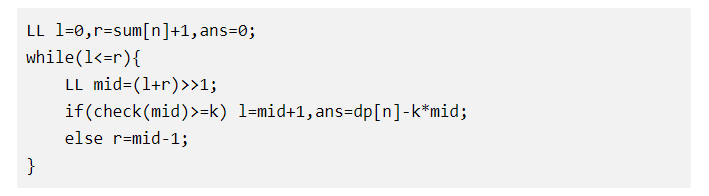
将x最右的0变为1： x |= x + 1;

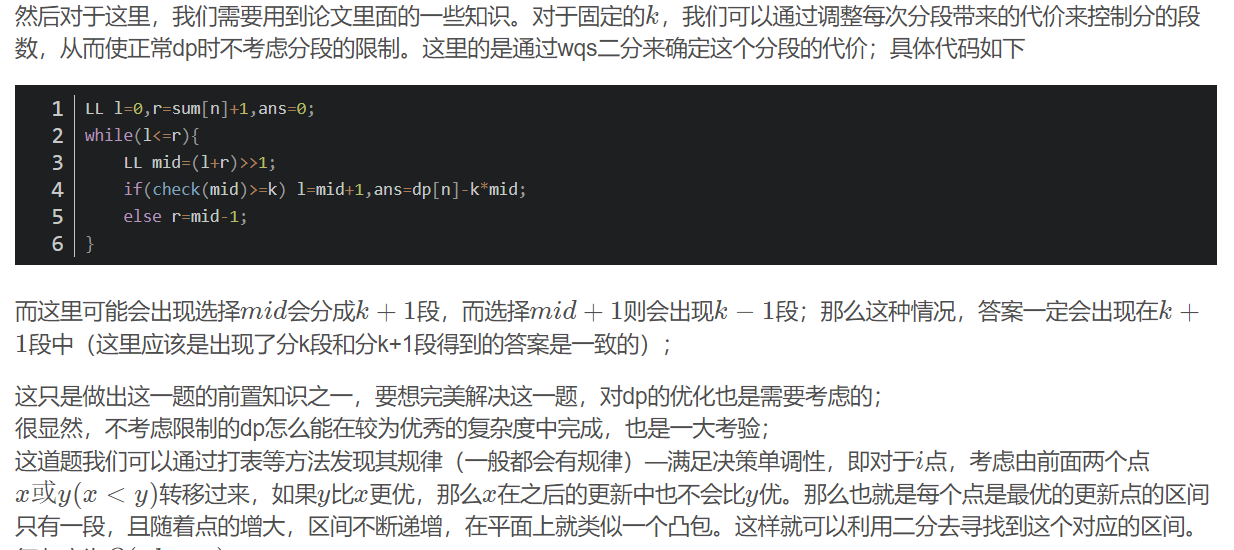
判断是否有两个连续的1： y = x & (x << 1) 可扩展到n个

Wqs二分、nlogn的dp优化：









int n,k;

int a[maxn],num[maxn];

LL sum[maxn],dp[maxn];

struct intv{//p+1 is the best left-point in interval [l,r]

int l,r,p;

};

LL calc(int l,int r){

if(l>r) return 0;

return sum[r]-sum[(l+r)>>1]-sum[(l+r-1)>>1]+sum[l-1];

}

bool cmp(int x,int y,int r){

LL sx=dp[x]+calc(x+1,r),sy=dp[y]+calc(y+1,r);

return sx<=sy;

}

int find(intv x,int y){

int l=x.l,r=x.r+1;

while(l<r){

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

if(cmp(y,x.p,mid)) r=mid;

else l=mid+1;

}

return l;

}

int check(LL p){

intv q[maxn];

int l=1,r=0;

dp[0]=num[0]=0;

q[++r]=(intv){1,n,0};

For(i,1,n){

while(l<=r && q[l].r<i) ++l;

dp[i]=dp[q[l].p]+calc(q[l].p+1,i)+p;

num[i]=num[q[l].p]+1;

while(l<=r && cmp(i,q[r].p,q[r].l)) --r;

int pos=i+1;

if(l<=r) pos=find(q[r],i),q[r].r=pos-1;

q[++r]=(intv){pos,n,i};

}

return num[n];

}

int main(){

#ifndef ONLINE\_JUDGE

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

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

#endif

scanf("%d%d",&n,&k);

For(i,1,n) scanf("%d",&a[i]),sum[i]=sum[i-1]+a[i];

LL l=0,r=sum[n]+1,ans=0;

while(l<=r){

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

if(check(mid)>=k) l=mid+1,ans=dp[n]-k\*mid;

else r=mid-1;

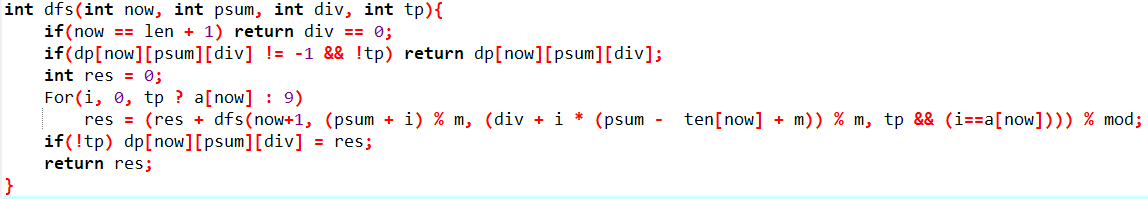
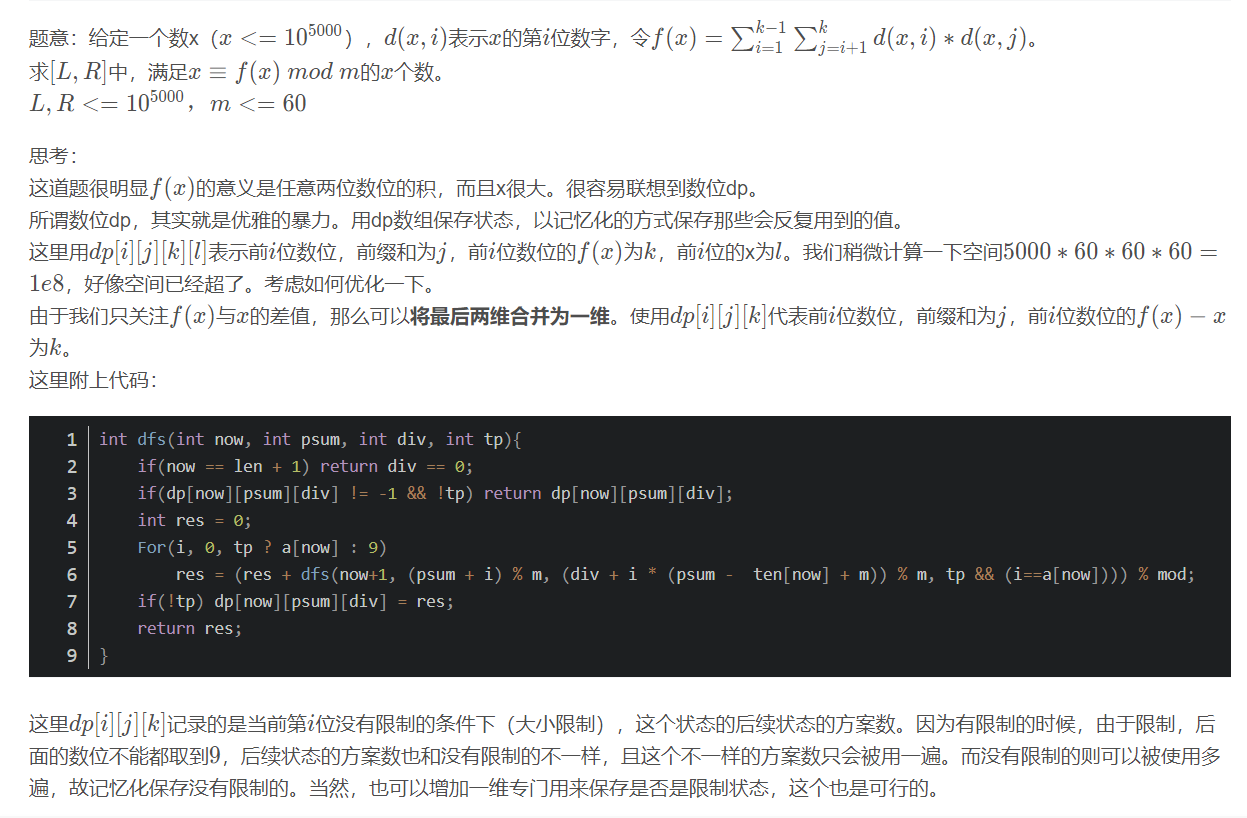
}

