List操作篇

Link初始化：

int num[10]={0,1,2,3,4,5,6,7,8,9};

list<int> link(&num[0],&num[9]+1);

print(link);

link 前插 前删 后插 后删

push\_front、pop\_front。。。。

link 删除

cout<<"删除第一个元素:"<<endl;

link.erase(link.begin());

cout<<"删除值为6的所有元素"<<endl;

link.remove(6);

link 排序

cout<<"sort之后的数据:"<<endl;

link.sort();

print(link);

link 反转

link.reverse

link 去重

link.unique();

link 拼接

link1.splice(link1.end(),link2,link2.begin(),link2.end());//把link2.begin()到link2.end()的数据拼到link1.end()的后面

link 融合 融合两个排序的list，融合的list依然是排序的

link1.merge(link2);

优化KMP

Int next[MAX];

String p,b;

Void get\_next(){

Next[0]=-1;

Int i=0,k=-1;

While(i<b.length()){

If(k==-1||b[k]==b[i]){

K++,j++;

If(b[k]==b[i])

Next[i]=next[k];

Else

Next[i]=k;

}

Else

K=next[k];

}

}

Void kmp(){

Int i=0,j=-1;

While(i<p.length()){

If(j==-1||p[i]==b[j]){

I++,j++;

}

Else

J=next[j];

If(j==b.length()) //计算第一次出现下标

Break;

If(j==b.length()){

Num++;

J=next[j];//可重复

J=0;//不可重复

}

}

Return i-j; //第一次出现位子

}

拓展KMP

#include<iostream>

#include<string>

#include<cstring>

#include<cstdio>

using namespace std;

const int K=100005;

int nt[K],extand[K];

char S[K],T[K];

void Getnext(char \*T,int \*next)

{

int len=strlen(T),a=0;

next[0]=len;

while(a<len-1 && T[a]==T[a+1]) a++;

next[1]=a;

a=1;

for(int k=2; k<len; k++)

{

int p=a+next[a]-1,L=next[k-a];

if( (k-1)+L >= p)

{

int j = (p-k+1)>0 ? (p-k+1) : 0;

while(k+j<len && T[k+j]==T[j]) j++;

next[k]=j;

a=k;

}

else

next[k]=L;

}

}

void GetExtand(char \*S,char \*T,int \*next)

{

Getnext(T,next);

int slen=strlen(S),tlen=strlen(T),a=0;

int MinLen = slen < tlen ? slen : tlen;

while(a<MinLen && S[a]==T[a]) a++;

extand[0]=a;

a=0;

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

{

int p=a+extand[a]-1, L=next[k-a];

if( (k-1)+L >= p)

{

int j= (p-k+1) > 0 ? (p-k+1) : 0;

while(k+j<slen && j<tlen && S[k+j]==T[j]) j++;

extand[k]=j;

a=k;

}

else

extand[k]=L;

}

}

int main(void)

{

while(scanf("%s%s",S,T)==2)

{

GetExtand(S,T,nt);

for(int i=0; i<strlen(T); i++)

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

puts("");

for(int i=0; i<strlen(S); i++)

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

puts("");

}

return 0;

}

Sunday算法

int MAXSIZE = 256;

　　int moveLength[MAXSIZE];

　　void getMoveLength(const string &T) {

　　　　int tLen = T.size();

　　　　for(int i = 0; i < MAXSIZE; ++ i)

　　　　　　moveLength[i] = tLen + 1;

　　　　for(int i = 0; T[i]; ++ i)

　　　　　　moveLength[T[i]] = tLen - i;

　　}

　　int Sunday(const string& S, const string &T) {

　　　　getMoveLength(T);

　　　　int tLen = T.size();

　　　　int sLen = S.size();

　　　　int i = 0;

　　　　while(i < sLen) {

　　　　　　int j = 0;

　　　　　　for( ; j < tLen && i + j < sLen && S[i + j] == T[j]; ++ j) ;

　　　　　　if(j >= tLen) return i;

　　　　　　if(i + tLen > sLen)

　　　　　　　　return -1;

　　　　　　i += moveLength[S[i + tLen]];

