ACM 常用算法模板

IMG_256IMG_256IMG_256IMG_256

**Author:pengge6666**

**图论**

**匈牙利匹配算法**

**结论：**

1. ****最小点覆盖=匹配数****
2. ****最大点独立集=总点数-匹配数（****方案：将最小点覆盖取反。）

**#include<bits/stdc++.h>**

**using namespace std;**

**const int maxn = 2555 ,N = maxn;**

**int vis[maxn];**

**int match[maxn];**

**int e[maxn][maxn];**

**int x[N],y[N];**

**int n,m;**

**int dfs(int u){ //匈牙利算法模板**

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

**if(vis[i]==0&&e[u][i]==1){**

**vis[i]=1;**

**if(match[i]==0||dfs(match[i])){**

**match[i]=u;**

**return 1;**

**}**

**}**

**}**

**return 0;**

**}**

**int main(){**

**cin>>n**

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

**memset(match,0,sizeof(match));**

**int ans=0;**

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

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

**if(dfs(i)){**

**ans++;**

**}**

**}**

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

**return 0;**

**}**

**LCA树上倍增**

1. 链上两点之间距离

int head[MAXN],cnt;

struct edge{

int to;

int next;

int w;

}e[MAXN\*2];

void addedge(int u,int v,int w){

e[++cnt].to=v;

e[cnt].w=w;

e[cnt].next=head[u];

head[u]=cnt;

}

int dep[MAXN],dis[MAXN],fa[MAXN][26];

void dfs(int root,int pre){

dep[root]=dep[pre]+1;

fa[root][0]=pre;

for(int i=1;(1<<i)<=dep[root];i++)

fa[root][i]=fa[fa[root][i-1]][i-1];

for(int i=head[root];i;i=e[i].next){

if(e[i].to==pre) continue;

dis[e[i].to]=dis[root]+e[i].w;

dfs(e[i].to,root);

}

}

int LCA(int a,int b)

{

if(dep[a]<dep[b])

swap(a,b);

for(int i=19;i>=0;i--)

{

if(dep[a]-(1<<i)>=dep[b])

a=fa[a][i];

}

if(a==b)

return a;

for(int i=19;i>=0;i--)

{

if(fa[a][i]!=fa[b][i])

{

a=fa[a][i];

b=fa[b][i];

}

}

return fa[a][0];

}

void init(){

memset(head,0,sizeof(head));

memset(dis,0,sizeof(dis));

memset(dep,0,sizeof(dep));

memset(fa,0,sizeof(fa));

cnt=0;

}

int DIS(int a,int b){

int lca=LCA(a,b);

return dis[a]+dis[b]-2\*dis[lca];

}

2.树上倍增求链上最大值或最小值

**倍增求链上点最大值(注意：1.cnt先++，2.head初始化为0)**

int deep[N];

int f[30][N],maxv[30][N],maxx[30][N];

void add(int u,int v,int w){

ed[++cnt].e=v;

ed[cnt].v=w;

ed[cnt].next=head[u];

head[u]=cnt;

}

int w[N];

inline int LCA(int x,int y)

{

int maxn=max(w[x],w[y]);

if (deep[x]>deep[y]) swap(x,y);

int k=log2(deep[y]);

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

if (deep[f[i][y]]>=deep[x])

{

maxn=max(maxn,maxx[i][y]);

y=f[i][y];

}

if (x==y) return max(maxn,w[x]);

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

if (f[i][y]!=f[i][x])

{

maxn=max(maxn,maxx[i][x]);

maxn=max(maxn,maxx[i][y]);

y=f[i][y];

x=f[i][x];

}

return max(maxn,maxx[0][x]);

}

inline void dfs(int x,int fa)

{

deep[x]=deep[fa]+1;

f[0][x]=fa;

maxx[0][x]=w[fa];

for (int i=1;(1<<i)<=deep[x];i++)

{

f[i][x]=f[i-1][f[i-1][x]];

maxx[i][x]=max(maxx[i-1][x],maxx[i-1][f[i-1][x]]);

}

for (int i=head[x];i;i=ed[i].next)

if (ed[i].e!=fa) dfs(ed[i].e,x);

return ;

}

**倍增求链上边最大值**

int deep[N];

int f[30][N],maxv[30][N];

void add(int u,int v,int w){

ed[++cnt].e=v;

ed[cnt].v=w;

ed[cnt].net=head[u];

head[u]=cnt;

}

inline int LCA(int x,int y){

int maxx=0;

if (deep[x]>deep[y]) swap(x,y);

int k=log2(deep[y]);

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

if (deep[f[i][y]]>=deep[x])

{

maxx=max(maxv[i][y],maxx);

y=f[i][y];

}

if (x==y) return maxx;

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

if (f[i][y]!=f[i][x])

{

maxx=max(maxv[i][y],maxx);

maxx=max(maxv[i][x],maxx);

y=f[i][y];

x=f[i][x];

}

return max(maxx,max(maxv[0][x],maxv[0][y]));

}

inline void dfs2(int x,int fa)

{

f[0][x]=fa;

for (int i=1;(1<<i)<=deep[x];i++)

{

f[i][x]=f[i-1][f[i-1][x]];

maxv[i][x]=max(maxv[i-1][x],maxv[i-1][f[i-1][x]]);

}

for (int i=head[x];i;i=ed[i].net)

if (ed[i].e!=fa) dfs2(ed[i].e,x);

return ;

}

inline void dfs1(int x,int fa)

{

deep[x]=deep[fa]+1;

for (int i=head[x];i;i=ed[i].net)

if (ed[i].e!=fa) dfs1(ed[i].e,x);

else maxv[0][x]=ed[i].v;

return ;

}

//Main()

dfs1,dfs2

**并查集**

**merge**

int getf(int v){

if(v==f[v]){

return f[v];

}else{

f[v]=getf(f[v]);return f[v];

}

}

void merge(int u,int v){

int t1=getf(u);

int t2=getf(v);

if(t1!=t2){

f[t1]=t2;

sum[t2]+=sum[t1];

}

}

for(int i=0;i<=n+1;i++) sum[i]=1,f[i]=i;

维护下一个数

unordered\_map<int,int> f,a;

int getf(int x){

if(!f.count(x))

return x;

else{

return f[x]=getf(f[x]);

}

}

//f[k]=getf(k+1);（貌似：f[i]=i+1）

FOR循环写法

for(int j=getf(st[i].l);j<=st[i].r;j=getf(j)){

f[j]=j+1;

}

**树的直径**

#include<stdio.h>

#define MAX 100000

using namespace std;

int head[MAX],vis[MAX];//标记当前节点是否已经用过

int dis[MAX];//记录最长距离

int n,m,ans,sum;//记录最长路径的长度

int aga;

struct node{

int u,v,w,next;

}edge[MAX];

void add(int u,int v,int w)//向邻接表中加边

{

edge[ans].u=u;

edge[ans].v=v;

edge[ans].w=w;

edge[ans].next=head[u];

head[u]=ans++;

}

void getmap(){

int i,j,a,b,c;

ans=0;

memset(head,-1,sizeof(head));

while(m--){

scanf("%d%d%d",&a,&b,&c);

char ss[5];

cin>>ss;

add(a,b,c);

add(b,a,c);

}

}

void bfs(int beg){

queue<int>q;

memset(dis,0,sizeof(dis));

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

int i,j;

while(!q.empty())

q.pop();

aga=beg;

sum=0;

vis[beg]=1;

q.push(beg);

int top;

while(!q.empty()){

top=q.front();

q.pop();

for(i=head[top];i!=-1;i=edge[i].next){

if(!vis[edge[i].v]){

dis[edge[i].v]=dis[top]+edge[i].w;

vis[edge[i].v]=1;

q.push(edge[i].v);

if(sum<dis[edge[i].v]){

sum=dis[edge[i].v];

aga=edge[i].v;

}

}

}

}

}

int main(){

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

getmap();

bfs(1);//搜索最长路径的一个端点

bfs(aga);//搜索另一个端点

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

return 0;

}

**树的重心**

定义：

定义1：树的重心也叫树的质心。找到一个点,其所有的子树中最大的子树节点数最少,那么这个点就是这棵树的重心,删去重心后，生成的多棵树尽可能平衡。

定义2(也可以这么描述)：树的重心也叫树的质心。对于一棵树n个节点的无根树，找到一个点，将无根树变为以该点为根的有根树时，最大子树的结点数最小。换句话说，删除这个点后最大连通块（一定是树）的结点数最小。

性质：

树中所有点到某个点的距离和中，到重心的距离和是最小的，如果有两个重心，他们的距离和一样。

把两棵树通过一条边相连，新的树的重心在原来两棵树重心的连线上。

一棵树添加或者删除一个节点，树的重心最多只移动一条边的位置。

一棵树最多有两个重心，且相邻。

#include<bits/stdc++.h>

using namespace std;