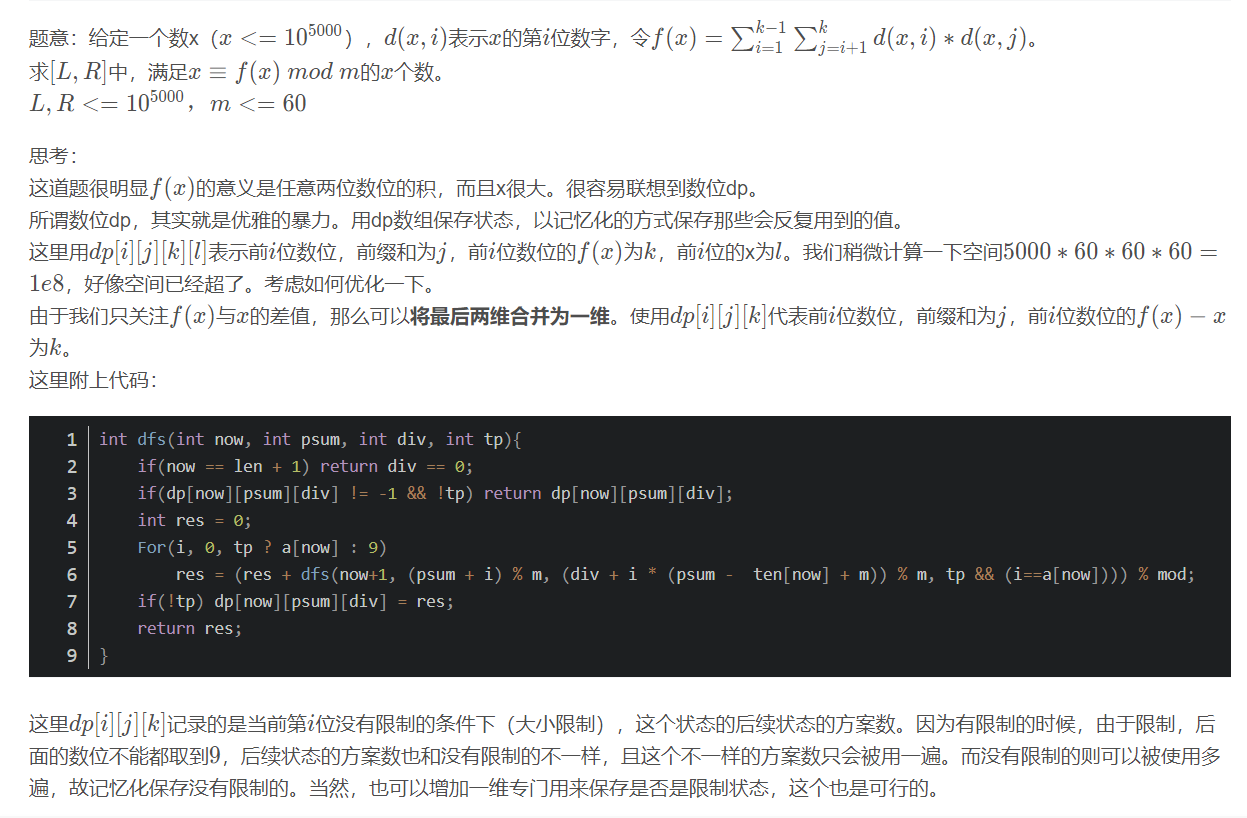
printf("%lld\n",ans);

return 0;

}

数位DP





# 4.树：

LCA：

倍增：

void dfs(int node,int fa,int d){

dep[node]=d;

f[node][0]=fa;

for(int i=0;f[f[node][i]][i];++i)

f[node][i+1]=f[f[node][i]][i];

for(int i=be[node];i;i=ne[i]){

int u=to[i];

if(u==fa) continue;

dfs(u,node,d+1);

}

}

int query(int x,int y){

if(dep[x]<dep[y]) swap(x,y);

int t=0;

while(f[x][t+1]) ++t;

Forr(i,t,0) if(dep[f[x][i]]>=dep[y]) x=f[x][i];

if(x==y) return x;

Forr(i,t,0) if(f[x][i]!=f[y][i]) x=f[x][i],y=f[y][i];

return f[x][0];

}

树链剖分：

void dfs1(int node,int father,int d){

dep[node]=d;

size[node]=1;

fa[node]=father;

for(int i=be[node];i;i=ne[i]){

int u=to[i];

if(u==father) continue ;

dfs1(u,node,d+1);

size[node]+=size[u];

if(!son[node] || size[u]>size[son[node]]) son[node]=u;

}

}

void dfs2(int node,int father){

top[node]=father;

tid[node]=++cnt;

rank[tid[node]]=node;

if(son[node]==-1) return ;

dfs2(son[node],father);

for(int i=be[node];i;i=ne[i]){

int u=to[i];

if(u==fa[node] || u==son[node]) continue ;

dfs2(u,u);

}

}

int query(int x,int y){

while(top[x]!=top[y]){

if(dep[top[x]]<dep[top[y]]) swap(x,y);

x=fa[top[x]];

}

return dep[x]>dep[y]?y:x;

}

DFS序：

子树在dfs序上连续；

欧拉序：

RMQ求LCA

树链剖分：

对点，正常做；

对边，考虑到最上面那条边下面那个点

void add(int x,int y){

to[++e]=y,next[e]=begin[x],begin[x]=e;

}

void dfs1(int node,int father,int deep){

dep[node]=deep;

fa[node]=father;

size[node]=1;

for(int i=begin[node];i;i=next[i]){

int u=to[i];

if(u!=father){

dfs1(u,node,deep+1);

size[node]+=size[u];

if(son[node]==-1 || size[u]>size[son[node]]) son[node]=u;

}

}

return ;

}

void dfs2(int node,int f){

tid[node]=++md;

rank[tid[node]]=node;

top[node]=f;

if(son[node]==-1) return ;

dfs2(son[node],f);

for(int i=begin[node];i;i=next[i]){

int u=to[i];

if(u==fa[node] || u==son[node]) continue;

dfs2(u,u);

}

return ;

}

void push\_up(int node){

tree[node].sum=tree[lr].sum+tree[rr].sum-(tree[lr].rc==tree[rr].lc);

tree[node].lc=tree[lr].lc,tree[node].rc=tree[rr].rc;

return ;

}

void create\_tree(int node,int l,int r){

lazy[node]=0;

if(l==r){

tree[node].sum=1,tree[node].lc=tree[node].rc=a[rank[l]];

return ;

}

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

create\_tree(lr,l,mid);

create\_tree(rr,mid+1,r);

push\_up(node);

return ;

}

void push\_down(int node){

if(!lazy[node]) return ;

tree[lr].sum=tree[rr].sum=1;

tree[lr].lc=tree[lr].rc=tree[rr].lc=tree[rr].rc=lazy[node];

lazy[lr]=lazy[rr]=lazy[node];

lazy[node]=0;

return ;

}

int query(int node,int l,int r,int be,int en){

if(be==l && en==r){

return tree[node].sum;

}

push\_down(node);

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

if(mid>=en) return query(lr,l,mid,be,en);

else if(mid<be) return query(rr,mid+1,r,be,en);

else{

int sb=query(lr,l,mid,be,mid)+query(rr,mid+1,r,mid+1,en);

if(tree[lr].rc==tree[rr].lc) --sb;

return sb;

}

}

int query\_color(int node,int l,int r,int mdzz){

if(l==r) return tree[node].lc;

push\_down(node);

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

if(mid>=mdzz) return query\_color(lr,l,mid,mdzz);

else return query\_color(rr,mid+1,r,mdzz);

}

int Query(int l,int r){

int ans=0;

while(top[l]!=top[r]){

if(dep[top[l]]<dep[top[r]]) swap(l,r);

ans+=query(1,1,n,tid[top[l]],tid[l]);

int lcc=query\_color(1,1,n,tid[top[l]]),rcc=query\_color(1,1,n,tid[fa[top[l]]]);

l=fa[top[l]];

if(lcc==rcc) --ans;

}

if(dep[l]>dep[r]) swap(l,r);

ans+=query(1,1,n,tid[l],tid[r]);

return ans==0?1:ans;

}

void update(int node,int l,int r,int be,int en,int ad){

if(be==l && en==r){

tree[node].sum=1,tree[node].lc=ad,tree[node].rc=ad;

lazy[node]=ad;

return ;

}

push\_down(node);

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

if(be>mid) update(rr,mid+1,r,be,en,ad);

else if(en<=mid) update(lr,l,mid,be,en,ad);

else{

update(lr,l,mid,be,mid,ad),update(rr,mid+1,r,mid+1,en,ad);

}

push\_up(node);

return ;

}

void change(int l,int r,int ad){

while(top[l]!=top[r]){

if(dep[top[l]]<dep[top[r]]) swap(l,r);

update(1,1,n,tid[top[l]],tid[l],ad);

l=fa[top[l]];

}

if(dep[l]>dep[r]) swap(l,r);

update(1,1,n,tid[l],tid[r],ad);

return ;

}

LCT：（待补）

**struct** LCT{

void **update**(int node){

pos[node]=node;

**if**(w[pos[node]]<w[pos[lr]]) pos[node]=pos[lr];

**if**(w[pos[node]]<w[pos[rr]]) pos[node]=pos[rr];

}

bool **is\_root**(int node){

**return** tree[fa[node]][0]!=node && tree[fa[node]][1]!=node;

}

void **Pushdown**(int node){

**if**(rev[node]){

rev[node]^=1;rev[tree[node][0]]^=1,rev[tree[node][1]]^=1;

**swap**(tree[node][0],tree[node][1]);

}

}

void **pushdown**(int node){

**if**(!**is\_root**(node)) **pushdown**(fa[node]);

**Pushdown**(node);

}

void **rotate**(int node){

int father=fa[node],grandfather=fa[father];

bool flag=tree[father][1]==node;

**if**(!**is\_root**(father)) tree[grandfather][tree[grandfather][1]==father]=node;

fa[node]=grandfather;

tree[father][flag]=tree[node][flag^1],fa[tree[father][flag]]=father;

tree[node][flag^1]=father,fa[father]=node;