　　　　}

　　　　return -1;

}

BM算法

void get\_bmB(const string& T, int bmB[]) {

　　　　int tLen = T.size();

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

　　　　　　bmB[i] = tlen;

　　　　}

　　　　for(int i = 0; i < tLen; ++i) {

　　　　　　bmB[T[i]] = tLen - 1 - i;

　　　　}

　　}

void get\_bmGLength(const string& T, int bmGLength[]) {

　　　　int tLen = T.size();

　　　　bmGLength[tLen - 1] = tLen;

　　　　int cur = tLen - 1;

　　　　int pre;

　　　　for(int i = tLen - 2; i >= 0; -- i) {

　　　　　　if(i > cur) {

　　　　　　　　bmGLength[i] = bmGLength[i + (tLen - 1 - pre)];

　　　　　　　　continue;

　　　　　　}

　　　　　　cur = min(cur, i);

　　　　　　pre = i;

　　　　　　while(cur >= 0 && T[cur] == T[tLen - 1 - (pre - cur)])

　　　　　　　　-- cur;

　　　　　　bmGLenth[i] = pre - cur;

　　　　}

　　}

int bmGlength[N];

void get\_bmG(const string& T, int bmG[]) {

　　　　get\_bmGLength(T, bmGLength);

　　　　int tLen = T.size();

　　　　for(int i = 0; i < tLen; ++ i) {

　　　　　　bmG[i] = tLen;

　　　　}

　　　　int j = 0;

　　　　for(int i = tLen - 2; i >= 0; -- i) {

　　　　　　if(bmGLength[i] == i + 1) {

　　　　　　　　for(; j < tLen - 1 - i; ++ j) {

　　　　　　　　　　if(bmG[j] == tLen)

　　　　　　　　　　　　bmG[j] = tLen - 1 - i;

　　　　　　　　}

　　　　　　}

　　　　}

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

　　　　　　bmG[tLen - 1 - bmGLenth[i]] = tLen - 1 - i;

　　　　}

　　}

　　int BM(const string& S, const string& T, int bmG[], int bmB[]) {

　　　　get\_bmB(T, bmB);

　　　　get\_bmG(T, bmG);

　　　　int sLen = S.size() - T.size();

　　　　int tLen = T.size();

　　　　int i = 0;

　　　　while(i <= sLen) {

　　　　　　int j = tLen - 1;

　　　　　　for(; j > -1 && S[i + j] == T[j]; --j) ;

　　　　　　if(j == -1)

　　　　　　　　return i;

　　　　　　i += max(bmG[j], bmB[S[i + j]] - (tLen - 1 - j));

　　　　}

　　　　return -1;

　　}

扩展KMP

给出一个长为N的字符串S，再给出一个长为M的字符串T   
求S的所有后缀中和T的最长公共前缀

#include <iostream>

#include <cstdio>

#include <cstring>

#include <string>

#include <algorithm>

#define mt(a) memset(a,0,sizeof(a))

using namespace std;

typedef long long ll;

const ll mod=1e9+7;

ll extend[1000001];

ll Next[1000001];

ll min(ll x,ll y)

{

if(x>y) return y;

return x;

}

void getNext(string t)

{

mt(Next);

ll len=t.length();

Next[0]=len;

ll a,p;

a=1;

while( a<len && t[a]==t[a-1]) a++; // 求出长度为1的时候 解为多少

Next[1]=a-1;

a=1;

for(ll i=2;i<len;i++) // 后续的按照算法来就好

{

p=a+Next[a]-1;

if((i-1)+Next[i-a] < p ) Next[i]=Next[i-a];// 第一种情况 没有超过等于的部分

else // 超过的话就不好直接用next的定义 需要后续的遍历

{

ll j = (p - i + 1) > 0 ? (p - i + 1) : 0;

