**2018暑期集训**

目录

[一、基础数据结构&语言 5](#_Toc520789031)

[补题 5](#_Toc520789032)

[铜牌题-前置：单调队列/单调栈/滑动窗口最大值问题 5](#_Toc520789033)

[<cstring> 5](#_Toc520789034)

[new&delete 6](#_Toc520789035)

[创建并释放一维数组 6](#_Toc520789036)

[创建并释放二维数组 6](#_Toc520789037)

[STL 7](#_Toc520789038)

[map 7](#_Toc520789039)

[next\_permutation 7](#_Toc520789040)

[优先队列（priority\_queue） 8](#_Toc520789041)

[JAVA 8](#_Toc520789042)

[BigDecimal 8](#_Toc520789043)

[可持久化平衡树（C++ Pb\_ds库 Rope） 8](#_Toc520789044)

[√-牛客网2018暑期多校第3场-C（可持久化平衡树） 13](#_Toc520789045)

[枚举贪心排序构造 18](#_Toc520789046)

[√-牛客网2018暑期多校第4场-F-Beautiful Garden-签到题 18](#_Toc520789047)

[模拟 20](#_Toc520789048)

[√-HDU-6330-Visual Cube-签到题 20](#_Toc520789049)

[二、搜索 23](#_Toc520789050)

[三、动态规划 23](#_Toc520789051)

[补题 23](#_Toc520789052)

[铜牌题-前置：状压DP 23](#_Toc520789053)

[递推 23](#_Toc520789054)

[√-HDU-2050-折线分割平面 23](#_Toc520789055)

[基础动态规划 24](#_Toc520789056)

[最大连续子段和 24](#_Toc520789057)

[LIS（最长上升子序列） 24](#_Toc520789058)

[LCIS（最长上升公共子序列） 25](#_Toc520789059)

[背包问题 27](#_Toc520789060)

[0-1背包 27](#_Toc520789061)

[完全背包 28](#_Toc520789062)

[多重背包 29](#_Toc520789063)

[数位DP 29](#_Toc520789064)

[√-HDU-2089-不要62 29](#_Toc520789065)

[概率DP 31](#_Toc520789066)

[HDU-5863-2016多校10-铜牌 31](#_Toc520789067)

[√-牛客网-第1场-E-Removal-铜牌 32](#_Toc520789068)

[√-牛客网-第2场-D-Money-签到题 33](#_Toc520789069)

[四、图论 35](#_Toc520789070)

[最短路 35](#_Toc520789071)

[SPFA 35](#_Toc520789072)

[Floyd 38](#_Toc520789073)

[负环判定 39](#_Toc520789074)

[√-POJ-3259-Wormholes 39](#_Toc520789075)

[网络流 41](#_Toc520789076)

[2-SAT 41](#_Toc520789077)

[图同构 41](#_Toc520789078)

[拓补排序 41](#_Toc520789079)

[最小生成树 41](#_Toc520789080)

[线段树优化建图 41](#_Toc520789081)

[五、字符串算法 41](#_Toc520789082)

[KMP 41](#_Toc520789083)

[√-HDU-1711-Number Sequence 41](#_Toc520789084)

[√-HDU-1686-Oulipo 44](#_Toc520789085)

[√-HDU-2087-剪花布条 46](#_Toc520789086)

[循环节问题 48](#_Toc520789087)

[√-HDU-2203-亲和串 53](#_Toc520789088)

[√-POJ-2752-Seek the Name,Seek the Fame 54](#_Toc520789089)

[扩展KMP 56](#_Toc520789090)

[字典树 56](#_Toc520789091)

[AC自动机 56](#_Toc520789092)

[后缀数组 56](#_Toc520789093)

[六、树状数组&线段树 56](#_Toc520789094)

[线段树：区间更新 56](#_Toc520789095)

[√-HDU-6315-多校2铜牌题 56](#_Toc520789096)

[线段树：区间合并 62](#_Toc520789097)

[√-POJ-3667-Hotel（线段树区间合并模板题） 62](#_Toc520789098)

[离散化&扫描线 65](#_Toc520789099)

[√-HDU-1542-Atlantis（矩形面积并） 65](#_Toc520789100)

[√-HDU-5862-2016多校10-铜牌 67](#_Toc520789101)

[二维线段树 70](#_Toc520789102)

[TLE-POJ-2155-Matrix 70](#_Toc520789103)

[√-HDU-5861-2016多校10-铜牌 70](#_Toc520789104)

[主席树 73](#_Toc520789105)

[七、数学 73](#_Toc520789106)

[补题 73](#_Toc520789107)

[银牌题-前置：指数循环节问题/扩展欧拉定理 73](#_Toc520789108)

[签到题-可做 78](#_Toc520789109)

[签到题-可做 78](#_Toc520789110)

[OEIS 78](#_Toc520789111)

[快速幂 78](#_Toc520789112)

[一般快速幂 78](#_Toc520789113)

[根据递推式构造系数矩阵 79](#_Toc520789114)

[矩阵快速幂 79](#_Toc520789115)

[数论 79](#_Toc520789116)

[逆元 79](#_Toc520789117)

[欧拉函数 79](#_Toc520789118)

[威尔逊定理 81](#_Toc520789119)

[幂取模(欧拉定理、费马小定理、扩展欧拉定理) 81](#_Toc520789120)

[不定方程 82](#_Toc520789121)

[组合数学 84](#_Toc520789122)

[卡特兰数 84](#_Toc520789123)

[Polya定理 84](#_Toc520789124)

[计算几何 86](#_Toc520789125)

[点到线段的最短距离——矢量法 86](#_Toc520789126)

[四点共面 89](#_Toc520789127)

[线段相交 90](#_Toc520789128)

[简单多边形与圆面积交 90](#_Toc520789129)

[其他题目 94](#_Toc520789130)

[综合题 95](#_Toc520789131)

[√-牛客网2018暑期多校第1场-A（OEIS打表&组合数递推&逆元） 95](#_Toc520789132)

[√-HDU-6304-2018多校1-铜牌题 98](#_Toc520789133)

[√-EOJ-110-2018月赛7-数蝌蚪-签到题 100](#_Toc520789134)

[八、匹配&网络流 101](#_Toc520789135)

[二分图匹配 101](#_Toc520789136)

[√-51nod-2006-飞行员配对 102](#_Toc520789137)

[未做-hdu-2063 105](#_Toc520789138)

[九、博弈 105](#_Toc520789139)

[√-51nod-1995-三子棋 105](#_Toc520789140)

[十、随机算法 107](#_Toc520789141)

[函数库 107](#_Toc520789142)

[十一、未分类 107](#_Toc520789143)

[莫队算法 107](#_Toc520789144)

[√-HDU-5857-2016多校10-签到 108](#_Toc520789145)

[√-HDU-5867-2016多校10-签到 108](#_Toc520789146)

[十二、题库 108](#_Toc520789147)

[签到题题库 108](#_Toc520789148)

[51nod（基础题） 108](#_Toc520789149)

[补题题库（签到或专项） 109](#_Toc520789150)

[动态规划 109](#_Toc520789151)

[一、简单基础dp 109](#_Toc520789152)

[二、区间dp 111](#_Toc520789153)

[三、树形dp 111](#_Toc520789154)

[四、数位dp 112](#_Toc520789155)

[五、概率(期望) dp 113](#_Toc520789156)

[六、状态压缩dp 113](#_Toc520789157)

[七、数据结构优化的dp 114](#_Toc520789158)

[图论 116](#_Toc520789159)

[并查集&最小生成树（学习区） 116](#_Toc520789160)

[拓补排序（学习区） 119](#_Toc520789161)

[最短路 120](#_Toc520789162)

[差分约束 123](#_Toc520789163)

[二分匹配（学习区） 124](#_Toc520789164)

[欧拉回路&哈密顿回路（待学习） 126](#_Toc520789165)

[连通&LCA（学习区） 126](#_Toc520789166)

[网络流（待学习） 126](#_Toc520789167)

[2-SAT（待学习） 126](#_Toc520789168)

[KM匹配（待学习） 126](#_Toc520789169)

[Dancing Links（待学习） 126](#_Toc520789170)

[一般图匹配（待学习） 126](#_Toc520789171)

[线段树 126](#_Toc520789172)

[字符串算法 128](#_Toc520789173)

[KMP 128](#_Toc520789174)

[字典树 128](#_Toc520789175)

[AC自动机 128](#_Toc520789176)

[后缀数组 129](#_Toc520789177)

[计算几何（模板库训练） 129](#_Toc520789178)

[金银牌题库 129](#_Toc520789179)

[温习题库 129](#_Toc520789180)

[索引 129](#_Toc520789181)

[十三、暑期集训每日训练安排 131](#_Toc520789182)

[学习训练实战补题循环队列 131](#_Toc520789183)

[7月31日 135](#_Toc520789184)

[8月1日 135](#_Toc520789185)

[8月2日 135](#_Toc520789186)

[8月3日 136](#_Toc520789187)

[8月4日 137](#_Toc520789188)

[8月5日 137](#_Toc520789189)

[8月6日 137](#_Toc520789190)

[十四、多校 137](#_Toc520789191)

[牛客网-第1场-2018.7.19（周四） 138](#_Toc520789192)

[ACM-ICPC-上海大都会赛-2018.7.21（周六） 138](#_Toc520789193)

[牛客网-第2场-2018.7.21（周六） 139](#_Toc520789194)

[HDU-1-2018.7.23（周一） 139](#_Toc520789195)

[HDU-2-2018.7.25（周三） 139](#_Toc520789196)

[牛客网-第3场-2018.7.26（周四） 140](#_Toc520789197)

[牛客网-第4场-2018.7.28（周六） 140](#_Toc520789198)

[HDU-3-2018.7.30（周一） 140](#_Toc520789199)

[HDU-4-2018.8.1（周三） 141](#_Toc520789200)

[牛客网-第5场-2018.8.2（周四） 141](#_Toc520789201)

[牛客网-第6场-2018.8.4（周六） 141](#_Toc520789202)

[HDU-5-2018.8.6（周一） 141](#_Toc520789203)

[HDU-6-2018.8.8（周三） 141](#_Toc520789204)

[HDU-7-2018.8.13（周一） 141](#_Toc520789205)

[HDU-8-2018.8.15（周三） 141](#_Toc520789206)

[HDU-9-2018.8.20（周一） 141](#_Toc520789207)

[HDU-10-2018.8.22（周三） 141](#_Toc520789208)

[2018中国大学生程序设计竞赛-网络选拔赛-2018.8.25（周六） 141](#_Toc520789209)

[十五、浙大模板库 141](#_Toc520789210)

[**1、几何** 144](#_Toc520789211)

[**1.5 浮点函数** 150](#_Toc520789212)

[**2、组合** 173](#_Toc520789213)

[**2.4 置换(polya)** 176](#_Toc520789214)

# 一、基础数据结构&语言

## 补题

### 铜牌题-前置：单调队列/单调栈/滑动窗口最大值问题

2018杭电暑期多校3A-Ascending Rating-HDU6319-铜牌

【题意】

【题解】

【代码】

## <cstring>

//strlen()函数的复杂度是O(n)要小心

//截取字符串

strncpy(char\* ch,const char\* pos,size\_t len);

char s[100];

char \*p;

strncpy(p,s+10,3);//这个函数不会自动加’\0’

## new&delete

### 创建并释放一维数组

#include<iostream>

using namespace std;

int main()

{

int n;

cin>>n;

//分配动态一维数组

int \*arr=new int[n];

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

cin>>arr[i];

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

cout<<arr[i]<<" ";

//释放arr数组

delete[] arr;

return 0;

}

### 创建并释放二维数组

#include<iostream>

using namespace std;

int main()

{

int row,col;

cin>>row>>col;

//为行指针分配空间

int \*\*arr=new int \*[row];

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

arr[i]= new int[col];//为每行分配空间（每行中有col个元素）

//输入二维数组的数

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

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

cin>>arr[i][j];

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

//输出二维数组中的数

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

{

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

cout<<arr[i][j]<<" ";

cout<<endl;

}

//释放二维数组（反过来）

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

delete[] arr[i];

delete[] arr;

return 0;

}

## STL

### map

//数据的插入

//示范1

map<int, string> mapStudent;

mapStudent.insert(pair<int, string>(1,“student\_one”));/

//示范2

map<int, string> mapStudent;

mapStudent.insert(map<int, string>::value\_type (1,“student\_one”));

//示范3

map<int, string> mapStudent;

mapStudent[1] = “student\_one”;

mapStudent[2] = “student\_two”;

map<string, int> mapStudent;

string s;//插入就用m[s]=1或者m[s]++之类;

//数据查找

map<int, string>::iterator iter=mapStudent.find(1);

if(iter != mapStudent.end())

cout<<"Find, the value is "<<iter->second<<endl;

else cout<<"Do not Find"<<endl;

### next\_permutation

自动生成全排列

#include <stdio.h>

#include <algorithm>

using namespace std;

int main(){

int n;

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

int a[1000];

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

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

}

sort(a,a+n);//可以自行测试一下删除后的结果

do{

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

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

printf("\n");

}while(next\_permutation(a,a+n));

}

return 0;

}

### 优先队列（priority\_queue）

struct cmp{

bool operator()(string &x,string &y){

return x>y;//从小到大排序

}

};

priority\_queue<string,vector<string>,cmp > q;

//这样的q是将存入的string从小到大

## JAVA

### BigDecimal

## 可持久化平衡树（C++ Pb\_ds库 Rope）

<https://www.cnblogs.com/keshuqi/p/6257642.html>

**声明**

1）头文件

#include<ext/rope>

2)调用命名空间

using namespace \_\_gnu\_cxx;

**底层原理**

查了资料，大概可以称作可持久化平衡树，因为rope适用于大量、冗长的串操作，而不适合单个字符操作官方说明如下：

Though ropes can be treated as [Container](http://www.sgi.com/tech/stl/Container.html)s of characters, and are almost [Sequence](http://www.sgi.com/tech/stl/Sequence.html)s, this is rarely the most efficient way to accomplish a task. Replacing an individual character in a rope is slow: each character replacement essentially consists of two substring operations followed by two concatenation operations. Ropes primarily target a more functional programming style.Inserting a character in the middle of a 10 megabyte rope should take on the order of 10s of microseconds, even if a copy of the original is kept, e.g. as part of an edit history.It is possible to view a function producing characters as a rope. Thus a piece of a rope may be a 100MByte file, which is read only when that section of the string is examined. Concatenating a string to the end of such a file does not involve reading the file. (Currently the implementation of this facility is incomplete.)

另，根据网上资料，rope本质是封装好的类似块状链表的东东，有人说是logn的，但也有说是n^0.5的。rope不支持一切数值操作，如第k大

**小知识**

先介绍几个可能使用到的函数

1）append()

string &append(const string &s,int pos,int n);//把字符串s中从pos开始的n个字符连接到当前字符串的结尾或a.append(b);  
2）substr()

s.substr(0,5);//获得字符串s中从第零位开始长度为5的字符串（默认时长度为刚好开始位置到结尾）

**定义/声明**

rope<char> str;  
also

<crope>r="abcdefg"

**具体内容**

总的来说，

1）运算符：rope支持operator += -= + - < ==

2）输入输出：可以用<<运算符由输入输出流读入或输出。

3）长度/大小：调用length()，size()都可以哦

4）插入/添加等：

push\_back(x);//在末尾添加x

insert(pos,x);//在pos插入x，自然支持整个char数组的一次插入

erase(pos,x);//从pos开始删除x个

copy(pos,len,x);//从pos开始到pos+len为止用x代替

replace(pos,x);//从pos开始换成x

substr(pos,x);//提取pos开始x个

at(x)/[x];//访问第x个元素

**访问**

1）迭代器：不说，在竞赛是超时大忌

2）单点访问，直接用数组形式调用下标即可

**应用**

**bzoj1269 文本编辑器**

如果想看正常版本的看我的[splay平衡树代码](http://www.cnblogs.com/keshuqi/p/6257215.html)

实现操作：  
1.（已知）move k：移动光标到目标，初始为0  
2.（已知）prev：光标前移一个字符  
3.（已知）next：光标后移一个字符  
4.insert n s：在光标后插入长度为n的字符串s光标位置不变  
5.delete n 删除光标后的n个字符，光标位置不变  
6.rotate n 反转光标后的n个字符，光标位置不变  
7.get 输出光标后一个字符，光标位置不变

**solution**

为实现反转操作且保证不超时，我们不调用rope自带的可怕函数，暴力构建两个rope，插入时一个正序插入一个倒序插入，区间即为子串赋值

#include<cstdio>

#include<ext/rope>

#include<iostream>

using namespace std;

using namespace \_\_gnu\_cxx;

inline int Rin(){

int x=0,c=getchar(),f=1;

for(;c<48||c>57;c=getchar())

if(!(c^45))f=-1;

for(;c>47&&c<58;c=getchar())

x=(x<<1)+(x<<3)+c-48;

return x\*f;

}

int n,pos,x,l;

rope<char>a,b,tmp;

char sign[10],ch[1<<22],rch[1<<22];

int main(){

n=Rin();

while(n--){

scanf("%s",sign);

switch(sign[0]){

case'M':pos=Rin();break;

case'P':pos--;break;

case'N':pos++;break;

case'G':putchar(a[pos]);putchar('\n');break;

case'I':

x=Rin();

l=a.length();

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

do{ch[i]=getchar();}

while(ch[i]=='\n');

rch[x-i-1]=ch[i];

}

ch[x]=rch[x]='\0';

a.insert(pos,ch);

b.insert(l-pos,rch);

break;

case'D':

x=Rin();

l=a.length();

a.erase(pos,x);

b.erase(l-pos-x,x);

break;

case'R':

x=Rin();

l=a.length();

tmp=a.substr(pos,x);

a=a.substr(0,pos)+b.substr(l-pos-x,x)+a.substr(pos+x,l-pos-x);

b=b.substr(0,l-pos-x)+tmp+b.substr(l-pos,pos);

break;

}

}

return 0;

}

基本操作：

rope test;

test.push\_back(x);//在末尾添加x

test.insert(pos,x);//在pos插入x

test.erase(pos,x);//从pos开始删除x个

test.copy(pos,len,x);//从pos开始到pos+len为止用x代替

test.replace(pos,x);//从pos开始换成x

test.substr(pos,x);//提取pos开始x个

test.at(x)/[x];//访问第x个元素

其算法复杂度n\*(n^0.5)，可以在很短的时间内实现快速的插入、删除和查找字符串，是一个很厉害的神器！

这里有英文版的详解：<http://www.sgi.com/tech/stl/Rope.html>

测试代码

#include <iostream>

#include <cstdio>

#include <cstring>

#include <cctype>

#include <ext/rope>

#include <algorithm>

using namespace std;

using \_\_gnu\_cxx::crope;

const int maxn = 10010;

char op[maxn];

crope text;

int main(){

text.clear();

scanf("%s",op);

int l = strlen(op);

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

text += op[i];

cout<<text<<endl;

cout<<"test.push\_back(x);//在末尾添加x"<<endl;

text.push\_back('a');

cout<<text<<endl;

/\*

text.push\_back("bbb");

cout<<text<<endl;//编译错误

\*/

cout<<"test.insert(pos,x);//在pos插入x"<<endl;

text.insert(1,'b');

cout<<text<<endl;

text.insert(2,"aaa");

cout<<text<<endl<<endl;

cout<<"test.erase(pos,x);//从pos开始删除x个"<<endl;

text.erase(1,3);

cout<<text<<endl<<endl;

cout<<"test.copy(pos,len,x);//从pos开始到pos+len为止用x代替"<<endl;

text.copy(1,5,op);

cout<<text<<endl<<endl;

cout<<"test.replace(pos,x);//从pos开始换成x"<<endl;

text.replace(1,'c');

cout<<text<<endl;

text.replace(1,"ccc");

cout<<text<<endl<<endl<<endl;

cout<<"test.substr(pos,x);//提取pos开始x个"<<endl;

//text = text.substr(2);这样默认为提取一个

cout<<text.substr(2)<<endl;

cout<<text.substr(2,3)<<endl<<endl;

cout<<"test.at(x)/[x];//访问第x个元素"<<endl;

cout<<text.at(4)<<endl<<endl;

}

### √-牛客网2018暑期多校第3场-C（可持久化平衡树）

【题意】n个数(1-n)，给m个操作，每次操作将当前的一段位置的数拿到最前面，要求输出进行完操作后依次位置的数

【题解】比赛的时候听说是splay树模板题，赛后又听说使用这个rope库（可持久化平衡树）就可以轻松AC

【代码1】//rope大法

#include <bits/stdc++.h>

#include <ext/rope>

using namespace std;

using namespace \_\_gnu\_cxx;

const int maxn=1e5+10;

rope<int> T;

int n,m;

int main()