**update**(father);

}

void **splay**(int node){

**pushdown**(node);

**while**(!**is\_root**(node)){

int father=fa[node],grandfather=fa[father];

**if**(!**is\_root**(father)){

**if**((tree[father][0]==node)^(tree[grandfather][0]==father)) **rotate**(node);

**else** **rotate**(father);

}

**rotate**(node);

}**update**(node);

}

void **access**(int x){

**for**(int t=0;x;t=x,x=fa[x])

**splay**(x),tree[x][1]=t,**update**(x);

}

void **makeroot**(int x){

**access**(x);**splay**(x);rev[x]^=1;

}

void **link**(int x,int y){

**makeroot**(x);fa[x]=y;

}

void **cut**(int x,int y){

**makeroot**(x);**access**(y);**splay**(y);

tree[y][0]=fa[x]=0;**update**(y);

}

int **query**(int x,int y){

**makeroot**(x),**access**(y),**splay**(y);

**return** pos[y];

}

}d;

点分治：

Root点的定义：Root的子树中节点最多的子树的节点数最小；

将问题分解到Root的子树中；最主要问题是怎么把重复的计算去掉；

以dis[u,v]<=k的题为例：

将每个点到Root的距离算出来，统计dis[i]+dis[j]<=k的对数(sort后从两端考虑)，然后会发现会有边被重复走了（选择的两个点在同一个子树中），这个在进入子树u时，将子树u与Root之间的边强制选择，就可以算出重复的部分；

void getroot(int node,int fa){

int u;

size[node]=1;f[node]=0;

for(int i=be[node];i;i=ne[i]){

u=to[i];

if(u==fa || vis[u]) continue;

getroot(u,node);

size[node]+=size[u];

f[node]=max(f[node],size[u]);

}

f[node]=max(f[node],sum-size[node]);

if(f[root]>f[node]) root=node;

}

void getdep(int node,int fa){

int u;

dep[++cnt]=dis[node];

for(int i=be[node];i;i=ne[i]){

u=to[i];

if(u==fa || vis[u]) continue;

dis[u]=dis[node]+w[i];

getdep(u,node);

}

}

int calc(int node,int d){

dis[node]=d;cnt=0;

getdep(node,0);

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

int res=0,l=1,r=cnt;

while(l<r){

if(dep[l]+dep[r]<=k) res+=r-l,++l;

else --r;

}

return res;

}

void dfs(int node){

int u;

ans+=calc(node,0);//统计所有

vis[node]=1;

for(int i=be[node];i;i=ne[i]){

u=to[i];

if(vis[u]) continue;

ans-=calc(u,w[i]);//减掉重复

sum=size[u];

root=0;

getroot(u,0);

dfs(root);

}

}

Dfs(root);

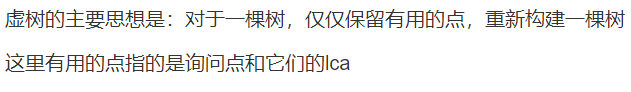
重链剖分：

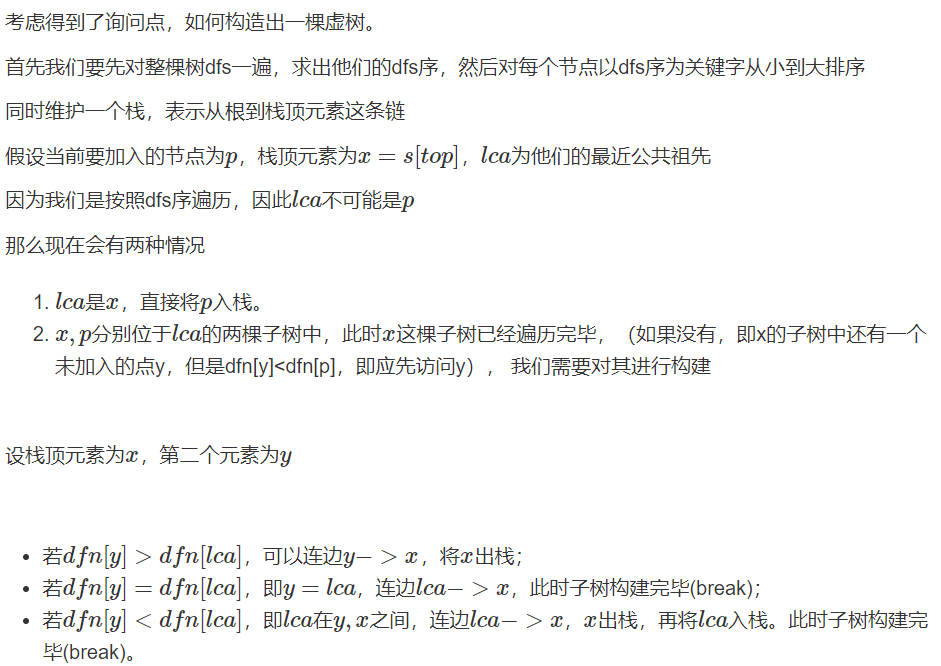
要求：没有修改，子树询问；

对树进行轻重链剖分，优先暴力处理轻儿子，然后暴力清空；最后处理重儿子，不进行清空，在处理父亲时重儿子就不用计算了。时间复杂度：O(nlog n)；

性质：一个节点到根的路径上轻边个数不会超过logn条；重边只会被访问一次，轻边最多logn次；

虚树：





#include<cstdio>

#include<iostream>

#include<algorithm>

#define For(aa,bb,cc) for(int aa=(bb);aa<=(int)(cc);++aa)

using namespace std;

typedef long long LL;

const int maxn=250010,maxm=500010,K=19;

const LL inf=1e18;

int n,m;

int be[maxn],ne[maxm],to[maxm],w[maxm],e;

int f[maxn][K+1],dfn[maxn],dfr[maxn],dep[maxn];

int s[maxn],top;

int a[maxn\*2],cnt;

bool vis[maxn];

LL val[maxn];

template<class T>void read(T &x){

x=0;char c=getchar();

while(!isdigit(c)) c=getchar();

while(isdigit(c)) x=(x<<1)+(x<<3)+(c^48),c=getchar();

}

inline void add\_edge(int x,int y,int z){

to[++e]=y,ne[e]=be[x],be[x]=e,w[e]=z;

to[++e]=x,ne[e]=be[y],be[y]=e,w[e]=z;

}

inline void add(int x,int y){ to[++e]=y,ne[e]=be[x],be[x]=e; }

void dfs(int node){

static int tot=0;

dfn[node]=++tot;

for(int i=be[node];i;i=ne[i]){

int u=to[i];

if(u==f[node][0]) continue;

f[u][0]=node;

dep[u]=dep[node]+1;

val[u]=min(val[node],(LL)(w[i]));

dfs(u);

}

dfr[node]=tot;

}

int lca(int x,int y){

if(dep[x]<dep[y]) swap(x,y);

int d=dep[x]-dep[y];

for(int i=0,p=1;i<=K;p<<=1,++i)

if(d&p) x=f[x][i];

if(x==y) return x;

for(int i=K;i>=0;--i){

if(f[x][i]==f[y][i]) continue;

x=f[x][i],y=f[y][i];

}

return f[x][0];

}

bool cmp(int x,int y){

return dfn[x]<dfn[y];

}

LL dp(int node){

if(vis[node]) return val[node];

LL sum=0;

for(int i=be[node];i;i=ne[i]) sum+=dp(to[i]);

return min(sum,val[node]);

}

int main(){

#ifndef ONLINE\_JUDGE

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

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

#endif

int x,y,z,k,res;

read(n);

For(i,1,n-1) read(x),read(y),read(z),add\_edge(x,y,z);

val[1]=inf;dfs(1);

For(j,1,K) For(i,1,n) f[i][j]=f[f[i][j-1]][j-1];

For(i,1,n) be[i]=0;

read(m);

while(m--){

read(k);cnt=0,top=0;e=0;

For(i,1,k) read(x),a[++cnt]=x,vis[x]=1;

sort(a+1,a+1+cnt,cmp);

res=cnt;

For(i,1,res-1) a[++cnt]=lca(a[i],a[i+1]);

a[++cnt]=1;

sort(a+1,a+1+cnt,cmp);

cnt=unique(a+1,a+1+cnt)-a-1;

For(i,1,cnt){

while(top && dfr[s[top]]<dfn[a[i]]) --top;

add(s[top],a[i]);

s[++top]=a[i];

}

printf("%lld\n",dp(1));

For(i,1,cnt) vis[a[i]]=0,be[a[i]]=0;

}

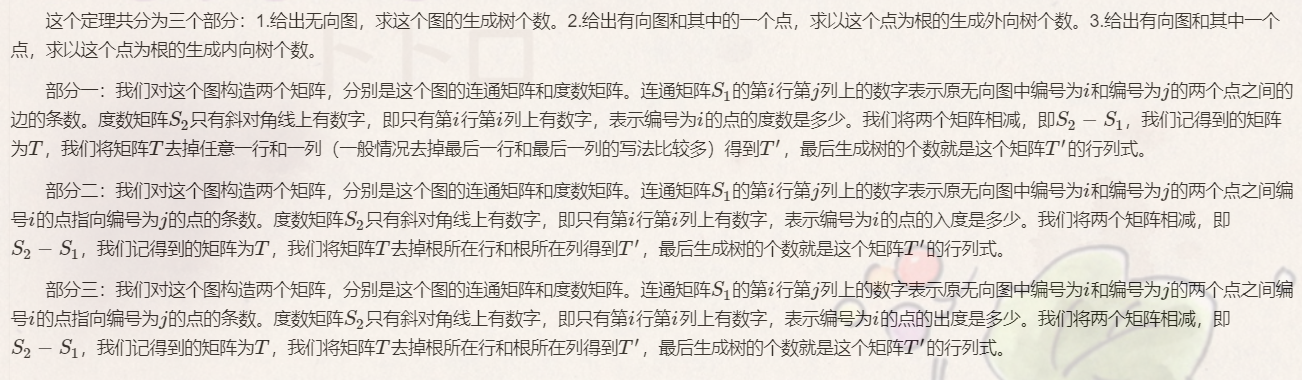
return 0;