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

Next[i]=j;

a=i;

}

}

}

void exkmp(string s,string t) // s->extend t->next

{

getNext(t);

ll a,p;//

ll slen=s.length();

ll tlen=t.length();

a=p=0;

ll len=min(s.length(),t.length());

while(p<len && t[p]==s[p]) p++; // after

extend[0]=p;

for(ll i=1;i<slen;i++)

{

p=a+extend[a]-1; // update

if( (i-1)+Next[i-a] < p) extend[i]=Next[i-a];

else

{

ll j = (p - i + 1) > 0 ? (p - i + 1) : 0;

while( j < tlen && i+j < slen && s[i + j] == t[j]) j++;

extend[i]=j;

a=i;

}

}

}

// 核心 一个起始位置a 一个最远匹配位置p 然后Next 和 extend数组

int main()

{

string s,t;// s->exkmp t->Next

int Case;

scanf("%d",&Case);

while(Case--)

{

cin>>s>>t;

exkmp(s,t);

}

return 0;

}

String操作

1. 初始化

string s;//默认初始化，一个空字符串

string s1("ssss");//s1是字面值“ssss”的副本

string s2(s1);//s2是s1的副本

string s3=s2;//s3是s2的副本

string s4(10,'c');//把s4初始化

string s5="hiya";//拷贝初始化

string s6=string(10,'c');//拷贝初始化，生成一个初始化好的对象，拷贝给s6

char cs[]="12345";

string s7(cs,3);//复制字符串cs的前3个字符到s当中

//string s(s2,pos2)

string s8="asac";

string s9(s8,2);//从s2的第二个字符开始拷贝，不能超过s2的size

//string s(s2,pos2,len2)

string s10="qweqweqweq";

string s11(s10,3,4);//s4是s3从下标3开始4个字符的拷贝，超过s3.size出现未定义

1. substr子串操作：

string s="abcdefg";

//s.substr(pos1,n)返回字符串位置为pos1后面的n个字符组成的串

string s2=s.substr(1,5);//bcdef

//s.substr(pos)//得到一个pos到结尾的串

string s3=s.substr(4);//efg

1. insert插入操作

string str="to be question";

string str2="the ";

string str3="or not to be";

string::iterator it;

//s.insert(pos,str)//在s的pos位置插入str

str.insert(6,str2); // to be the question

//s.insert(pos,str,a,n)在s的pos位置插入str中插入位置a到后面的n个字符

str.insert(6,str3,3,4); // to be not the question

//s.insert(pos,cstr,n)//在pos位置插入cstr字符串从开始到后面的n个字符str.insert(10,"that is cool",8); // to be not that is the question

//s.insert(pos,cstr)在s的pos位置插入cstr

str.insert(10,"to be "); // to be not to be that is the question

//s.insert(pos,n,ch)在s.pos位置上面插入n个ch

str.insert(15,1,':'); // to be not to be: that is the question

//s.insert(s.it,ch)在s的it指向位置前面插入一个字符ch，返回新插入的位置的迭代器

it = str.insert(str.begin()+5,','); // to be, not to be: that is the question

//s.insert(s.it,n,ch)//在s的it所指向位置的前面插入n个ch

str.insert (str.end(),3,'.'); // to be, not to be: that is the question...

//s.insert(it,str.ita,str.itb)在it所指向的位置的前面插入[ita,itb)的字符串

str.insert (it+2,str3.begin(),str3.begin()+3); // to be, or not to be: that is the question...

1. erase删除操作

std::string str ("This is an example sentence.");

// "This is an example sentence."

str.erase (10,8);

//直接指定删除的字符串位置第十个后面的8个字符

// "This is an sentence."

str.erase (str.begin()+9);// ^

//删除迭代器指向的字符

// "This is a sentence."

str.erase (str.begin()+5, str.end()-9);

//删除迭代器范围的字符

// "This sentence."