#define pb push\_back

#define int long long

const int N = 3e5+100 , inf = 0x3f3f3f3f;

int n;

int sz[N];

vector<int> g[N];

int bnode=0,Maxnode=0;

void dfs(int u,int fa){

sz[u]=1;

int cur=0;

int sum=1;

for(auto to:g[u]){

if(to==fa) continue;

dfs(to,u);

sz[u]+=sz[to];

cur=max(cur,sz[to]);

sum+=sz[to];

}

cur=max(cur,n-sum);

// cout<<u<<" "<<fa<<" "<<cur<<endl;

if(Maxnode>=cur){

if(bnode==0){

bnode=u;

}else{

bnode=min(bnode,u);

}

Maxnode=min(Maxnode,cur);

}

}

signed main(){

while(cin>>n){

bnode=0,Maxnode=n;

for(int i=0;i<=n;i++) g[i].clear();

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

sz[i]=0;

int u,v;

cin>>u>>v;

g[u].pb(v),g[v].pb(u);

}

dfs(1,-1);

cout<<bnode<<" "<<Maxnode<<endl;

}

return 0;

}

**Tarjan**

**Tarjan染色缩点（有向图）//可以考虑度**

**dfn[i]就是时间戳,即在什么时刻搜索到了点i。**

**low[i]则是i点能回溯到的dfn最小的祖先,**

#include<bits/stdc++.h>

using namespace std;

#define int long long

#define N 666666

vector<int> v[N];

int low[N],dfn[N],cnt;

int num[N];

int sig;

int vis[N],used[N];

stack<int> s;

int color[N];

void paint(int x){

s.pop();

color[x]=sig;

num[sig]++;//联通块的大小

vis[x]=0;

}

void tarjan(int x){ //**如果是无向图，则写tarjan(int x,int fa)**

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

s.push(x);

vis[x]=used[x]=1;

for(int i=0;i<v[x].size();i++){

int to=v[x][i];

//**如果是无向图，则加上if(to==fa) continue;**

if(!dfn[to]){

tarjan(to); //**如果是无向图，tarjan(to,x);**

low[x]=min(low[x],low[to]);

}else if(vis[to]){ //**无向图就不用if(vis[to])的判断**

low[x]=min(low[x],dfn[to]);

}

}

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

sig++;//强联通分量总个数

while(s.top()!=x){

int temp=s.top();

paint(temp);

}paint(x);

}

}

signed main(){

int n,m;

cin>>n>>m;

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

int a,b;

cin>>a>>b;

v[a].push\_back(b);//无向图：v[a].pb v[b].pb

}

sig=0;

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

if(!used[i]) tarjan(i);

}

int ans=0;

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

if(num[i]>1){ //

ans++;

}

}

cout<<ans;

return 0;

}

**Tarjan求割边（无向图）**

#include<bits/stdc++.h>

using namespace std;

#define int long long

#define N 666666

int head[N],cnt1=1;

struct edge{

int u;

int to;

int next;

}e[N\*2];

void addedge(int u,int v){

e[++cnt1].to=v;

e[cnt1].next=head[u];

head[u]=cnt1;

}

vector<int> v[N];

int low[N],dfn[N],cnt;

int n,m;

int vis[N];

struct str{

int u,v;

}st[N];

int res;

void add(int u,int v){

st[res].u=min(u,v);

st[res].v=max(u,v);

res++;

}

bool cmp(str a,str b){

if(a.u!=b.u){

return a.u<b.u;

}else{

return a.v<b.v;

}

}

int id;

void tarjan(int u)

{

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

for(int i=head[u];i;i=e[i].next)

{

if(vis[i^1])

continue;

int v=e[i].to;

vis[i]=1;

if(!dfn[v])

{

tarjan(v);

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

if(dfn[u]<low[v])

add(u,v);

}

else

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

}

}

signed main(){

cin>>n>>m;

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

int a,b;

cin>>a>>b;

addedge(a,b);

addedge(b,a);

}

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

if(!dfn[i])

tarjan(i);

sort(st,st+res,cmp);

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

cout<<st[i].u<<" "<<st[i].v<<'\n';

return 0;

}

**Tarjan求割点（无向图）**

**我们删除一个点,要使原本互通的两个(或以上)的节点不互通**

#include<bits/stdc++.h>

using namespace std;

#define int long long

#define N 666666

vector<int> v[N];

map<int,int> mp;

int dfn[N],low[N],cnt;

void tarjan(int u,int fa){

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

int child=0;

for(int i=0;i<v[u].size();i++){

int to=v[u][i];

if(!dfn[to]){

tarjan(to,u);

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

if(low[to]>=dfn[u]&&u!=fa){ //

mp[u]=1;//标记为割点

}

if(u==fa) child++;

}

low[u]=min(low[u],dfn[to]);

}

if(child>=2&&u==fa) mp[u]=1;//标记为割点

}

signed main(){

int n,m;

cin>>n>>m;

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

int a,b;

cin>>a>>b;

v[a].push\_back(b);

v[b].push\_back(a);

}

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

if(!dfn[i]) tarjan(i,i);

}

int sum=0;

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

if(mp[i]) sum++;//计算割点数量

}

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

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

if(mp[i])

printf("%lld ",i);

return 0;

}

**DFS序**

void dfs(int x)

{

in[x] = ++tot;//第一次遇到

num[tot] = x;//num[tot]=x表示在新的序列中，tot这个点是原先的x这个点

for(int i = head[x]; i != -1; i = edge[i].nxt)

dfs(edge[i].to);

out[x] = tot;//最后一次遇到

}

**线段树模板**

/\*\* 有 n 个数和 5 种操作

add a b c：把区间[a,b]内的所有数都增加 c

set a b c：把区间[a,b]内的所有数都设为 c

sum a b：查询区间[a,b]的区间和

max a b：查询区间[a,b]的最大值

min a b：查询区间[a,b]的最小值

\*/

#include <bits/stdc++.h>

using namespace std;

const int maxn = 1e6 + 7;

const long long INF = 1LL << 62;

struct Segment\_tree

{

struct Node

{

int l, r;

long long sum, max, min, set\_lazy, add\_lazy;

} tre[maxn << 2];

long long arr[maxn];//需要输入的数组

inline void push\_up(int rt)

{

if(tre[rt].l == tre[rt].r)

{

return ;

}

tre[rt].sum = tre[rt<<1].sum + tre[rt<<1|1].sum;

tre[rt].max = max(tre[rt<<1].max, tre[rt<<1|1].max);

tre[rt].min = min(tre[rt<<1].min, tre[rt<<1|1].min);

}

inline void push\_down(int rt)

{

if(tre[rt].set\_lazy) {

///if set\_lazy add\_lazy = 0

tre[rt<<1].set\_lazy = tre[rt].set\_lazy;

tre[rt<<1].sum = (tre[rt<<1].r - tre[rt<<1].l + 1) \* tre[rt].set\_lazy;

tre[rt<<1].max = tre[rt].set\_lazy;

tre[rt<<1].min = tre[rt].set\_lazy;

tre[rt<<1|1].set\_lazy = tre[rt].set\_lazy;

tre[rt<<1|1].sum = (tre[rt<<1|1].r - tre[rt<<1|1].l + 1) \* tre[rt].set\_lazy;

tre[rt<<1|1].max = tre[rt].set\_lazy;

tre[rt<<1|1].min = tre[rt].set\_lazy;

tre[rt].add\_lazy = 0;

tre[rt<<1].add\_lazy = tre[rt<<1|1].add\_lazy = 0;

tre[rt].set\_lazy = 0;

return ;

}

if(tre[rt].add\_lazy)

{

tre[rt<<1].add\_lazy += tre[rt].add\_lazy;

tre[rt<<1].sum += (tre[rt<<1].r - tre[rt<<1].l + 1) \* tre[rt].add\_lazy;

tre[rt<<1].max += tre[rt].add\_lazy;

tre[rt<<1].min += tre[rt].add\_lazy;

tre[rt<<1|1].add\_lazy += tre[rt].add\_lazy;

tre[rt<<1|1].sum += (tre[rt<<1|1].r - tre[rt<<1|1].l + 1) \*

tre[rt].add\_lazy;

tre[rt<<1|1].max += tre[rt].add\_lazy;

tre[rt<<1|1].min += tre[rt].add\_lazy;

tre[rt].add\_lazy = 0;

}

}

void build(int rt,int l,int r)

{

tre[rt].l = l;

tre[rt].r = r;

tre[rt].set\_lazy = 0;

tre[rt].add\_lazy = 0;

if(l == r)

{

tre[rt].sum = tre[rt].max = tre[rt].min = arr[l];

return ;

}

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

build(rt<<1,l,mid);

build(rt<<1|1,mid+1,r);

push\_up(rt);

}

void update1(int rt,int l,int r,long long val)///add

{

push\_down(rt);

if(l == tre[rt].l && tre[rt].r == r)

{

tre[rt].add\_lazy = val;

tre[rt].sum += (tre[rt].r - tre[rt].l + 1) \* val;

tre[rt].max += val;

tre[rt].min += val;

return ;

