ACM模板

**数据结构：**

**单调队列：**

push:平均O(1),查询:O（1）

struct Node

{

int val,t; //val是值，t是加入的时间

Node(int val\_,int t\_):val(val\_),t(t\_){}

Node(){}

};

Node maxq[maxn];

int maxf,maxr;

void push(int n,int t)

{

if(maxq[maxf].t<t-k+1) //如果头超出范围，则pop

maxf++;

for(int i=maxr-1;i>=maxf;--i)//把队列中小于要push进去的值的点都pop掉

if(maxq[i].val<n)

maxr--;

else break;

maxq[maxr++]=Node(n,t);

}

**线段树：**

struct Node

{

int l,r; //表示的线段范围

int color; //存的值，这里为二进制表示的存在color数（状态压缩）

bool lazy; //lazy标记

};

Node tree[4\*maxn];

void buildtree(int l,int r,int n)

{

tree[n].r=r;

tree[n].l=l;

tree[n].color=1;

tree[n].lazy=0;

if(r==l)

{

return;

}

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

buildtree(l,m,n<<1);

buildtree(m+1,r,n<<1|1);

}

void pushDown(int n)

{

if(tree[n].lazy)

{

tree[n<<1].color=tree[n<<1|1].color=tree[n].color;

tree[n].lazy=0;

tree[n<<1].lazy=tree[n<<1|1].lazy=1;

}

}

void pushUp(int n)

{

tree[n].color=tree[n<<1].color|tree[n<<1|1].color;

}

void change(int l,int r,int col,int n)

{

if(tree[n].l==l && tree[n].r==r)

{

tree[n].color=col;

if(l!=r)

tree[n].lazy=1;

return;

}

if(tree[n].color==col) return;

pushDown(n);

int m=(tree[n].l+tree[n].r)>>1;

if(r<=m) change(l,r,col,n<<1);

else if(l<=m && r>m)

{

change(l,m,col,n<<1);

change(m+1,r,col,n<<1|1);

}

else change(l,r,col,n<<1|1);

pushUp(n);

}

inline int toBin(const int n) //把颜色转换为二进制表示

{

return 1<<(n-1);

}

int count(int n) //计算一个color里有多少种颜色

{

int res=0;

while(n)

{

if(n&1) res++;

n>>=1;

}

return res;

}

**堆排序：**

void maxsort(float a[],int count);

void maxbuild(float a[],int count);

void heapify(float a[],int start,int count);

int main()

{

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

float a[100];

float \*p=a+1;

int count=0;

printf("please input the data,end up by 0\n");

scanf("%f",p);

while(\*p++)

{

scanf("%f",p);

count++;

}

maxbuild(a,count);

maxsort(a,count);

p=a+1;

for(;count>0;count--)

printf("%f\t",\*p++);

return 0;

}

void heapify(float a[],int start,int count)

{

int m=start;

int k,max;

float t;

while(1)

{

k=m\*2;

if(k>count)

break;

else if(k+1>count)

{

if(a[k]>a[m])

{

t=a[k];

a[k]=a[m];

a[m]=t;

}

break;

}

else

{

if(a[k]>a[k+1])

max=k;

else max=k+1;

if(a[max]>a[m])

{

t=a[max];

a[max]=a[m];

a[m]=t;

}

}

m=max;

}

}

void maxbuild(float a[],int count)

{

int i;

for(i=count/2;i>0;i--)

heapify(a,i,count);

}

void maxsort(float a[],int count)

{

float t;

while(count>1)

{

t=a[count];

a[count]=a[1];

a[1]=t;

heapify(a,1,--count);

}

}

**并查集：**

//两个基本功能：并/查~

int father[maxn];

int num[maxn];

int n,m;

void init()

{

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

num[i]=1;

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

father[i]=i;

}

int find(int a)

{

while(father[a]!=a)

{

a=father[a];

}

return a;

}

void Union(int a,int b)