1. replace替换操作

std::string base="this is a test string.";

std::string str2="n example";

std::string str3="sample phrase";

std::string str4="useful.";

// Using positions: 0123456789\*123456789\*12345

std::string str=base; // "this is a test string."

//第9个字符以及后面的4个字符被str2代替

str.replace(9,5,str2); // "this is an example string." (1)

//第19个字符串以及后面的5个字符用str的第7个字符以及后面的5个字符代替

str.replace(19,6,str3,7,6); // "this is an example phrase." (2)

//第8个字符以及后面的9个字符用字符串参数代替

str.replace(8,10,"just a"); // "this is just a phrase." (3)

//第8个字符以及后面的5个字符用字符串参数的前7个字符替换

str.replace(8,6,"a shorty",7); // "this is a short phrase." (4)

//第22以及后面的0个字符用3个叹号替换

str.replace(22,1,3,'!'); // "this is a short phrase!!!" (5)

//迭代器的原理同上

str.replace(str.begin(),str.end()-3,str3); // "sample phrase!!!" (1)

str.replace(str.begin(),str.begin()+6,"replace"); // "replace phrase!!!" (3)

str.replace(str.begin()+8,str.begin()+14,"is coolness",7); // "replace is cool!!!" (4)

str.replace(str.begin()+12,str.end()-4,4,'o'); // "replace is cooool!!!" (5)

str.replace(str.begin()+11,str.end(),str4.begin(),str4.end());// "replace is useful." (6)

1. assign赋值操作

std::string str;

std::string base="The quick brown fox jumps over a lazy dog.";

//直接把base赋值给str

str.assign(base);

//把base第10个字符以及后面的8个字符赋给str

str.assign(base,10,9); // "brown fox"

//把参数中的0到6个字符串赋给str

str.assign("pangrams are cool",7); // "pangram"

//直接使用参数赋值

str.assign("c-string"); // "c-string"

//给str赋值10个'\*'字符

str.assign(10,'\*'); // "\*\*\*\*\*\*\*\*\*\*"

//赋值是10个'-'

str.assign<int>(10,0x2D); // "----------"

//指定base迭代器范围的字符串

str.assign(base.begin()+16,base.end()-12); // "fox jumps over"

1. find查找函数

std::string str ("There are two needles in this haystack with needles.");

std::string str2 ("needle");

// different member versions of find in the same order as above:

//在str当中查找第一个出现的needle，找到则返回出现的位置，否则返回结尾

std::size\_t found = str.find(str2);

if (found!=std::string::npos)

std::cout << "first 'needle' found at: " << found << '\n';

//在str当中，从第found+1的位置开始查找参数字符串的前6个字符

found=str.find("needles are small",found+1,6);

if (found!=std::string::npos)

std::cout << "second 'needle' found at: " << found << '\n';

//在str当中查找参数中的字符串

found=str.find("haystack");

if (found!=std::string::npos)

std::cout << "'haystack' also found at: " << found << '\n';

//查找一个字符

found=str.find('.');

if (found!=std::string::npos)

std::cout << "Period found at: " << found << '\n';

//组合使用，把str2用参数表中的字符串代替

// let's replace the first needle:

str.replace(str.find(str2),str2.length(),"preposition");

std::cout << str << '\n';

RMQ

void ST(int n){

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

MIN[i][0]=MAX[i][0]=a[i];

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

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

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

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

}

}

}

int RMQ(int l,int r,int c){

if(!c) return min(MIN[l][k],MIN[r-(1<<k)+1][k]);

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

}

前缀表达式求值

#include<bits/stdc++.h>

using namespace std;

const int inf=1e9+7;

char s[2000];

int i;

int eval(){

int x = 0;

while( s[ i ] == ' ') {

i++;

}

if( s[ i ] == '\*' ) {

i++;

return eval() \* eval();

}

if( s[ i ] == '+' ) {

i++;

return eval() + eval();

}

while( s[ i ] <= '9' && s[ i ] >= '0' ) {

x = x \* 10 + s[ i++ ] - '0';

}

return x;