}

int mid = (tre[rt].l + tre[rt].r) >> 1;

if(r <= mid)

{

update1(rt<<1,l,r,val);

}

else if(l > mid)

{

update1(rt<<1|1,l,r,val);

}

else

{

update1(rt<<1,l,mid,val);

update1(rt<<1|1,mid+1,r,val);

}

push\_up(rt);

}

void update2(int rt,int l,int r,long long val)///set

{

push\_down(rt);

if(l == tre[rt].l && tre[rt].r == r) {

tre[rt].set\_lazy = val;

tre[rt].sum = (tre[rt].r - tre[rt].l + 1) \* val;

tre[rt].max = val;

tre[rt].min = val;

tre[rt].add\_lazy = 0;

return ;

}

int mid = (tre[rt].l + tre[rt].r) >> 1;

if(r <= mid) {

update2(rt<<1,l,r,val);

} else if(l > mid) {

update2(rt<<1|1,l,r,val);

} else {

update2(rt<<1,l,mid,val);

update2(rt<<1|1,mid+1,r,val);

}

push\_up(rt);

}

long long query1(int rt,int l,int r)///sum

{

push\_down(rt);

if(l == tre[rt].l && tre[rt].r == r) {

return tre[rt].sum;

}

int mid = (tre[rt].l + tre[rt].r) >> 1;

if(r <= mid) {

return query1(rt<<1,l,r);

} else if(l > mid) {

return query1(rt<<1|1,l,r);

} else {

return query1(rt<<1,l,mid) + query1(rt<<1|1,mid+1,r);

}

}

long long query2(int rt,int l,int r)///max

{

push\_down(rt);

if(l == tre[rt].l && tre[rt].r == r) {

return tre[rt].max;

}

int mid = (tre[rt].l + tre[rt].r) >> 1;

if(r <= mid) {

return query2(rt<<1,l,r);

} else if(l > mid) {

return query2(rt<<1|1,l,r);

} else {

return max(query2(rt<<1,l,mid), query2(rt<<1|1,mid+1,r));

}

}

long long query3(int rt,int l,int r)///min

{

push\_down(rt);

if(l == tre[rt].l && tre[rt].r == r) {

return tre[rt].min;

}

int mid = (tre[rt].l + tre[rt].r) >> 1;

if(r <= mid) {

return query3(rt<<1,l,r);

} else if(l > mid) {

return query3(rt<<1|1,l,r);

} else {

return min(query3(rt<<1,l,mid), query3(rt<<1|1,mid+1,r));

}

}

} S[5];

**离散化模板**

void LS(){

m=0;

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

b[++m]=a[i];

sort(b+1,b+1+m);

m=unique(b+1,b+1+m)-b-1;

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

a[i]=lower\_bound(b+1,b+1+m,a[i])-b;

return ;

}

**树状数组（树状数组处理不了值为0的情况）**

**求逆序数**

#define lowbit(x) (x&(-x))

int tr[N];

void upd(int x,int v) {

for(int i=x;i<N;i+=lowbit(i))

tr[i]+=v;

}

int qy(int x) {

int res=0;

for(int i=x;i>0;i-=lowbit(i))

res+=tr[i];

return res;

}

signed main(){

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

ans+=i-qy(arr[i])-1;

upd(arr[i],1);

}

return 0;

}

模板2

#define lowbit(x) (x&(-x))

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*树状数组

struct BIT{

int tree[N];

void init(){

for(int i=0;i<N;i++) tree[i]=0;

}

void add(int x,int y){

for(int i=x;i<N;i+=lowbit(i)) tree[i]+=y;

}

int ask(int x){

int sum=0;

for(int i=x;i;i-=lowbit(i)) sum+=tree[i];

return sum;

}

int query(int x,int y){

if(x>y) return 0;

return ask(y)-ask(x-1);

}

}t1,t2;

**主席树(可能要离散化)**

1.动态查询区间中有多少不同的数

**Input**

5

1 1 2 1 3

3

1 5

2 4

3 5

**Output**

3

2

3

#include<bits/stdc++.h>

using namespace std;

const int N = 6e5+100;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*主席树(动态查询区间中有多少不同的数)

struct Node{

int l,r;

int cnt;

}tr[N\*20];

int root[N],idx;

int build(int l,int r){

int p=++idx;

tr[p].cnt=0;

if(l==r) return p;

int mid=(l+r)/2;

tr[p].l=build(l,mid);

tr[p].r=build(mid+1,r);

return p;

}

int insert(int l,int r,int p,int pos,int val){

int q=++idx;

tr[q]=tr[p];

if(l==r){

tr[q].cnt+=val;

return q;

}

int mid=(l+r)/2;

if(pos<=mid){

tr[q].l=insert(l,mid,tr[p].l,pos,val);

}else{

tr[q].r=insert(mid+1,r,tr[p].r,pos,val);

}

tr[q].cnt=tr[tr[q].l].cnt+tr[tr[q].r].cnt;

return q;

}

int query(int pos,int x,int l,int r){

if(l==r){

return tr[x].cnt;

}

int mid=(l+r)/2;

if(pos<=mid){

return tr[tr[x].r].cnt+query(pos,tr[x].l,l,mid);

}else{

return query(pos,tr[x].r,mid+1,r);

}

}

int n,m;

int a[N];

map<int,int> vis;

void init(){

vis.clear();

for(int i=0;i<N;i++) root[i]=0;

idx=0;

}

signed main(){

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

init();

for(int i=1;i<=n;i++) scanf("%d",&a[i]);

scanf("%d",&m);

root[0]=build(1,n);

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

if(!vis[a[i]]){

root[i]=insert(1,n,root[i-1],i,1);

}else{

int tmp=0;

tmp=insert(1,n,root[i-1],vis[a[i]],-1);

root[i]=insert(1,n,tmp,i,1);

}

vis[a[i]]=i;

}

while(m--){

int l,r;

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

int ans=query(l,root[r],1,n);

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

}

}

return 0;

}

1. 查询第K小数

input

7 3

1 5 2 6 3 7 4

2 5 3

4 4 1

1 7 3

output

5

6

3

#include<bits/stdc++.h>

using namespace std;

#define int long long

#define N 666666

int n,m;

int a[N];

vector<int> nums;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*主席树

struct Node{

int l,r;//左右儿子编号

int cnt; //权值线段树上的数字个数

}tr[N\*22];

int root[N],idx;//每个起始根节点的下标

int build(int l,int r){

int p=++idx;

if(l==r) return p;

int mid=(l+r)/2;

tr[p].l=build(l,mid);

tr[p].r=build(mid+1,r);

return p;

}

int insert(int p,int l,int r,int x){

int q=++idx;

tr[q]=tr[p];

if(l==r){

tr[q].cnt++;

return q;

}

int mid=(l+r)/2;

if(x<=mid) tr[q].l=insert(tr[p].l,l,mid,x);

else tr[q].r=insert(tr[p].r,mid+1,r,x);

tr[q].cnt=tr[tr[q].l].cnt+tr[tr[q].r].cnt;

return q;

}

int query(int q,int p,int l,int r,int k){

if(l==r) return r;

int cnt=tr[tr[q].l].cnt-tr[tr[p].l].cnt;

int mid=(l+r)/2;

if(k<=cnt) return query(tr[q].l,tr[p].l,l,mid,k);

else return query(tr[q].r,tr[p].r,mid+1,r,k-cnt);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*离散化

int find(int x){

return lower\_bound(nums.begin(),nums.end(),x)-nums.begin();

}

signed main(){

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

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

scanf("%lld",&a[i]);

nums.push\_back(a[i]);

}

sort(nums.begin(),nums.end());

nums.erase(unique(nums.begin(),nums.end()),nums.end());

root[0]=build(0,nums.size()-1);

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

root[i]=insert(root[i-1],0,nums.size()-1,find(a[i]));

}

while(m--){

int l,r,k;

scanf("%lld%lld%lld",&l,&r,&k);

int ans=nums[query(root[r],root[l-1],0,nums.size()-1,k)];

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

}

return 0;

}

3查询区间小于等于K的个数(或者查询区间大于等于K的个数)

**查找区间小于等于 k 的数的个数(build , insert和前面一样都得有)**

int query(int q,int p,int l,int r,int k){

if(l==r){

return tr[q].cnt-tr[p].cnt;

}

int mid=(l+r)/2;

if(k<=mid) return query(tr[q].l,tr[p].l,l,mid,k);

else{

int sum=0;

sum=tr[tr[q].l].cnt-tr[tr[p].l].cnt;

sum+=query(tr[q].r,tr[p].r,mid+1,r,k);

return sum;

}

}

查找区间大于等于 k 的数的个数

int query(int q,int p,int l,int r,int num){

if(l==r){

return tr[q].cnt-tr[p].cnt;

}

int mid=(l+r)/2;

if(num<=mid){

return query(tr[q].l,tr[p].l,l,mid,num)+(tr[tr[q].r].cnt-tr[tr[p].r].cnt);

}else{

return query(tr[q].r,tr[p].r,mid+1,r,num);