{

a=find(a);

b=find(b);

if(a==b)

return;

father[a]=b;

num[b]+=num[a];

return;

}

**数论：**

**GCD/exGCD:**

int gcd(int a,int b)

{

int t;

for(;b;t=b,b=a%b,a=t);

return a;

}

int exgcd(int a,int b,int &x,int &y)

{

int res;

if(b==0)

{

x=1;

y=0;

res=a;

}

else

{

res=exgcd(b,a%b,x,y);

int t=x;

x=y;

y=t-a/b\*y;

}

return res;

}

**大整数类：**

（加减乘除开方）

void rev(int\* m,int l)

{

int t;

for(int i=0; i<l/2; ++i)

{

t=m[i];

m[i]=m[l-i-1];

m[l-i-1]=t;

}

}

struct LNI

{

int num[maxn];

int n;

int f;

int read()

{

memset(num,0,sizeof(num));

char \*tem;

tem=new char[maxn];

if(scanf("%s",tem)==EOF) return EOF;

change(tem);

rev(num,n);

//反过来存，加减乘法会快，除法会略慢，若只有除法，可以正着存

delete [] tem;

return 0;

}

void change(char \*tem)

{

f=1;

if(\*tem=='-')

{

f=-1;

tem++;

}

while(\*tem=='0')

tem++;

int l=strlen(tem);

int p=l%4;

int k=-1;

if(p)

{

k++;

num[k]=tem[0]-'0';

for(int i=1; i<p; ++i)

{

num[k]=num[k]\*10+tem[i]-'0';

}

}

for(int i=p; i<l; ++i)

{

if(i%4==p)

{

k++;

num[k]=tem[i]-'0';

}

else

{

num[k]=num[k]\*10+tem[i]-'0';

}

}

n=(l-1)/4+1;

}

void print()

{

rev(num,n);

int i=0;

if(f<0) putchar('-');

printf("%d",num[i++]);

for(; i<n; ++i)

printf("%04d",num[i]);

printf("\n");

}

};

LNI operator-(LNI a,LNI b);

LNI operator/(LNI a,int n);

LNI operator+(LNI a,LNI b)

{

LNI res;

if(a.f\*b.f==1)

{

res.f=a.f;

}

else

{

if(a.f<0)

{

a.f=1;

return b-a;

}

else

{

b.f=1;

return a-b;

}

}

memset(res.num,0,sizeof(res.num));

int jw=0;

res.n=max(a.n,b.n);

for(int i=0; i<res.n; ++i)

{

if(i<a.n && i<b.n)

{

res.num[i]=a.num[i]+b.num[i]+jw;

jw=res.num[i]/10000;

res.num[i]%=10000;

}

else if(i>=a.n)

{

res.num[i]=b.num[i]+jw;

jw=res.num[i]/10000;

res.num[i]%=10000;

}

else if(i>=b.n)

{

res.num[i]=a.num[i]+jw;

jw=res.num[i]/10000;

res.num[i]%=10000;

}

}

if(jw)

{

if(res.num[res.n-1]/1000)

{

res.num[res.n]=jw;

res.n++;

}

else

{

int k=1;

for(int i=0; i<k%4; ++i)

k\*=10;

res.num[k-1]+=jw\*k;

}

}

return res;

}

bool operator<(const LNI &a,const LNI &b)

{

if(a.f!=b.f) return a.f<b.f;

if(a.n!=b.n) return a.f>0?(a.n<b.n):(a.n>b.n);

for(int i=0; i<a.n; ++i)

if(a.num[i]!=b.num[i]) return a.f>0?(a.num[i]<b.num[i]):(a.num[i]>b.num[i]);

return 0;

}

LNI operator-(LNI a,LNI b)

{

LNI res;

if(a.f\*b.f==-1)

{

b.f\*=-1;

return a+b;

}

res.f=a.f;

if((a<b&&a.f>0)||(b<a&&a.f<0))