}

启发式合并：

将小的和到大的中去，每个元素最多被合并log次（在树上有点类似轻重链剖分）

生成树计数：矩阵树定理



# 5.数学：

逆元：

Qpow(a,mod-2);//费马小定理 a^(p-1) ≡1（mod p）

多项式乘法FFT：

const double pi = acos(-1);

const int maxn = 2e5 + 10;

int n, n1, n2;

int rev[maxn << 1];

struct Complex{

double x, y;

Complex operator +(Complex tmp){

return (Complex){x + tmp.x, y + tmp.y};

}

Complex operator -(Complex tmp){

return (Complex){x - tmp.x, y - tmp.y};

}

Complex operator \*(Complex tmp){

return (Complex){x \* tmp.x - y \* tmp.y, x \* tmp.y + y \* tmp.x};

}

}a[maxn], b[maxn];

void init\_rev(){

For(i, 0, n - 1) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) \* (n >> 1));

}

void FFT(Complex \*y, int tp){

For(i, 0, n - 1) if(i < rev[i]) swap(y[i], y[rev[i]]);

for(int k = 2; k <= n; k <<= 1){

Complex wn = (Complex){cos(2 \* pi / k), sin(2 \* pi / k) \* tp};

for(int i = 0; i < n; i += k){

Complex x = (Complex){1, 0};

for(int j = 0; j < (k >> 1); ++j, x = x \* wn){

Complex y0 = y[i + j], y1 = x \* y[i + j + (k >> 1)];

y[i + j] = y0 + y1;

y[i + j + (k >> 1)] = y0 - y1;

}

}

}

}

int main(){

#ifndef ONLINE\_JUDGE

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

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

#endif

char s1[maxn], s2[maxn];

while(scanf("%s%s", s1, s2) != EOF){

int l = 0, r = 0;

if(s1[0] == '-') l = 1;

if(s2[0] == '-') r = 1;

n1 = strlen(s1) - 1 - l;

n2 = strlen(s2) - 1 - r;

n = 1;

while(n <= (n1 + n2)) n <<= 1;

For(i, 0, n) a[i].x = a[i].y = b[i].x = b[i].y = 0;

For(i, 0, n1) a[n1 - i].x = s1[i + l] - '0';

For(i, 0, n2) b[n2 - i].x = s2[i + r] - '0';

init\_rev();

FFT(a, 1);

FFT(b, 1);

For(i, 0, n - 1) a[i] = a[i] \* b[i];

FFT(a, -1);

int ans[maxn << 1], pre = 0;

For(i, 0, n1 + n2){

int x = (int)(a[i].x / n + 0.5) + pre;

pre = x / 10;

x %= 10;

ans[n1 + n2 - i] = x;

}

if(l ^ r) putchar('-');

if(pre) printf("%d", pre);

For(i, 0, n1 + n2) printf("%d", ans[i]);

puts("");

}

}

素数筛：

For(I,2,N){

If(!np[i]) pr[++tot]=I;

For(int j=1;j<=tot && 1ll\*i\*pr[j]<=N;++j){

Np[i\*pr[j]]=1;

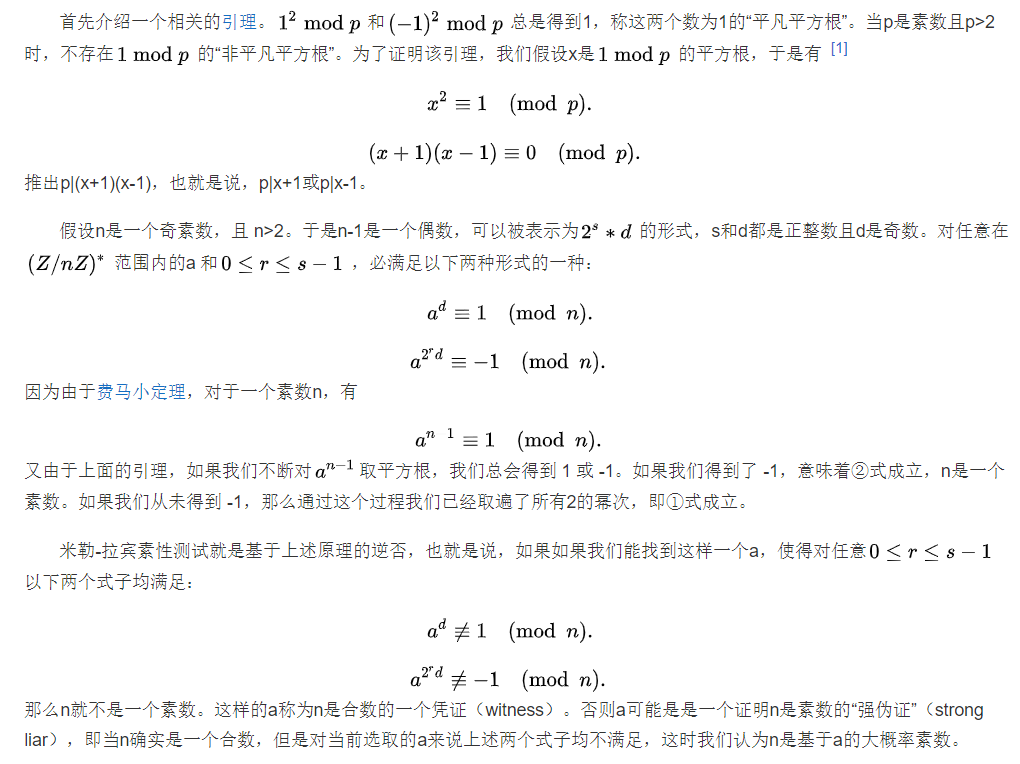
If(i%pr[j]==0) break;  
}

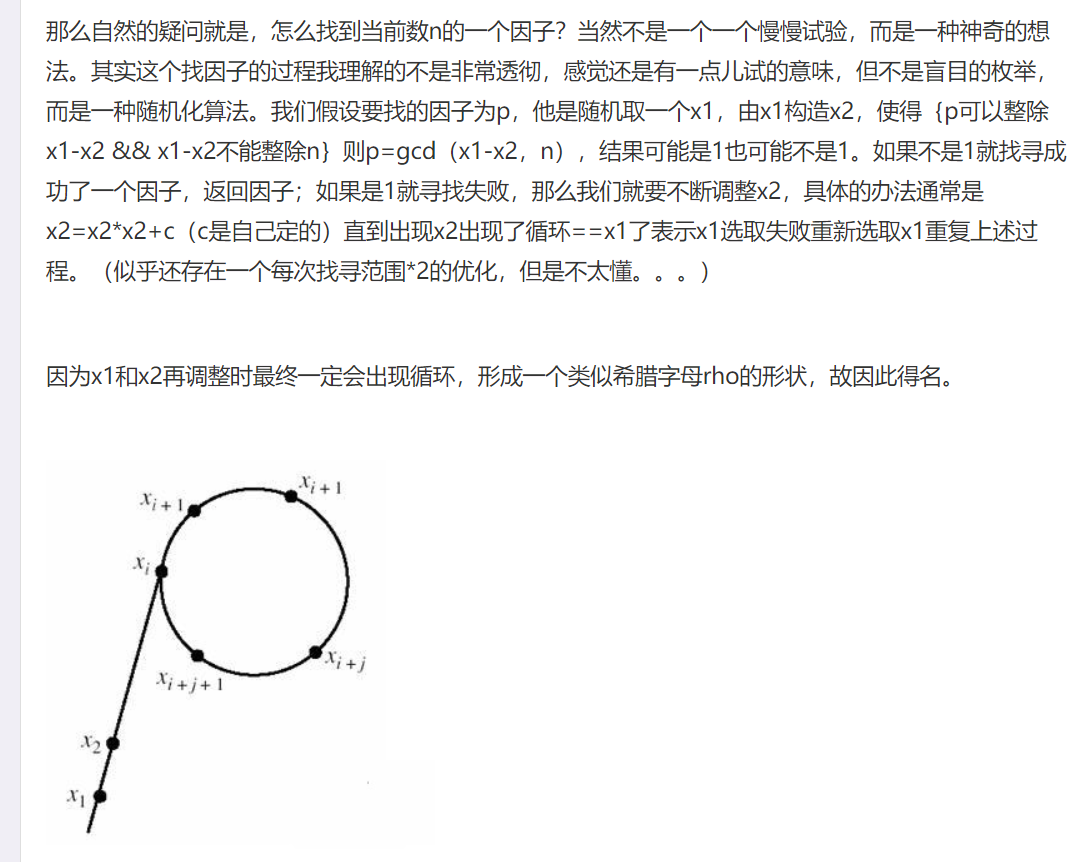
}

Miller Rabin素数测试&pollard-rho:

Poj1811: 2<=n<2^54，如果n是素数直接输出，否则求N的最小质因数。

如果要分解质因数的话，用一个map保存质数因子就可以了





#include<bits/stdc++.h>

#define For(aa, bb, cc) for(int aa = (bb); aa <= (int)(cc); ++aa)

using namespace std;

typedef long long LL;

const int maxn = 50, N = 1e5;

LL num[maxn];

map<LL, int> P;

LL gcd(LL x, LL y){

return y ? gcd(y, x % y) : x;

}

LL qmul(LL x, LL y, LL z){

LL res = 0;

while(y){

if(y & 1) res = (res + x) % z;

x = (x + x) % z;

y >>= 1;

}

return res;

}

LL qpow(LL x, LL y, LL z){

LL res = 1;

while(y){

if(y & 1) res = qmul(res, x, z);

x = qmul(x, x, z);

y >>= 1;

}

return res;

}

LL pollard\_rho(LL x, int c){

LL i = 1, y = rand() % (x - 1) + 1, z = y, k = 2;

while(1){

++i;

y = (qmul(y, y, x) + c) % x;

LL p = gcd((z - y + x) % x, x);

if(p != 1 && p != x) return p;

if(z == y) return x;

if(i == k){

z = y;

k <<= 1;

}

}

}

bool miller\_rabin(LL x){

if(x == 2) return 1;

int tot = 0;

LL y = x - 1;

while(!(y & 1)){

++tot;

y >>= 1;

}

For(i, 1, 20){

LL z = rand() % (x - 2) + 2;

num[0] = qpow(z, y, x);

For(j, 1, tot){

num[j] = qmul(num[j - 1], num[j - 1], x);

if(num[j] == 1 && num[j - 1] != 1 && num[j - 1] != x - 1) return 0;

}

if(num[tot] != 1) return 0;

}

return 1;

}

void find(LL x, int c){

if(x == 1) return ;

if(miller\_rabin(x)){

if(!P.count(x)) P[x] = 1;

else P[x] = P[x] + 1;

return ;

}

LL p = x, k = c;

while(p >= x) p = pollard\_rho(p, c--);

find(p, k);

find(x / p, k);

}

int main(){

#ifndef ONLINE\_JUDGE

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

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

#endif

LL n;

scanf("%lld", &n);

LL sum = 1;

P.clear();

memset(num, 0, sizeof num);

find(n, 120);

map<LL, int>::iterator it;

for(it = P.begin(); it != P.end(); ++it){

pair<LL, int> tmp = \*it;

LL x = tmp.first, y = tmp.second;

For(j, 1, y) n /= x;

sum \*= (y + 1);

}

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

return 0;

}

ull pollard(ull n) {  
auto f = [n](ull x) { return modmul(x, x, n) + 1; };  
ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;  
while (t++ % 40 || \_\_gcd(prd, n) == 1) {  
if (x == y) x = ++i, y = f(x);  
if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;  
x = f(x), y = f(f(y));  
}  
return \_\_gcd(prd, n);  
}  
vector<ull> factor(ull n) {  
if (n == 1) return {};  
if (isPrime(n)) return {n};  
ull x = pollard(n);  
auto l = factor(x), r = factor(n / x);  
l.insert(l.end(), all(r));  
return l;  
}

Min\_25:计算n以内素数和

const int N = 1e7 + 1e3, Lim = 1e7;

typedef long long LL;

typedef \_\_int128 ll;

int prime[N], cnt = 0; bitset<N> is\_prime;

ll sump[N];

void Init(int maxn) {

is\_prime.set(); is\_prime[0] = is\_prime[1] = false;

For (i, 2, maxn) {

sump[i] = sump[i - 1];

if (is\_prime[i])

prime[++ cnt] = i, sump[i] += i;

For (j, 1, cnt) {

register int res = i \* prime[j];

if (res > maxn) break;

is\_prime[res] = false;

if (!(i % prime[j])) break;

}

}

}

inline ll Sum(ll x) { return x \* (x + 1) / 2 - 1; }

int tot = 0;

ll Sump(ll n, int m) {

if (n <= Lim && n < (ll)prime[m + 1] \* prime[m + 1]) return sump[n];

for (; n < (ll)prime[m] \* prime[m]; -- m);

ll res = Sum(n);

for (; m; -- m)

res -= (Sump(n / prime[m], m - 1) - /\*Sump(prime[m] - 1, m - 1)\*/ sump[prime[m] - 1]) \* prime[m];

return res;

}

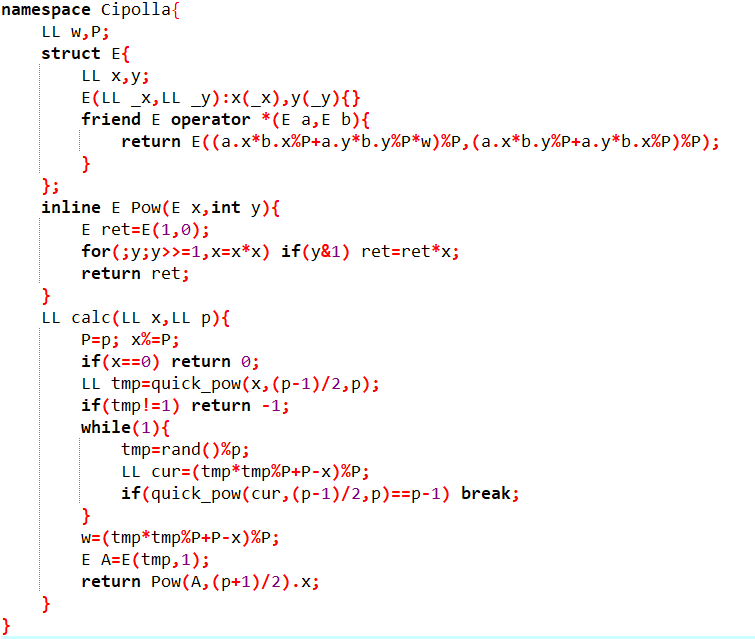
inline ll Calc(ll x) { return Sump(x, cnt - 1); }//计算x以内

BSGS：（待补）

扩展BSGS：（待补）

二次剩余：

计算n在模p意义下的开根号；



CRT：待补

拉格朗日插值：

重心插值；

while (n--) {

int tp, X;

scanf("%d", &tp);

if (tp == 1) {

++cnt;

scanf("%lld%lld", &x[cnt], &y[cnt]);

For(i, 1, cnt - 1) w[i] = w[i] \* qpow(x[i] - x[cnt], mod - 2) % mod;

w[cnt] = y[cnt];

For(i, 1, cnt - 1) w[cnt] = w[cnt] \* qpow(x[cnt] - x[i], mod - 2) % mod;

} else {

scanf("%d", &X);

bool flag = 0;

For(i, 1, cnt) {

if (X == x[i]) {

flag = 1;

printf("%lld\n", y[i]);

break;

}

}

if (flag)

continue;

LL ans = 1;

For(i, 1, cnt) ans = ans \* (X - x[i]) % mod;

LL sum = 0;

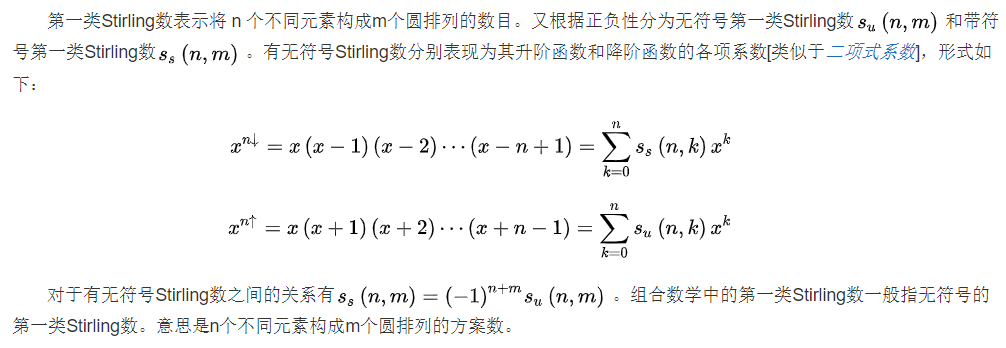
For(i, 1, cnt) sum = (sum + w[i] \* qpow(X - x[i], mod - 2) % mod) % mod;

printf("%lld\n", (ans \* sum % mod + mod) % mod);