}

}

**树链剖分**

重结点：子树结点数目最多的结点；

轻节点：父亲节点中除了重结点以外的结点；

重边：父亲结点和重结点连成的边；

轻边：父亲节点和轻节点连成的边；

重链：由多条重边连接而成的路径；

轻链：由多条轻边连接而成的路径；

名称 解释

f[u] 保存结点u的父亲节点

d[u] 保存结点u的深度值

size[u] 保存以u为根的子树节点个数

son[u] 保存重儿子//叶节点没有重儿子，非叶节点有且只有一个重儿子

rk[u] 保存当前dfs标号在树中所对应的节点

top[u] 保存当前节点所在链的顶端节点

id[u] 保存树中每个节点剖分以后的新编号（DFS的执行顺序）

(简单说)

dep[x]:x的深度

size[x]:以x为根的子树节点个数总和

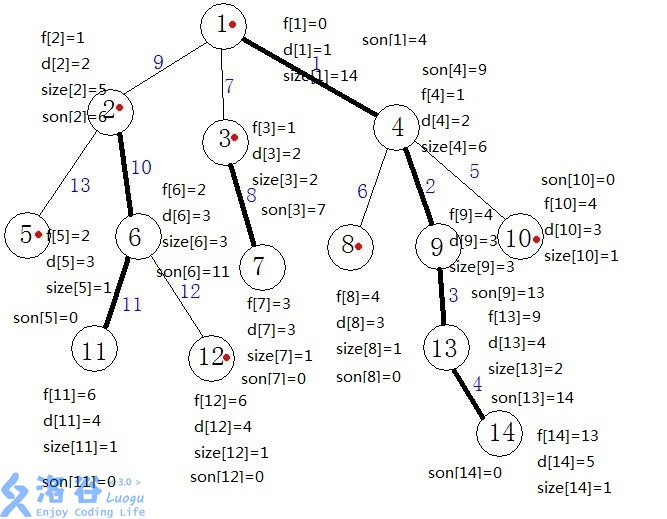
id[x]:dfs序所访问的编号

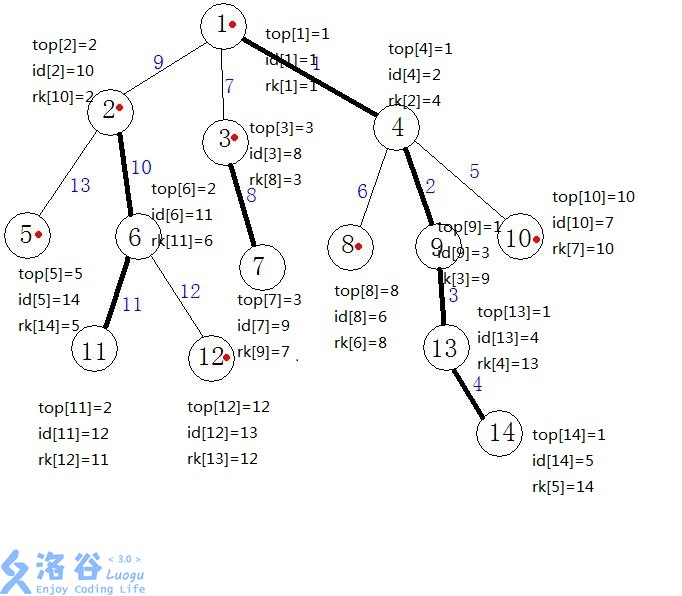
fa[x]:保存u的父亲节点

son[x]:保存重儿子

rk[x]:保存当前dfs序标号在树中所对应的节点

top[x]:保存重链的顶点





/\*

1 x y z，表示将树从 xx 到 yy 结点最短路径上所有节点的值都加上 zz。

2 x y，表示求树从 xx 到 yy 结点最短路径上所有节点的值之和。

3 x z，表示将以 xx 为根节点的子树内所有节点值都加上 zz。

4 x 表示求以 xx 为根节点的子树内所有节点值之和

\*/

#include<bits/stdc++.h>

using namespace std;

const int N=1e5+1000;

struct edge

{

int next,to;

}e[N<<2];

struct node

{

int l,r,w,siz,lazy;

}tr[N<<2];

int idx,root,MOD,cnt=0,n,m;

int a[N];

int dep[N],fa[N],son[N],size[N],top[N],id[N],rk[N];

int h[N];

//深度 ，父节点，重链儿子节点， ，重链的顶点，每个节点编号 ,编号对应的节点

void add(int u,int v)

{

e[++idx].to=v;e[idx].next=h[u];h[u]=idx;

}

void dfs1(int u,int father,int depth )//处理深度，父亲节点，u为根的节点个数

{

dep[u]=depth;fa[u]=father;size[u]=1;

int maxn=-1;

for(int i=h[u];~i;i=e[i].next)

{

int j=e[i].to;

if(j==father) continue;

dfs1(j,u,depth+1);

size[u]+=size[j];

if(size[j]>maxn) //重链的儿子

{

maxn=size[j];

son[u]=j;

}

}

}

void dfs2(int u,int v)

{

top[u]=v;//标记重链的顶点

id[u]= ++cnt;//节点对应编号

rk[cnt]=a[u];//编号对应节点建树的关键一员

if(!son[u]) return ;

dfs2(son[u],v);

for(int i=h[u];~i;i=e[i].next)

{

int j=e[i].to;

if(j!=son[u]&&j!=fa[u]) dfs2(j,j);//非重链的顶点就是自己

}

}

void push\_up(int u)

{

tr[u].w=(tr[u<<1].w+tr[u<<1|1].w+MOD)%MOD;

}

void build(int u,int l,int r)

{

tr[u].l=l;tr[u].r=r;tr[u].siz=r-l+1;

if(l==r)

{

**tr[u].w=rk[l];//注意：如果题目给了点权，则有需要可以更改位arr[rk[l]]**

return ;

}

int mid=l+r>>1;

build(u<<1,l,mid);build(u<<1|1,mid+1,r);

push\_up(u);

}

void push\_down(int u)//

{

if(tr[u].lazy)

{

tr[u<<1].w=(tr[u<<1].w+tr[u<<1].siz\*tr[u].lazy)%MOD;

tr[u<<1|1].w=(tr[u<<1|1].w+tr[u<<1|1].siz\*tr[u].lazy)%MOD;

tr[u<<1].lazy=(tr[u<<1].lazy+tr[u].lazy)%MOD;

tr[u<<1|1].lazy=(tr[u<<1|1].lazy+tr[u].lazy)%MOD;

tr[u].lazy=0;

}

}

void Interadd(int u,int l,int r,int val)//

{

if(l<=tr[u].l&&tr[u].r<=r)

{

tr[u].w+=tr[u].siz\*val;

tr[u].lazy+=val;

return ;

}

push\_down(u);

int mid=tr[u].l+tr[u].r>>1;

if(l<=mid) Interadd(u<<1,l,r,val);

if(r>mid) Interadd(u<<1|1,l,r,val);

push\_up(u);

}

void treeadd(int x,int y,int val)//

{

while(top[x]!=top[y])//将两个节点跳到同一重链上

{

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

Interadd(1,id[top[x]],id[x],val);

x=fa[top[x]];

}

if(dep[x]>dep[y]) swap(x,y);//节点编号的顺序依照深度大小，深度越大节点编号越大

Interadd(1,id[x],id[y],val);//因为遍历线段树区间[l,r]必须从小到大

}

int Insum(int u,int l,int r)//

{

int ans=0;

if(l<=tr[u].l&&tr[u].r<=r) return tr[u].w;

push\_down(u);

int mid=tr[u].l+tr[u].r>>1;

if(l<=mid) ans=(ans+Insum(u<<1,l,r))%MOD;

if(r>mid) ans=( ans+Insum(u<<1|1,l,r))%MOD;

return ans;

}

void querysum(int x,int y)//

{

int ans=0;

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

{

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

ans=(ans+Insum(1,id[top[x]],id[x]))%MOD;

x=fa[top[x]];

}

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

ans=(ans+Insum(1,id[x],id[y]))%MOD;

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

}

int main()

{

memset(h,-1,sizeof h);

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

for(int i=1;i<=n;i++) scanf("%d",&a[i]);

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

{

int x,y;

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

add(x,y);add(y,x);

}

dfs1(root,0,1);

dfs2(root,root);

build(1,1,n);

while(m--)

{

int op,x,y,z;

cin>>op;

if(op==1)

{

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

treeadd(x,y,z);

}

else if(op==2)

{

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

querysum(x,y);

}

else if(op==3)

{

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

Interadd(1,id[x],size[x]+id[x]-1,y%MOD);

}

else if(op==4)

{

scanf("%d",&x);

printf("%d\n",Insum(1,id[x],id[x]+size[x]-1));

}

}

}

//如果题目给了边权，可以转化为点权

将边权赋值给深度大的节点（在build建完树之后操作）

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

if(dep[st[i].x]>dep[st[i].y]){//x上y下

swap(st[i].x,st[i].y);