{

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

for (int i=1;i<=n;i++) T.push\_back(i);

while (m--)

{

int p,s;

scanf("%d%d",&p,&s);

p--;

T=T.substr(p,s)+T.substr(0,p)+T.substr(p+s,n-p-s);

}

for (int i=0;i<n;i++) printf("%d ",T.at(i));

return 0;

}

【代码2】//splay树版本

#include<iostream>

#include<cstring>

#include<queue>

#include<cstdio>

#include<algorithm>

#define N 300015

#define inf 1<<29

#define MOD 100000007

#define LL long long

#define Key\_value ch[ch[root][1]][0]

#define \_match(a,b) ((a)==(b))

using namespace std;

int n,q;

int size[N],pre[N],key[N],num[N],rev[N];

int ch[N][2],tot,root,node[N];

void NewNode(int &r,int k,int father){

r=++tot;

ch[r][0]=ch[r][1]=0;

pre[r]=father;

rev[r]=0;

key[r]=k;

}

void Push\_Up(int r){

size[r]=size[ch[r][0]]+size[ch[r][1]]+1;

}

void Push\_Down(int r){

if(rev[r]){

swap(ch[r][0],ch[r][1]);

rev[ch[r][0]]^=1;

rev[ch[r][1]]^=1;

rev[r]=0;

}

}

void Bulid(int &r,int L,int R,int father){

if(L>R)

return ;

int mid=(L+R)/2;

NewNode(r,mid,father);

Bulid(ch[r][0],L,mid-1,r);

Bulid(ch[r][1],mid+1,R,r);

Push\_Up(r);

}

void Init(){

tot=root=0;

ch[root][0]=ch[root][1]=pre[root]=rev[root]=size[root]=0;

NewNode(root,-1,0);

NewNode(ch[root][1],-1,root);

size[root]=2;

Bulid(Key\_value,1,n,ch[root][1]);

Push\_Up(ch[root][1]);

Push\_Up(root);

}

void Rotate(int x,int kind){

int y=pre[x];

Push\_Down(y);

Push\_Down(x);

ch[y][!kind]=ch[x][kind];

pre[ch[x][kind]]=y;

if(pre[y])

ch[pre[y]][ch[pre[y]][1]==y]=x;

pre[x]=pre[y];

ch[x][kind]=y;

pre[y]=x;

Push\_Up(y);

}

void Splay(int r,int goal){

Push\_Down(r);

while(pre[r]!=goal){

if(pre[pre[r]]==goal)

Rotate(r,ch[pre[r]][0]==r);

else{

int y=pre[r];

int kind=(ch[pre[y]][0]==y);

if(ch[y][kind]==r){

Rotate(r,!kind);

Rotate(r,kind);

}

else{

Rotate(y,kind);

Rotate(r,kind);

}

}

}

Push\_Up(r);

if(goal==0) root=r;

}

int Get\_Kth(int r,int k){

Push\_Down(r);

int t=size[ch[r][0]];

if(t==k-1)

return r;

if(t>=k)

return Get\_Kth(ch[r][0],k);

else

return Get\_Kth(ch[r][1],k-t-1);

}

int Get\_Min(int r){

Push\_Down(r);

while(ch[r][0]){

r=ch[r][0];

Push\_Down(r);

}

return r;

}

int Get\_Max(int r){

Push\_Down(r);

while(ch[r][1]){

r=ch[r][1];

Push\_Down(r);

}

return r;

}

void Reversal(int a,int b){

int x=Get\_Kth(root,a);

int y=Get\_Kth(root,b+2);

Splay(x,0);

Splay(y,root);

rev[Key\_value]^=1;

}

void Cut(int a,int b,int c){

int x=Get\_Kth(root,a);

int y=Get\_Kth(root,b+2);

Splay(x,0);

Splay(y,root);

int tmp=Key\_value;

Key\_value=0;

Push\_Up(ch[root][1]);

Push\_Up(root);

int z=Get\_Kth(root,c+1);

Splay(z,0);

int m=Get\_Min(ch[root][1]);

Splay(m,root);

Key\_value=tmp;

pre[Key\_value]=ch[root][1];

Push\_Up(ch[root][1]);

Push\_Up(root);

}

int cnt;

void InOrder(int r){

if(r==0)

return;

Push\_Down(r);

InOrder(ch[r][0]);

if(cnt>=1&&cnt<=n){

if(cnt>1) printf(" ");

printf("%d",key[r]);

}

cnt++;

InOrder(ch[r][1]);

}

int main(){

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

Init();

while(q--){

int a,b;

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

b=a+b-1;

Cut(a,b,0);

}

cnt=0;

InOrder(root);

printf("\n");

}

return 0;

}

## 枚举贪心排序构造

### √-牛客网2018暑期多校第4场-F-Beautiful Garden-签到题

【题意】

【题解】

【自测数据】

/\*

input

5

6 8

acbbbbca

dcaccacd

cdaddadc

cdaddadc

dcaccacd

acbbbbca

6 8

acbcbbca

dcaccacd

cdaddadc

cdaddadc

dcaccacd

acbbbbca

6 8

acbbbbca

dcadcacd

cdaddadc

cdaddadc

dcaccacd

acbbbbca

4 4

aaaa

aaaa

aaaa

aaaa

4 6

abccba

dcfccd

defced

abccba

output

6

0

3

1

1

\*/

【代码】

#include <bits/stdc++.h>

#define MAXN 2010

using namespace std;

int main()

{

int t;

char garden[MAXN][MAXN];

scanf("%d",&t);

while(t--){

int n,m;

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

for (int i=0;i<n;i++) scanf("%s",garden[i]);

int cntn=0,cntm=0;

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

bool same=true;

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

if(garden[i][j]!=garden[n-i-1][j]){

same=false;

break;

}

}

if(!same) break;

for (int j=0;j<m/2;j++){

if(garden[i][j]!=garden[i][m-j-1]){

same=false;

break;

}

}

if(same) cntn++;else break;

}

for (int j=0;j<m/2;j++){

bool same=true;

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

if(garden[i][j]!=garden[i][m-j-1]){

same=false;

break;

}

}

if(!same) break;

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

if(garden[i][j]!=garden[n-i-1][j]){

same=false;

break;

}

}

if(same) cntm++;else break;

}

cntn=min(cntn,n/2-1);

cntm=min(cntm,m/2-1);

printf("%d\n",cntn\*cntm);

}

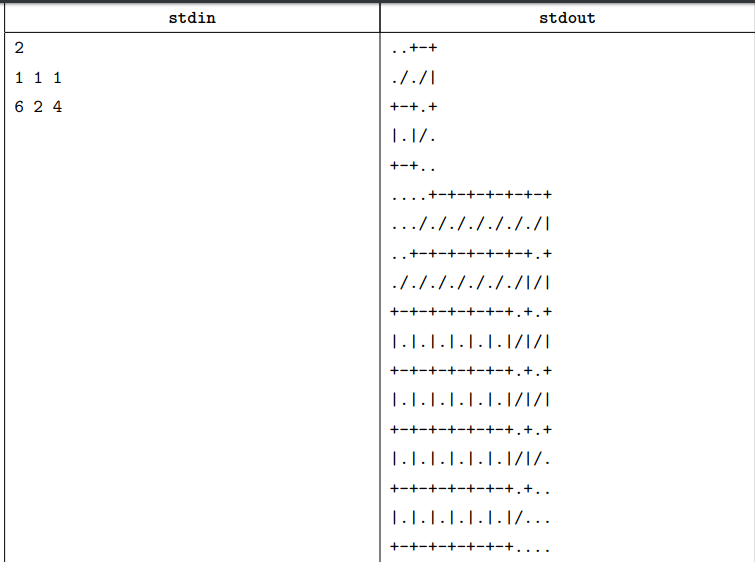
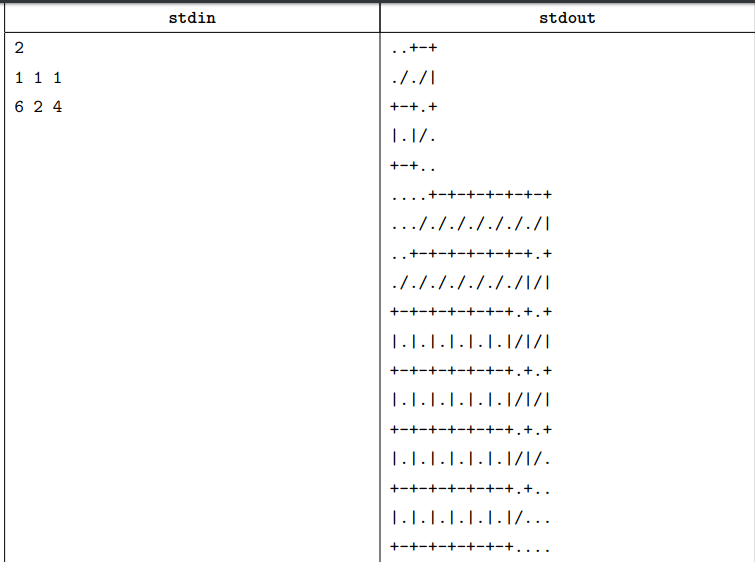
return 0;

}

## 模拟

### √-HDU-6330-Visual Cube-签到题

【题意】给定长宽高，输出立方体模拟图



【题解】2018杭电暑期多校L题签到题；直接找规律输出；每一行计算一下加在前面的点和加在后面的点数。

【代码】

#include <bits/stdc++.h>

using namespace std;

int main()

{

int t;

cin>>t;

while(t--){

int a,b,c;

cin>>a>>b>>c;

int rowlen=2\*a+1+2\*b,rownum=0,colen=2\*c+2\*b+1;

for(int row=0;row<2\*b;row++){

int backdotnum=rowlen-(2\*a+1)-(colen-rownum)+1;

if(row%2==0){

for(int j=0;j<2\*b-row;j++) printf(".");

for(int j=0;j<a;j++) printf("+-");

printf("+");

int cnt=0;

if(backdotnum<=0){

for(int j=0;j<row/2;j++) printf(".+");

}else{

cnt=rowlen-backdotnum-(2\*b-row)-(2\*a+1);

for(int i=0;i<cnt/2;i++) printf(".+");

}

}else{

for(int j=0;j<2\*b-row;j++) printf(".");

for(int j=0;j<a;j++) printf("/.");

int cnt=0;

if(backdotnum<=0){

for(int j=0;j<row/2+1;j++) printf("/|");

}else{

cnt=rowlen-backdotnum-(2\*b-row)-(2\*a+1);

for(int i=0;i<cnt/2;i++) printf("/|");

printf("/");

}

}

if(backdotnum>0) for(int i=0;i<backdotnum;i++) printf(".");

printf("\n"),rownum++;

}

for(int row=0;row<2\*c;row++){

int backdotnum=rowlen-(2\*a+1)-(colen-rownum)+1;

if(row%2==0){

for(int j=0;j<a;j++) printf("+-");

printf("+");

int cnt=0;

if(backdotnum<=0){

cnt=rowlen-2\*a-1;

}else{

cnt=rowlen-2\*a-backdotnum-1;

}

for(int i=0;i<cnt/2;i++) printf(".+");

}else{

for(int j=0;j<a;j++) printf("|.");

printf("|");

int cnt=0;

if(backdotnum<=0){

cnt=rowlen-2\*a-1;

}else{

cnt=rowlen-2\*a-1-backdotnum;

}

for(int i=0;i<cnt/2;i++) printf("/|");

if(backdotnum>0) printf("/");

}

if(backdotnum>0){

for(int i=0;i<backdotnum;i++) printf(".");

}

printf("\n"),rownum++;

}

for(int i=0;i<a;i++) printf("+-");

printf("+");

int dotnum=rowlen-(2\*a+1);

for(int i=0;i<dotnum;i++) printf(".");

printf("\n");

}

return 0;

}

# 二、搜索

# 三、动态规划

## 补题

### 铜牌题-前置：状压DP

2018杭电暑期多校3C-HDU6321-Dynamic Graph Matching

【题意】

【题解】

【代码】

## 递推

### √-HDU-2050-折线分割平面

【题意】求解n个折线能把平面分为几个区域

【题解】递推式f(n)=f(n-1)+4(n-1)+1=f(n-1)+4n-3,f(1)=2

具体的推导思路大概是考虑前面已经有n-1个折线了，然后现在又来一个折线，看成一个起点的2条射线，这样一交（尽可能往密集了交），按照新的射线大概可以分为3个区域，左射线左增加的区域数，右射线右增加的区域数，两射线之间增加的区域数，左边是n-1，右边是n-1，中间是2(n-1)，再加1个自带的。反正是看着n=1,n=2,n=3的情况连蒙带猜的，n=4再网上画也不好画了，这题也没法打表，还没有想到比较严谨的思路。

## 基础动态规划

### 最大连续子段和

#### √-HDU-1003

【题意】n个数求最大连续子段和，同时输出子段的起始终止点（存在多个按照起始点顺序输出第1个，起始点相同的终止点尽可能小）

【题解】最大连续子段和模板题

【代码】

#include <bits/stdc++.h>

#define MAXN 100010

using namespace std;

int main()

{

int t;

scanf("%d",&t);

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

int n,maxsum=INT\_MIN,nowsum=-1,maxfirst=0,maxlast=0,nowfirst=1,nownum;

scanf("%d",&n);

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

scanf("%d",&nownum);

if(nowsum<0) nowsum=nownum,nowfirst=i;else nowsum+=nownum;

if(nowsum>maxsum) maxsum=nowsum,maxfirst=nowfirst,maxlast=i;

}

printf("Case %d:\n%d %d %d\n",casenum,maxsum,maxfirst,maxlast);

if(casenum!=t) printf("\n");

}

return 0;

}

### LIS（最长上升子序列）

#### √-HDU-1087

【题意】n个正数，找到一个上升子序列，使得和最大，输出这个最大值，n规模1000

【题解】dp[i]=max(dp[j])+num[i](num[j]<num[i]&&j<i)，复杂度

【代码】

#include <bits/stdc++.h>

#define MAXN 1010

using namespace std;

int main()

{

int n,num[MAXN],dp[MAXN],maxans;

while(cin>>n&&n){

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

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

cin>>num[i];

for (int j=i-1;j>=0;j--) if(num[j]<num[i]&&dp[j]>dp[i]) dp[i]=dp[j];

dp[i]+=num[i],maxans=max(maxans,dp[i]);

}

cout<<maxans<<endl;

}

return 0;

}

### LCIS（最长上升公共子序列）

#### √-HDU-1503-Advanced Fruits

【题意】给定2个字符串；要求1个最短的字符串，使得给定的2个字符串都是所求字符串的子序列。Special Judge。字符串长度范围1-100.

【题解】LCIS问题，对于非LCIS的部分，2个字符串的字母都要写。

dp[i][j]表示第1个字符串考虑前i个字符，第2个字符串考虑前j个字符时的最大上升公共子序列长度

s[i][j]表示考虑第1个字符串前i个字符和第2个字符串前j个字符时的题目所求字符串（也就是最大公共子串，还要加上非公共子串的部分）

当a[i]==b[j]时

dp[i][j]=dp[i-1][j-1]+1,s[i][j]=s[i-1][j-1]+a[i]或b[j]

当a[i]!=b[j]时

dp[i][j]=max(dp[i-1][j],dp[i][j-1])

若选取dp[i-1][j]则s[i][j]=s[i-1][j]+a[i]

若选取dp[i][j-1]则s[i][j]=s[i][j-1]+b[j]

初始时dp[0][j]和dp[i][0]均为0

s[0][0]=””,s[0][j]为b串的前j个字符，a[i][0]为a串的前i个字符

【自测数据】

【代码】

#include <bits/stdc++.h>

#define MAXN 110

using namespace std;

int main()

{

int dp[MAXN][MAXN],n,m;

string s[MAXN][MAXN],a,b;

while(cin>>a>>b){

n=a.size(),m=b.size();

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

s[0][0]="";

for (int i=1;i<=n;i++) s[i][0]=s[i-1][0]+a[i-1];

for (int j=1;j<=m;j++) s[0][j]=s[0][j-1]+b[j-1];

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

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

char cha=a[i-1],chb=b[j-1];

if(cha==chb){

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

s[i][j]=s[i-1][j-1]+cha;

}else{

if(dp[i-1][j]>dp[i][j-1]){

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

s[i][j]=s[i-1][j]+cha;

}else{

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

s[i][j]=s[i][j-1]+chb;

}

}

}

}

cout<<s[n][m]<<endl;

}

return 0;

}

#### √-HDU-1159-LCIS模板题

【题意】求最长上升公共子序列

【题解】模板题

【代码】

#include <bits/stdc++.h>

#define MAXN 1010

using namespace std;

int main()

{

string sa,sb;

int n,m,dp[MAXN][MAXN];

while(cin>>sa>>sb){

n=sa.size(),m=sb.size(),memset(dp,0,sizeof(dp));

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

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

char a=sa[i-1],b=sb[j-1];

if(a==b) dp[i][j]=dp[i-1][j-1]+1;

else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

}

}

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

}

return 0;

}

## 背包问题

<https://blog.csdn.net/yoer77/article/details/70943462>

### 0-1背包

#### √-HDU-2602-Bone Collector

【题意】n个物品，告知n个物品价值、体积，给定背包容量，求解最大存放价值

【题解】0-1背包裸题，题目要求背包不必放满（初始化全0）

【代码】

#include <cstdio>

#include <cstring>

#include <algorithm>

#define MAXN 1010

#define MAXV 1010

using namespace std;

int main()

{

int t,N,V,dp[MAXV],w[MAXN],v[MAXN];

scanf("%d",&t);

while(t--){

scanf("%d%d",&N,&V);

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

for (int i=0;i<N;i++) scanf("%d",&w[i]);

for (int i=0;i<N;i++) scanf("%d",&v[i]);

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

for (int j=V;j>=0;j--){

if (j<v[i]) continue;

dp[j]=max(dp[j],dp[j-v[i]]+w[i]);

}

}

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

}

return 0;

}

### 完全背包

#### HDU-1114-Piggy Bank

【题意】给定背包容量，n个物品的体积和价值，要求背包刚好装满的最小价值

【题解】完全背包，修改状态转移方程为min，恰好装满，初始化除容量为0之外为非法

【代码】

#include <bits/stdc++.h>

#define MAXV 10010

using namespace std;

int main(){

int t,dp[MAXV],empty\_v,full\_v,N,V,w,v;

scanf("%d",&t);

while(t--){

scanf("%d%d%d",&empty\_v,&full\_v,&N);

V=full\_v-empty\_v,memset(dp,-1,sizeof(dp)),dp[0]=0;

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

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

for (int j=0;j<=V;j++){

if(j<v||dp[j-v]<0) continue;

if(dp[j]>=0) dp[j]=min(dp[j],dp[j-v]+w);else dp[j]=dp[j-v]+w;

}

}

if(dp[V]==-1) printf("This is impossible.\n");

else printf("The minimum amount of money in the piggy-bank is %d.\n",dp[V]);

}

return 0;

}

### 多重背包

## 数位DP

### √-HDU-2089-不要62

【题意】求[n,m]区间中不含‘4’和‘62’的数字

【题解】数位DP经典入门题，dp[i][j]表示长度为i开头填j这个数字的方案数



初始化dp[0][j]为1(dp[0][4]为0)

WA了1发之后写了个测试函数，暴力答案对照，随机试了很多组数据都对之后又交了WA了一发，于是写了个遍历n以内所有的对照debug程序，只要有错就跳出来，果然又找到了一些隐藏的错误。

【代码】

#include <bits/stdc++.h>

#define MAXN 20

#define debug false

#define hugedebug false

using namespace std;

int dp[MAXN][10],digit[MAXN];

int solve(int x){

memset(digit,0,sizeof(digit));

int len=0,ans=0;

while(x){

digit[len++]=x%10;

x/=10;

}

for(int i=len-1;i>=0;i--){

for (int j=0;j<digit[i];j++){

if(!(digit[i+1]==6&&j==2)) ans+=dp[i][j];

}

if(digit[i]==4) break;

if(digit[i+1]==6&&digit[i]==2) break;

}

return ans;

}

int debug\_print(int n,int m){

int cnt=0;

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

char temp[20];

sprintf(temp,"%d",i);

bool check=true;

for (int j=0;j<(int)strlen(temp);j++){

if(temp[j]=='4'||((j+1<(int)strlen(temp))&&(temp[j]=='6'&&temp[j+1]=='2'))){

check=false;

break;

}

}

if(check) cnt++;

}

if(!hugedebug) cout<<"ans="<<cnt<<endl;

return cnt;

}

int main(){

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

int lastsum,sum;