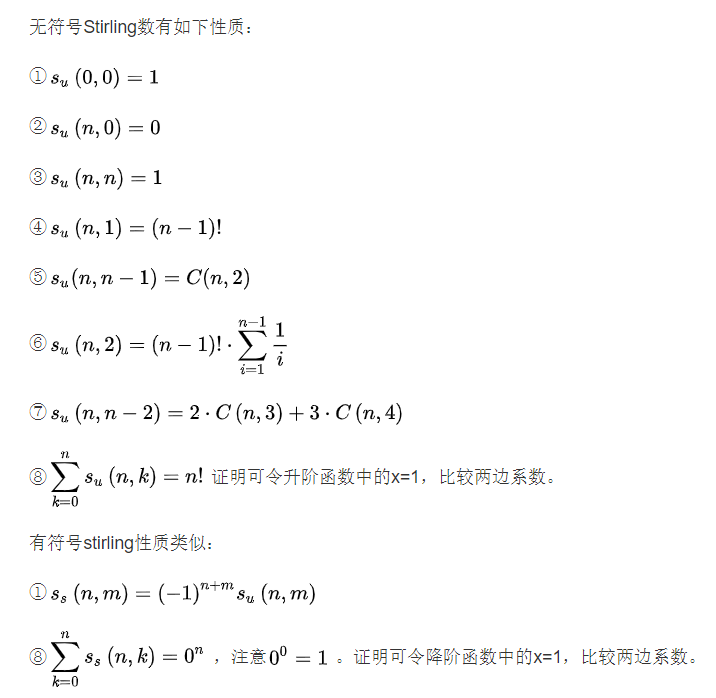
}

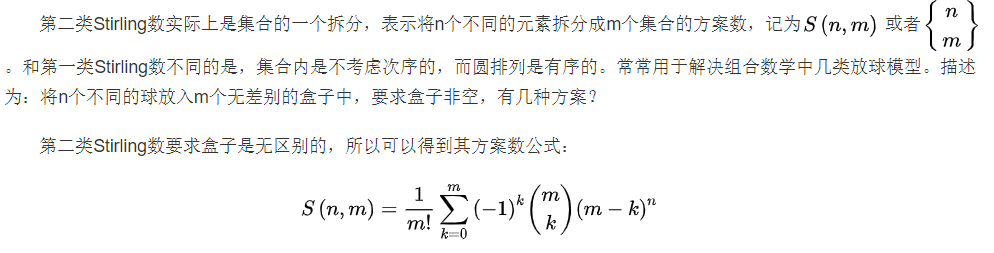
}

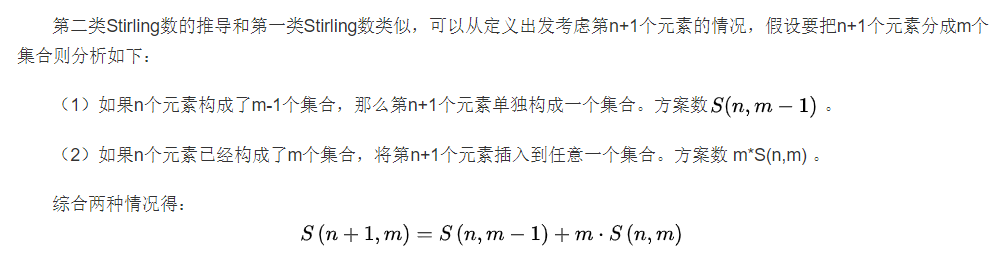
斯特林数：



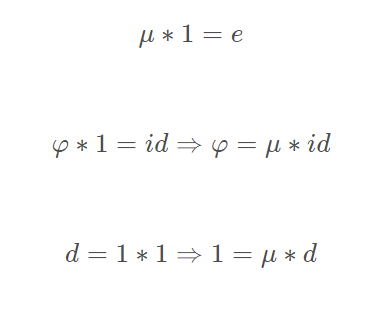


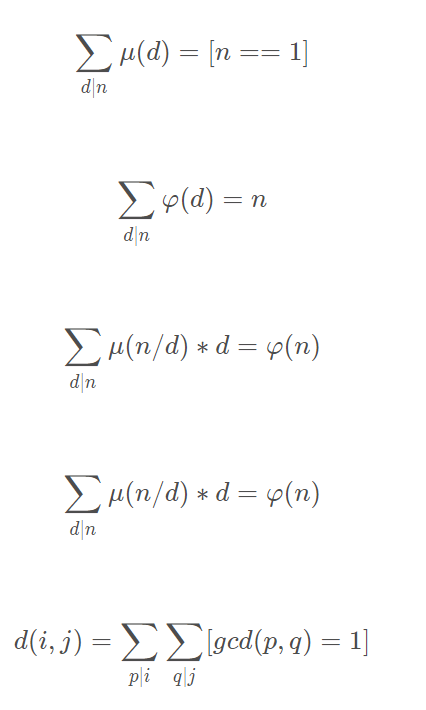


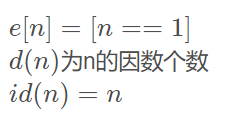




杜教筛：







求1-n的欧拉函数和：O(n^(2/3))

（欧拉函数，phi(n)与n互质的数；莫比乌斯函数：质因数的指数为1的质因数的个数为k，(-1)^k）

LL n;

LL phi[maxn+10];

int prime[maxn+10],tot;

bool np[maxn+10];

map<LL,LL>M;

inline void init(){//预处理phi

phi[1]=1;

For(i,2,maxn){

if(!np[i]) prime[++tot]=i,phi[i]=i-1;

for(int j=1;j<=tot && i\*prime[j]<=maxn;++j){

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

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

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

break;

}

phi[i\*prime[j]]=phi[i]\*(prime[j]-1);

}

}

For(i,1,maxn) phi[i]=(phi[i]+phi[i-1])%mod;

}

inline void init(){//预处理mu

mu[1]=1;np[1]=1;

For(i,2,maxn){

if(!np[i]) prime[++tot]=i,mu[i]=-1;

for(int j=1;j<=tot && i\*prime[j]<=maxn;++j){

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

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

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

break;

}

mu[i\*prime[j]]=-mu[i];

}

}

For(i,1,maxn-1) mu[i]+=mu[i-1];

}

LL calc(LL now){

if(now<=maxn) return phi[now];

if(M.count(now)) return M[now];

LL res=now%mod\*(now%mod+1)/2%mod,tmp=0;

for(LL l=2,r;l<=now;l=r+1)

r=now/(now/l),tmp=(tmp+(r-l+1)%mod\*calc(now/l)%mod)%mod;

res=((res-tmp)%mod+mod)%mod;

M[now]=res;

return res;

}

int main(){

init();

scanf("%lld",&n);

printf("%lld\n",(calc(n)%mod+mod)%mod);

return 0;

}

欧几里得算法：

Gcd(x, y) = gcd(y, x % y)

X = a \* y + b;

考虑提取公因式的角度，a \* y与y的因子相同，需要考虑b包含的y的因子，即等价于gcd(y, b)

扩展欧几里得算法：

求a \* x + b \* y = c的通解

贝祖定理：给予二整数 a 与 b, 必存在有整数 x 与 y 使得ax + by = gcd(a,b)。

可以将a’ , b’, c同时除以k = c / gcd(a, b)，化简到ax + by = gcd(a, b)；根据贝祖定理，c一定是gcd(a, b)的整数倍，否则无解。由辗转相除法递归的最后状态：a \* 1 + b \* 0 = gcd(a, b)；可以回溯算出来x, y；

在求a和b的最大公约数回溯时，我们已经求出了下一个状态：b和a%b的最大公因数，并且求出了一组x1和y1使得b \* x1 + (a % b) \* y1 = gcd；将a % b = a - (a / b) \* b带入：

上式=b \* x1 + (a - (a / b) \* b) \* y1 = b \* x1 + a \* y1 – (a / b) \* b \* y1 = a \* y1 + b \* (x1 – a / b \* y1)  =  gcd   ；

发现 x = y1 , y = x1 – a/b\*y1（一组特解）

int Exgcd(int a, int b, int &x, int &y){//x, y为一组特解

if(b==0) {

x = 1,y = 0 ;

return a ;

}

else {

int r = Exgcd(b, a % b, x, y); /\* r = GCD(a, b) = GCD(b, a%b) \*/

int t = x ; x = y; y = t - a / b \* y ;

return r ;

}

}

不定方程的特解：将特解都乘以k，即可得到不定方程的特解

不定方程的通解：a \* (x – n \* b) + b \* (y + n \* a) = c ，n为整数

通过这样可以得到方程通解：x = x0 – n \* b， y = y0 + n \* a

超级幂

a^b ≡ a ^ (b % phi(p)) (mod p)

一些数学结论：

**剩余系**

所谓“剩余系”，就是指对于某一个特定的正整数n，一个整数集中的数模n所得的余数域；

如果一个剩余系中包含了这个正整数所有可能的余数（一般地，对于任意正整数n，有n个余数：0,1,2,...,n-1），那么就被称为是模n的一个**完全剩余系；**

在与模n互素的全体剩余类中，从每一个类中各任取一个数作为代表组成的集合，叫做**模n的一个简化剩余系**，大小为φ(n)

简化剩余系：元素的乘法在模n下封闭

结论1：1~n中与n互素的数乘起来模n与1同余（当n为4,p^t,2\*p^t除外，这些为-1）

结论2：当一个序列存在严格众数，那么不同的两个数两两相消，剩下的就是众数。

# 6.数据结构：

主席树：

区间第k大，历史信息

void update(int &node,int pre,int x,int l,int r){

node=++tot;

t[node]=t[pre];++t[node].sum;

if(l==r) return ;

if(x<=mid) update(t[node].lr,t[pre].lr,x,l,mid);

else{

update(t[node].rr,t[pre].rr,x,mid+1,r);

}

}

int query(int u,int v,int l,int r,int k){

if(l==r) return l;

int res=t[t[v].lr].sum-t[t[u].lr].sum;

if(k<=res) return query(t[u].lr,t[v].lr,l,mid,k);

else{

return query(t[u].rr,t[v].rr,mid+1,r,k-res);

}

}

动态开点：

int n, m;

int rt[maxn], tot;

struct node{

int lr, rr;

LL sum;

}tr[maxn << 5];

void update(int &now, int pre, int l, int r, int val){

now = ++tot;

tr[now] = tr[pre];

tr[now].sum += val;

if(l == r) return ;