}

update1(1,id[st[i].y],id[st[i].y],st[i].z);//区间修改---》区间加数

}

**最短路**

struct node{

int v,w;

bool operator < (const node &t) const

{

return w>t.w;

}

};

int n,m,c;

int u[M],v[M],p[M];

vector<node> g[N];

int vis[N];

int dis[N][N];

void dijkstra(int s){

for(int i=1;i<=n;i++) vis[i]=0,dis[s][i]=inf;

dis[s][s]=0;

priority\_queue<node> q;

q.push({s,0});

while(!q.empty()){

node t=q.top();

q.pop();

int o=t.v;

if(vis[o]) continue;

vis[o]=1;

for(int i=0;i<g[o].size();i++){

int to=g[o][i].v;

int wo=g[o][i].w;

if(dis[s][to]>dis[s][o]+wo){

dis[s][to]=min(dis[s][to],dis[s][o]+wo);

q.push({to,dis[s][to]});

}

}

}

return ;

}

**网络流(范围小，大多N在500以下)**

**Dinic最大流**

#include<bits/stdc++.h>

using namespace std;

#define int long long

const int maxn= 10090 , inf = 99999999;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Dinic最大流

int head[maxn\*3],sign,cur[maxn\*3];

int s,t,d[maxn\*3];

struct node

{

int to,w,next;

}edge[maxn\*500];

void creat()

{

memset(head,-1,sizeof(head));

sign=0;

}

void add\_edge(int u,int v,int w)

{

edge[sign].to=v;

edge[sign].w=w;

edge[sign].next=head[u];

head[u]=sign++;

edge[sign].to=u;

edge[sign].w=0;

edge[sign].next=head[v];

head[v]=sign++;

}

int bfs()

{

queue<int>q;

memset(d,0,sizeof(d));

d[s]=1;

q.push(s);

while(!q.empty())

{

int top=q.front();

q.pop();

for(int i=head[top];~i;i=edge[i].next)

{

int to=edge[i].to;

if(edge[i].w>0&&d[to]==0)

{

d[to]=d[top]+1;

if(to==t)

return 1;

q.push(to);

}

}

}

return d[t]!=0;

}

int dfs(int top,int flow)

{

if(top==t)

return flow;

int ans=0,x=0;

for(int i=cur[top];~i;i=edge[i].next)

{

int to=edge[i].to;

if(edge[i].w>0&&d[to]==d[top]+1)

{

x=dfs(to,min(flow-ans,edge[i].w));

edge[i].w-=x;

edge[i^1].w+=x;

if(edge[i].w)

cur[top] = i;

ans+=x;

if(ans==flow)

return flow;

}

}

if(ans==0)

d[top]=0;

return ans;

}

int dinic(int n)

{

int ans=0;

while(bfs())

{

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

cur[i]=head[i];

ans+=dfs(s,inf);

}

return ans;

}

signed main(){

creat();

s=6666,t=s+1;

add\_edge(i+n+1,t,m);//加边

int ans=dinic(t);

return 0;

}

**最小费用最大流**

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define N 6666

#define inf 999999999999

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*建图

int head[N],cnt=0;

struct Node{

int v;

int w;//指权值

int f;//指流量

int next;

}edge[N\*2];

void addedge(int u,int v,int flow,int cost){

edge[cnt].v=v;

edge[cnt].w=cost;

edge[cnt].f=flow;

edge[cnt].next=head[u];

head[u]=cnt++;

//添加反向边

edge[cnt].v=u;

edge[cnt].w=-cost;

edge[cnt].f=0;

edge[cnt].next=head[v];

head[v]=cnt++;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Dinic+SPFA

int ans,mincost;

int n,m,s,t;

int dis[N],vis[N];

bool spfa(int s,int t){

for(int i=0;i<N;i++) dis[i]=inf,vis[i]=0;

queue<int> q;

q.push(s);

vis[s]=1;

dis[s]=0;

while(!q.empty()){

int u=q.front();

q.pop();

vis[u]=0;

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

int v=edge[i].v;

int w=edge[i].w;

int f=edge[i].f;

if(f>0&&dis[v]>dis[u]+w){

dis[v]=dis[u]+w;

if(!vis[v]) q.push(v),vis[v]=1;

}

}

}

return (dis[t]!=inf);

}

int dfs(int u,int flow){

if(u==t){

return flow;

}

int detla=flow;

vis[u]=1;

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

int v=edge[i].v;

int w=edge[i].w;

int f=edge[i].f;

if(f>0&&!vis[v]&&dis[v]==dis[u]+w){

int d=dfs(v,min(detla,f));

edge[i].f-=d;edge[i^1].f+=d;

detla-=d;

mincost+=d\*w;

if(detla==0) break;

}

}

vis[u]=0;

return flow-detla;

}

void Dinic(){

ans=0;

mincost=0;

while(spfa(s,t)){

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

ans+=dfs(s,inf);

}

return ;

}

void init(){

memset(head,-1,sizeof(head));

cnt=0;

}

int M,A,B,C,a,b,c;

void slove1(){

//addedge(int u,int v,int flow,int cost)

s=7,t=8;

addedge(s,1,A,0);

addedge(s,2,B,0);

Dinic();

}

signed main(){

init();

cin>>M>>A>>B>>C>>a>>b>>c;

slove1();

cout<<mincost<<" "<<min(A,b)+min(B,c)+min(C,a);

return 0;

}

**ST表**

#include<bits/stdc++.h>

using namespace std;

#define int long long

#define maxn 666666

int a[maxn];

int stmin[maxn][20];

int stmax[maxn][20];

int n,m;

//st[i][j]表示[i,i+2^j-1]的最值

void Init(){

int i,j;

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

stmin[i][0]=stmax[i][0]=a[i];

for(i=1;(1<<i)<=n;i++){ //按区间长度递增顺序递推

for(j=1;j+(1<<i)-1<=n;j++){ //区间起点

stmin[j][i]=min(stmin[j][i-1],stmin[j+(1<<(i-1))][i-1]);

stmax[j][i]=max(stmax[j][i-1],stmax[j+(1<<(i-1))][i-1]);

}

}

}

int Query\_MAX(int l,int r){//得到[l,r]的最值

int k=(int)(log(r-l+1)/log(2));//log也是需要时间的，有些题目会T，预处理向下取整更好

return max(stmax[l][k],stmax[r-(1<<k)+1][k]);

// return min(stmin[l][k],stmin[r-(1<<k)+1][k]);

}

int Query\_MIN(int l,int r){//得到[l,r]的最值

int k=(int)(log(r-l+1)/log(2));//log也是需要时间的，有些题目会T，预处理向下取整更好

// return max(stmax[l][k],stmax[r-(1<<k)+1][k]);

return min(stmin[l][k],stmin[r-(1<<k)+1][k]);

}

signed main(){

int i,l,r;

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

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

scanf("%lld",&a[i]);

Init();

for(i=1;i<=m;i++){

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

printf("%lld\n",Query\_MAX(l,r)-Query\_MIN(l,r));

}

return 0;

}

**最大权闭合子图 (最大流最小割模型)**

引入闭合子图的概念 : 通俗点说就是选出一个图的子图，使得子图中的所有点出度指向的点依旧在这个子图内，则说明此子图是闭合子图。

最大权闭合子图 : 假设每个点具有点权值，在一个图的所有闭合子图中，点权之和最大的即是最大权闭合子图。

求取最大权闭合子图的权值之和是有一个结论的

一、先抽象出一个超级源、汇点

二、将权值为正的点和超级源点连接、容量为权值

三、将权值为负的点和超级汇点连接、容量为权值的绝对值

四、然后除了源、汇之外的点原本怎么连泽怎么连、且容量为无穷大

五、最大权闭合子图权值 = 所有权值为正的权值之和 - 最大流

/\*

洛谷：P2762 太空飞行计划问题（**模型转换**）

题意：给你若干个设备，若干个任务。

每个任务需要若干设备，设备可重复利用。

完成任务有钱，买设备要钱。

问最大总收益(可以什么任务都不做)。

sample:

input

2 3

10 1 2

25 2 3

5 6 7

output

1 2

1 2 3

17

\*/

#include<bits/stdc++.h>

#define inf 0x3fffffff

using namespace std;

const int maxn=555;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Dinic最大流

int head[maxn],sign,cur[maxn];

int s,t,d[maxn];

int mp[maxn][maxn];

struct node

{

int to,w,next;

}edge[maxn\*50];

void creat()

{

memset(head,-1,sizeof(head));

sign=0;

}

void add\_edge(int u,int v,int w)

{

edge[sign].to=v;

edge[sign].w=w;

edge[sign].next=head[u];

head[u]=sign++;

edge[sign].to=u;

edge[sign].w=0;

edge[sign].next=head[v];

head[v]=sign++;

}

int bfs()

{

queue<int>q;

memset(d,0,sizeof(d));

d[s]=1;

q.push(s);

while(!q.empty())

{

int top=q.front();

q.pop();

for(int i=head[top];~i;i=edge[i].next)