}

int main(){

while( gets(s) ){

i = 0;

int ans = eval();

cout<< ans << endl;

}

return 0;

}

SPFA

#include<bits/stdc++.h>

using namespace std;

const int inf=1e9+7;

struct Edge{

int u,to,next,w;

}edge[40005];

int head[105];

int dis[105];

int n,m;

void spfa() {

queue < int > que;

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

dis[ i ] = inf;

}

dis[ 1 ] = 0;

que.push( 1 );

while( que.size() ) {

int x = que.front();

que.pop();

x = head[ x ];

while( x != -1 ) {

int u = edge[ x ].u;

int v = edge[ x ].to;

if( dis[ v ] > dis[ u ] + edge[ x ].w ) {

dis[ v ] = dis[ u ] + edge[ x ].w;

que.push( v );

}

x = edge[ x ].next;

}

}

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

}

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

edge[ i ].u = u;

edge[ i ].to = v;

edge[ i ].w = w;

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

head[ u ] = i;

}

int main(){

int u,v,w;

while( scanf("%d%d", &n, &m) && ( n || m )){

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

for(int i=1;i<=2\*m;i+=2){

scanf("%d%d%d",&u,&v,&w);

add( u, v, w, i);

add( v, u, w, i+1);

}

spfa();

}

return 0;

}

SPFA判负环

#include <cstdio>

#include <cmath>

#include <cstdlib>

#include <cstring>

#include <string>

#include <algorithm>

#include <iomanip>

#include <iostream>

using namespace std ;

const int maxn = 200011,maxm = 200011,inf = 1e9 ;

struct node{

int to,val,pre ;

}e[2\*maxm];

int T,n,m,x,y,val,cnt ;

int head[maxn],dist[maxn] ;

bool flag ;

bool vis[maxn] ;

inline void addedge(int x,int y,int v)

{

e[++cnt] = (node){ y,v,head[x] } ;

head[ x ] = cnt ;

}

inline int read()

{

char ch = getchar() ;

int x = 0 , f = 1 ;

while(ch<'0'||ch>'9') { if(ch=='-') f = -1 ; ch = getchar() ; }

while(ch>='0'&&ch<='9') { x = x\*10+ch-48 ; ch = getchar() ; }

return x\*f ;

}

inline void SPFA(int u)

{

int v ;

vis[ u ] = 1 ;

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

{

v = e[ i ].to ;

if( dist[ u ] + e[ i ].val < dist[ v ] )

{

dist[ v ] = dist[ u ] + e[ i ].val ;

if(vis[ v ]||flag)

{

flag = 1 ;

break ;

}

SPFA( v ) ;

}

}

vis[ u ] = 0 ;

}

int main()

{

T = read() ;

while(T--)

{

flag = 0 ;

cnt = 0 ;

n = read() ; m = read() ;

for(int i=1;i<=n;i++) dist[ i ] = 0,vis[ i ] = 0,head[ i ] = 0 ; //0

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

{

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

addedge(x,y,val) ;

if(val>=0) addedge(y,x,val) ;

}

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

{

SPFA( i ) ;

if(flag) break ;

}

if(flag)

printf("YE5\n") ;

else

printf("N0\n") ;

}

return 0 ;

}

Prim算法

nt mp[ 50 ][ 50 ];

int n;

int vis[ 50 ];

int dis[ 50 ];

void prim() {

int ans = 0;

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

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

dis[ i ] = MAX ;

// vis[ 1 ] = 1;

dis[ 1 ] = 0;

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

int k = 0, mmin = MAX;

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

if(mmin > dis[ i ] && !vis[ i ]) {

k = i;

mmin = dis[ i ];

}

}

// cout<< k << ' ' << mmin << endl;

if( k == 0 ) break;

ans += mmin ;

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

if(dis[ i ] > mp[ k ][ i ] ) {

dis[ i ] = mp[ k ][ i ];

}

}

vis[ k ] = 1;

}

cout<< ans <<endl;