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

if(val <= mid) update(tr[now].lr, tr[pre].lr, l, mid, val);

else update(tr[now].rr, tr[pre].rr, mid + 1, r, val);

}

LL query (int L, int R, int l, int r, LL x){

if(l == r) return tr[R].sum - tr[L].sum;

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

if(x <= mid) return query(tr[L].lr, tr[R].lr, l, mid, x);

else return tr[tr[R].lr].sum - tr[tr[L].lr].sum +

query(tr[L].rr, tr[R].rr, mid + 1, r, x);

}

线段树优化建图：

#define mid ((l+r)>>1)

int build(int l,int r,int tp){

if(l==r) return l;

int node=++tot;

t[tp][node].lr=build(l,mid,tp);

t[tp][node].rr=build(mid+1,r,tp);

int lr=t[tp][node].lr,rr=t[tp][node].rr;

if(tp==1){

G[node].push\_back(mk(lr,0));

G[node].push\_back(mk(rr,0));

// cout<<node<<" "<<lr<<" 0"<<endl;

// cout<<node<<" "<<rr<<" 0"<<endl;

}

else{

G[lr].push\_back(mk(node,0));

G[rr].push\_back(mk(node,0));

// cout<<lr<<" "<<node<<" 0"<<endl;

// cout<<rr<<" "<<node<<" 0"<<endl;

}

return node;

}

void link(int node,int l,int r,int L,int R,int x,int w,int tp){

if(l==L && r==R){

if(tp==1){

G[x].push\_back(mk(node,w));

//cout<<x<<" "<<node<<" "<<w<<endl;

}

else{

G[node].push\_back(mk(x,w));

//cout<<node<<" "<<x<<" "<<w<<endl;

}

return ;

}

int lr=t[tp][node].lr,rr=t[tp][node].rr;

//cout<<l<<" "<<r<<" "<<node<<" "<<lr<<" "<<rr<<endl;

if(R<=mid) link(lr,l,mid,L,R,x,w,tp);

else if(L>mid) link(rr,mid+1,r,L,R,x,w,tp);

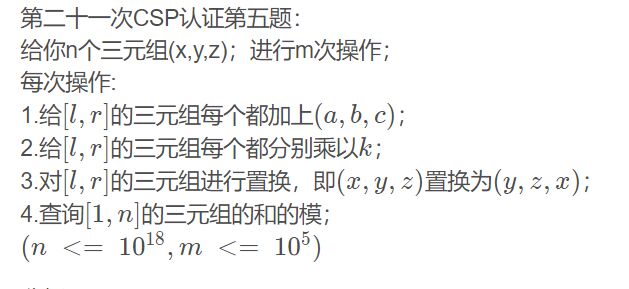
else link(lr,l,mid,L,mid,x,w,tp),link(rr,mid+1,r,mid+1,R,x,w,tp);

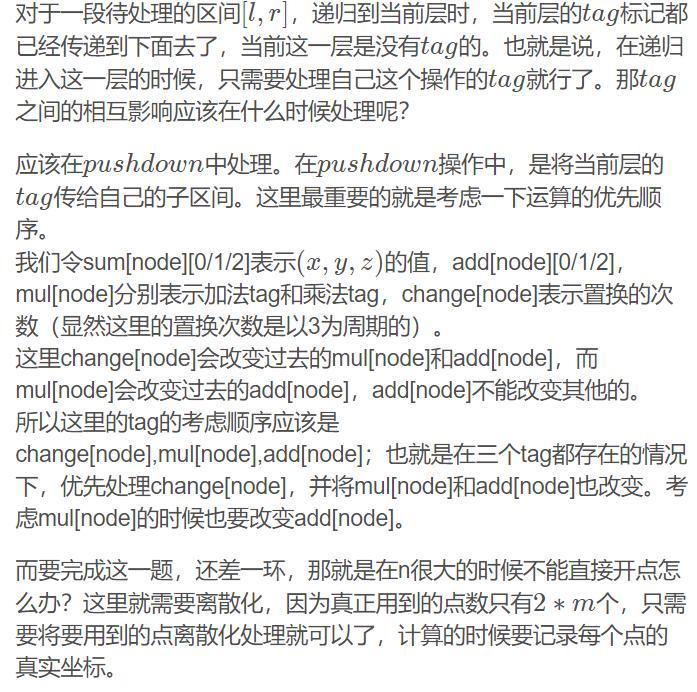
}

#undef mid

线段树：

写线段树的lazy\_tag，需要着重考虑tag之间的影响，这里的影响主要是在push-down的时候带来的影响。因为传递下去后，当前层就已经变成了新的了，不会受到影响。





#define lr (node<<1)

#define rr (node<<1|1)

#define mid ((l+r)>>1)

struct segment\_tree{

int tree[maxn<<2],lazy[maxn<<2];

void init(){

memset(tree,0,sizeof tree);

memset(lazy,0,sizeof lazy);

}

inline void push\_down(int node){

lazy[lr]+=lazy[node],lazy[rr]+=lazy[node];

tree[lr]+=lazy[node],tree[rr]+=lazy[node];

lazy[node]=0;

}

inline void push\_up(int node){

tree[node]=max(tree[lr],tree[rr]);

}

void update(int node,int l,int r,int L,int R,int w){//区间修改

if(L<=l && r<=R){

lazy[node]+=w;

tree[node]+=w;

return ;

}

push\_down(node);

if(L<=mid) update(lr,l,mid,L,R,w);

if(R>mid) update(rr,mid+1,r,L,R,w);

push\_up(node);

}

int query(int node,int l,int r,int L,int R){//区间查询

if(L<=l && r<=R) return tree[node];

push\_down(node);

int ans=0;

if(L<=mid) ans=max(ans,query(lr,l,mid,L,R));

if(R>mid) ans=max(ans,query(rr,mid+1,r,L,R));

push\_up(node);

return ans;

}

}T;

#undef lr

#undef rr

#undef mid

对于swap相关的做法，维护矩阵是一个不错的选择

# 7.不方便归类的算法：

**01分数规划：**

求sum(a[i]\*x[i])/sum(b[i]\*x[i])的最值；设结果为y,转为sum{(a[i]-y\*b[i])\*x[i]}，其中x[i]为0/1，二分y就可以求得答案；

**差分约束：**

已知n个变量，m个不等式，;这个系统可以变为差分约束系统，通过变为，有点类似最短路中的，只要满足最短路就一定满足这个条件，可以将，跑最短路。

高斯消元：

高斯消元满足的形式是： ;

Ans[i]是最后方程的解；

复杂度O(n^3)

void Guass(){

int col,line;

for(col=1,line=1;col<=n && line<=n;++col,++line){

while(fabs(matrix[line][col])<eps && col<=n) ++col;

if(col>n) break;

For(i,line+1,n){

if(fabs(matrix[i][col])<eps) continue;

double res=matrix[i][col]/matrix[line][col];

For(k,1,n+1) matrix[i][k]-=res\*matrix[line][k];

}

}

Forr(i,n,1){

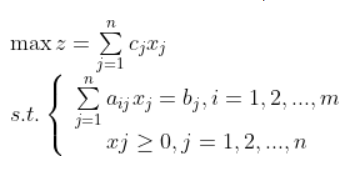
For(j,i+1,n) matrix[i][n+1]-=matrix[i][j]\*ans[j];

ans[i]=matrix[i][n+1]/matrix[i][i];

}

}

线性规划：





求最小值就直接对a[0][i]取负即可

inline void pivot(int line,int col){

swap(id[col],id[n+line]);

double res=a[line][col];

a[line][col]=1;

For(i,0,n) a[line][i]/=res;

For(i,0,m){

if(i!=line){

res=a[i][col],a[i][col]=0;

For(j,0,n) a[i][j]-=res\*a[line][j];

}

}

}

inline void init\_simplex(){

int line,col;double res;

while(1){

line=col=0;res=-eps;

For(i,1,m)

if(a[i][0]<res)

line=i,res=a[i][0];

if(!line) break;

For(i,1,n)

if(a[line][i]<-eps) col=i;

if(!col){

puts("Infeasible");

return ;

}

pivot(line,col);

}

}

inline void simplex(){

int line,col;double res;

while(1){

line=col=0;

For(i,1,n)

if(a[0][i]>eps){

col=i;

break;

}

if(!col) break;

res=inf;

For(i,1,m)

if(a[i][col]>eps && a[i][0]/a[i][col]<res)

res=a[i][0]/a[i][col],line=i;

if(!line){

puts("Unbounded");

return ;

}

pivot(line,col);

}

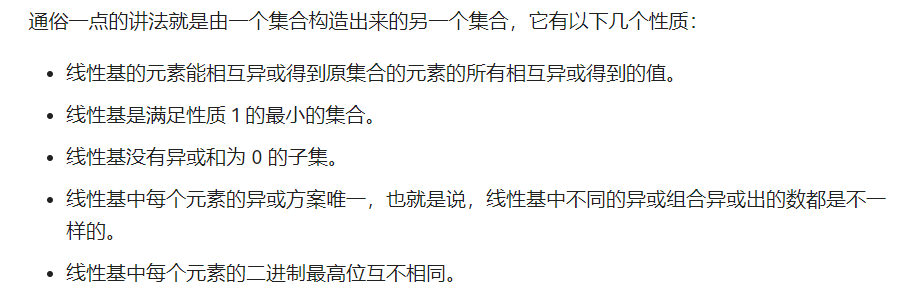
}

a[0][i]//c[i]