{

int to=edge[i].to;

if(edge[i].w>0&&d[to]==0)

{

d[to]=d[top]+1;

if(to==t)

return 1;

q.push(to);

}

}

}

return d[t]!=0;

}

int dfs(int top,int flow)

{

if(top==t)

return flow;

int ans=0,x=0;

for(int i=cur[top];~i;i=edge[i].next)

{

int to=edge[i].to;

if(edge[i].w>0&&d[to]==d[top]+1)

{

x=dfs(to,min(flow-ans,edge[i].w));

edge[i].w-=x;

edge[i^1].w+=x;

if(edge[i].w)

cur[top] = i;

ans+=x;

if(ans==flow)

return flow;

}

}

if(ans==0)

d[top]=0;

return ans;

}

int dinic(int n)

{

int ans=0;

while(bfs())

{

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

cur[i]=head[i];

ans+=dfs(s,inf);

}

return ans;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*slove

int n,m;

int cost[maxn];

int pay[maxn];

vector<int> res;

signed main(){

creat();

scanf("%d%d",&m,&n);//m 是实验数，n 是仪器数

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

int sum=0;

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

scanf("%d",&cost[i]);

sum+=cost[i];

add\_edge(s,i,cost[i]);

while(1){

int num;

char ch;

scanf("%d%c",&num,&ch);

add\_edge(i,num+m,inf);

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

}

}

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

scanf("%d",&pay[i]);

add\_edge(i+m,t,pay[i]);

}

int ANS=sum-dinic(t);

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

if(d[i]) printf("%d ",i);

}

printf("\n");

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

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

}

printf("\n");

printf("%d",ANS);

return 0;

}

// Clever King：同侧点也可以相连

**动态规划**

背包问题（可求方案数）

01背包

题目背景：有N件物品和一个大小为M的背包，以及若干物品，每种物品只有一件，大小分别为w[i] ，其价值分别为val[i]。问题：将哪些物品装入背包，可使得背包内的物品价值总和最大？

#include <stdio.h>

int max(int a,int b)

{

if(a>b) return a;

else return b;

}

int bag[1000];//降维到一维的数组

int w[1000],val[1000];//每件物品的大小与价值

int main()

{

int n,m;

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

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

scanf("%d%d",&w[i],&val[i]);//读入数据

for(int i=1;i<=n;i++) //从第一件物品一直处理到第i件物品

for(int j=m;j>0;j--) //倒序处理

if(j>=w[i])//塞得进去这件物品的话

bag[j]=max(bag[j],bag[j-w[i]]+val[i]);//就判断，处理后覆盖上一次处理的数据

else//塞不进去就是原来的值

bag[j]=bag[j];//这行代码完全没有意义，但是为了方便对比，留在这里

printf("%d\n",bag[m]);//全部处理完后，输出最后的答案

return 0;

}

完全背包

题目背景：将0/1背包问题中的物品，全部改为可以取无数件，则在这种情况下，求能获得的最值。

#include <stdio.h>

int max(int a,int b)

{

return a>b?a:b;

}

int bag[1000];//降维到一维的数组

int w[1000],val[1000];//每件物品的大小与价值

int main()

{

int n,m;

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

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

scanf("%d%d",&w[i],&val[i]);//读入数据

for(int i=1;i<=n;i++) //从第一件物品一直处理到第i件物品

for(int j=w[i];j<=m;j++) //改为正序处理，由于仅装得下时数据才会覆盖更新，所以从w[i]开始循环

bag[j]=max(bag[j],bag[j-w[i]]+val[i]);//此时必定装得下，于是直接处理，覆盖上一次处理的数据

printf("%d\n",bag[m]);//全部处理完后，输出最后的答案

return 0;

}

多维费用背包

题目背景：此时背包的花费是多维情况，比如说每件物品有一个大小和一个重量，而背包所能装的大小和重量都有限。其他与完全、0/1背包相同。

思路：费用变成多维，那么状态也变成多维即可。

假设有二维费用,那么就设bag[v][w]来存储在背包所装物品大小不超过v，重量不超过w时的最大价值。

只需要在循环求解时，多加上一层循环即可，两层循环一层来考虑v，一层来考虑w

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int vi[50],wi[50],k[50];

int bag[400][400];

int max(int a,int b)

{

return a>b?a:b;

}

int main()

{

int v,w,n;

while(scanf("%d%d%d",&v,&w,&n)!=EOF)//最大可装体积，最大可装重量，物品件数

{

memset(bag,0,sizeof(bag));

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

scanf("%d%d%d",&vi[i],&wi[i],&k[i]);

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

for(int j=w;j>=wi[i];j--)//一层循环考虑重量

for(int l=v;l>=vi[i];l--)//一层循环考虑体积

bag[j][l]=max(bag[j][l],bag[j-wi[i]][l-vi[i]]+k[i]);

printf("%d\n",bag[w][v]);

}

return 0;

}

多重背包

题目背景：在0/1背包的基础上加上一个条件：此时每种物品不再只有一件，而是最多有n[i]件可用。

#include <cstdio>

#include <algorithm> //使用c++的标准库，内置max函数

int bag[1000];//已经降维为一维的背包

int w[1000],val[1000],num[1000];//每件物品的大小与价值、最多能装的数量

int main(){

int n,m;

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

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

scanf("%d%d%d",&w[i],&val[i],&num[i]);//读入数据

for(int i=1;i<=n;i++) //从第一件物品一直处理到第i件物品

for(int j=m;j>=w[i];j--) //倒序处理

for(int l=1;l<=num[i];l++)//多了一层循环

if(j>=l\*w[i])//装得进去l件物品的话

bag[j]=max(bag[j],bag[j-w[i]\*l]+val[i]\*l);//就判断是塞还是不塞比较好，注意乘l

printf("%d\n",bag[m]);//全部处理完后，输出最后的答案

return 0;

}

分组背包

题目背景：在分组背包问题中，n件物品被分为了k 组，其中每组内的物品最多只能取一件，求所能拿的最大价值。

#include <stdio.h>

#include <algorithm>

#include <string.h>

using namespace std;

struct node{int w;int val;int t;}art[1005];

int num[105];

int dp[1005];

int cmp(node a,node b)

{

return a.t<b.t;

}

int main()

{

int n,m;

while(scanf("%d%d",&m,&n)!=EOF)//n件物品，背包大小为m

{

int k=0;

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

{

scanf("%d%d%d",&art[i].w,&art[i].val,&art[i].t);//物品的大小、价值、组别

num[art[i].t]++;//该组别所含物品数增多

k=max(k,art[i].t);//判定有多少个组

}

sort(art,art+n,cmp);//按照组别进行排序，同组的排在一起

int p=0,q;//用于顺序处理的指针

for(int i=1;i<=k;i++)//k个组

{

for(int j=m;j>=0;j--)//01背包的倒序处理方案

{

q=p;

for(int l=1;l<=num[i];l++)//每个组有num[i]件物品

{

if(j>=art[q].w)

dp[j]=max(dp[j],dp[j-art[q].w]+art[q].val);//这种处理方案能保证一组里只取一件

q++;

}

if(j==0) p=q;

}

}

printf("%d\n",dp[m]);

}

}

//字符串中子序列出现的次数

signed main(){

string s;

cin>>s;

for(int i=0;i<s.size();i++){

s[i]=tolower(s[i]);

dp[4]=(dp[4]+(s[i]=='v')\*dp[3])%mod;

dp[3]=(dp[3]+(s[i]=='o')\*dp[2])%mod;

dp[2]=(dp[2]+(s[i]=='l')\*dp[1])%mod;

dp[1]=(dp[1]+(s[i]=='i'))%mod;

}

cout<<dp[4];

return 0;

}

**数位DP**

/\*

给定两个正整数 a 和 b，求在[a,b]中的所有整数中，每个数码(digit)各出现了多少次。

sample

input

1 99

output

9 20 20 20 20 20 20 20 20 20

\*/

#include<bits/stdc++.h>

using namespace std;

#define int long long

const int N = 666 ;

int dp[N][2][2][20];

int a[N];

int p;

class Solution {

public:

//sta:表示前导0 1:表示有前导0 0:表示没有前导0

//limit:表示当前位置是否到达数位的上界

//count:计算符合条件的数（10^count） (count指的是指数上的数)

int DP(int pos,int sta,int limit,int count){

if(!pos) return count;

if(!limit&&sta&&dp[pos][sta][limit][count]!=-1)

return dp[pos][sta][limit][count];

int high=limit?a[pos]:9;

int tot=0;

for(int i=0;i<=high;i++){

tot+=DP(pos-1,i||sta,limit&&(i==high),count+(sta||i)\*(i==p));

}

if(!limit&&sta)

dp[pos][sta][limit][count]=tot;

return tot;

}

int Count(int n) {

memset(dp,-1,sizeof(dp));

int len=0;

int m=n;

while(m){

a[++len]=m%10,m/=10;

}

return DP(len,0,1,0);