{

LNI t=a;

a=b;

b=t;

res.f=-1\*a.f;

}

memset(res.num,0,sizeof(res.num));

int jw=0;

res.n=max(a.n,b.n);

for(int i=0; i<res.n; ++i)

{

if(i<a.n && i<b.n)

{

res.num[i]=a.num[i]-b.num[i]+jw;

jw=0;

if(res.num[i]<0)

{

jw=-1;

res.num[i]+=10000;

}

}

else if(i>=b.n)

{

res.num[i]=a.num[i]+jw;

jw=0;

if(res.num[i]<0)

{

jw=-1;

res.num[i]+=10000;

}

}

}

while(res.num[res.n-1]==0)

res.n--;

return res;

}

LNI operator\*(LNI a,LNI b)

{

LNI res;

res.f=a.f\*b.f;

memset(res.num,0,sizeof(res.num));

int jw,k;

for(int i=0; i<a.n; ++i)

{

for(int j=0; j<b.n; ++j)

{

res.num[i+j]+=a.num[i]\*b.num[j];

jw=res.num[i+j]/10000;

res.num[i+j]%=10000;

k=1;

while(jw)

{

res.num[i+j+k]+=jw;

jw=res.num[i+j+k]/10000;

res.num[i+j+k]%=10000;

k++;

}

}

}

int n=a.n+b.n-1;

while(res.num[n++]);

res.n=n-1;

return res;

}

LNI operator^(LNI a,int n)

{

LNI res;

res.n=1;

res.num[0]=1;

while(n)

{

if(n&1)

res=res\*a;

a=a\*a;

n>>=1;

}

return res;

}

LNI sqr(const LNI &a) //使用二分法找平方根

{

LNI res=a/2;

LNI dif;

LNI one;

one.f=1;

one.num[0]=1;

one.n=1;

LNI Max=a,Min=one;

while(1)

{

dif=res\*res-a;

if(dif.n==1&&dif.num[0]==0)

break;

if(dif.f<0)

{

Min=res;

dif.f=1;

}

else

Max=res;

if(!(one<(Max-Min)))

{

LNI dif1=Max\*Max-a;

LNI dif2=Min\*Min-a;

dif1.f=1;

dif2.f=1;

if(dif1<dif2)

return Max;

else return Min;

}

res=(Max+Min)/2;

}

return res;

}

LNI operator/(LNI a,int n)

{

LNI res=a;

rev(res.num,res.n);

for(int i=0;i<res.n;++i)

{

res.num[i+1]+=res.num[i]%n\*10000;

res.num[i]/=n;

}

if(res.num[0]==0)

{

for(int i=0;i<res.n-1;++i)

res.num[i]=res.num[i+1];

res.n--;

}

rev(res.num,res.n);

return res;

}

**带小数的大数乘法：**

void rev(int\* m,int l)

{

int t;

for(int i=0; i<l/2; ++i)

{

t=m[i];

m[i]=m[l-i-1];

m[l-i-1]=t;

}

}

struct LargeNum

{

int num[maxn];

int n;

int xp; //小数点在倒数第几位

int read()

{

memset(num,0,sizeof(num));

char \*tem,\*tmm,tem2[maxn];

tem=new char[maxn];

tmm=tem;

tem2[0]=0;

// if(scanf("%s.%s",tem,tem2)==EOF) return EOF;

char ch,\*z=tem;

while(1)

{

if(scanf("%c",&ch)==EOF) return -1;

if(ch=='\n' || ch==' ') break;

if(ch!='.')