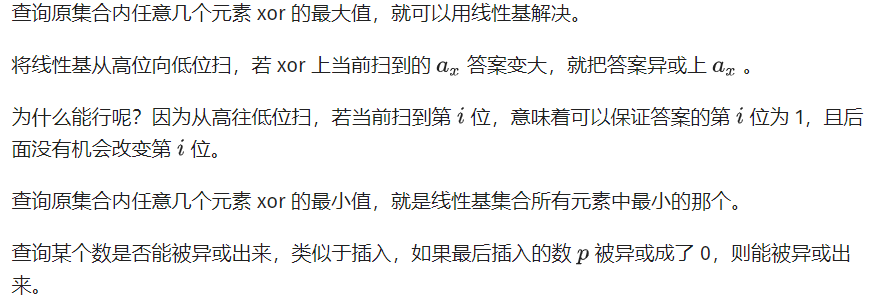
a[i][0]//b[i]

ans=-a[0][0];

线性基：







#include<bits/stdc++.h>

#define ll long long

#define MAXN 10005

using namespace std;

int t,n,q;

ll k,tmp;

struct L\_B{

ll b[65],p[65];

int cnt,flag;

L\_B(){

memset(p,0,sizeof(p));

memset(b,0,sizeof(b));

cnt = flag = 0;

}

inline bool insert(ll x){

for(int i = 62;i >= 0;--i)

if(x & (1ll <<i)){

if(b[i]) x ^= b[i];

else{

b[i] = x;

return true;

}

}

flag = 1;//有没有0

return false;

}

ll get\_max(){

ll ret = 0;

for(int i = 62;i >= 0;--i)

if((ret^b[i]) > ret)

ret ^= b[i];

return ret;

}

ll get\_min(){

if(flag)

return 0;

for(int i = 0;i <= 62;++i)

if(b[i])

return b[i];

return 0;

}

inline void rebuild(){

for(int i = 62;i >= 1;--i)

if(b[i]){

for(int j = i-1;j >= 0;--j)

if(b[i] & (1ll << j))

b[i] ^= b[j];

}

for(int i = 0;i <= 62;++i) if(b[i]) p[cnt++] = b[i];

}

ll kth(ll k){

if(flag)//有没有0

--k;

if(k == 0)

return 0;

ll ret = 0;

if(k >= (1ll << cnt))

return -1;

for(int i = 0;i <= cnt-1;++i) if(k & (1ll << i)) ret ^= p[i];

return ret;

}

};

L\_B merge(const L\_B &n1,const L\_B &n2){

L\_B ret = n1;

for(int i = 0;i <= 62;++i) if(n2.b[i]) ret.insert(n2.b[i]);

ret.flag = n1.flag | n1.flag;

return ret;

}

int main(){

scanf("%d",&t);

for(int cas = 1;cas <= t;++cas){

L\_B lis;

scanf("%d",&n);

for(int i = 1;i <= n;++i){

scanf("%lld",&tmp);

lis.insert(tmp);

}

scanf("%d",&q);

lis.rebuild();

printf("Case #%d:\n",cas);

while(q--){

scanf("%lld",&k);

printf("%lld\n",lis.kth(k));

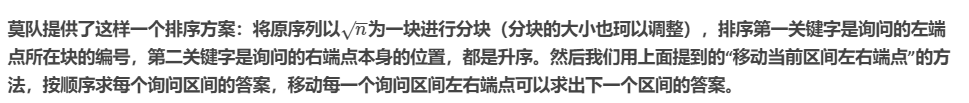
}

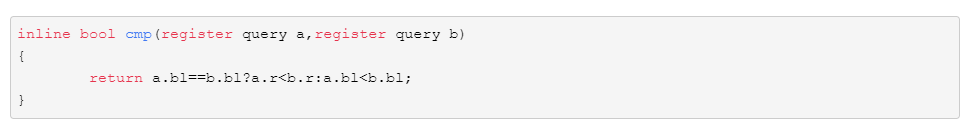
}

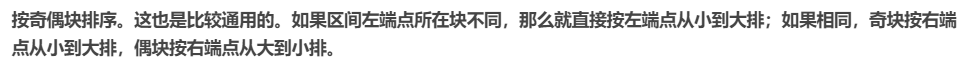
}

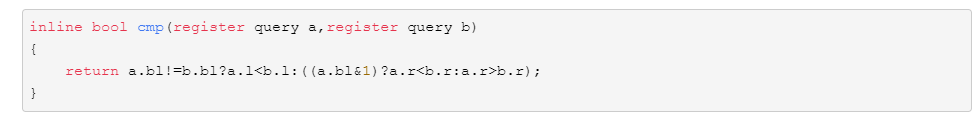
莫队：

对询问进行排序，优雅的使用暴力求解；这里要求每次移动的时候，处理的复杂度为O(1);

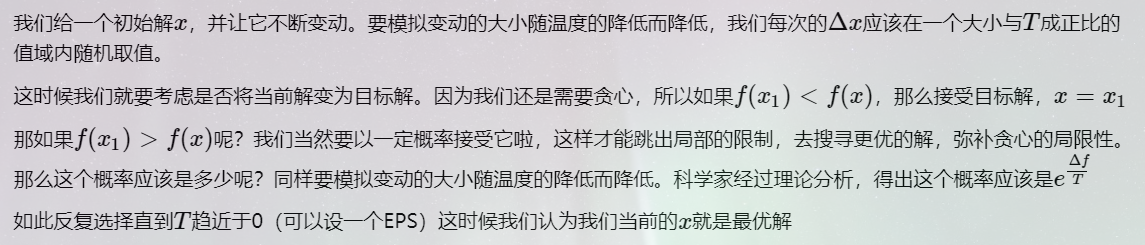








模拟退火：



#include<bits/stdc++.h>

#define For(aa,bb,cc) for(int aa=(bb);aa<=(int)(cc);++aa)

#define Rand T\*(rand()\*2-RAND\_MAX)//[-T\*RM,T\*RM)

using namespace std;

typedef long double LD;

const int maxn=110;

const double eps=1e-14,D=0.97;

int n;

double best,ans;//best ans,now ans

struct Point{

double x,y,z;

Point(){ x=y=z=0; }

}p[maxn];

LD calc(LD xx,LD yy,LD zz){//Min max\_dis(p0,p[i])

LD res=0;

For(i,1,n)

res=max(res,sqrt((xx-p[i].x)\*(xx-p[i].x)+

(yy-p[i].y)\*(yy-p[i].y)+(zz-p[i].z)\*(zz-p[i].z)));

return res;

}

int main(){

#ifndef ONLINE\_JUDGE

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

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

#endif

scanf("%d",&n);

Point ap,p0,p1;

For(i,1,n){

scanf("%lf%lf%lf",&p[i].x,&p[i].y,&p[i].z);

ap.x+=p[i].x,ap.y+=p[i].y,ap.z+=p[i].z;

}

ap.x/=n,ap.y/=n,ap.z/=n;

best=ans=calc(ap.x,ap.y,ap.z);

srand(time(NULL));

int tim=100;//模拟次数

while(tim--){

ans=best,p0=ap;

for(LD T=1e5;T>eps;T\*=D){

p1.x=p0.x+Rand;

p1.y=p0.y+Rand;

p1.z=p0.z+Rand;

LD res=calc(p1.x,p1.y,p1.z);

if(best>res) best=res,ap=p1;

if(ans>res || exp((ans-res)/T)>(LD)rand()/RAND\_MAX) ans=res,p0=p1;

}

}

printf("%.8lf\n",best);

cerr<<1.0\*clock()/CLOCKS\_PER\_SEC<<endl;

return 0;

}

二分：

bool check(int x,int mid){//upper\_bound

return x<a[mid];

}

bool check(int x,int mid){//lower\_bound

return x<a[mid];

}

int solve(int x){

int l=1,r=n;

while(l<r){

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

if(check(x,mid)) r=mid;

else l=mid+1;

}

return l;

}

Cdq分治、整体二分

二进制分组

# 8.辅助功能：

对拍程序：

Check.bat

:loop

gen.exe

j.exe

j2.exe

fc out.txt ans.txt

if %errorlevel%==0 goto loop

pause

读入优化：

**char** BUF\_R[1 << 22], \*csy1, \*csy2;

**#define** GC (csy1 == csy2 && (csy2 = (csy1 = BUF\_R) + fread(BUF\_R, 1, 1 << 22, stdin), csy1 == csy2) ? EOF : \*csy1 ++)

**template** <**typename** **Ty**> **inline** **void** RI(**Ty** &t) {

**char** c = GC;

**for** (t = 0; c < 48 || c > 57; c = GC);

**for** (; c > 47 && c < 58; c = GC) t = t \* 10 + (c ^ 48);

}

template<class T>void read(T &x){

x=0;char c=getchar();

while(!isdigit(c)) c=getchar();

while(isdigit(c)) x=(x<<1)+(x<<3)+(c^48),c=getchar();

}

std::ios::sync\_with\_stdio(false);

std::cin.tie(nullptr);

优化cin（开O2）

随机数

mt19937 rnd(chrono::steady\_clock::now().time\_since\_epoch().count());

或者

mt19937 rnd(chrono::system\_clock::now().time\_since\_epoch().count());

数组内移动：shuffle(a+1, a+n+1, rnd);

随机范围[l,r]：uniform\_int\_distribution<int>(l,r)(rnd)

random\_device seed;

default\_random\_engine rd(seed());

uniform\_int\_distribution<> di(1, 1000);

di(rd)