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

sum=0;

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

if(i==0){

if(j!=4) dp[i][j]=1;

}else{

if(j==4) continue;

if(j!=6) dp[i][j]=lastsum;else dp[i][j]=lastsum-dp[i-1][2];

}

sum+=dp[i][j];

}

lastsum=sum;

}

if(hugedebug){

int hugenum=1000;

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

for (int m=n;m<=hugenum;m++){

int ans1=debug\_print(n,m),ans2=solve(m+1)-solve(n);

if(ans1!=ans2){

cout<<n<<" "<<m<<endl;

cout<<"ans1:"<<ans1<<endl;

cout<<"ans2:"<<ans2<<endl;

return 0;

}

}

}

cout<<"correct"<<endl;

return 0;

}

int n,m;

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

if (n==0&&m==0) break;

if(debug) debug\_print(n,m);

printf("%d\n",solve(m+1)-solve(n));

}

return 0;

}

## 概率DP

## HDU-5863-2016多校10-铜牌

【题意】

【题解】

【代码】

## √-牛客网-第1场-E-Removal-铜牌

【题意】n个数字组成的序列，最多有k种数字（1-k），要求删掉m个数字后，不同的序列个数。数据规模n为，k为10，m为10

【题解】

铜牌题

动态规划dp[i][j]表示长度为i结尾为数字j的方案数，sum[i]表示长度为i的方案数。

那么sum[i]=dp[i][1]+……+dp[i][k]

dp[i][j]=sum[i-1]

答案就是sum[n-m]

伪代码：

从第1给定序列的第1个数字开始…，每次多取1个{

（考虑前i-1个数字的序列的答案都已经最新）

则（同时要首先更新）

依次求得……（同时更新sum数组）

}

考虑到我们需要的是取到第n个数字时的sum[n-m]，也就是在最后一轮，我们在求dp时，求dp[n][c]……到dp[n-m][c]的时候就可以停止了，因为再更新下去也不会改变sum[n-m]的值，这个时候会用到的值就是取第n-1个数字时的sum[n-1]……sum[n-m-1]，那么也就是说在取第n-1个数字时，求dp[n-1][c]……到dp[n-m-1][c]就可以停止了，因为再更新下去也不会影响到我们之后要用的值，以此类推，求第i个数字的时候，最多只需要求dp[i][c]……到dp[i-m][c]。这样总的时间复杂度就是,如果内层循环不是只求m个就停止的话，那么这样的时间复杂度就是

【代码】

#include <bits/stdc++.h>

#define MAXN 10010

#define N 1000000007

#define MAXK 15

#define ll long long

using namespace std;

int main()

{

int n,m,k,dp[MAXN][MAXK],sum[MAXN],num[MAXN];

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

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

memset(dp,0,sizeof(dp)),memset(sum,0,sizeof(sum)),sum[0]=1;

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

for (int j=i;j>=max(1,i-m);j--){

int change=(sum[j-1]-dp[j][num[i]]+N)%N;

dp[j][num[i]]=sum[j-1];

sum[j]=(sum[j]+change)%N;

}

}

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

}

return 0;

}

## √-牛客网-第2场-D-Money-签到题

【题意】n家店，每个店给一个price[i]，可以在一家店以price[i]买入，在另一家店以price[i]卖出，必须按照1-n的顺序访问各家店，起始资金无穷大，同一时刻只能携带1个物品，求最大利润和最大利润时的最小交易次数

【题解】buy[i]表示在第I家店进行买入时的最大利润

buy\_op[i]表示在第I家店买入且获得最大利润时的最小交易册数

sell[i]表示在第I家店卖出时的最大利润

sell\_op[i]表示在第I家店卖出且获得最大利润时的最小交易次数

buy[i]=max(sell[0]…sell[i-1])-price[i]

当选定某个sell[k]得到的buy[i]超过原本值时，更新操作数，如果与原来持平，则选取较小的操作数。

sell[i]=max(buy[0]…buy[i-1])+price[i]

sell\_op与buy\_op类似

可以发现维护4个最值（买入卖出时的最大利润和最小操作数）即可，因为之后的更新只会使用到这些值。

【自测数据】

/\*

input

7

5

9 10 7 6 8

6

9 10 9 10 9 11

6

9 9 9 10 10 10

6

9 9 9 11 10 10

6

9 9 9 10 11 10

2

0 2147483647

2

0 0

output

3 4

4 6

1 2

2 2

2 2

2147483647 2

0 0

\*/

【代码】

#include <bits/stdc++.h>

#define MAXN 100010

#define ll long long

using namespace std;

int main()

{

int t;

cin>>t;

while(t--){

ll n,max\_sell=0,min\_sell\_op=0,max\_buy=INT\_MIN,min\_buy\_op=0;

cin>>n;

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

ll p;

cin>>p;

ll new\_max\_sell=max\_sell,new\_min\_sell\_op=min\_sell\_op,new\_max\_buy=max\_buy,new\_min\_buy\_op=min\_buy\_op;

if(max\_sell<=max\_buy+p){

new\_max\_sell=max\_buy+p;

if(max\_sell==max\_buy+p) new\_min\_sell\_op=min(min\_sell\_op,min\_buy\_op+1);else new\_min\_sell\_op=min\_buy\_op+1;

}

if(max\_buy<=max\_sell-p){

new\_max\_buy=max\_sell-p;

if(max\_buy==max\_sell-p) new\_min\_buy\_op=min(min\_sell\_op+1,min\_buy\_op);else new\_min\_buy\_op=min\_sell\_op+1;

}

max\_sell=new\_max\_sell,min\_sell\_op=new\_min\_sell\_op,max\_buy=new\_max\_buy,min\_buy\_op=new\_min\_buy\_op;

}

cout<<max\_sell<<" "<<min\_sell\_op<<endl;

}

return 0;

}

# 四、图论

## 最短路

### SPFA

#### √-51nod-1459-迷宫游戏

【题意】给定n个房间，m条路（2个房间之间只有1条路），到达每个房间有一个正的得分，每条路有一个时间消耗，给定起点、终点，求最短的时间和最短时间下的最大得分

【题解】使用SPFA求解最短路，但是这里最短路的定义（也就是在做松弛操作的时候），以时间消耗越短越好为第一判断，分数得分越大越好为第二判断。

【代码】

#include <bits/stdc++.h>

#define MAXROOM 510

#define MAXROAD 610

using namespace std;

int main()

{

int roomnum,roadnum,from,to,score[MAXROOM],cost[MAXROOM][MAXROOM],spath[MAXROOM],mscore[MAXROOM];

memset(cost,-1,sizeof(cost)),memset(mscore,0,sizeof(mscore));

for (int i=0;i<MAXROOM;i++) spath[i]=INT\_MAX;

cin>>roomnum>>roadnum>>from>>to;

for (int i=0;i<roomnum;i++) cin>>score[i];

while(roadnum--){

int x,y,z;

cin>>x>>y>>z;

cost[x][y]=cost[y][x]=z;

}

spath[from]=0,mscore[from]=score[from];

queue<int> q;

bool exist[MAXROOM];

memset(exist,false,sizeof(exist));

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

q.push(from);

exist[from]=true;

while(!q.empty()){

int nowroom=q.front();

q.pop();

exist[nowroom]=false;

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

if (cost[nowroom][i]>=0&&((spath[nowroom]+cost[nowroom][i]<spath[i])||((spath[nowroom]+cost[nowroom][i]==spath[i])&&(mscore[nowroom]+score[i]>mscore[i])))){

spath[i]=spath[nowroom]+cost[nowroom][i],mscore[i]=mscore[nowroom]+score[i];

if (!exist[i]) q.push(i),exist[i]=true;

}

}

}

cout<<spath[to]<<" "<<mscore[to]<<endl;

return 0;

}

#### √-UVA-762-We Ship Cheap

【题意】告知若干个点之间连通，求指定起终点之间的最短路径（存在与否、输出路径），连通距离为1

【题解】SPFA模板题（要输出路径，从to逆着找到from，这样可以直接用pre的记录顺着输出），输入没有指定V的个数，但是规定了一定是2个大写字母，所以使用使用哈希思想将其看作是26进制的数表示，直接使用邻接矩阵即可，没有超时

【代码】

#include <bits/stdc++.h>

#define MAXN 26\*26

#define FARDIS 10000

using namespace std;

int string\_2\_num(string s){

return (s[0]-'A')\*26+s[1]-'A';

}

string num\_2\_string(int x){

string s="";

s+='A'+x/26,s+='A'+x%26;

return s;

}

int main()

{

int n,cost[MAXN][MAXN],spath[MAXN],pre[MAXN],from,to;

bool first=true;

while(cin>>n){

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

for (int i=0;i<MAXN;i++) spath[i]=FARDIS;

while(n--){

string a,b;

cin>>a>>b;

int numa=string\_2\_num(a),numb=string\_2\_num(b);

cost[numa][numb]=cost[numb][numa]=1;

}

string sfrom,sto;

cin>>sfrom>>sto;

from=string\_2\_num(sfrom),to=string\_2\_num(sto);

queue<int> q;

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

bool exist[MAXN],have\_path=false;

memset(exist,false,sizeof(exist));

exist[to]=true;

q.push(to);

spath[to]=0;

while(!q.empty()){

int now=q.front();

q.pop();

exist[now]=false;

for (int i=0;i<26\*26;i++){

if(cost[now][i]>=0&&(spath[now]+cost[now][i]<spath[i])){

spath[i]=spath[now]+cost[now][i];

pre[i]=now;

if (!exist[i]) q.push(i),exist[i]=true;

if(i==from) have\_path=true;

}

}

}

int numa=from,numb=pre[numa];

if(first) first=false;else cout<<endl;

if (have\_path){

while(true){

string sa=num\_2\_string(numa),sb=num\_2\_string(numb);

cout<<sa<<" "<<sb<<endl;

if(numb==to) break;

numa=numb,numb=pre[numa];

}

}else cout<<"No route"<<endl;

}

return 0;

}

### Floyd

#### √-UVA-567-Risk

【题意】给定20个城市之间的连通情况，就给定起终点间的最短距离（连通距离为1）

【题解】floyd模板题目，设置100为任意两城市之间的不连通距离，1为直接连通距离

【代码】

#include <bits/stdc++.h>

#define FARDIS 100

#define MAXN 30

using namespace std;

int main()

{

int f[MAXN][MAXN],num,casenum=1,qnum;

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

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

for (int j=0;j<MAXN;j++) f[i][j]=FARDIS;

f[i][i]=0;

}

for (int i=1;i<=19;i++){

if(i>1) scanf("%d",&num);

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

int temp;

scanf("%d",&temp);

f[i][temp]=f[temp][i]=1;

}

}

//floyd

for (int k=1;k<=20;k++) for (int i=1;i<=20;i++) for (int j=1;j<=20;j++) f[i][j]=min(f[i][j],f[i][k]+f[k][j]);

scanf("%d",&qnum);

printf("Test Set #%d\n",casenum++);

while(qnum--){

int s,t;

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

printf("%2d to %2d: %d\n",s,t,f[s][t]);

}

printf("\n");

}

return 0;

}

## 负环判定

使用SPFA判负环（BFS形式），进入队列超过n次即有负环

### √-POJ-3259-Wormholes

【题意】一张有向图，给定每条边的权值，判断是否有负环

【题解】使用spfa的负环判定，用邻接表可以减少时间复杂度，本题的数据给定的每张图都是连通图 ，故判定时只要将1作为起始点放入即可，不必再将其他点作为起始点放入，因为但凡是进入过队列的点，都可以认为以其起始过了，所以如果给定的数据不是连通图，则要确保每个点至少进入队列一次

【代码】

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <queue>

#include <cstring>

#include <vector>

#define MAXN 510

#define FARDIS 50000

using namespace std;

int main()

{

int F,f[MAXN][MAXN],n,m,w,s,e,t,cnt[MAXN],d[MAXN],in[MAXN];

vector<int> edge[MAXN];

bool exist;

scanf("%d",&F);

while(F--){

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

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

for (int j=1;j<=n;j++) f[i][j]=FARDIS;

f[i][i]=0,edge[i].clear();

}

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

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

if(f[s][e]==FARDIS) edge[s].push\_back(e),edge[e].push\_back(s);

f[s][e]=min(f[s][e],t),f[e][s]=min(f[e][s],t);

}

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

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

if(f[s][e]==FARDIS) edge[s].push\_back(e);

f[s][e]=min(f[s][e],-t);

}

exist=false;

queue<int> q;

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

memset(cnt,0,sizeof(cnt)),memset(in,0,sizeof(in));

for(int i=1;i<=n;i++) d[i]=FARDIS;

q.push(1),cnt[1]++,in[1]=true,d[1]=0;

while(!q.empty()){

int now=q.front();

q.pop(),in[now]=false;

for (int i=0;i<(int)edge[now].size();i++){

int to=edge[now][i];

if(d[now]+f[now][to]<d[to]){

d[to]=d[now]+f[now][to];

if(!in[to]){

q.push(to),cnt[to]++;

if(cnt[to]>n){

exist=true;

break;

}

}

}

}

if(exist) break;

}

if(exist) printf("YES\n");else printf("NO\n");

}

return 0;

}

## 网络流

## 2-SAT

## 图同构

## 拓补排序

## 最小生成树

## 线段树优化建图

# 五、字符串算法

## KMP

<https://www.bilibili.com/video/av3246487?from=search&seid=8616183992466782178>

这个视频讲的很详细…比较容易懂

### √-HDU-1711-Number Sequence

【题意】给定数字序列a和b，如果b序列是a序列的子序列，输出其在a序列中第1个出现的位置（下标从1开始），否则输出-1。

【题解】KMP模板题，在交之前写了个随机化测试以验证程序是否正确。

【代码】

#include <bits/stdc++.h>

#define MAXN 100010

#define MAXM 10010

#define random\_debug false

using namespace std;

void cal\_next(int n,int num[],int next[]){

next[0]=0;

int i=1,j=0;

while(i<n){

if(num[i]==num[j]){

next[i]=j+1,i++,j++;

continue;

}

while(num[i]!=num[j]){

if(j==0){

next[i++]=0;

break;

}

j=next[j-1];

}

}

return;

}

int kmp(int n,int m,int s[],int pattern[],int next[]){

cal\_next(m,pattern,next);

int j=0;

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

while(s[i]!=pattern[j]){

if(j==0) break;

j=next[j-1];

}

if(s[i]==pattern[j]) j++;

if(j==m) return i-m+2;

}

return -1;

}

int main()

{

if(random\_debug){

int checktime=100000;

srand((unsigned)time(NULL));//这个必须放在外面才能在while语句内每次生成随机的序列

while(checktime--){

int nmin=1,nmax=1000,mmin=1,mmax=1000,a[MAXN],b[MAXM],nummin=-3,nummax=3,next[MAXM];

int n=rand()%(nmax-nmin+1)+nmin,m=rand()%(mmax-mmin+1)+mmin;

for(int i=0;i<n;i++) a[i]=rand()%(nummax-nummin+1)+nummin;

for(int i=0;i<m;i++) b[i]=rand()%(nummax-nummin+1)+nummin;

int ans1=-1,ans2=kmp(n,m,a,b,next);

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

bool found=true;

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

if(i+j>=n){

found=false;

break;

}

if(a[i+j]!=b[j]){

found=false;

break;

}

}

if(found){

ans1=i+1;

break;

}

}

if(ans1!=ans2){

printf("error\na:");

for (int i=0;i<n;i++) printf("%d ",a[i]);

printf("\nb:");

for (int i=0;i<m;i++) printf("%d ",b[i]);

printf("\nnext:");

for (int i=0;i<m;i++) printf("%d ",next[i]);

printf("ans1:%d,ans2:%d\n",ans1,ans2);

return 0;

}

if(ans1!=-1) printf("%d %d %d\n",n,m,ans1);

}

printf("correct\n");

return 0;

}

int t;

scanf("%d",&t);

while(t--){

int n,m,a[MAXN],b[MAXM],next[MAXM];

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

for (int i=0;i<n;i++) scanf("%d",&a[i]);

for (int i=0;i<m;i++) scanf("%d",&b[i]);

printf("%d\n",kmp(n,m,a,b,next));

}

return 0;

}

### √-HDU-1686-Oulipo

【题意】求b字符串在a字符串中的出现次数，字符为大写字母，长度范围1e6

【题解】KMP简单题，发现之前做过一次，将b字符串末尾添加1个不可能被匹配到的字符（如’#’），正常求next数组，在作匹配时b字符串只要’#’之前的字符都被匹配就计数1次，继续匹配，而不return；在提交前通过了随机测试

【代码】

#include <bits/stdc++.h>

#define MAXLEN 1000010

#define random\_debug false

using namespace std;

void cal\_next(char s[],int len,int next[]){

next[0]=0;

int i=1,j=0;

while(i<len){

if(s[i]==s[j]){

next[i++]=(j++)+1;

continue;

}

while(s[i]!=s[j]){

if(j==0){

next[i++]=0;

break;

}

j=next[j-1];

}

}

return;

}

int kmp(char a[],char b[],int lena,int lenb,int next[]){

b[lenb++]='#',b[lenb]='\0';

cal\_next(b,lenb,next);

int cnt=0,j=0;

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

while(a[i]!=b[j]){

if(j==0) break;

j=next[j-1];

}

if(a[i]==b[j]){

j++;

if(j==lenb-1) cnt++;

continue;

}

}

return cnt;

}

int main(){

if(random\_debug){

int debug\_case=10000;

srand((unsigned)time(NULL));

while(debug\_case--){

int lenamin=1,lenamax=1000,lena,lenbmin=1,lenbmax=10,lenb,chmin='A',chmax='Z',next[MAXLEN];

lena=rand()%(lenamax-lenamin+1)+lenamin,lenb=rand()%(lenbmax-lenbmin+1)+lenbmin;

char a[MAXLEN],b[MAXLEN];

for (int i=0;i<lena;i++) a[i]=rand()%(chmax-chmin+1)+chmin;

for (int i=0;i<lenb;i++) b[i]=rand()%(chmax-chmin+1)+chmin;

int ans1=0,ans2=kmp(a,b,lena,lenb,next);

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

bool check=true;

for(int j=0;j<lenb;j++){

if(i+j>=lena){

check=false;

break;

}

if(a[i+j]!=b[j]){

check=false;

break;

}

}

if(check) ans1++;

}

if(ans1!=ans2){

printf("a:%d %s\nb:%d %s\nans1=%d,ans2=%d\nnext:",lena,a,lenb,b,ans1,ans2);

for(int i=0;i<lenb;i++) printf("%d ",next[i]);

printf("\n");

return 0;

}

if(ans1>1&&lenb>1) printf("%d %d %d\n",ans1,lena,lenb);

}

printf("correct\n");

return 0;

}

int t;

scanf("%d",&t);

while(t--){

char a[MAXLEN],b[MAXLEN];

scanf("%s%s",b,a);

int next[MAXLEN],lena=strlen(a),lenb=strlen(b);

printf("%d\n",kmp(a,b,lena,lenb,next));

}

return 0;

}

### √-HDU-2087-剪花布条

【题意】给定字符串a和b，问a最多能拆出多少个b字符串

【题解】KMP简单题。a中可能有多种选择方案，按顺序从前往后匹配一定是数量最多的。考虑这样证明。假定我们已经在a中选择了n个不想交的b串b1,b2…bn（从前往后）。对于某一个串，如果同时有2个串与它相交（不可能有3个或以上的串与它相交还保持互不相交），且替换为这2个串之后n+1个串都互不相交，则应该作出这样的替换，这种情况被替换的串一定在替换后的（更优解的中间），因此如果按照顺序选取就一定会先选到更优解而不是这个应该被替换的串。

【代码】

#include <bits/stdc++.h>

#define MAXN 1010

#define random\_debug false

using namespace std;

void cal\_next(char s[],int len,int next[]){

next[0]=0;

int i=1,j=0;

while(i<len){

if(s[i]==s[j]){

next[i]=j+1,i++,j++;

continue;

}

while(s[i]!=s[j]){

if(j==0){

next[i++]=0;

break;

}

j=next[j-1];

}

}

return;

}

int kmp(char a[],char b[],int lena,int lenb,int next[]){

cal\_next(b,lenb,next);

int cnt=0;

int j=0;

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

while(a[i]!=b[j]){

if(j==0) break;

j=next[j-1];

}

if(a[i]==b[j]){

j++;

if(j==lenb){

cnt++;

j=0;

}

}

}

return cnt;

}

int main(){

if(random\_debug){

srand((unsigned)time(NULL));

int debugcase=1000;

while(debugcase--){

char a[MAXN],b[MAXN],chmin='a',chmax='d';

int lenamin=1,lenamax=1000,lena=rand()%(lenamax-lenamin+1)+lenamin,lenbmin=1,lenbmax=100,lenb=rand()%(lenbmax-lenbmin+1)+lenbmin,next[MAXN];

for(int i=0;i<lena;i++) a[i]=rand()%(chmax-chmin+1)+chmin;

for(int i=0;i<lenb;i++) b[i]=rand()%(chmax-chmin+1)+chmin;

a[lena]=b[lenb]='\0';

int ans1=0,ans2=kmp(a,b,lena,lenb,next);

int i=0;

while(i<lena){

bool match=true;

int j;

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

if(i+j>=lena){

match=false;

break;

}

if(a[i+j]!=b[j]){

match=false;

break;

}

}

if(match){

ans1++,i=i+j;

}else{

i++;

}

}

if(ans1!=ans2){

printf("error\na:%s\nb:%s\nans1:%d,ans2:%d\n",a,b,ans1,ans2);

return 0;

}

//if(ans1>0&&lenb>1) printf("a:%s\nb:%s\nans1:%d,ans2:%d\n",a,b,ans1,ans2);

}

printf("correct\n");

return 0;

}

char a[MAXN],b[MAXN];

int lena,lenb,next[MAXN];

while(~scanf("%s",a)){

if(a[0]=='#'&&strlen(a)==1) break;

scanf("%s",b);

lena=strlen(a),lenb=strlen(b);

printf("%d\n",kmp(a,b,lena,lenb,next));

}

return 0;

}

### 循环节问题

#### √-HDU-3746-Cyclic Nacklace

【题意】给定1个字符串（长度范围1e5），要求在左侧或右侧补上最少的字符，使得它的循环节数量超过1.