\*z++=ch;

else

{

\*z=0;

z=tem2;

}

}

\*z=0;

xp=strlen(tem2);

strcat(tem,tem2);

while(\*tem=='0')

tem++;

int l=strlen(tem);

int p=l%4;

int k=-1;

if(p)

{

k++;

num[k]=tem[0]-'0';

for(int i=1; i<p; ++i)

{

num[k]=num[k]\*10+tem[i]-'0';

}

}

for(int i=p; i<l; ++i)

{

if(i%4==p)

{

k++;

num[k]=tem[i]-'0';

}

else

{

num[k]=num[k]\*10+tem[i]-'0';

}

}

delete [] tmm;

n=(l-1)/4+1;

rem0();

return 0;

}

void print()

{

int i=0,k;

if(n==1 && num[0]==0)

{

printf("0\n");

return;

}

if(xp==0)

{

printf("%d",num[i++]);

for(; i<n; ++i)

printf("%04d",num[i]);

}

else

{

if((n-xp/4-1)>0)

{

printf("%d",num[i++]);

for(; i<n-xp/4-1; ++i)

printf("%04d",num[i]);

k=1;

for(int j=0; j<xp%4; ++j)

k\*=10;

char pstr[10]="%0 d";

pstr[2]=4-xp%4+'0';

printf(pstr,num[i]/k);

if(i<n-1)

{

char pstr[10]=".%0 d";

if(xp%4)

{

pstr[3]=xp%4+'0';

printf(pstr,num[i]%k);

}

else putchar('.');

i++;

k=1000;

}

else

{

k/=10;

if(i==n-1)

{

if(num[i]%(k\*10)==0)

{

putchar('\n');

return;

}

putchar('.');

}

}

for(; i<n-1; ++i)

printf("%04d",num[i]);

while(k)

{

if(num[i]%k)

{

printf("%d",(num[i]/k)%10);

k/=10;

}

else

{

printf("%d",(num[i]/k)%10);

break;

}

}

}

else if((n-xp/4-1)==0)

{

k=1;

for(int j=0; j<xp%4; ++j)

k\*=10;

if(num[i]/k!=0)

printf("%d",num[i]/k);

if(i<n-1)

{

char pstr[10]=".%0 d";

if(xp%4)

{

pstr[3]=xp%4+'0';

printf(pstr,num[i]%k);

}

else putchar('.');

i++;

k=1000;

}

else

{

k/=10;

if(i==n-1)

{

if(num[i]%(k\*10)==0)

{

putchar('\n');

return;

}

putchar('.');

}

}

for(; i<n-1; ++i)

printf("%04d",num[i]);

while(k)

{

if(num[i]%k)

{

printf("%d",(num[i]/k)%10);

k/=10;

}

else

{

printf("%d",(num[i]/k)%10);

break;

}

}

}

else

{

putchar('.');

for(int j=0; j<xp-4\*n; ++j)

putchar('0');

for(; i<n-1; ++i)

printf("%04d",num[i]);

k=1000;

while(k)

{

if(num[i]%k)

{

printf("%d",(num[i]/k)%10);

k/=10;

}

else

{

printf("%d",(num[i]/k)%10);

break;

}

}

}

}

putchar('\n');

}

void rem0()

{

while(xp>4)

{

if(num[n-1]==0)

{

n--;

xp-=4;

}

else break;

}

}

};

LargeNum operator\*(LargeNum a,LargeNum b)

{

LargeNum res;

res.xp=a.xp+b.xp;

rev(a.num,a.n);

rev(b.num,b.n);

memset(res.num,0,sizeof(res.num));

int jw,k;

for(int i=0; i<a.n; ++i)

{

for(int j=0; j<b.n; ++j)

{

res.num[i+j]+=a.num[i]\*b.num[j];

jw=res.num[i+j]/10000;

res.num[i+j]%=10000;

k=1;

while(jw)

{

res.num[i+j+k]+=jw;

jw=res.num[i+j+k]/10000;

res.num[i+j+k]%=10000;

k++;

}

}

}

int n=a.n+b.n-1;;

while(res.num[n++]);

res.n=n-1;

rev(res.num,res.n);

res.rem0();

return res;

}

LargeNum operator^(LargeNum a,int n)