}

树状数组

#include <cstdio>

#include <cstring>

#define maxn 50047

int c[maxn], a[maxn];

int n,t;

int Lowbit(int x) // 2^k{

return x&(-x);

}

void update(int i, int x)//i点增量为x{

while(i <= n){

c[i] += x;

i += Lowbit(i);

}

}

int sum(int x)//区间求和 [1,x]{

int sum=0;

while(x>0){

sum+=c[x];

x-=Lowbit(x);

}

return sum;

}

int Getsum(int x1,int x2) //求任意区间和{

return sum(x2) - sum(x1-1);

}

int main() {

int i , j;

scanf("%d",&t);

int count = 0;

while(t--) {

count++;

memset(a,0,sizeof(a));

memset(c,0,sizeof(c));

scanf("%d",&n);

for(i = 1; i <= n; i++) //i须从1开始 {

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

update(i,a[i]); //初始的人数

}

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

char oper[11];

while(scanf("%s",oper)==1) {

if(strcmp(oper,"End")==0)

break;

scanf("%d%d",&i,&j);

if(strcmp(oper,"Query")==0) {

printf("%d\n",Getsum(i,j));

}

if(strcmp(oper,"Add")==0) //表示第i个营地增加j个人{

a[i] += j;

update(i,j);

}

if(strcmp(oper,"Sub")==0) //表示第i个营地减少j个人{

a[i] -= j;

update(i,-j);

}

}

}

return 0;

}

最小&&次小生成树

#define Del(a,b) memset(a,b,sizeof(a))

const int inf = 0x3f3f3f3f;

const int N = 550;

int mp[N][N];

bool vis[N],used[N][N];

int pre[N],ma[N][N],cost[N];

int n,m;

int Prim(int x){

int ans = 0;

Del(ma,0);

Del(used,false);

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

cost[i] = mp[x][i];

pre[i] = 1;

vis[i] = false;

}

vis[x] = true;

pre[x] = -1;

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

int minc = inf;

int p = -1;

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

if(vis[j]==false && minc>cost[j])

{

minc = cost[j];

p = j;

}

}

if(p==-1)

return -1;

ans+=minc;

vis[p] = true;

int tmp = pre[p];

used[p][tmp] = used[tmp][p] = true; //MST上的边

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

if(vis[j])

ma[j][p] = ma[p][j] = max(ma[j][tmp],cost[p]);

if(vis[j]==false && cost[j]>mp[p][j])

{

cost[j] = mp[p][j];

pre[j] = p;

}

}

}

return ans;

}

int Next\_Prim(int x){

int ans = inf;

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

{

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

{

if(mp[i][j]!=inf && !used[i][j]) //枚举不在MST上的边替换

ans = min(ans,x+mp[i][j]-ma[i][j]);

}

}

return ans;

}

int main(){

int T;

scanf("%d",&T);

while(T--)

{

memset(mp,inf,sizeof(mp));

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

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

{

int x,y,z;

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

mp[x][y] = mp[y][x] = z;

}

int ans = Prim(1);

int next = Next\_Prim(ans);

if(ans!=next)

puts("No");

else

puts("Yes");

}

return 0;

}

带权并查集（hdu3038 区间和判错）

给出区间[1,n],下面有m组数据，l r v区间[l,r]之和为v,每输入一组数据，判断此组条件是否与前面冲突 ,最后输出与前面冲突的数据的个数.比如 [1 5]区间和为100 然后后面给出区间[1,2]的和为 200 那肯定就是有问题的了

对于A~B之间的和是S，其实可以理解成B比A-1大S，这样可以转换为带权并查集的问题

const ll MAX=2e5+5;

int fa[MAX];

int sum[MAX];

int n,m,ans;

int Find(int x){

if(fa[x]==-1) return x;

int xx=Find(fa[x]);

sum[x]+=sum[fa[x]];//更新当前节点到根节点的权值？

return fa[x]=xx;

}

int main()