【题解】首先对于这个问题，在左侧补和在右侧补是一样的（包括在两侧同时补），因为假定我们在一个字符串s的左侧补上了a个字符，右侧补上了b个字符，此时它成为了一个循环串，由循环串的性质，当我们把左侧a个字符顺次添加到右侧时，不改变它的循环性质。因此我们只需要考虑添加字符在右侧的情形。

考虑1个字符串s和它自身按如下方式进行匹配。（不含完全覆盖本身的情况）

aaaassssssssss

ssssssssssbbbb

设j为公共部分的最大长度.

考虑前面的串aaaa和后面的串bbbb

如果他们是一样的话，那么s就是一个循环串，并且len(s)-j或者说a或b串的长度就是其最小循环节长度.

证明的思路如下

|aaaa|ssssssssss

ssssssssss|aaaa|  
推出

|aaaa|aaaa|ssssss

|aaaa|ssssss|aaaa|  
以此类推，证明当中的长度恰好被len(s)-j还没有想好

当a串和b串不相等时,len(s)-j就是s串通过增加最少字符成为循环串后的循环节大小

也可以类似上面的方式理解。

设unit=len(s)-j表示循环节大小

可以证明加上unit-len%unit个字符后可以变成循环节大小为unit的循环串。

不过要严格证明其为最小的话还要反证出更小的不可能。

【代码】

#include <bits/stdc++.h>

using namespace std;

const int MAXLEN=1e5+10;

void cal\_next(char s[],int len,int next[]){

next[0]=0;

int i=1,j=0;

while(i<len){

if(s[i]==s[j]){

next[i++]=(j++)+1;

continue;

}

while(s[i]!=s[j]){

if(j==0){

next[i++]=0;

break;

}

j=next[j-1];

}

}

return;

}

int kmp(char s[]){

int len=strlen(s),next[MAXLEN];

cal\_next(s,len,next);

int j=0;

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

while(s[i]!=s[j]){

if(j==0) break;

j=next[j-1];

}

if(s[i]==s[j]){

j++;

}

}

int unit=len-j;

if(unit==len) return len;

char s1[MAXLEN],s2[MAXLEN];

strncpy(s1,s,unit),strncpy(s2,s+j,unit);

if(strcmp(s1,s2)==0) return 0;

return unit-len%unit;

}

int main()

{

int t;

scanf("%d",&t);

while(t--){

char s[MAXLEN];

scanf("%s",s);

printf("%d\n",kmp(s));

}

return 0;

}

#### √-HDU-1358-Period

【题意】给定字符串s（长度范围1e6），求它所有前缀是否是循环串，如果是，输出循环节组数

【题解】KMP循环节问题的经典结论题。在HDU3746的基础上对时间复杂度有了更高要求，在求一个串是否为循环串时，我们之前用了自己匹配自己后，前后多出来的部分是否相同来判断是否为循环串。

aaaaassssssssssss

ssssssssssssbbbbb

这里实际上并不需要判断，只要看循环节长度（假定的）是否整除串的长度，如果是的话，那么前后的a串和b串一定会相同，否则就一定不相同。而且也不用再拿一个自己来匹配了，直接使用next数组即可，因为这个自己和自己匹配的过程本身就是在求next数组，对于0,…,i-1，到第i-1个字符时，总共有i个字符，考虑next[i-1]的含义就是如果下一个字符不匹配的话，就要跳到next[i-1]，也就是说，next[i-1]是前i个字符的最大前后缀相同长度，也就是我们上图中s串的大小，那么循环节长度就是i-next[i-1]，不过要排除一个next[i-1]等于0的情况，其实广义上说这种情况也是满足循环节的，只不过循环次数是1，而题目中一般要求循环次数要大于1，所以要特判一下下。

【代码】

#include <bits/stdc++.h>

#define MAXLEN 1000010

using namespace std;

void cal\_next(char s[],int len,int next[]){

next[0]=0;

int i=1,j=0;

while(i<len){

if(s[i]==s[j]){

next[i++]=(j++)+1;

continue;

}

while(s[i]!=s[j]){

if(j==0){

next[i++]=0;

break;

}

j=next[j-1];

}

}

return;

}

void kmp(char s[],int len){

int next[MAXLEN];

cal\_next(s,len,next);

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

if((i+1)%(i-next[i]+1)==0&&next[i]!=0){

printf("%d %d\n",i+1,(i+1)/(i-next[i-1]));

}

}

return;

}

int main()

{

int len,casenum=1;

char s[MAXLEN];

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

scanf("%s",s);

printf("Test case #%d\n",casenum++);

kmp(s,len);

printf("\n");

}

return 0;

}

#### √-POJ-2406-Power Strings

【题意】给定字符串，求最大循环次数

【题解】铁铜题难度；利用len-next[len-1]求出最小循环节长度，判断一下是否整除len即可。学了一下如何用new，果然直接开1e6本地codeblocks会爆栈，用了new之后就不会了。

【代码】

#include <cstring>

#include <cstdio>

#define MAXLEN 1000010

using namespace std;

void cal\_next(char s[],int len,int next[]){

next[0]=0;

int i=1,j=0;

while(i<len){

if(s[i]==s[j]){

next[i++]=(j++)+1;

continue;

}

while(s[i]!=s[j]){

if(j==0){

next[i++]=0;

break;

}

j=next[j-1];

}

}

return;

}

int kmp(char s[],int len){

int \*next=new int[len];

cal\_next(s,len,next);

int unit=len-next[len-1];

delete[] next;

if(len%unit) return 1;

return len/unit;

}

int main()

{

char \*s=new char[MAXLEN];

while(~scanf("%s",s)){

int len=strlen(s);

if(s[0]=='.') break;

printf("%d\n",kmp(s,len));

}

delete[] s;

return 0;

}

### √-HDU-2203-亲和串

【题意】给定字符串a和b判断b是否为a在经过任意次循环移位后的子串

【题解】处理序列循环移位的惯用套路-写2遍，这样的一个新的序列一定包含了所有的循环移位序列，之后就变成KMP模板题

【代码】

#include <bits/stdc++.h>

#define MAXLEN 100010

using namespace std;

void cal\_next(char s[],int len,int next[]){

next[0]=0;

int j=0,i=1;

while(i<len){

if(s[i]==s[j]){

next[i++]=(j++)+1;

continue;

}

while(s[i]!=s[j]){

if(j==0){

next[i++]=0;

break;

}

j=next[j-1];

}

}

return;

}

bool kmp(char s1[],char s2[],int len1,int len2){

int \*next=new int[len2];

strncpy(s1+len1,s1,len1+1);

len1=strlen(s1);

cal\_next(s2,len2,next);

int j=0;

for (int i=0;i<len1;i++){

while(s1[i]!=s2[j]){

if(j==0) break;

j=next[j-1];

}

if(s1[i]==s2[j]){

j++;

if(j==len2) return true;

}

}

return false;

}

int main()

{

char \*s1=new char[MAXLEN<<1],\*s2=new char[MAXLEN];

while(~scanf("%s%s",s1,s2)){

int len1=strlen(s1),len2=strlen(s2);

if(kmp(s1,s2,len1,len2)) printf("yes\n");else printf("no\n");

}

return 0;

}

### √-POJ-2752-Seek the Name,Seek the Fame

【题意】给定字符串s（长度范围4e5）要求输出所有的前缀和后缀相同的字符串长度（升序）

【题解】考虑KMP的next数组本质，对于长度为len的字符串0..len-1，next[len-1]的意思就是它（除自己本身）的最大前后缀相同的长度是next[len-1]，接着我们就只考虑这个相同的前后缀，再看除它自己本身之外的最大前后缀相同的长度，这样就是next[next[len-1]-1]…依次类推可以求出所有的前后缀相同数值。复杂度O(n)

【代码】

#include <cstdio>

#include <cstdlib>

#include <cstring>

#define MAXLEN 400010

using namespace std;

void cal\_next(char s[],int len,int next[]){

next[0]=0;

int i=1,j=0;

while(i<len){

if(s[i]==s[j]){

next[i++]=(j++)+1;

continue;

}

while(s[i]!=s[j]){

if(j==0){

next[i++]=0;

break;

}

j=next[j-1];

}

}

return;

}

void kmp(char s[],int len){

int \*next=new int[len],\*ans=new int[len],anssz=1;

ans[0]=len;

cal\_next(s,len,next);

int i=len-1;

while(i>0&&next[i]){

ans[anssz++]=next[i];

i=next[i]-1;

}

for(int i=anssz-1;i>=0;i--){

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

if(i==0) printf("\n");else printf(" ");

}

delete[] next;

delete[] ans;

return;

}

int main()

{

char \*s=new char[MAXLEN];

while(~scanf("%s",s)){

int len=strlen(s);

kmp(s,len);

}

delete[] s;

return 0;

}

### 最长公共子串问题

时间复杂度O(n^2)

## 扩展KMP

## 字典树

## AC自动机

## 后缀数组

# 六、树状数组&线段树

<https://blog.csdn.net/trapper_c/article/details/51919980>

## 线段树：区间更新

### √-HDU-6315-多校2铜牌题

【题意】b数组是1-n的一个排列，a数组开始全0，更新操作为将一段区间的a数组都加1，查询操作为查询一段区间的

【题解】每个区间维护差几加几（差left加add）

【自测数据】

debug设为true，生成数据的答案，debughuge可以测试大n下程序的性能（自动生成b数组）

/\*

input

5 12

1 5 2 4 3

add 1 4

query 1 4

add 2 5

query 2 5

add 3 5

query 1 5

add 2 4

query 1 4

add 2 5

query 2 5

add 2 2

query 1 5

6 8

6 5 4 3 2 1

add 1 3

add 1 3

query 2 4

add 2 4

query 1 6

add 1 6

add 2 6

query 3 6

7 8

3 5 1 2 4 7 6

add 1 3

add 1 3

query 2 4

add 2 4

query 1 6

add 1 6

add 2 6

query 3 6

100000 8

add 1 3

add 1 3

query 2 4

add 2 4

query 1 6

add 1 6

add 2 6

query 3 6

output

1

1

2

4

4

6

0

0

5

2

3

6

1

4

1

\*/

【代码】

#include <bits/stdc++.h>

#define MAXN 100010

#define ll long long

#define debug false

#define debughuge false

using namespace std;

struct node{

int l,r;

ll sum,store,lazy,left,add;

}tree[MAXN<<2];

ll b[MAXN];

void push\_up(int id);

void push\_down(int id);

void build\_tree(int id,int l,int r);

void update(int id,int l,int r,ll num);

ll query(int id,int l,int r);

void push\_up(int id){

tree[id].sum=tree[id<<1].sum+tree[id<<1|1].sum;

tree[id].store=min(tree[id<<1].store,tree[id<<1|1].store);

if(tree[id<<1].left==tree[id<<1|1].left){

tree[id].left=tree[id<<1].left,tree[id].add=tree[id<<1].add+tree[id<<1|1].add;

}else if(tree[id<<1].left<tree[id<<1|1].left){

tree[id].left=tree[id<<1].left,tree[id].add=tree[id<<1].add;

}else{

tree[id].left=tree[id<<1|1].left,tree[id].add=tree[id<<1|1].add;

}

return;

}

void push\_down(int id){

if(tree[id].lazy){

update(id<<1,tree[id<<1].l,tree[id<<1].r,tree[id].lazy);

update(id<<1|1,tree[id<<1|1].l,tree[id<<1|1].r,tree[id].lazy);

tree[id].lazy=0;

}

return;

}

void build\_tree(int id,int l,int r){

tree[id].l=l,tree[id].r=r,tree[id].lazy=tree[id].store=tree[id].sum=0;

if(l==r){

tree[id].left=b[l],tree[id].add=1;

return;

}

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

build\_tree(id<<1,l,mid),build\_tree(id<<1|1,mid+1,r);

push\_up(id);

return;

}

void update(int id,int l,int r,ll num){

if(tree[id].l==l&&tree[id].r==r){

if(l==r){

tree[id].store+=num;

while(tree[id].store>=tree[id].left){

tree[id].store-=tree[id].left;

tree[id].sum++;

tree[id].left=b[l];

tree[id].add=1;

}

tree[id].left-=tree[id].store;

tree[id].store=0;

}else{

tree[id].lazy+=num;

tree[id].store+=num;

while(tree[id].store>=tree[id].left){

tree[id].store-=tree[id].left;

tree[id].sum+=tree[id].add;

tree[id].left=0;

push\_down(id);

push\_up(id);

}

tree[id].left-=tree[id].store;

tree[id].store=0;

}

return;

}

push\_down(id);

int leftr=tree[id<<1].r,rightl=tree[id<<1|1].l;

if(r<=leftr){

update(id<<1,l,r,num);

}else if(l>=rightl){

update(id<<1|1,l,r,num);

}else{

update(id<<1,l,leftr,num),update(id<<1|1,rightl,r,num);

}

push\_up(id);

return;

}

ll query(int id,int l,int r){

if(tree[id].l==l&&tree[id].r==r){

return tree[id].sum;

}

push\_down(id);

ll ans;

int leftr=tree[id<<1].r,rightl=tree[id<<1|1].l;

if(r<=leftr){

ans=query(id<<1,l,r);

}else if(l>=rightl){

ans=query(id<<1|1,l,r);

}else{

ans=query(id<<1,l,leftr)+query(id<<1|1,rightl,r);

}

push\_up(id);

return ans;

}

int main()

{

if(debug){

int n,q;

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

if(debughuge) for (int i=1;i<=n;i++) b[i]=i;

else{

for (int i=1;i<=n;i++) scanf("%I64d",&b[i]);

}

int a[MAXN];

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

char op[20];

int l,r;

while(q--){

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

if(op[0]=='a'){

for (int i=l;i<=r;i++) a[i]++;

}else if(op[0]=='q'){

ll ans=0;

for (int i=l;i<=r;i++) ans+=a[i]/b[i];

printf("%I64d\n",ans);

}

}

}

return 0;

}

int n,q;

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

if(debughuge) for (int i=1;i<=n;i++) b[i]=i;

else{

for (int i=1;i<=n;i++) scanf("%I64d",&b[i]);

}

build\_tree(1,1,n);

char op[20];

int l,r;

while(q--){

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

if(op[0]=='a'){

update(1,l,r,1);

}else if(op[0]=='q'){

printf("%I64d\n",query(1,l,r));

}

}

}

return 0;

}

## 线段树：区间合并

### √-POJ-3667-Hotel（线段树区间合并模板题）

【题意】

n个房间，初始时均为空，2种操作，第1种操作是找到指定的k个 连续空房间，如果找得到的话输出第1个房间号，否则输出0，优先找房间号小的，第2种操作是指定连续的x号到y号房间清空。数据规模n为50000.

【代码】

#include <iostream>

#include <cstdio>

#include <algorithm>

#define MAXN 50010

#define IN 1

#define OUT 2

using namespace std;

struct node{

int l,r,lmax,rmax,totmax,lazytag;

}tree[MAXN<<2];

void build\_tree(int id,int l,int r);

void push\_up(int id);

void push\_down(int id);

void check\_in(int id,int l,int r);

void check\_out(int id,int l,int r);

int query\_first(int id,int d);

void push\_up(int id){

tree[id].lmax=(tree[id<<1].lmax==tree[id<<1].r-tree[id<<1].l+1)?tree[id<<1].lmax+tree[id<<1|1].lmax:tree[id<<1].lmax;

tree[id].rmax=(tree[id<<1|1].rmax==tree[id<<1|1].r-tree[id<<1|1].l+1)?tree[id<<1].rmax+tree[id<<1|1].rmax:tree[id<<1|1].rmax;

tree[id].totmax=max(max(tree[id<<1].totmax,tree[id<<1|1].totmax),tree[id<<1].rmax+tree[id<<1|1].lmax);

return;

}

void push\_down(int id){

if (tree[id].lazytag==IN){

tree[id<<1].lmax=tree[id<<1].rmax=tree[id<<1].totmax=tree[id<<1|1].lmax=tree[id<<1|1].rmax=tree[id<<1|1].totmax=0;

tree[id<<1].lazytag=tree[id<<1|1].lazytag=IN;

}else if (tree[id].lazytag==OUT){

tree[id<<1].lmax=tree[id<<1].rmax=tree[id<<1].totmax=tree[id<<1].r-tree[id<<1].l+1;

tree[id<<1|1].lmax=tree[id<<1|1].rmax=tree[id<<1|1].totmax=tree[id<<1|1].r-tree[id<<1|1].l+1;

tree[id<<1].lazytag=tree[id<<1|1].lazytag=OUT;

}

tree[id].lazytag=0;

return;

}

void build\_tree(int id,int l,int r){

tree[id].l=l,tree[id].r=r,tree[id].lazytag=0;

if (tree[id].l==tree[id].r){

tree[id].lmax=tree[id].rmax=tree[id].totmax=1;

return;

}

int mid=(l+r)/2;

build\_tree(id<<1,l,mid),build\_tree(id<<1|1,mid+1,r);

push\_up(id);

return;

}

int query\_first(int id,int d){

if (tree[id].totmax<d) return 0;

if (tree[id].totmax==d&&tree[id].totmax==tree[id].r-tree[id].l+1){

return tree[id].l;

}

push\_down(id);

if (tree[id<<1].totmax>=d){

return query\_first(id<<1,d);

}else if (tree[id<<1].rmax+tree[id<<1|1].lmax>=d){

return tree[id<<1].r-tree[id<<1].rmax+1;

}else{

return query\_first(id<<1|1,d);

}

return 0;

}

void check\_in(int id,int l,int r){

if (tree[id].l==l&&tree[id].r==r){

tree[id].lmax=tree[id].rmax=tree[id].totmax=0;

tree[id].lazytag=IN;

return;

}

push\_down(id);

int leftr=tree[id<<1].r,rightl=tree[id<<1|1].l;

if (r<=leftr){

check\_in(id<<1,l,r);

}else if (l>=rightl){

check\_in(id<<1|1,l,r);

}else{

check\_in(id<<1,l,leftr),check\_in(id<<1|1,rightl,r);

}

push\_up(id);

return;

}

void check\_out(int id,int l,int r){

if (tree[id].l==l&&tree[id].r==r){

tree[id].lmax=tree[id].rmax=tree[id].totmax=tree[id].r-tree[id].l+1;

tree[id].lazytag=OUT;

return;

}

push\_down(id);

int leftr=tree[id<<1].r,rightl=tree[id<<1|1].l;

if (r<=leftr){

check\_out(id<<1,l,r);

}else if (l>=rightl){

check\_out(id<<1|1,l,r);

}else{

check\_out(id<<1,l,leftr),check\_out(id<<1|1,rightl,r);

}

push\_up(id);

return;

}

int main()

{

int n,m;

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

build\_tree(1,1,n);

while (m--){

int op;

scanf("%d",&op);

switch(op)

{

case 1:

{

int d;

scanf("%d",&d);

int ans=query\_first(1,d);

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

if (ans>0) check\_in(1,ans,ans+d-1);

break;

}

case 2:

{

int x,d;

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

check\_out(1,x,x+d-1);

break;