}

}S;

signed main(){

int L,R;

cin>>L>>R;

for(int i=0;i<=9;i++){

p=i;

cout<<S.Count(R)-S.Count(L-1)<<" ";

}

return 0;

}

# **上下界模板数位dp**

#include<bits/stdc++.h>

using namespace std;

#define int long long

int a[33],b[33];

int dp[66][2][2][2][2][2][2];

int DP(int pos,int limit1,int limit2,int limit3,int ismin1,int ismin2,int ismin3){

if(pos==-1) return 1;

if(!limit1&&!limit2&&!limit3&&dp[pos][limit1][limit2][limit3][ismin1][ismin2][ismin3]!=-1)

return dp[pos][limit1][limit2][limit3][ismin1][ismin2][ismin3];

int high1=limit1?a[pos]:1;

int high2=limit2?a[pos]:1;

int high3=limit3?a[pos]:1;

int low1=ismin1?b[pos]:0;

int low2=ismin2?b[pos]:0;

int low3=ismin3?b[pos]:0;

int ans=0;

for(int i=low1;i<=high1;i++){

for(int j=low2;j<=high2;j++){

for(int k=low3;k<=high3;k++){

if((i^j^k)==(i|j|k))

ans=ans+DP(pos-1,(i==high1)&&limit1,(j==high2)&&limit2,(k==high3)&&limit3,(i==low1)&&ismin1,(j==low2)&&ismin2,(k==low3)&&ismin3);

}

}

}

if(!limit1&&!limit2&&!limit3)

dp[pos][limit1][limit2][limit3][ismin1][ismin2][ismin3]=ans;

return ans;

}

int slove(int L,int R){

for(int i=0;i<=30;i++){

if((1<<i)&R) a[i]=1;else a[i]=0;

if((1<<i)&L) b[i]=1;else b[i]=0;

}

return DP(30,1,1,1,1,1,1);

}

signed main(){

memset(dp,-1,sizeof(dp));

int L,R;

cin>>L>>R;

cout<<slove(L,R);

return 0;

}

区间DP

for(int len = 1;len<=n;len++){//枚举长度

for(int j = 1;j+len<=n+1;j++){//枚举起点，ends<=n

int ends = j+len - 1;

for(int i = j;i<ends;i++){//枚举分割点，更新小区间最优解

dp[j][ends] = min(dp[j][ends],dp[j][i]+dp[i+1][ends]+something);

}

}

}

树形DP

1.

#include<bits/stdc++.h>

using namespace std;

#define int long long

const int N = 2e5+100;

#define fi first

#define se second

vector<pair<int,int> > g[N];

int dp[N][6],p[N];

class Solution {

public:

/\*\*

\* 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可

\*

\* @param n int整型 节点个数

\* @param u int整型vector

\* @param v int整型vector

\* @return int整型

\*/

/\*

dp[u][0]:表示第u节点向下的最长链

dp[u][1]:表示第u节点向下的次长链

dp[u][2]:表示第u节点向上的最长链

\*/

void dfs1(int u,int fa){

int len=g[u].size();

for(int i=0;i<len;i++){

int v=g[u][i].fi;

int w=g[u][i].se;

if(v==fa) continue;

dfs1(v,u);

if(dp[u][0]<dp[v][0]+w){

p[u]=v;

dp[u][1]=dp[u][0];

dp[u][0]=dp[v][0]+w;

}else if(dp[u][1]<dp[v][0]+w){

dp[u][1]=dp[v][0]+w;

}

}

}

void dfs2(int u,int fa){

int len=g[u].size();

for(int i=0;i<len;i++){

int v=g[u][i].fi;

int w=g[u][i].se;

if(v==fa) continue;

if(p[u]==v) dp[v][2]=max(dp[u][1],dp[u][2])+w;

else dp[v][2]=max(dp[u][0],dp[u][2])+w;

dfs2(v,u);

}

}

int PointsOnDiameter(int n, vector<int>& u, vector<int>& v) {

for(int i=0;i<=n+10;i++) g[i].clear();

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

int ui=u[i],vi=v[i];

g[ui].push\_back(make\_pair(vi,1));

g[vi].push\_back(make\_pair(ui,1));

}

memset(dp,0,sizeof(dp));

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

dfs1(1,-1);

dfs2(1,-1);

int mx=0;

for(int i=1;i<=n;i++) mx=max(mx,dp[i][0]+dp[i][2]);

int ans=0;

//for(int i=1;i<=n;i++) cout<<i<<" "<<dp[i][0]<<" "<<dp[i][2]<<'\n';

//cout<<mx<<'\n';

for(int i=1;i<=n;i++) if(dp[i][0]+dp[i][2]+dp[i][1]-min(min(dp[i][0],dp[i][1]),dp[i][2])==mx) ans++;

return ans;

}

};

signed main(){

vector<int> s;s.push\_back(1),s.push\_back(2);

vector<int> t;t.push\_back(2),t.push\_back(3);

cout<<T.PointsOnDiameter(3,s,t);

return 0;

}

1. 寻找最长链和次长链

int dp[N];//以u为根节点的子树的最长边

int dfs(int u,int fa){

int sum=0,dis1=0,dis2=0;

dp[u]=a[u];

for(int i=0;i<g[u].size();i++){

int v=g[u][i].v;

int w=g[u][i].w;

if(v==fa) continue;

dfs(v,u);

int d=dp[v]-w;

if(d>dis1){

swap(dis1,dis2);

dis1=d;

}else if(d>dis2){

dis2=d;

}

}

dp[u]+=dis1;

ans=max(dis1+dis2+a[u],ans);

ans=max(ans,dp[u]);

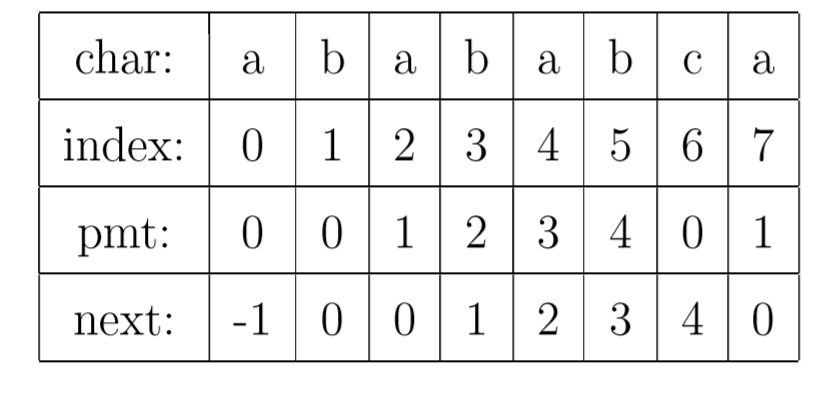
}

字符串

KMP

长度为len字符串的最短循环节长度为len-Next[len]

Next[i]:表示i-1为结尾的字符串的最长公共前后缀（不算当前这位（本身））



模式串T在主串S中首次出现的位置和模式串T在主串S中出现的次数的模板

/\*

pku3461(Oulipo), hdu1711(Number Sequence)

这个模板 字符串是从0开始的

Next数组是从1开始的

\*/

#include <iostream>

#include <cstring>

using namespace std;

const int N = 1000002;

int next[N];

char S[N], T[N];

int slen, tlen;

void getNext()

{

int j, k;

j = 0; k = -1; next[0] = -1;

while(j < tlen)

if(k == -1 || T[j] == T[k])

next[++j] = ++k;

else

k = next[k];

}

/\*

返回模式串T在主串S中首次出现的位置

返回的位置是从0开始的。

\*/

int KMP\_Index()

{

int i = 0, j = 0;

getNext();

while(i < slen && j < tlen)

{

if(j == -1 || S[i] == T[j])

{

i++; j++;

}

else

j = next[j];

}

if(j == tlen)

return i - tlen;

else

return -1;

}

/\*

返回模式串在主串S中出现的次数

\*/

int KMP\_Count()

{

int ans = 0;

int i, j = 0;

if(slen == 1 && tlen == 1)

{

if(S[0] == T[0])

return 1;

else

return 0;

}

getNext();

for(i = 0; i < slen; i++)

{

while(j > 0 && S[i] != T[j])

j = next[j];

if(S[i] == T[j])

j++;

if(j == tlen)

{

ans++;

j = next[j];

/\*

1)

j=next[j];

testcase:

aaaaaa aa

5

2)

j=j+1;

testcase:

aaaaaa aa

3

\*/

}

}

return ans;

}

int main()

{

int TT;

int i, cc;

cin>>TT;

while(TT--)

{

cin>>S>>T;

slen = strlen(S);

tlen = strlen(T);

cout<<"模式串T在主串S中首次出现的位置是: "<<KMP\_Index()<<endl;

cout<<"模式串T在主串S中出现的次数为: "<<KMP\_Count()<<endl;

}

return 0;

}

/\*

test case

4

aaaaaa a

abcd d

aabaa b

\*/

manacher算法模板

S中最长回文串长度

#include<bits/stdc++.h>

using namespace std;