{

LargeNum res;

res.xp=0;

res.n=1;

res.num[0]=1;

while(n)

{

if(n&1)

res=res\*a;

a=a\*a;

n>>=1;

}

return res;

}

**伪素数判断（64位）：**

long long multiply( long long a,long long b,long long n)

{

long long exp,res=0;

//这个就是用加法，来模拟乘法，为了防止运算的数太大，

exp=a%n;

while(b)

{

if(b&1)

{

res+=exp;

if(res>n)

res-=n;

}

exp<<=1;

if(exp>n)

exp-=n;

b/=2;

}

return res;

}

long long paw(long long a,long long t,long long n)

{

long long ans=1;

a=a%n;

while(t)

{

if(t%2==1)

ans=multiply(ans,a,n);

a=multiply(a,a,n);

t/=2;

}

return ans;

}

bool miller\_rabin(long long n,long long ti)

{

if(n==2)

return true;

if(n<2||!(n&1))

return false;

long long p=n-1;

long long k=0,a,x;

while(p%2==0)

{

p/=2;

k++;

}

long long ans;

srand(time(0));

for(int t=1; t<=ti; t++)

{

a=rand()%(n-1)+1;

x=paw(a,p,n);

for(int i=1; i<=k; i++)

{

ans=multiply(x,x,n);

if(ans==1&&x!=1&&x!=n-1)

return false;

x=ans;

}

if(ans!=1)

//注意这个判断！

return false;

}

return true;

}

**分解质因数（pollar\_rho）：**

（需要用到伪素数判断）

分解出的数存在pri里面

调用：search(n,180);

long long top=0,pri[100];

long long pollard\_rho(long long a,long long k,long long n)

{

srand(time(0));

long long x=rand()%(n-1)+1;

long long i=1,t=2;

long long y=x;

while(1)

{

i++;

x=(multiply(x,x,n)+k)%n;

long long ans=gcd(y-x,n);

if(ans>1&&ans<n)

return ans;

if(x==y)

return n;

if(t==i)

{

y=x;

t\*=2;

}

}

}

void search(long long n,long long k)

{

if(miller\_rabin(n,10))

{

pri[++top]=n;

return ;

}

else

{

long long p=n;

while(p>=n)

p=pollard\_rho(p,k--,n);

search(p,k);

search(n/p,k);

}

}

**图论：**

**链式前向星：**

struct Node

{

int to,next;

};

Node edge[MAXN];

int box[MAXN];

int ent=1;

void add(int from,int to)

{

edge[ent].to=to;

edge[ent].next=box[from];

box[from]=ent++;

}

void edgefrom(int n) //输出以n为开头的所有边

{

for(int j=0; j<10; ++j)

for(int i=box[j]; i; i=edge[i].next)

{

cout<<j<<"to"<<edge[i].to<<endl;

}

}

**dijkstra（优先队列优化）：**

//如果不用优先队列，则去掉优先队列，在每次循环时找到没有被visited的点中dis值最小的

int dis[maxn2];

bool vis[maxn2];

const int Inf=;

struct FSO

{

int from,dis;

FSO(int from\_,int dis\_):from(from\_),dis(dis\_) {}

bool operator<(const FSO &n)const {return dis>n.dis;}

};

int dijkstra(const int &n,int from,int to)

{

for(int i=0; i<maxn2; ++i)

dis[i]=Inf;

memset(vis,0,sizeof(vis));

priority\_queue<FSO> que;

que.push(FSO(from,0));

dis[from]=0;

int mx,now;

while(1)

{

if(que.empty()) break;

mx=que.top().from;

if(vis[mx])

{

que.pop();

continue;

}

que.pop();

for(int j=box[mx]; j; j=edge[j].next)

{

now=edge[j].to;

if(dis[now]>dis[mx]+1)

{

dis[now]=dis[mx]+1;

que.push(FSO(edge[j].to,dis[now]));