}

}

}

return 0;

}

## 离散化&扫描线

### √-HDU-1542-Atlantis（矩形面积并）

【题意】

给定n个矩形（题目给出每个矩形的左下和右上点坐标，浮点数，坐标范围0-100000，最多100个），求面积并

【题解】

选定横坐标离散化（这样最多100\*2个），自下而上扫描，碰到下面的边就把离散化后的相应区间+1，碰到上面的边就-1，从0到1时当前横坐标和增加该离散化区间的原始长度，从1到0则是减少。

【代码】

#include <cstdio>

#include <algorithm>

#include <cstring>

#define MAXN 110

using namespace std;

struct node{

int id,fs,ft;

double x,y;

}dot[MAXN<<2];

typedef struct node node;

bool cmp\_x(node x,node y){return x.x<y.x;}

bool cmp\_y(node x,node y){return x.y<y.y;}

bool cmp\_id(node x,node y){return x.id<y.id;}

int main()

{

int n,testcase=1,cnt[MAXN<<2];

double ans,totx,lasty,f[MAXN<<2];

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

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

dot[2\*i-1].id=2\*i-1,dot[2\*i].id=2\*i;

scanf("%lf%lf%lf%lf",&dot[2\*i-1].x,&dot[2\*i-1].y,&dot[2\*i].x,&dot[2\*i].y);

}

sort(dot+1,dot+2\*n+1,cmp\_x);

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

if (dot[i].id%2==1) dot[i].fs=i; else dot[i].ft=i;

if(i<2\*n) f[i]=dot[i+1].x-dot[i].x;

}

sort(dot+1,dot+2\*n+1,cmp\_id);

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

if (dot[i].id%2==1) dot[i].ft=dot[i+1].ft;else dot[i].fs=dot[i-1].fs;

}

sort(dot+1,dot+2\*n+1,cmp\_y);

ans=totx=lasty=0;

memset(cnt,0,sizeof(cnt));

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

ans+=totx\*(dot[i].y-lasty);

lasty=dot[i].y;

for (int j=dot[i].fs;j<dot[i].ft;j++){

if (dot[i].id%2==0){

cnt[j]--;

if (cnt[j]==0) totx-=f[j];

}else{

cnt[j]++;

if (cnt[j]==1) totx+=f[j];

}

}

}

printf("Test case #%d\nTotal explored area: %.2lf\n\n",testcase++,ans);

}

return 0;

}

### √-HDU-5862-2016多校10-铜牌

【题意】给定n个与坐标轴平行的线段（以端点坐标给出），求解交点个数，n的规模为10^5，坐标的原始规模为-10^9-10^9

【题解】首先离散化，使得坐标规模缩小到10^5，接着使用扫描线，一开始用了错误的算法，本想以up\_num[y]表示在y以上的那些以竖线上端点在y以上的个数，down\_num[y]表示竖线下端点在y以下的个数，这样2者相加就是完全在上+2\*相交+完全在下，再减去整体的数量，就可以得到相交到的数量，但是这里由于是线段，所以要求的上述数量都是在一个范围的，总体数量还可以用线段树在nlogn级别解决，但是在上和在下就涉及到二维线段树，时间复杂度无法降低。

正确算法是采用扫描线，自上而下扫描，对于同一纵坐标，先处理竖直线上端点，单点更新线段树+1，再处理水平线，区间查询增加ans值，再处理竖直线下端点，单点更新-1。

另外这里线段树可以不用建树，只需维护cnt即可，可以节省一些时间和空间的复杂度。

要注意最后的ans是会爆int的，水平线和竖直线各n/2完全相交的情况可以达到n^2/4。

HDU上提交的时候选用C++编译器会TLE，选用G++编译器可AC

【代码】

#include <cstdio>

#include <algorithm>

#include <iostream>

#include <cstring>

#define MAXN 100010

#define ll long long//会爆int,可能的最大值是n^2/4

#define VERTICAL\_UP 1

#define HORIZONTAL 2

#define VERTICAL\_DOWN 3

using namespace std;

/\*

vertex从1开始标号，奇数代表左（上）端点，偶数是右（下）端点

按照id排序的话,1..2\*hori\_num都是水平线的端点,2\*hori\_num+1..2\*n都是竖直线的端点,n-hori\_num=verti\_num

\*/

struct node{

int id,x,y,type;

}vertex[MAXN<<2];

bool cmp\_id(node x,node y){return x.id<y.id;}

bool cmp\_x(node x,node y){return x.x<y.x;}

bool cmp\_y(node x,node y){return x.y<y.y;}

bool cmp(node x,node y){

if (x.y==y.y){

if(x.type==y.type) return x.id<y.id;

return x.type<y.type;

}

return x.y>y.y;

}

int cnt[MAXN<<4];

void update(int id,int l,int r,int num,int flag){

if(l==r&&l==num){cnt[id]+=flag;return;}

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

if(num<=mid) update(id<<1,l,mid,num,flag);else update(id<<1|1,mid+1,r,num,flag);

cnt[id]=cnt[id<<1]+cnt[id<<1|1];

return;

}

int query(int id,int l,int r,int ql,int qr){

if(l==ql&&r==qr) return cnt[id];

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

if(qr<=mid) return query(id<<1,l,mid,ql,qr);

else if (ql>=mid+1)return query(id<<1|1,mid+1,r,ql,qr);

else return query(id<<1,l,mid,ql,mid)+query(id<<1|1,mid+1,r,mid+1,qr);

}

int main()

{

int t;

scanf("%d",&t);

while(t--){

int n,verti\_num=0,hori\_num=0;

scanf("%d",&n);

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

int x1,y1,x2,y2;

scanf("%d%d%d%d",&x1,&y1,&x2,&y2);

if(x1==x2){//竖直线从后往前存

if(y1<y2) swap(y1,y2);//确保奇数点是上端点

verti\_num++;

vertex[2\*(n-verti\_num+1)-1].type=VERTICAL\_UP,vertex[2\*(n-verti\_num+1)].type=VERTICAL\_DOWN;

vertex[2\*(n-verti\_num+1)-1].id=2\*(n-verti\_num+1)-1,vertex[2\*(n-verti\_num+1)].id=2\*(n-verti\_num+1);

vertex[2\*(n-verti\_num+1)-1].x=x1,vertex[2\*(n-verti\_num+1)-1].y=y1;

vertex[2\*(n-verti\_num+1)].x=x2,vertex[2\*(n-verti\_num+1)].y=y2;

}else if(y1==y2){//水平线从前往后存

if(x1>x2) swap(x1,x2);//确保奇数点是左端点

hori\_num++;

vertex[2\*hori\_num-1].type=vertex[2\*hori\_num].type=HORIZONTAL;

vertex[2\*hori\_num-1].id=2\*hori\_num-1,vertex[2\*hori\_num].id=2\*hori\_num;

vertex[2\*hori\_num-1].x=x1,vertex[2\*hori\_num-1].y=y1;

vertex[2\*hori\_num].x=x2,vertex[2\*hori\_num].y=y2;

}

}

//离散化,x坐标和y坐标分别离散化,注意sort的数组下标范围,排序0..n-1时下标为+0,+n,排序1..n时下标为+1,+n+1

sort(vertex+1,vertex+2\*n+1,cmp\_x);

int last\_x=vertex[1].x,cnt\_x=1;

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

if(vertex[i].x==last\_x) vertex[i].x=cnt\_x; else last\_x=vertex[i].x,vertex[i].x=++cnt\_x;

}

sort(vertex+1,vertex+2\*n+1,cmp\_y);

int last\_y=vertex[1].y,cnt\_y=1;

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

if(vertex[i].y==last\_y) vertex[i].y=cnt\_y;else last\_y=vertex[i].y,vertex[i].y=++cnt\_y;

}

sort(vertex+1,vertex+2\*n+1,cmp\_id);

//开始建树，用于查询一段x区间上竖直线的个数

memset(cnt,0,sizeof(cnt));

//正确算法为自上而下扫描，遇到竖直线的上端点则update+1,有水平线则查询,有竖直线下端点则update-1，顺序不能错

ll ans=0;

sort(vertex+1,vertex+2\*n+1,cmp);

int nowk=1;

while(nowk<=2\*n){

if(vertex[nowk].type==VERTICAL\_UP) update(1,1,cnt\_x,vertex[nowk++].x,1);

else if (vertex[nowk].type==VERTICAL\_DOWN) update(1,1,cnt\_x,vertex[nowk++].x,-1);

else if (vertex[nowk].type==HORIZONTAL){

ans+=query(1,1,cnt\_x,vertex[nowk].x,vertex[nowk+1].x),nowk+=2;

}

}

cout<<ans<<endl;

}

return 0;

}

## 二维线段树

### TLE-POJ-2155-Matrix

【题意】

【题解】

<https://blog.csdn.net/u013761036/article/details/46363605>

## √-HDU-5861-2016多校10-铜牌

【题干】

一条直线上n个村庄，n-1个道路，每个道路有一个每天开放的费用。开始的时候所有道路关闭，现在已知m天的通行计划，每天 给出a和b，要求a到b之间的道路开放，每个道路只能开放和关闭一次，要求总费用最小。

样例输入为4个村庄，3天计划，道路费用1,2,3，接下来3天计划

**Sample Input**

4 3

1 2 3

1 3

3 4

2 4

**Sample Output**

3

5

5

【题解】

200000的数据规模，想知道一个道路最早被使用的时间和最晚被使用的时间（如果被使用到的话），那么道路在此之前和之后都是关闭的，这样是最小的费用，假设我们知道了每个道路的最早和最晚使用时间，那么扫一遍时间，每天加上开放的，减去关闭的，这样输出费用复杂度为O（3\*m）（每个道路最多出现2次）；它每次给出信息对一段区间进行操作，使用线段树，维护每个从开始到结束时每个节点得到的最大值和最小值（注意这里的最大值最小值不是区间的最大值和最小值，而是整个历史记录的最大值和最小值）

【代码】

#include <cstdio>

#include <vector>

#include <algorithm>

#define MAXN 200010

using namespace std;

struct node{

int left,right,maxnum,minnum;

bool used,lazy;

}tree[MAXN<<2];

vector<int> open[MAXN],close[MAXN];

void build\_tree(int id,int l,int r){

tree[id].left=l,tree[id].right=r,tree[id].used=tree[id].lazy=false;

if (tree[id].left==tree[id].right){

return;

}

int mid=(l+r)/2;

build\_tree(id<<1,l,mid),build\_tree(id<<1|1,mid+1,r);

return;

}

void update(int id,int l,int r,int t){

if (tree[id].left==l&&tree[id].right==r){

if (tree[id].used){

tree[id].maxnum=max(tree[id].maxnum,t),tree[id].minnum=min(tree[id].minnum,t);

}else{

tree[id].used=true,tree[id].maxnum=tree[id].minnum=t;

}

tree[id].lazy=true;

return;

}

if (tree[id].lazy){

update(id<<1,tree[id<<1].left,tree[id<<1].right,tree[id].maxnum);

update(id<<1,tree[id<<1].left,tree[id<<1].right,tree[id].minnum);

update(id<<1|1,tree[id<<1|1].left,tree[id<<1|1].right,tree[id].maxnum);

update(id<<1|1,tree[id<<1|1].left,tree[id<<1|1].right,tree[id].minnum);

tree[id].lazy=false;

}

int leftr=tree[id<<1].right,rightl=tree[id<<1|1].left;

if (r<=leftr){

update(id<<1,l,r,t);

}else if (l>=rightl){

update(id<<1|1,l,r,t);

}else{

update(id<<1,l,leftr,t),update(id<<1|1,rightl,r,t);

}

return;

}

void query(int id,int l,int r,int &minnum,int &maxnum,bool &used){

if (tree[id].left==l&&tree[id].right==r){

maxnum=tree[id].maxnum,minnum=tree[id].minnum,used=tree[id].used;

return;

}

if (tree[id].lazy){

update(id<<1,tree[id<<1].left,tree[id<<1].right,tree[id].maxnum);

update(id<<1,tree[id<<1].left,tree[id<<1].right,tree[id].minnum);

update(id<<1|1,tree[id<<1|1].left,tree[id<<1|1].right,tree[id].maxnum);

update(id<<1|1,tree[id<<1|1].left,tree[id<<1|1].right,tree[id].minnum);

tree[id].lazy=false;

}

int leftr=tree[id<<1].right,rightl=tree[id<<1|1].left;

if (r<=leftr){

query(id<<1,l,r,minnum,maxnum,used);

}else if (l>=rightl){

query(id<<1|1,l,r,minnum,maxnum,used);

}else{

query(id<<1,l,leftr,minnum,maxnum,used),query(id<<1|1,rightl,r,minnum,maxnum,used);

}

return;

}

int main()

{

int n,m,w[MAXN];

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

build\_tree(1,1,n-1);

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

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

int a,b;

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

if (a>b) swap(a,b);

update(1,a,b-1,t);

}

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

open[i].clear(),close[i].clear();

}

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

int minnum,maxnum;

bool used;

query(1,i,i,minnum,maxnum,used);

if (used) open[minnum].push\_back(i),close[maxnum+1].push\_back(i);

}

int money=0;

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

for (int j=0;j<(int)open[i].size();j++) money+=w[open[i][j]];

for (int j=0;j<(int)close[i].size();j++) money-=w[close[i][j]];

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

}

}

return 0;

}

## 主席树

# 七、数学

## 补题

### 银牌题-前置：指数循环节问题/扩展欧拉定理

牛客网2018暑期多校第4场-A

【题意】一个包含0、1、2的字符串，每秒每个1后面会多1个0，每个2后面会多1个1，在每一秒结束时，第1个字符会被删除。给定字符串（长度1e5，所有测试数据总长度数量级1e6），输出该字符串多久变为空串，如果永远不会变为空串，输出-1.

【题解】

解决本题要经过3道关卡

第一关：找规律得到递推式

首先，分析出一个字符串不会永远为空串。考虑1个字符串在第1个2之前出现的第1个1，其之前只有0，所以这个1会在有限时间内被消灭，考虑第1个2之前出现的第2个1，由于它前面的1会在有限的时间被消灭，之后第2个1也会变为与第1个1相同的情形，依次类推，第1个2之前的所有1都会在有限时间内被消灭，因此经过有限时间第1个2之前只有0，因此第1个2也会在有限时间被消灭，此时之后的1和2就相继在有限时间内变成了第1个1和第1个2，同理，所有的数都将在有限时间内被消灭。

考虑当消灭初始串的1个0时，需要花1个单位时间。

而当消灭初始串的1个1时，假定之前已经过去了n个单位时间，那么这个1已经繁衍出了n个0，于是要花费n+1单位时间才能将初始串的这个1及其后代都消灭。

当消灭初始串的1个2时，假定之前已经过去了n个单位时间，则2在这n个单位时间的繁衍规律如下：

2

21

2110

2101001000

2110100100010000

输入模拟程序，得到存活序列为

8,19,42，89,184…

（0）找（E）规（I）律（S）后可以发现满足如下式子

a[n]=3\*2^（n+1）-（n+1）-2

即扫描原串，记录当前已经经过的时间t

遇到0 t=t+1

遇到1 t=2\*t+2

遇到2 t=3\*2^(t+1)-3

第二关：正确取模-应用扩展欧拉定理（指数循环节问题）

当通过第1关后就可以过样例了，然后就可喜的WA了。然后就会发现这里会有指数循环取模的问题，所以应用扩展欧拉定理进行解决，要维护模phi(N)、phi(phi(N))……的值。

第三关：优化时间复杂度

当弥补了取模的BUG后，就会可喜的发现有TLE了，于是要进行时间复杂度的优化……这里并没有理解的很到位，首先预处理掉要用到的phi值是很好理解的，毕竟这里phi值不多，能节省一些时间复杂度，单单这样还是不能解决TLE的问题。发现了网上可以AC的程序中有一个使用了DFS，可能由于对于上一层phi的维护值不需要算完，于是就可以剪枝。至于标程中顺推过程遇到2的优化，还没有很理解。可以考虑先解决以下问题（指数循环节）：

<https://blog.csdn.net/acdreamers/article/details/8236942>

【代码1】//dfs版本

#include <bits/stdc++.h>

#define MAXN 100010

#define ll long long

#define debug false

using namespace std;

char s[MAXN];

const ll MODN=1000000007ll;

map<ll,ll> phi;

const ll modnum[29] = {

1000000007, 1000000006, 500000002, 243900800, 79872000,

19660800, 5242880, 2097152, 1048576, 524288,

262144, 131072, 65536, 32768, 16384,

8192, 4096, 2048, 1024, 512,

256, 128, 64, 32, 16,

8, 4, 2, 1

};

const ll modcnt=28;

ll fast\_pow(ll a,ll b,ll N){

ll eN=phi[N];

if(b<eN){

a%=N;

ll base=a,ans=1ll;

while(b){

if(b&1) ans=(ans\*base)%N;

b>>=1;

base=(base\*base)%N;

}

return ans;

}else{

b=b%eN+eN;

a%=N;

ll base=a,ans=1ll;

while(b){

if(b&1) ans=(ans\*base)%N;

b>>=1;

base=(base\*base)%N;

}

return ans;

}

}

ll dfs(ll i,ll mod){

if(i==-1) return 0;

if(s[i]=='0') return (dfs(i-1,mod)+1)%mod;

if(s[i]=='1') return (2\*dfs(i-1,mod)+2)%mod;

return ((3\*fast\_pow(2,dfs(i-1,phi[mod])+1,mod))%mod-3+mod)%mod;

}

int main(){

int t;

scanf("%d",&t);

phi.clear();

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

phi[modnum[i]]=modnum[i+1];

}

phi[1]=1;

while(t--){

scanf("%s",s);

ll len=strlen(s),m[MAXN];

memset(m,0,sizeof(m));

cout<<dfs(len-1,MODN)<<endl;

}

return 0;

}

【代码2】//标程-顺推版本（优化方法不明）

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using int64 = long long;

const int mods[29] = {

1000000007, 1000000006, 500000002, 243900800, 79872000,

19660800, 5242880, 2097152, 1048576, 524288,

262144, 131072, 65536, 32768, 16384,

8192, 4096, 2048, 1024, 512,

256, 128, 64, 32, 16,

8, 4, 2, 1

};

const int N = 1e6 + 10, S = 262144 \* 3;

int p2[29][S];

char s[N];

int64 pow\_mod(int64 a, int64 n, int64 mod) {

if ((mod & (mod - 1)) == 0 && n > 30) return 0;

int64 r = 1 % mod;

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

if (n & 1) r = r \* a % mod;

a = a \* a % mod;

}

return r;

}

int main() {

for (int i = 0; i < 28; ++i) {

p2[i][0] = 1;

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

p2[i][j] = p2[i][j - 1] \* 2;

if (p2[i][j] >= mods[i]) p2[i][j] -= mods[i];

}

}

int T;

scanf("%d", &T);

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

std::vector<int> ret(29);

scanf("%s", s);

int n = strlen(s), now = 0;

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

if (s[i] == '0') {

now = std::min(now + 1, mods[0]);

for (int j = 0; j < 28; ++j) {

++ret[j];

if (ret[j] >= mods[j]) ret[j] -= mods[j];

}

} else if (s[i] == '1') {

now = std::min(now \* 2 + 2, mods[0]);

for (int j = 0; j < 28; ++j) {

ret[j] = ret[j] \* 2 + 2;

ret[j] %= mods[j];

}

} else if (s[i] == '2') {

// x -> 3 \* (2 ^ (x + 1) - 1)

for (int j = 0, e; j < 28; ++j) {

if (now + 1 < mods[j + 1]) e = now + 1;

else e = ret[j + 1] + 1 + mods[j + 1];

if (e < S) ret[j] = 3ll \* (p2[j][e] - 1) % mods[j];

else ret[j] = 3 \* (pow\_mod(2, e, mods[j]) - 1) % mods[j];

if (ret[j] < 0) ret[j] += mods[j];

}

if (now <= 30) {

now = std::min<long long>(mods[0], (3ll << (now + 1)) - 3);

} else {

now = mods[0];

}

}

}

printf("%d\n", ret[0]);

}

return 0;

}

### 签到题-可做

2018杭电暑期多校3F-Grab The Tree-HDU6324

【题意】

【题解】

【代码】

### 签到题-可做

2018杭电暑期多校3D-HDU6322-Euler Function

【题意】

【题解】

【代码】

## OEIS

<https://oeis.org/>

## 快速幂

### 一般快速幂

ll fast\_pow(ll a,ll b,ll N){

a%=N;

ll base=a,ans=1;

while(b){

if(b&1) ans=(ans\*base)%N;

b>>=1;

base=(base\*base)%N;

}

return ans;