{

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

while(cin>>n>>m){

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

memset(sum,0,sizeof(sum));

ans=0;

int a,b,c;

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

cin>>a>>b>>c;

a--;//这样使得(1,5)和(6,19)能合并

int u=Find(a);

int v=Find(b);

if(u!=v){

fa[u]=v;

sum[u]=c+sum[b]-sum[a];

}

else if(sum[a]-sum[b]!=c)

ans++;

}

cout<<ans<<endl;

}

return 0;

}

二分图匹配

const ll MAX=1005;

int mp[MAX][MAX];

int n;

int linker[MAX];

bool used[MAX];

bool dfs(int a){

for(int i=0;i<n;i++)if(mp[a][i]&&!used[i]){

used[i]=true;

if(linker[i]==-1||dfs(linker[i])){

linker[i]=a;

return true;

}

}

return false;

}

int hungary(){ //返回最大匹配数

int res=0;

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

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

memset(used,0,sizeof(used));

if(dfs(i)) res++;

}

return res;

}

int main()

{

int i,j,a,b,num;

while(scanf("%d",&n)!=EOF){

memset(mp,0,sizeof(mp));

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

scanf("%d: (%d)",&a,&num);

for( j=0;j<num;j++){

scanf("%d",&b);

mp[a][b]=1;

}

}

int cnt=hungary();

printf("%d\n",n-cnt/2); //n-cnt==最大独立集

}

return 0;

}

Polay定理

给你n种颜色和m个小球，问你有多少种不同的方案

int n, m;

int gcd(int a, int b)

{

b = b % a;

while (b)

{

a = a % b;

swap(a, b);

}

return a;

}

int main()

{

while (scanf("%d%d", &n, &m), n | m)

{

int ans = 0;

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

ans += pow(n, gcd(i, m));

if (m & 1)

ans += m \* pow(n, m / 2 + 1);

else

ans += m / 2 \* pow(n, m / 2) + m / 2 \* pow(n, m / 2 + 1);

ans /= m \* 2;

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

}

return 0;

}

后缀数组

#include <cstdio>

#include <cstring>

#include <algorithm>

#define N 1000

using namespace std;

int n,m=200,c[N],x[N],y[N],sa[N]; //sa[i] 以i为头的后缀排第几

char s[N];

void build\_sa()

{

for (int i=0;i<m;i++) c[i]=0;

for (int i=0;i<n;i++) c[x[i]=s[i]]++;

for (int i=1;i<m;i++) c[i]+=c[i-1];

for (int i=n-1;i>=0;i--)sa[--c[x[i]]]=i;

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

int p=0;

for (int i=n-k;i<n;i++) y[p++]=i;

for (int i=0;i<n;i++) if (sa[i]>=k) y[p++]=sa[i]-k;

for (int i=0;i<m;i++) c[i]=0;

for (int i=0;i<n;i++) c[x[i]]++;

for (int i=1;i<m;i++) c[i]+=c[i-1];

for (int i=n-1;i>=0;i--)sa[--c[x[y[i]]]]=y[i];

swap(x,y);

p=1; x[sa[0]]=0;

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

x[sa[i]]=y[sa[i-1]]==y[sa[i]]&&((sa[i-1]+k>=n?-1:y[sa[i-1]+k])==(sa[i]+k>=n?-1:y[sa[i]+k]))?p-1:p++;

if (p>n) break;

m=p;

}

}

int rank[N],height[N]; //rank[i] 排第i的是从哪开始的

void build\_height() //height[i] 相邻sa的最长公共前缀

{

for (int i=0;i<n;++i) rank[sa[i]]=i;

int k=0;height[0]=0;

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

if (!rank[i]) 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;

}

}

main()

{

scanf("%s",s);

n=strlen(s);

build\_sa();

build\_height();

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

printf("%d%c",sa[i]+1," \n"[i==n-1]);

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

printf("%d%c",rank[i]+1," \n"[i==n-1]);

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

printf("%d%c",height[i]+1," \n"[i==n-1]);

}