#define int long long

const int N=36666666;

char s[N],str[N];

int p[N];

int slove(){

int len=strlen(str);

s[0]='$',s[1]='#';

int j=2;

for(int i=0;i<len;i++){

s[j++]=str[i];

s[j++]='#';

}

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

return j;

}

int manacher(){

int len=slove();

int ans=-N;

int id=0,mx=0;

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

if(i<mx) p[i]=min(p[2\*id-i],mx-i);

else p[i]=1;

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

if(mx<i+p[i]) mx=i+p[i],id=i;

ans=max(ans,p[i]-1);

}

return ans;

}

signed main(){

scanf("%s",str);

int ans=manacher();

cout<<ans<<'\n';

return 0;

}

Exkmp（Z函数）

对于一个字符串S，和一个下标i(i>1)

令Next[i]表示S和S[i...|S|]的最长公共前缀。

例如：S=aabcaabxaaz

Next[5]=3(因为下标[1-3] 和从5开始的下标[5-8]) 注意下标0还是1

给定2个字符串S和T(长度分别为n和m)

定义extend[i]为S[i...n]与T的最长公共前缀的长度

求出extend数组

S:a b a b a c a

T:a b a a c

extend:3 0 4 0 1 0 1

ababaca

abac

#include<bits/stdc++.h>

using namespace std;

#define int long long

const int maxn=100010; //字符串长度最大值

int nt[maxn],ex[maxn]; //ex数组即为extend数组

///预处理计算next数组

void GETNEXT(char \*str)

{

int i=0,j,po,len=strlen(str);

nt[0]=len;///用自己作为后缀与自己匹配

while(str[i]==str[i+1]&&i+1<len) i++;///暴力求next[1]

nt[1]=i;

po=1;///从此点出发next数组延伸位置最远

for(i=2;i<len;i++)

{

if(nt[i-po]< nt[po]+po-i )///第一种情况，可以直接得到next[i]的值

nt[i]=nt[i-po];

else///第二种情况，要继续匹配才能得到next[i]的值

{

j=nt[po]+po-i;

if(j<0)j=0; ///小于0表示没有已知相同部分，重新开始匹配

while(i+j<len&&str[j]==str[j+i])

j++;

nt[i]=j;

po=i;///更新po的位置

}

}

}

///计算extend数组

void EXKMP(char \*s1,char \*s2)

{

int i=0,j,po,len=strlen(s1),l2=strlen(s2);

GETNEXT(s2);

while(s1[i]==s2[i]&&i<l2&&i<len)

i++;

ex[0]=i;

po=0;

for(i=1;i<len;i++)

{

if(nt[i-po]<ex[po]+po-i)

ex[i]=nt[i-po];

else

{

j=ex[po]+po-i;

if(j<0)j=0;

while(i+j<len&&j<l2&&s1[j+i]==s2[j])

j++;

ex[i]=j;

po=i;

}

}

}

char S[maxn],T[maxn];

signed main(){

cin>>S>>T;

GETNEXT(S);

int Slen=strlen(S);

for(int i=0;i<Slen;i++) cout<<nt[i]<<" ";cout<<endl;

EXKMP(S,T);

for(int i=0;i<Slen;i++) cout<<ex[i]<<" ";cout<<endl;

return 0;

}

/\*

ababaca

abac

\*/

**数学部分**

**素数筛**

#define N 6666666

int su[N],cnt;

int n;

bool isprime[N];

void prime(){

cnt=1;

memset(isprime,1,sizeof(isprime));//初始化认为所有数都为素数

isprime[0]=isprime[1]=0;//0和1不是素数

for(int i=2;i<=N;i++){

if(isprime[i])

su[cnt++]=i;//保存素数i

for(int j=1;j<cnt&&su[j]\*i<N;j++){

isprime[su[j]\*i]=0;//筛掉小于等于i的素数和i的积构成的合数

}

}

}

**C(m,n)预处理版本**

using namespace std;

#define int long long

const int N = 5e4+10 , M = 1e5+100;

const int mod=998244353;

int h,w,a,b;

int fac[N],fnv[N];

int ans;

int quickmod(int x,int y){

int ans=1;

for(;y;y>>=1){

if(y&1) ans=ans\*x%mod;

x=x\*x%mod;

}

return ans;

}

void build()

{

fac[0]=1;

for(int i=1;i<N;i++) fac[i]=1ll\*fac[i-1]\*i%mod;

fnv[N-1]=quickmod(fac[N-1],mod-2);

for(int i=N-2;i>=0;i--) fnv[i]=1ll\*fnv[i+1]\*(i+1)%mod;

}

int C(int x,int y)

{

if(x<y) return 0;

return 1ll\*fac[x]\*fnv[y]%mod\*fnv[x-y]%mod;

}

二维前缀和

#include<iostream>

#include<cstring>

using namespace std;

int dp[2000][2000],map[2000][2000];

int main(){

int m,n,k;//所给的矩阵是n\*m的,有k组查询

cin >>n>>m>>k;

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

for(int j=1;j<=m;j++)

cin >>map[i][j];

memset(dp,0,sizeof(dp));

for(int i=1;i<=n;i++)//预处理一波

for(int j=1;j<=m;j++)

dp[i][j]=dp[i-1][j]+dp[i][j-1]-dp[i-1][j-1]+map[i][j];

for(int i=1;i<=k;i++){//接受查询

int x1,x2,y1,y2;

cin >>x1>>y1>>x2>>y2;

cout <<(dp[x2][y2]+dp[x1-1][y1-1]-dp[x1-1][y2]-dp[x2][y1-1])<<endl;//O（1）查询

}

return 0;

}

单调栈

//左边第一个小于a[i],右边第一个大于a[i]

using namespace std;

#define int long long

#define N 2666666

stack<int> s;

int l[N],r[N];

int arr[N],sum[N];

signed main(){

int n;

scanf("%lld",&n);

sum[0]=0;

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

scanf("%lld",&arr[i]);

sum[i]=sum[i-1]+1;

}

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

while(!s.empty()&&arr[s.top()]>=arr[i]) s.pop();

if(s.empty()) l[i]=1;

else l[i]=s.top()+1;

s.push(i);

}

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

for(int i=n;i>=1;i--){

while(!s.empty()&&arr[s.top()]>=arr[i]) s.pop();

if(s.empty()) r[i]=n;

else r[i]=s.top()-1;

s.push(i);

}

for(int i=1;i<=n;i++) cout<<l[i]<<" ";cout<<endl;

for(int i=1;i<=n;i++) cout<<r[i]<<" ";cout<<endl;

return 0;

}

三分

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*三分求最大值

#define eps 1e-9

void slove(){

scanf("%d",&n);

double l,r;

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

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

scanf("%lf",&arr[i]);

}

while(l+1e-6<r){

double lmid=l+(r-l)/3.0,rmid=r-(r-l)/3.0;

double t1=check(lmid);

double t2=check(rmid);

if(t1<t2){

l=lmid;

}else{

r=rmid;

}

}

printf("%.5lf",l);

return ;

}

signed main(){

slove();

return 0;

}

1.整数三分

int calc(int M)

{

}

int solve(int L, int R)

{

int M, RM;

while (L + 1 < R)

{

M = (L + R) / 2;

RM = (M + R) / 2;

if (calc(M) < calc(RM)) //计算最小值,若为最大值则>

R = RM;

else

L = M;

}

return L;

}

For(int i=l;i<=r;i++) ans=m(ans,cal)

2.小数三分

注意一点，double的循环条件中最好使用以下的表示（L + EPS < R），如果使用dcmp（R - L）> 0判断也可以，但是使用dcmp函数时候一定不能写等号，因为如果两个double数在EPS范围内相等的还继续的话会导致死循环

const double EPS = 1e-10;

double calc(double n)

{

return;

}

double solve(double L, double R)

{

double M, RM;

while (L + EPS < R)

{

M = (L + R) / 2;

RM = (M + R) / 2;

if (calc(M) < calc(RM)) //计算最小值

R = RM;

else

L = M;

}

return L;

}

3.限定次数的三分

const double EPS = 1e-10;

double calc(double n)

{

return;

}

double solve(double L, double R)

{

double M, RM;

for (int i = 1; i <= 300; i++)

{

M = (L + R) / 2;

RM = (M + R) / 2;

if (calc(M) < calc(RM)) R = RM;

else L = M;

}

return L;

}

整数分块

1.基础问题（可以求解[1,x]的因数个数的和）

_0V~0`N8LVQF%F%9X42MQ0D

ll ans=0;

for(ll l=1,r;l<=n;l=r+1)

{

r=n/(n/l);

ans+=(r-l+1)\*(n/l);

}

2.变形

R]DG_MM11BCD)0BTPE)0@SN

ll ans=0;

for(ll l=1,r;l<=n;l=r+1)

{

if((w-1)/l)r=(w-1)/((w-1)/l);

else r=n;

r=min(r,n);

ans+=(r-l+1)\*((w-1)/l+1);

}