}

### 根据递推式构造系数矩阵

<https://blog.csdn.net/u012061345/article/details/52224623>

### 矩阵快速幂

## 数论

### 逆元

x对mod求逆元，应用费马小定理，x和mod互素，一般题目给的mod本身就是素数，于是所以x的逆元就是

使用快速幂求解

ll inv(ll x,ll mod)

{

ll k=mod-2,ans=1;

while(k)

{

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

x=(x\*x)%mod;

k>>=1;

}

return ans;

}

### 欧拉函数

//单个欧拉值

ll euler(ll x)

{

ll num=x;

for (ll i=2ll;i\*i<=x;i++)

{

if (x%i==0ll)

{

num=(num/i)\*(i-1ll);

while (x%i==0ll) x/=i;

}

}

if (x!=1ll) num=(num/x)\*(x-1ll);

return num;

}

//单个欧拉值

//**模板未验证**

#include <cmath>

int GetPhi(int n)

{

int m = sqrt(n+0.5);

int ans = n;

for(int i=2; i<=m; ++i) if(n%i == 0)

{

ans = ans/i \* (i-1);

while(n%i == 0) n /= i;

}

if(n > 1) ans = ans/n \* (n-1);

return ans;

}

//下面的代码求出了1~n中所有数的欧拉函数值，并保存在了phi数组中

//**模板未验证**

const int MAXN = 3e6+5;

\_\_int64 phi[MAXN];

void PhiTable(int n)

{

for(int i=2; i<=n; ++i)

phi[i] = 0;

phi[1] = 1;

for(int i=2; i<=n; ++i) if((!phi[i]))

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

{

if(!phi[j]) phi[j] = j;

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

}

}

//求出了1~n中所有数的欧拉函数值的前缀和，并保存在了f数组中

//**模板未验证**

const int MAXN = 3e6+5;

\_\_int64 f[MAXN];

void SumPhiTable(int n)

{

f[1] = 1;

for(int i=2; i<n; ++i)

{

if(!f[i])

{

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

{

if(!f[j]) f[j] = j;

f[j] = f[j]/i \* (i-1);

}

}

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

}

}

### 威尔逊定理

p为素数，则((p-1)! + 1) % p == 0

### 幂取模(欧拉定理、费马小定理、扩展欧拉定理)

#### 欧拉定理



#### 费马小定理



#### 扩展欧拉定理

（不要求a,c互质）

http://dl.iteye.com/upload/attachment/531761/b2c93169-8323-37f9-8dda-4999cb233343.jpg

当x<phi(C)时直接使用快速幂即可

#### 指数循环节问题

<https://blog.csdn.net/acdreamers/article/details/8236942>

FZU-1759

HDU-2837

ZOJ-1674

HDU-4335

<https://blog.csdn.net/wust_zzwh/article/details/51966450>

（HDU-5728-2016多校）

### 不定方程

#### √-HDU-6298-2018多校1-签到题

【题意】



【题解】















【代码】

#include <bits/stdc++.h>

#define ll long long

using namespace std;

int main()

{

int t;

scanf("%d",&t);

while(t--){

ll n;

scanf("%I64d",&n);

if(n%3==0) cout<<n\*n\*n/27<<endl;

else if(n%4==0)cout<<n\*n\*n/32<<endl;

else cout<<-1<<endl;

}

return 0;

}

## 组合数学

### 卡特兰数

出栈次序

二叉树构成

凸多边形的三角形划分

### Polya定理

<https://www.bilibili.com/video/av6357073?from=search&seid=4827637457650479048>

#### 原理不会-HDU-3923-Invoker

【题意】n种元素（可以同一种元素拿多个）放m个位置，经过旋转和翻转重合的算一种，统计总的个数

【题解】应用（Burnsid引理）Polya定理，目前还不会

【参考AC代码】

1. #include <cstdio>
3. #define LL \_\_int64
4. const LL mm=1000000007;
5. LL mod, pow[10005];
6. int n, m;
8. LL gcd(LL a, LL b)
9. {
10. return b==0 ? a : gcd(b,a%b);
11. }
13. LL rotat() *//旋转时的不动点*
14. {
15. LL ans = 0;
16. for (int i=0; i<m; ++i)
17. ans = (ans+pow[gcd(i,m)]) % mod;
18. return ans;
19. }
21. LL overturn() *//翻转时的不动点，分奇偶讨论*
22. {
23. LL ans = 0;
24. if(m & 1)
25. ans = (ans+m\*pow[(m+1)/2]) % mod;
26. else
27. ans = (ans+m/2\*(pow[m/2+1]+pow[m/2])) % mod;
28. return ans;
29. }
31. int main ()
32. {
33. int cas, t=0;
34. scanf("%d", &cas);
35. while(cas--)
36. {
37. LL ans = 0;
38. scanf("%d%d", &n, &m);
39. mod = 2\*m\*mm;
40. pow[0] = 1;
41. for(int i=1; i<=m; ++i)
42. pow[i] = (pow[i-1]\*n) % mod;
43. ans += rotat();
44. ans = (ans + overturn()) % mod;
45. printf("Case #%d: %I64d\n", ++t, (ans/2/m)%mm);
46. }
47. return 0;
48. }

#### HDU-1812-Count Teris

【题意】n\*n的棋盘，用c种颜色染色，（旋转、反射算一种），问有多少种不同棋盘。

数据规模 n,c<31

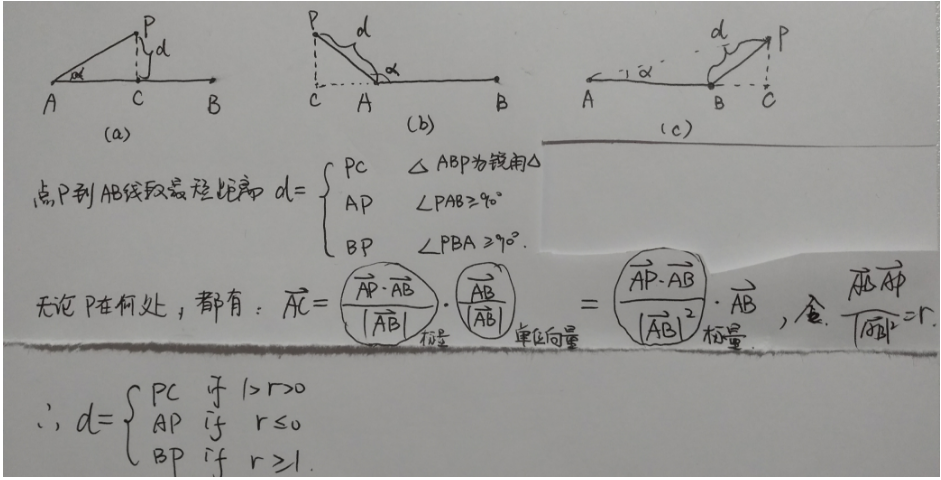
【题解】

【代码】

## 计算几何

[http://dev.gameres.com/Program/Abstract/Geometry.htm#](http://dev.gameres.com/Program/Abstract/Geometry.htm)

### 点到线段的最短距离——矢量法



const double eps=1e-6;

int fcmp(double x){

if (fabs(x)<=eps) return 0;

return x<0?-1:1;

}

double PointToSegDist(double px, double py, double xa, double ya, double xb, double yb){

double AP\_AB=(px-xa)\*(xb-xa)+(py-ya)\*(yb-ya);

double AB\_AB=(xb-xa)\*(xb-xa)+(yb-ya)\*(yb-ya);

double AB=sqrt(AB\_AB);

double k=AP\_AB/AB\_AB;

double AP\_AP=(px-xa)\*(px-xa)+(py-ya)\*(py-ya);

double AP=sqrt(AP\_AP);

double BP\_BP=(px-xb)\*(px-xb)+(py-yb)\*(py-yb);

double BP=sqrt(BP\_BP);

if (fcmp(k)<0){

return AP;

}else if (fcmp(k-1)<=0){

double d=sqrt(AP\_AP-(AP\_AB/AB)\*(AP\_AB/AB));

return d;

}else{

return BP;

}

}

#### √-51nod-1298-圆与三角形

【题干】

给出圆的圆心和半径，以及三角形的三个顶点，问圆同三角形是否相交。相交输出"Yes"，否则输出"No"。（三角形的面积大于0）。

【题解】

圆与三角形是否相交转换为圆与三角形的每一条边是否相交，圆与一条线段相交的条件是圆心到线段的距离小于等于半径且线段的端点至少有一个在圆外（含边）

【代码】

#include <bits/stdc++.h>

using namespace std;

const double eps=1e-6;

int fcmp(double x){

if (fabs(x)<=eps) return 0;

return x<0?-1:1;

}

double PointToSegDist(double px, double py, double xa, double ya, double xb, double yb){

double AP\_AB=(px-xa)\*(xb-xa)+(py-ya)\*(yb-ya);

double AB\_AB=(xb-xa)\*(xb-xa)+(yb-ya)\*(yb-ya);

double AB=sqrt(AB\_AB);

double k=AP\_AB/AB\_AB;

double AP\_AP=(px-xa)\*(px-xa)+(py-ya)\*(py-ya);

double AP=sqrt(AP\_AP);

double BP\_BP=(px-xb)\*(px-xb)+(py-yb)\*(py-yb);

double BP=sqrt(BP\_BP);

if (fcmp(k)<0){

return AP;

}else if (fcmp(k-1)<=0){

double d=sqrt(AP\_AP-(AP\_AB/AB)\*(AP\_AB/AB));

return d;

}else{

return BP;

}

}

bool in\_circle(double cx,double cy,double cr,double x,double y){

return fcmp((cx-x)\*(cx-x)+(cy-y)\*(cy-y)-cr\*cr)<=0;

}

bool is\_intersect(double cx,double cy,double cr,double x1,double y1,double x2,double y2){

return fcmp(PointToSegDist(cx,cy,x1,y1,x2,y2)-cr)<=0&&(!in\_circle(cx,cy,cr,x1,y1)||!in\_circle(cx,cy,cr,x2,y2));

}

int main()

{

int t;

cin>>t;

while (t--){

bool ans=false;

int cx,cy,cr,x[3],y[3];

cin>>cx>>cy>>cr;

for (int i=0;i<3;i++) cin>>x[i]>>y[i];

for (int i=0;i<2;i++){

for (int j=i+1;j<3;j++){

if (is\_intersect(cx,cy,cr,x[i],y[i],x[j],y[j])){

ans=true;

break;

}

}

if (ans) break;

}

if (ans) cout<<"Yes"<<endl;else cout<<"No"<<endl;

}

return 0;

}

### 四点共面

#### √-51nod-1265-四点共面

【题意】输入4个点判断是否在同一平面

【题解】使用如下公式进行判断



【代码】

#include <bits/stdc++.h>

using namespace std;

const double eps=1e-6;

int fcmp(double x){

if (fabs(x)<eps) return 0;

return x<0?-1:1;

}

double matrix(double x11,double x12,double x13,double x21,double x22,double x23,double x31,double x32,double x33){

return x11\*x22\*x33+x21\*x32\*x13+x31\*x12\*x23-x31\*x22\*x13-x11\*x32\*x23-x21\*x12\*x33;

}

int main()

{

int t;

cin>>t;

while (t--){

double x[4],y[4],z[4];

for (int i=0;i<4;i++) cin>>x[i]>>y[i]>>z[i];

double s=-matrix(x[1],y[1],z[1],x[2],y[2],z[2],x[3],y[3],z[3])

+matrix(x[0],y[0],z[0],x[2],y[2],z[2],x[3],y[3],z[3])

-matrix(x[0],y[0],z[0],x[1],y[1],z[1],x[3],y[3],z[3])

+matrix(x[0],y[0],z[0],x[1],y[1],z[1],x[2],y[2],z[2]);

if (fcmp(s)==0) cout<<"Yes"<<endl;else cout<<"No"<<endl;

}

return 0;

}

### 线段相交

#### √（模板）-51nod-1264

【题意】输入2个线段的4个端点，判断是否相交

【题解】直接使用浙大计算几何模板（1.5浮点数），注意该模板使用时不能加using namespace std;

### 简单多边形与圆面积交

#### √-牛客网2018暑期多校第3场-J（简单多边形与圆面积交）

【题意】给定简单多边形（定点顺时针或逆时针给出），给定圆心，要求圆的和简单多边形的面积交为给定值，输出圆的半径,坐标绝对值1000以内,n最多300

【题解】使用模板简单多边形与圆面积交和凸多边形面积，这里二分半径r即可（最小0，最大3000）

【代码】

#include<bits/stdc++.h>

using namespace std;

const double eps = 1e-12 ;

const double PI = acos( -1.0 ) ;

inline double sqr( double x ){ return x \* x ; }

inline int sgn( double x ){

if ( fabs(x) < eps ) return 0 ;

return x > 0? 1 : -1 ;

}

struct Point{

double x , y ;

Point(){}

Point( double \_x , double \_y ): x(\_x) , y(\_y) {}

void input() { scanf( "%lf%lf" ,&x ,&y ); }

double norm() { return sqrt( sqr(x) + sqr(y) ); }

friend Point operator + ( const Point &a , const Point &b ) { return Point( a.x + b.x , a.y + b.y ) ; }

friend Point operator - ( const Point &a , const Point &b ) { return Point( a.x - b.x , a.y - b.y ) ; }

friend Point operator \* ( const Point &a , const double &b ) { return Point( a.x \* b , a.y \* b ) ; }

friend Point operator \* ( const double &a , const Point &b ) { return Point( b.x \* a , b.y \* a ) ; }

friend Point operator / ( const Point &a , const double &b ) { return Point( a.x / b , a.y / b ) ; }

friend bool operator == ( const Point &a , const Point &b ) { return sgn( a.x - b.x ) == 0 && sgn( a.y - b.y ) == 0 ; }

bool operator < ( const Point &a )const{

return ( sgn( x - a.x ) < 0 ) || ( sgn( x - a.x ) == 0 && sgn( y - a.y ) < 0 ) ;

}

};

double dot( Point a , Point b ) { return a.x \* b.x + a.y \* b.y ; }

double det( Point a , Point b ) { return a.x \* b.y - a.y \* b.x ; }

double dist( Point a , Point b ) { return ( a - b ).norm() ; }

int n ;

double k ;

Point A,B ;

Point p[505] ;

Point o ;

double r ;

int CircleInterLine( Point a, Point b, Point o, double r, Point \*p )

{

Point p1 = a - o ;

Point d = b - a ;

double A = dot( d, d ) ;

double B = 2 \* dot( d, p1 ) ;

double C = dot( p1, p1 ) - sqr(r) ;

double delta = sqr(B) - 4\*A\*C ;

if ( sgn(delta) < 0 ) return 0 ;//相离

if ( sgn(delta) == 0 ) { //相切

double t = -B / (2\*A) ; // 0 <= t <= 1说明交点在线段上

if ( sgn( t - 1 ) <= 0 && sgn( t ) >= 0 ) {

p[0] = a + t \* d ;

return 1 ;

}

}

if ( sgn(delta) > 0 ) { //相交

double t1 = ( -B - sqrt(delta) ) / (2\*A) ;

double t2 = ( -B + sqrt(delta) ) / (2\*A) ; //0 <= t1, t2 <= 1说明交点在线段上

int k = 0 ;

if ( sgn( t1 - 1 ) <= 0 && sgn( t1 ) >= 0 )

p[k++] = a + t1 \* d ;

if ( sgn( t2 - 1 ) <= 0 && sgn( t2 ) >= 0 )

p[k++] = a + t2 \* d ;

return k ;

}

return 0;

}

double Triangle\_area( Point a, Point b )

{

return fabs( det( a , b ) ) / 2.0 ;

}

double Sector\_area( Point a, Point b )

{

double ang = atan2( a.y , a.x ) - atan2( b.y, b.x ) ;

while ( ang <= 0 ) ang += 2 \* PI ;

while ( ang > 2 \* PI ) ang -= 2 \* PI ;

ang = min( ang, 2\*PI - ang ) ;

return sqr(r) \* ang/2 ;

}

double calc( Point a , Point b , double r )

{

Point pi[2] ;

if ( sgn( a.norm() - r ) < 0 ) {

if ( sgn( b.norm() - r ) < 0 ) {

return Triangle\_area( a, b ) ;

}

else {

CircleInterLine( a, b, Point(0,0), r, pi) ;

return Sector\_area( b, pi[0] ) + Triangle\_area( a, pi[0] ) ;

}

}

else {

int cnt = CircleInterLine( a, b, Point(0,0), r, pi ) ;

if ( sgn( b.norm() - r ) < 0 ) {

return Sector\_area( a, pi[0] ) + Triangle\_area( b, pi[0] ) ;

}

else {

if ( cnt == 2 )

return Sector\_area( a, pi[0] ) + Sector\_area( b, pi[1] ) + Triangle\_area( pi[0], pi[1] ) ;

else

return Sector\_area( a, b ) ;

}

}

}

double area\_CircleAndPolygon( Point \*p , int n , Point o , double r )

{

double res = 0 ;

p[n] = p[0] ;

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

int tmp = sgn( det( p[i] - o , p[i+1] - o ) ) ;

if ( tmp )

res += tmp \* calc( p[i] - o , p[i+1] - o , r ) ;

}

return fabs( res ) ;

}

double area\_polygon(int n,Point\* p){

double s1=0,s2=0;

int i;

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

s1+=p[(i+1)%n].y\*p[i].x,s2+=p[(i+1)%n].y\*p[(i+2)%n].x;

return fabs(s1-s2)/2;

}

int main()

{

cin>>n;

for (int i=0;i<n;i++) p[i].input();

int m;

cin>>m;

while(m--){

o.input();

//cout<<area\_CircleAndPolygon(p,n,o,0.7324)<<endl;

double pp,qq;

scanf("%lf%lf",&pp,&qq);

double targets=area\_polygon(n,p)\*(1-pp/qq);

//cout<<targets<<endl;

double rmin=0,rmax=3000,small=1e-12;

r=(rmin+rmax)/2.0;

while(sgn(rmin-rmax)<=0){

double nows=area\_CircleAndPolygon(p,n,o,r);

if(sgn(nows-targets)<0){

rmin=r+small;

r=(rmin+rmax)/2.0;

}else if(sgn(nows-targets)>0){

rmax=r-small;

r=(rmin+rmax)/2.0;

}else{

break;

}

}

printf("%.12lf\n",r);

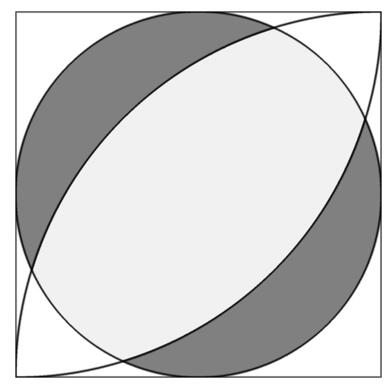
}

return 0;

}

### 其他题目

#### √-HDU-5858-2016多校10-签到

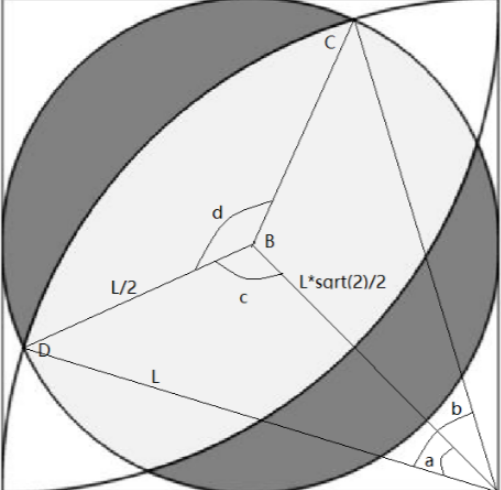


【题意】求阴影部分面积

【公式】



需要使用余弦定理与三角形面积公式



## 综合题

### √-牛客网2018暑期多校第1场-A（OEIS打表&组合数递推&逆元）

【题意】一个n\*m的矩阵，只包含0,1,2这3种数字，要求从左至右递增，从上至下递增，求符合条件的矩阵个数，数据规模，测试数据组数最多

【题解】

允许使用OEIS是铜牌题，不允许则是金牌题

合理的可以做的解题思路：

首先打表，接着在OEIS上查询

打表代码如下（dfs）

#include <bits/stdc++.h>

#define MAXN 3010

#define ll long lon

ll n,m,cnt;

int num[MAXN][MAXN];

void dfs(int x,int y){

for (int i=max(num[x-1][y],num[x][y-1]);i<=2;i++){

num[x][y]=i;

if(y<m) dfs(x,y+1); else if(x<n) dfs(x+1,1); else cnt++;

}

return;

}

int main()

{

int maxn,maxm;

cin>>maxn>>maxm;

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

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

n=i,m=j;