}

}

vis[mx]=1;

}

if(dis[to]!=Inf)

return dis[to];

else return -1;

}

**迭代加深搜索：（搜分解分数式）：**

注意剪枝….

bool ids(int d,double sum,int now) //深度，和，现在的值

{

if(d==maxd)

{

if(abs(sum-val)<dev)

{

return 1;

}

else return 0;

}

int k=(maxd-d)/abs(val-sum)+1; //迭代上限，还剩下maxd-d个数,到val的值为val-sum，为能到达val现在可以填的最小的数即1/k

for(int i=now;i<=k;++i)

{

if(ids(d+1,sum+1.0/i,i+1))

{

cout<<"1/"<<i<<" ";

return 1;

}

}

return 0;

}

Int main()

{

While(ids(0,0,1))

deep++;

}

**第k短路（A\*算法）：**

首先对反图做一遍dijkstra,作为预处理

（存图的时候就存反边

void add(int from,int to,int weight)

{

edge[ent].to=to;

edge[ent].from=from;

edge[ent].next=box[from];

edge[ent].last=rebox[to];

edge[ent].weight=weight;

box[from]=ent;

rebox[to]=ent++;

}

//这个是打印路径的，如果不需要打印路径，则吧关于vector的东西去掉

struct ASO

{

int from,dis,f;

vector<int> luj;

ASO(int from\_,int dis\_,int f\_,vector<int> l):from(from\_),dis(dis\_),f(f\_),luj(l){}

bool operator<(const ASO &a)const

{

return f>a.f;

}

};

int Astar(int from,int to,int k)

{

int mx,now,ndis;

int c=0;

if(from==to)

c—

//这个判断有争议，用不用都可以试下

priority\_queue<ASO> que;

vector<int> tem;

tem.push\_back(from);

que.push(ASO(from,0,dis[from],tem));

while(1)

{

if(que.empty()) break;

mx=que.top().from;

ndis=que.top().dis;

tem=que.top().luj;

que.pop();

if(mx==to)

{

c++;

printf("%d : %d\n",c,ndis);

printf("lujing:");

for(vector<int>::iterator iter=tem.begin();iter!=tem.end();++iter)

cout<<\*iter<<" ";

cout<<endl;

}

if(c==k)

return ndis;

tem.push\_back(1);

for(int i=box[mx]; i; i=edge[i].next)

{

now=edge[i].to;

tem.pop\_back();

tem.push\_back(now);

que.push(ASO(now,ndis+edge[i].weight,ndis+edge[i].weight+dis[now],tem));

}

}

return -1;

}

**最小生成树：**

**krusakal：**

//把所有边按长度排序，然后从头开始加边，如果存在环路则跳过，判断环路用并查集

bool used[maxn]

void kruskal()

{

sort(edge,edge+ent);

ub=0;

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

{

if(find(edge[i].u)==find(edge[i].v))

continue;

Uni(edge[i].u,edge[i].v);

used[ub++]=i;

if(ub==p-1) break;

}

}

**Prim:**

//使用邻接表来存（内存消耗很大，但速度较快）

struct FSO //原来的版本没有dis,直接用dp[p]来判断，这导致队列不能有效更新

{

int p,dis;

FSO(int p\_,int dis\_):p(p\_),dis(dis\_) {}

bool operator<(const FSO &a)const

{

return dis>a.dis;

}

};

void prim()

{

int now;

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

dp[i]=Inf;

memset(vis,0,sizeof(vis));

priority\_queue<FSO> que;

que.push(FSO(0,0));

while(1)

{

if(que.empty()) break;

now=que.top().p;

if(vis[now])

{

que.pop();

continue;

}

que.pop();

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

{

if(i==now) continue;

if(vis[i]) continue;

if(dis[now][i]<dp[i])

{

dp[i]=dis[now][i];

que.push(FSO(i,dp[i]));

}

}

vis[now]=1;

}

}