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

cnt=0;

dfs(1,1);

cout<<cnt<<" ";

}

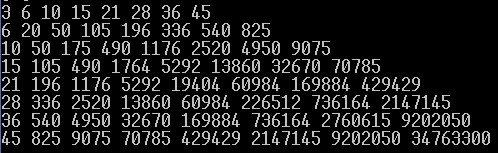
cout<<endl;

}

return 0;

}

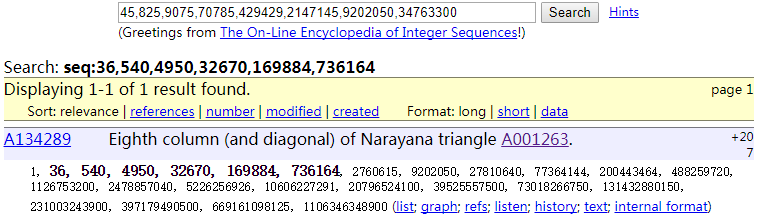
短时间大约能打8\*8左右



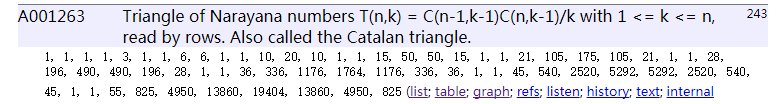
拿着结果一行一行的到OEIS上查询

第1行到第6行每一行都能查询出一个不同的公式，但并没有明显的通用规律

第7行开始输进去，神奇的事情发生了

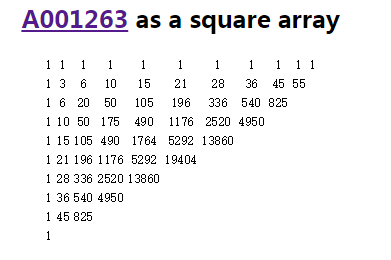


它说这是一个什么三角形的第8行，点开这个A001263



看到了一个通用公式，且其别名为Catalan三角形

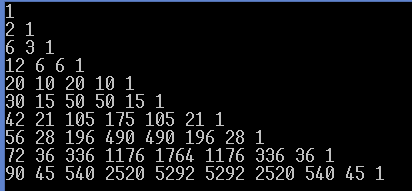
以table形式显示



这显然就是我们打表出来的答案，所以初步肯定上面的通式就是本题的答案

不过按照这个通式打表

（这里用到，记忆化搜索）



并不是直接与我们上面那个表一一对应的，找到这里面出现的数字在我们上一张表中出现的位置的规律，发现这里面的每一行都是我们打表答案中的每一个斜行（对角行），再调整一下行列偏移，得到公式



这里面由于数字较大要对N取模，考虑到T的公式中出现除法，故还要使用逆元将/k变为\*(inv(k))，同时开long long，注意分步取模

这里逆元用的板子的思路是根据费马小定理，a对p的逆元就是a^(p-2)，使用快速幂求解

标准官方解题思路（看看就好）：

考虑 01 和 12 的分界线

是 (n, 0) 到 (0, m) 的两条不相交（可重合）路径

平移其中一条变成 (n-1, -1) 到 (-1, m-1)

变成起点 (n, 0) 和 (n-1, -1)，终点 (0, m) 和 (-1, m-1) 的严格不相交路径

套 Lindström–Gessel–Viennot lemma

答案是 Cn+m, n 2 - Cn+m, m - 1 Cn+m, n-1

【代码】

#include <bits/stdc++.h>

#define MAXN 3010

#define ll long long

#define N 1000000007

using namespace std;

//逆元

ll inv(ll x,ll mod)

{

ll k=mod-2,ans=1;

while(k)

{

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

x=(x\*x)%mod;

k>>=1;

}

return ans;

}

//组合数&记忆化搜索

ll rc[MAXN][MAXN];

ll c(int n,int m){

if(rc[n][m]>=0) return rc[n][m];

if(n==m) return rc[n][m]=1;

if(m==1||m==0) return rc[n][m]=n;

return rc[n][m]=(c(n-1,m-1)+c(n-1,m))%N;

}

ll t(int n,int k){

return (((c(n-1,k-1)\*c(n,k-1))%N)\*inv(k,N))%N;

}

int main(){

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

int n,m;

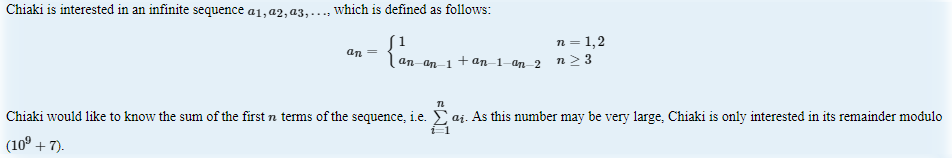
while(cin>>n>>m) cout<<t(n+m+1,m+1)<<endl;

return 0;

}

### √-HDU-6304-2018多校1-铜牌题

【题意】



n范围10^18，测试数据10^5

【题解】使用了OEIS打表找规律，发现了sum[2^k]的值可以在log级别求出，同时发现a[2^k]=2^(k-1)且a[2^k+1]=2^(k-1)+1，于是

sum[n]=sum[2^k+newn]=sum[2^k]+a[2^k+1]+……a[n]，后面的数值如果都减去a[2^k]，则会发现与a[2]…开始的序列相同，所以可以不断将问题规模以log级别缩小

【代码】

#include <bits/stdc++.h>

#define ll long long

#define MAXN 100

#define MODN 1000000007

using namespace std;

ll fast\_pow(ll a,ll b,ll N,bool modflag){

if(modflag){

a%=N;

ll base=a,ans=1;

while(b){

if(b&1) ans=(ans\*base)%N;

b>>=1;

base=(base\*base)%N;

}

return ans;

}

ll base=a,ans=1;

while(b){

if(b&1) ans\*=base;

b>>=1;

base\*=base;

}

return ans;

}

ll f(ll n){

if(n==1) return 4;

return (3\*fast\_pow(2,2\*(n-1),MODN,true))%MODN+((n+1)\*fast\_pow(2,n-2,MODN,true))%MODN;

}

ll s[MAXN]={1,2};

int main(){

for (int i=2;i<=70;i++){

s[i]=(s[i-1]+f(i-1)%MODN)%MODN;

}

int t;

cin>>t;

while(t--){

ll n;

cin>>n;

if(n==1){

cout<<1<<endl;

continue;

}

int cnt=0;

while(fast\_pow(2,cnt,MODN,false)<=n) cnt++;

cnt--;

ll ans=s[cnt];

n=n-fast\_pow(2,cnt,MODN,false);

while(n){

n++;

ll add=(fast\_pow(2,cnt-1,MODN,false))%MODN;

ans=(ans+((add\*((n-1)%MODN))%MODN))%MODN;

while(fast\_pow(2,cnt,MODN,false)>n) cnt--;

ans=((ans+s[cnt])%MODN+MODN-1)%MODN;

n=n-fast\_pow(2,cnt,MODN,false);

}

cout<<ans<<endl;

}

return 0;

}

### √-EOJ-110-2018月赛7-数蝌蚪-签到题

【题意】n个数，要求变为公差为k的等差数列（均要为正），对每个数可以加1或减1算1次操作，求最小操作数

【题解】

【代码】

#include <bits/stdc++.h>

#define ll long long

#define MAXN 300010

using namespace std;

int main(){

ll n,k,a[MAXN],b[MAXN];

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

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

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

a[i]-=i\*k;//每个数减去它所要求的公差后应该要相等(这样把这个公差加回去之后就是它所要的等差数列)

}

/\*

找到a数组的中位数

\*/

memcpy(b,a,sizeof(a));

sort(b,b+n);

ll mid=b[n/2];

if(mid<0) mid=0;//特判中位数<0的情况，因为蝌蚪数不能为负

ll ans=0;

/\*计算操作数\*/

for (int i=0;i<n;i++) ans+=(a[i]>mid)?a[i]-mid:mid-a[i];

cout<<ans<<endl;

return 0;

}

# 八、匹配&网络流

## 二分图匹配

**定理**

无向图G=<V,E>为二分图的充要条件是G的所有回路的长度均为偶数。

**最大匹配**

在G的一个子图M中，M的边集中的任意两条边都不依附于同一个顶点，则称M是一个匹配。选择这样的边数最大的子集称为图的最大匹配问题,最大匹配的边数称为最大匹配数.如果一个匹配中，图中的每个顶点都和图中某条边相关联，则称此匹配为完全匹配，也称作完备匹配。如果在左右两边加上源汇点后，图G等价于一个网络流，最大匹配问题可以转为最大流的问题。解决此问的[匈牙利](http://blog.sina.com.cn/s/blog_60707c0f0100o976.html)[算法](http://lib.csdn.net/base/31)的本质就是寻找最大流的增广路径。

**最优匹配**

最优匹配又称为带权最大匹配，是指在带有权值边的二分图中，求一个匹配使得匹配边上的权值和最大。一般X和Y集合顶点个数相同，最优匹配也是一个完备匹配，即每个顶点都被匹配。如果个数不相等，可以通过补点加0边实现转化。一般使用[KM算法](http://blog.sina.com.cn/s/blog_60707c0f01010633.html)解决该问题。

**最小覆盖**

二分图的最小覆盖分为最小顶点覆盖和最小路径覆盖：

**最小顶点覆盖**是指最少的顶点数使得二分图G中的每条边都至少与其中一个点相关联，二分图的最小顶点覆盖数=二分图的最大匹配数；

**最小路径覆盖**也称为最小边覆盖，是指用尽量少的不相交简单路径覆盖二分图中的所有顶点。二分图的最小路径覆盖数=|V|-二分图的最大匹配数；

**最大独立集**

最大独立集是指寻找一个点集，使得其中任意两点在图中无对应边。对于一般图来说，最大独立集是一个NP完全问题，对于二分图来说最大独立集=|V|-二分图的最大匹配数。

**匈牙利算法**

初始时最大匹配为空  
while 找得到增广路径 do 把增广路径加入到最大匹配中去

**增广路径**

(1)有奇数条边。  
(2)起点在二分图的左半边，终点在右半边。  
(3)路径上的点一定是一个在左半边，一个在右半边，交替出现。

(4)整条路径上没有重复的点。  
(5)起点和终点都是目前还没有配对的点，而其它所有点都是已经配好对的。

(6)路径上的所有第奇数条边都不在原匹配中，所有第偶数条边都出现在原匹配中。

(7)把增广路径上的所有第奇数条边加入到原匹配中去，并把增广路径中的所有第偶数条边从原匹配中删除（这个操作称为增广路径的**取反**），则新的匹配数就比原匹配数增加了1个。**定理**

如果从一个点A出发，没有找到增广路径，那么无论再从别的点出发找到多少增广路径来改变现在的匹配，从A出发都永远找不到增广路径。

**应用定理描述匈牙利算法**

初始时最大匹配为空  
for 二分图左半边的每个点i  
    do 从点i出发寻找增广路径。如果找到，则把它取反（即增加了总了匹配数）

### √-51nod-2006-飞行员配对

**Input**

第1行有2个正整数 m 和 n。n 是皇家空军的飞行 员总数(n<100);m 是外籍飞行员数。外籍飞行员编号为 1~m;英国飞行员编号为 m+1~n。接下来每行有 2 个正整数 i 和 j，表示外籍飞行员 i 可以和英国飞行员 j 配合。输入最后以 2 个-1 结束。

**Output**

第 1 行是最佳飞行 员配对方案一次能派出的最多的飞机数 M。如果所求的最佳飞行员配对方案不存在，则输出‘No Solution!’。

**Input示例**

5 10

1 7

1 8

2 6

2 9

2 10

3 7

3 8

4 7

4 8

5 10

-1 -1

**Output示例**

4

【题解】

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

#define MAXN 110

using namespace std;

int rednum,bluenum,tot,choose[MAXN],ans=0;

vector<int> match[MAXN];

bool exist[MAXN];

bool dfs(int x){

exist[x]=true;

if (x<=rednum){

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

if (match[x][i]>rednum&&!exist[match[x][i]]&&dfs(match[x][i])){

ans++;

choose[x]=match[x][i];

choose[match[x][i]]=x;

exist[x]=false;

return true;

}

}

}else{

if (choose[x]==-1){

exist[x]=false;

return true;//找到增广路径

}else{

if (!exist[choose[x]]&&dfs(choose[x])){//蓝到红取反消除

ans--;

//choose[choose[x]]=-1;

//choose[x]=-1;

exist[x]=false;

return true;

}

}

}

exist[x]=false;

return false;

}

int main()

{

memset(match,false,sizeof(match));

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

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

match[i].clear();

}

scanf("%d%d",&rednum,&tot);

bluenum=tot-rednum;

int tempx,tempy;

while (scanf("%d%d",&tempx,&tempy)&&!(tempx==-1&&tempy==-1)){

match[tempx].push\_back(tempy);

match[tempy].push\_back(tempx);

}

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

memset(exist,false,sizeof(exist));

if (choose[i]==-1) dfs(i);

if (ans==min(rednum,bluenum)) break;

//printf("%d:%d\n",i,ans);

//while(dfs(i));

}

if (ans==0) printf("No Solution!\n");else printf("%d\n",ans);

return 0;

}

### 未做-hdu-2063

# 九、博弈

## √-51nod-1995-三子棋

【题干】

**Input**

第一行输入一个整数T，表示数据组数（1<T<10000）；

第二行输入两个整数x,y,表示3×4格子里面的一个坐标(x,y)（1<=x<=3,1<=y<=4）；

**Output**

每组数据输出最后小明输赢的结果，如果小明一定能赢，第一行输出“Win”,第二行输出小明所需要花的最少步数；如果小明跟小花只能打成平手，第一行输出“Equal”，第二行输出数字0；如果小明不能赢也不能跟小花打成平手，第一行输出“Lose”,第二行输出小花赢小明所需要花的最少步数。

**Input示例**

2

2 1

2 4

**Output示例**

Equal

0

Equal

0

【思路】

将3\*4的棋盘分成4种类型的位置（对称）

第1种 （1,1）（1,4）（3,1）（3,4）枚举后手的第1个位置，都有必赢策略，其中最多需要6步

第2种 （1,2）（1,3）（3,2）（3,3）枚举后手的第1个位置，都有必赢策略，其中最多需要4步

第3种 （2,2）（2，3）枚举后手的第1个位置，都有必赢策略，其中最多需要4步

第4种 （2,1）（2，4）根据题目样例，平局

【AC代码】

#include <iostream>

using namespace std;

int main()

{

const int type[3][4]={{1,2,2,1},{4,3,3,4},{1,2,2,1}};

int t;

cin>>t;

while (t--){

int x,y;

cin>>x>>y;

x--,y--;

switch(type[x][y]){

case 1:

{

cout<<"Win"<<endl<<"6"<<endl;

break;

}

case 4:

{

cout<<"Equal"<<endl<<"0"<<endl;

break;

}

case 2:case 3:

{

cout<<"Win"<<endl<<"4"<<endl;

break;

}

}

}

return 0;

}

√-HDU-6312-多校2-签到题

【题意】1-n这n个正数，每个人可以拿走1个数连带它的所有因子都被拿走，谁先拿完谁赢，Alice先手，给定n问Alice是否有必赢策略。数据规模n为500

【题解】

比赛的时候先手算打表，一直算到n=7都是Alice赢，这时才开始比赛没几分钟，这题过的人已经井喷了，于是队友说估计是全Yes了，又稍微想了一会儿，交了果然全Yes…

具体严格的原因还没有想清楚。当时考虑到的一个就是如果剩下的都是互相互素的数的话，那么轮到一个人又偶数个的局就必输，反之奇数个就必赢。

赛后群里发的题解是“考虑将游戏变成初始时只有2~n，如果先手必胜的话，那么先手第一步按这样取就获胜了；如果后手必胜的话，那么先手第一步取走1就获胜了。所以全输出Yes就行了。”这有点强。。不过这个里面似乎没有出现题目中所要求的什么拿走因子之类的。。所以这个套路看来对很多种博弈都有效？（开局1-n，先手可以拿走1，谁先拿光谁赢，其他的拿法可以是任意某种确定的拿法） 哦，不对，这里因为2-n里面随便拿走某一个都会导致1的被拿走，这一点是由本题的性质所决定的。

# 十、随机算法

随机除了可应用与某些随机算法的题目，还可以在代码提交前对代码进行大量的随机测试以判定代码是否正确

## 函数库

//随机序列

#include <algorithmn>

int a[100];

random\_shuffle(a,a+n);

vector<int> a;

random\_shuffle(a.begin(),a.end());

//生成一定范围内随机数

#include <ctime>

#include <cstdlib>

srand((unsigned)time(NULL));//用这个提交OJ可能会错，换成srand(i)尝试；

产生一定范围随机数的通用表示公式是：

取得(0,x)的随机整数：rand()%x；

取得(a,b)的随机整数：rand()%(b-a)；

取得[a,b)的随机整数：rand()%(b-a)+a；

取得[a,b]的随机整数：rand()%(b-a+1)+a；

取得(a,b]的随机整数：rand()%(b-a)+a+1；

取得0-1之间的浮点数：rand()/double(RAND\_MAX)。

# 十一、未分类

## 莫队算法

<https://blog.csdn.net/hnshhslsh/article/details/50582926>

<https://www.cnblogs.com/137shoebills/p/7783739.html>

<https://blog.csdn.net/thinfatty/article/details/72581276>

## √-HDU-5857-2016多校10-签到

## √-HDU-5867-2016多校10-签到

# 十二、题库

<https://blog.csdn.net/liuqiyao_01/article/details/9079611>

**ACM题集以及各种总结大全**

[**https://blog.csdn.net/lingzidong/article/details/78472208**](https://blog.csdn.net/lingzidong/article/details/78472208)

**2013-2017 ACM/ICPC 区域赛&final 水题和铜牌题**

[**https://blog.csdn.net/acm\_1361677193/article/details/42873141**](https://blog.csdn.net/acm_1361677193/article/details/42873141)

**叉姐的训练指南**

[**https://blog.csdn.net/qq\_40688707/article/details/80602064**](https://blog.csdn.net/qq_40688707/article/details/80602064)

**ACM训练史上最详细计划（大神养成记）**

## 签到题题库

### 51nod（基础题）

1995-√

2006-√

1459-√

1384-√

1298-√

1265-√

1264-√（计算几何模板）

1205

1256

1240

1242

1212

1185

1183

1181

1079

1174

1137

1136

1135

1134

未完，共44题，上述快要做完时再补充展开，基础级完成后进入1级

## 补题题库（签到或专项）

**ACM作业20180613-数学专题**

FZU-1851-ACM作业20180613-签到/铜

HDU-2204-ACM作业20180613-签到/铜

HDU-1796-ACM作业20180613-签到/铜

HDU-1685-ACM作业20180613-签到/铜

UVA-1025-ACM作业20180606-签到/铜

UVA-437-ACM作业20180606-签到/铜

UVA-11584-ACM作业20180606-签到/铜

UVA-1256-ACM作业20180606-签到/铜

UVA-10003-ACM作业20180606-签到/铜

UVA-11400-ACM作业20180606-签到/铜

HDU-1502-ACM作业20180606-签到/铜

## 动态规划

背包-进阶

（<https://blog.csdn.net/eagle_or_snail/article/details/50987044>）

一、简单基础dp

这类dp主要是一些状态比较容易表示，转移方程比较好想，问题比较基本常见的。主要包括递推、背包、LIS（最长递增序列），LCS（最长公共子序列），下面针对这几种类型，推荐一下比较好的学习资料和题目。

1、递推：

递推一般形式比较单一，从前往后，分类枚举就行。

推荐：

[zoj 3747 Attack on Titans](http://blog.csdn.net/cc_again/article/details/24841249)

[uva 10328 Coin Toss](http://blog.csdn.net/cc_again/article/details/24844911)

[hdu 4747 Mex](http://blog.csdn.net/cc_again/article/details/11856847)

[hdu 4489 The King's Ups and Downs](http://blog.csdn.net/cc_again/article/details/9918313)

[hdu 4054 Number String](http://blog.csdn.net/cc_again/article/details/10858813)

hdu 3369 矩阵快速幂（递推式构造系数矩阵）

hdu 3483 矩阵快速幂（递推式构造系数矩阵）

2、背包

经典的背包九讲：<http://love-oriented.com/pack/>

推荐博客：<http://blog.csdn.net/woshi250hua/article/details/7636866>

主要有0-1背包、完全背包、分组背包、多重背包。

简单：

[hdu 2955 Robberies](http://acm.hdu.edu.cn/showproblem.php?pid=2955)

[hdu 1864 最大报销额](http://acm.hdu.edu.cn/showproblem.php?pid=1864)

[hdu 2602 Bone Collector](http://acm.hdu.edu.cn/showproblem.php?pid=2602)

[hdu 2159 FATE](http://acm.hdu.edu.cn/showproblem.php?pid=2159)

推荐：

[woj 1537 A Stone-I](http://blog.csdn.net/cc_again/article/details/22728273)

[woj 1538 B Stone-II](http://blog.csdn.net/cc_again/article/details/22728273)

[poj 1170 Shopping Offers](http://blog.csdn.net/cc_again/article/details/12200343) 状压+背包

[zoj 3769 Diablo III](http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=3769) 带限制条件的背包

[zoj 3638 Fruit Ninja](http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=3638)背包的转化成组合数学

[hdu 3092 Least common multiple](http://blog.csdn.net/cc_again/article/details/11518329) 转化成完全背包问题

[poj 1015 Jury Compromise](http://blog.csdn.net/cc_again/article/details/25426159) 扩大区间+输出路径

3、LIS

最长递增子序列，朴素的是o(n^2)算法，二分下可以写成o(nlgn)：维护一个当前最优的递增序列——找到恰好大于它更新

推荐：

[uva 10635 Prince and Princess](http://blog.csdn.net/cc_again/article/details/18372521) LCS转化成LIS

[hdu 4352 XHXJ's LIS](http://blog.csdn.net/cc_again/article/details/11821361)　数位dp+LIS思想

[srm div2 1000](http://blog.csdn.net/cc_again/article/details/12113809) 状态压缩+LIS

[poj 1239 Increasing Sequence](http://blog.csdn.net/cc_again/article/details/12208725) 两次dp

4、LCS

最长公共子序列，通常o(n^2)的算法

[uva 111 History Grading](http://blog.csdn.net/cc_again/article/details/8554454) 要先排个序

[poj 1080 Human Gene Functions](http://poj.org/problem?id=1080)

二、区间dp

推荐博客：<http://blog.csdn.net/woshi250hua/article/details/7969225>

区间dp,一般是枚举区间，把区间分成左右两部分，然后求出左右区间再合并。

[hdu 4745 Two Rabbits](http://blog.csdn.net/cc_again/article/details/11852367) 转化成求回文串

[zoj 3541 The Last Puzzle](http://blog.csdn.net/cc_again/article/details/10977751) 贪心+区间dp

[poj 2955 Brackets](http://poj.org/problem?id=2955)

[hdu 4283 You Are the One](http://blog.csdn.net/woshi250hua/article/details/7973824)  常见写法

[hdu 2476 String Printer](http://acm.hdu.edu.cn/showproblem.php?pid=2476)

[zoj 3537 Cake](http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=3537)

[CF 149D Coloring Brackets](http://codeforces.com/problemset/problem/149/D)

[zoj 3469 Food Delivery](http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=3469)

三、树形dp

比较好的博客：<http://blog.csdn.net/woshi250hua/article/details/7644959>

一篇论文：<http://doc.baidu.com/view/f3b19d0b79563c1ec5da710e.html>

树形dp是建立在树这种数据结构上的dp,一般状态比较好想，通过dfs维护从根到叶子或从叶子到根的状态转移。

[hdu 4514](http://blog.csdn.net/cc_again/article/details/8911480)  求树的直径

[poj 1655 Balancing Act](http://blog.csdn.net/cc_again/article/details/13004997)

[hdu 4714 Tree2Cycle](http://blog.csdn.net/cc_again/article/details/11407157) 思维

[hdu 4616 Game](http://blog.csdn.net/cc_again/article/details/10312393)

[hdu 4126 Genghis Kehan the Conqueror](http://blog.csdn.net/cc_again/article/details/12060191) MST+树形dp 比较经典

[hdu 4756 Install Air Conditioning](http://blog.csdn.net/cc_again/article/details/12092021) MST+树形dp 同上

[hdu 3660 Alice and Bob's Trip](http://blog.csdn.net/cc_again/article/details/12346065) 有点像对抗搜索

[CF 337D Book of Evil](http://blog.csdn.net/cc_again/article/details/10226673) 树直径的思想 思维

[hdu 2196 Computer](http://acm.hdu.edu.cn/showproblem.php?pid=2196) 搜两遍

四、数位dp

推荐一篇论文：<http://wenku.baidu.com/view/d2414ffe04a1b0717fd5dda8.html>

数位dp,主要用来解决统计满足某类特殊关系或有某些特点的区间内的数的个数，它是按位来进行计数统计的，可以保存子状态，速度较快。数位dp做多了后，套路基本上都差不多，关键把要保存的状态给抽象出来，保存下来。

[CF 401D Roman and Numbers](http://blog.csdn.net/cc_again/article/details/25053071) 状压+数位dp

[hdu 4398 X mod f(x)](http://blog.csdn.net/cc_again/article/details/8872355) 把模数加进状态里面

[hdu 4734 F(x)](http://blog.csdn.net/cc_again/article/details/11747555) 简单数位dp

[hdu 3693 Math teacher's homework](http://blog.csdn.net/cc_again/article/details/12257445) 思维变换的数位dp

[hdu 4352 XHXJ's LIS](http://blog.csdn.net/cc_again/article/details/11821361)　数位dp+LIS思想

[CF 55D Beautiful Numbers](http://blog.csdn.net/cc_again/article/details/8815450)  比较巧妙的数位dp

[hdu 3565 Bi-peak Numbers](http://blog.csdn.net/cc_again/article/details/8872073) 比较难想

[CF 258B Little Elephant and Elections](http://blog.csdn.net/cc_again/article/details/8877603) 数位dp+组合数学+逆元

五、概率(期望) dp

推荐博客：<http://www.cnblogs.com/kuangbin/archive/2012/10/02/2710606.html>

推荐博客：<http://blog.csdn.net/woshi250hua/article/details/7912049>

推荐论文：

[《走进概率的世界》](http://wenku.baidu.com/view/1c41152de2bd960590c677a8.html)

[《浅析竞赛中一类数学期望问题的解决方法》](http://wenku.baidu.com/view/90adb02acfc789eb172dc8a8.html)

[《有关概率和期望问题的研究》](http://wenku.baidu.com/view/56147518a8114431b90dd81e.html)

一般来说概率正着推，期望逆着推。有环的一般要用到高斯消元解方程。期望可以分解成多个子期望的加权和，权为子期望发生的概率，即 E(aA+bB+...) = aE(A) + bE(B) +...

[ural 1776 Anniversiry Firework](http://blog.csdn.net/cc_again/article/details/8974277) 比较基础

[hdu 4418 Time travel](http://blog.csdn.net/cc_again/article/details/10493543) 比较经典BFS+概率dp+高斯消元

[hdu 4586 Play the Dice](http://blog.csdn.net/cc_again/article/details/10456837) 推公式比较水

[hdu 4487 Maximum Random Walk](http://blog.csdn.net/cc_again/article/details/9926597)

[jobdu 1546 迷宫问题](http://blog.csdn.net/cc_again/article/details/12408505) 高斯消元+概率dp+BFS预处理

[hdu 3853 LOOPS](http://blog.csdn.net/cc_again/article/details/11536347) 简单概率dp

[hdu 4405 Aeroplane chess](http://blog.csdn.net/cc_again/article/details/11554945) 简单概率dp,比较直接

[hdu 4089 Activation](http://blog.csdn.net/cc_again/article/details/10431451) 比较经典

[poj 2096 Collecting Bugs](http://blog.csdn.net/cc_again/article/details/9936197) 题目比较难读懂

[zoj 3640 Help me Escape](http://blog.csdn.net/cc_again/article/details/11532517) 从后往前，比较简单

[hdu 4034 Maze](http://blog.csdn.net/cc_again/article/details/11544753) 经典好题，借助树的概率dp

[hdu 4336 Card Collector](http://blog.csdn.net/cc_again/article/details/11099749) 状态压缩+概率dp

六、状态压缩dp

这类问题有TSP、插头dp等。

推荐论文：<http://wenku.baidu.com/view/ce445e4f767f5acfa1c7cd51.html>

推荐博客：<http://blog.csdn.net/sf____/article/details/15026397>

推荐博客：<http://www.notonlysuccess.com/index.php/plug_dp/>

[hdu 4568 Hunter](http://blog.csdn.net/cc_again/article/details/9984961) 最短路+TSP

[hdu 4539](http://blog.csdn.net/cc_again/article/details/9954921) 插头dp

[hdu 4529 状压dp](http://blog.csdn.net/cc_again/article/details/9060019)

[poj 1185 炮兵阵地](http://poj.org/problem?id=1185)

[hdu 3811 Permutation](http://acm.hdu.edu.cn/showproblem.php?pid=3811)

[poj 2411 Mandriann's Dream](http://poj.org/problem?id=2411)

[poj 1038](http://poj.org/problem?id=1038)

[poj 2441](http://poj.org/problem?id=2441)

[hdu 2167](http://acm.hdu.edu.cn/showproblem.php?pid=2167)

[hdu 4026](http://acm.hdu.edu.cn/showproblem.php?pid=4026)

[hdu 4281](http://acm.hdu.edu.cn/showproblem.php?pid=4281)

七、数据结构优化的dp

有时尽管状态找好了，转移方程的想好了，但时间复杂度比较大，需要用数据结构进行优化。常见的优化有二进制优化、单调队列优化、斜率优化、四边形不等式优化等。

1、二进制优化

主要是优化背包问题，背包九讲里面有介绍，比较简单，这里只附上几道题目。

[hdu 1059 Diving](http://acm.hdu.edu.cn/showproblem.php?pid=1059)

[hdu 1171 Big Event in Hdu](http://acm.hdu.edu.cn/showproblem.php?pid=1059)

[poj 1048 Follow My Magic](http://poj.org/problem?id=1048)

2、单调队列优化

推荐论文：<http://wenku.baidu.com/view/4d23b4d128ea81c758f578ae.html>

推荐博客：<http://www.cnblogs.com/neverforget/archive/2011/10/13/ll.html>

[hdu 3401 Trade](http://blog.csdn.net/cc_again/article/details/9328243)

[poj 3245 Sequece Partitioning](http://blog.csdn.net/cc_again/article/details/9335795) 二分+单调队列优化

3、斜率优化

推荐论文：[用单调性优化动态规划](http://wenku.baidu.com/view/ef259400bed5b9f3f90f1c3a.html)

推荐博客：<http://www.cnblogs.com/ronaflx/archive/2011/02/05/1949278.html>

[hdu 3507 Print Article](http://acm.hdu.edu.cn/showproblem.php?pid=3507)

[poj 1260 Pearls](http://poj.org/problem?id=1260)

[hdu 2829 Lawrence](http://acm.hdu.edu.cn/showproblem.php?pid=2829)

[hdu 2993 Max Average Problem](http://acm.hdu.edu.cn/showproblem.php?pid=2993)

4、四边形不等式优化

推荐博客：<http://www.cnblogs.com/ronaflx/archive/2011/03/30/1999764.html>

推荐博客：<http://www.cnblogs.com/zxndgv/archive/2011/08/02/2125242.html>

[hdu 2952 Counting Sheep](http://acm.hdu.edu.cn/showproblem.php?pid=2952)

[poj 1160 Post Office](http://poj.org/problem?id=1160)

[hdu 3480 Division](http://acm.hdu.edu.cn/showproblem.php?pid=3480)

[hdu 3516 Tree Construction](http://acm.hdu.edu.cn/showproblem.php?pid=3516)

[hdu 2829 Lawrence](http://acm.hdu.edu.cn/showproblem.php?pid=2829)

## 图论

POJ综合题

<https://blog.csdn.net/liuqiyao_01/article/details/8477827>

### 并查集&最小生成树（学习区）

<https://blog.csdn.net/liuqiyao_01/article/details/8797356>

【HDU】

1213 How Many Tables 基础并查集★

1272 小希的迷宫 基础并查集★

1325&&poj1308 Is It A Tree? 基础并查集★

1856 More is better 基础并查集★

1102 Constructing Roads 基础最小生成树★

1232 畅通工程 基础并查集★

2120 Ice\_cream's world I 基础并查集★

2122 Ice\_cream’s world III 基础最小生成树★

1233 还是畅通工程 基础最小生成树★

1863 畅通工程 基础最小生成树★

1875 畅通工程再续 基础最小生成树★

1879 继续畅通工程 基础最小生成树★

3371     Connect the Cities 简单最小生成树★

1301 Jungle Roads 基础最小生成树★

1162 Eddy's picture 基础最小生成树★

1198 Farm Irrigation 基础最小生成树★

[1598 find the most comfortable road 枚举+最小生成树★★](http://acm.hdu.edu.cn/showproblem.php?pid=1598)[解题报告](http://blog.csdn.net/liuqiyao_01/article/details/8867249)

[1811 Rank of Tetris 并查集+拓扑排序★★](http://acm.hdu.edu.cn/showproblem.php?pid=1811) [解题报告](http://blog.csdn.net/liuqiyao_01/article/details/8867346)

3926 Hand in Hand 同构图★

3938 Portal 离线+并查集★★

2489     Minimal Ratio Tree dfs枚举组合情况+最小生成树★

4081     Qin Shi Huang's National Road System 最小生成树+DFS★★

4126     Genghis Khan the Conqueror 枚举+最小生成树+DFS(难)★★★★

2860 并查集  
2144 LCS+并查集

1829&&poj2492 A Bug's Life 基础种类并查集★

1558 Segment set 计算几何+并查集★

3461 Code Lock 并查集(有点难想到)★★

3367 Pseudoforest 最大生成树★

2473 Junk-Mail Filter 并查集+设立虚父节点(马甲)★★

3172 Virtual Friends 带权并查集★

3635 Dragon Balls 带权并查集★

3047 Zjnu Stadium 带权并查集★

3038 How Many Answers Are Wrong 种类并查集★★

2818 Building Block 带权并查集★

3234 Exclusive-OR 异或并查集(难)★★★

2121 Ice\_cream’s world II 最小树形图(要输出根有点恶心)★★

4009   Transfer water 最小树形图(模板题)★

3311  Dig The Wells 斯坦纳树(状压DP)(模板题)★★

4085 Peach Blossom Spring 斯坦纳树(状压DP)(有可能是森林...)★★★

4253 Two Famous Companies

4263 Red/Blue Spanning Tree

4313 Matrix [最大生成森林]

4424 Conquer a New Region [最大生成森林]

4509 湫湫系列故事——减肥记II [可以用并查集，区间合并]   
【POJ】

1258 最经典的MST★

1789 Truck History 最小生成树★

1287 Networking 简单★

2349 Arctic Network 简单★

1611 The Suspects 并查集★

2377 kruskal★

2524 Ubiquitous Religions 并查集★

[2236 Wireless Network 并查集+计算几何★](http://poj.org/problem?id=2236)   [解题报告](http://blog.csdn.net/liuqiyao_01/article/details/8867670)

2560 Kruskal 并查集★

1861 Kruskal★  
3625 prim★

1679 - The Unique MST(基础) 判断MST是否唯一★

3522 - Slim Span(基础) 求一颗生成树，让最大边最小边差值最小★

2485 Highways MST中的最长边★

2395 最小生成树的最长边★

1751 Highways 求出方案★

POJ-1182 食物链 种类并查集★★

POJ 1456 Supermarket 贪心+区间合并★

POJ-1703 种类并查集★  
POJ-1988 种类并查集★

POJ-1733 Parity game 种类并查集，先要离散化一下，不影响结果★

POJ-1417 True Liars(难) 并查集+DP 种类并查集★★

POJ-2912 Rochambeau(难) baidu的题,很不错...是食物链的加强版.判断裁判比较难想.★★★

POJ 2728 - Desert King(中等) 最优比率生成树★★

POJ 1639 - Picnic Planning(较难) 顶点度数有限制的最小生成树★★

POJ 3164 - Command Network(难) 最小树形图★★  
poj3723好题!!! ★★  
poj3228好好题!!! ★★  
POJ-1984 Navigation Nightmare二维曼哈顿距离并查集.   
【ZOJ】

ZOJ-3261 逆向并查集 ★★

ZOJ-3613 斯坦纳树

### 拓补排序（学习区）

<https://blog.csdn.net/liuqiyao_01/article/details/8478726>

【HDU】  
1285确定比赛名次  
2094产生冠军  
2647Reward  
3342Legal or Not  
1811Rank of Tetris 拓扑+并查集  
3231 Box Relations 三维拓扑  
【POJ】  
1094 Sorting It All Out  Floyd+拓扑  
2367 Genealogical tree  
3660 Cow Contest  
3687 Labeling Balls 神奇的拓扑  
1128Frame Stacking  DFS版拓扑  
1270Following Orders  拓扑+回溯  
1420Spreadsheet  模拟拓扑  
2762Going from u to v or from v to u?  强连通+拓扑

3553 Task schedule

最短路&差分约束（学习区）

<https://blog.csdn.net/liuqiyao_01/article/details/8797353>

**最短路**

【HDU】

[1548](http://acm.hdu.edu.cn/showproblem.php?pid=1548)    A strange lift基础最短路(或bfs)★  
[2544](http://acm.hdu.edu.cn/showproblem.php?pid=2544)    最短路  基础最短路★  
[3790](http://acm.hdu.edu.cn/status.php?user=671coder&pid=3790&status=5)   最短路径问题基础最短路★  
[2066](http://acm.hdu.edu.cn/showproblem.php?pid=2066)    一个人的旅行基础最短路(多源多汇,可以建立超级源点和终点)★  
[2112](http://acm.hdu.edu.cn/showproblem.php?pid=2112)    HDU Today基础最短路★  
[1874](http://acm.hdu.edu.cn/showproblem.php?pid=1874)    畅通工程续基础最短路★  
[1217](http://acm.hdu.edu.cn/showproblem.php?pid=1217)    Arbitrage   货币交换 Floyd (或者 Bellman-Ford 判环)★  
[1245](http://acm.hdu.edu.cn/showproblem.php?pid=1245)    Saving James Bond计算几何+最短路★  
1317    XYZZY  Bellman-Ford判环,有负权★  
1535    Invitation Cards   有向图的来回最短路,(反向建图)★  
1546    Idiomatic Phrases Game  最短路★  
2680    Choose the best route   最短路★  
2923    Einbahnstrasse最短路★  
3339    In Action  最短路+背包★  
2224    The shortest path双调旅行商问题★★  
2807    The Shortest Path矩阵运算+最短路(floyd)★★  
1595    find the longest of the shortest枚举+最短路(删掉任意一条边的最长最短路)★★  
3986    Harry Potter and the Final Battle 枚举+最短路(删掉任意一条边的最长最短路)★★  
1599    find the mincost routefloyd求最小环★

1839 Delay Constrained... 二分下限+最短路(带限制最短路)★★

3631 Shortest Path Floyd插点法★★

4114 Disney's FastPass 最短路+二维状压DP(难)★★★

3832    Earth Hour 三点连通(斯坦纳树)★

3873    Invade the Mars Dij变体(好题!,带限制最短路)★★★

4063  Aircraft 几何构图+最短路★★★★

hdu4179 Difficult Routes dis[][]开二维状态的最短路(带限制最短路)★★

2145 zz's Mysterious Present 反向建边. 最短路

3268 最短路  
3143 最短路  
1869 六度分离 Floyd最短路★  
1385    Minimum Transport Cost 最短路+输出路径(输出字典序最小路径,有点恶心)★★  
1224    free DIY Tour 最短路+输出路径★  
1142    A Walk Through the Forest  最短路+记忆搜索★★  
1596    find the safest road   乘积最小的最短路★  
[1598](http://acm.hdu.edu.cn/showproblem.php?pid=1598)    find the most comfortable road二分速度差+最短路(带限制最短路)★★  
2722    Here We Go(relians) Again最短路★  
2962    Trucking 二分+最短路(带限制最短路)★★  
1690    Bus System 最短路★  
2433    Travel 删边+最短路之和(预处理桥边)★★★  
2363    Cycling 二分+最短路(带限制最短路)★★  
2377    Bus Pass 最短路(寻找一个点的最长最短路最小)★★  
2833    WuKong 最短路+记忆化搜索(求两条最短路的最多公共点)★★  
1688    Sightseeing最短次短路条数★★  
3191    How Many Paths Are There 次短路条数★★  
2482    Transit search最短路★★★

3768  Shopping 最短路+dfs(或最短路+状压DP)★★

3035    War 平面图最小割(建图麻烦)★★  
3870    Catch the Theves 平面图最小割(建图麻烦)★★

3860 Circuit Board 平面图最小割(建图麻烦)★★

4308 Saving Princess claire\_ 最短路构图

4034 Graph [Floyd应用]

4157 Slalom 计算几何+最短路

4280 Island Transport [抠图+平面图最小割]

4293 Groups [最长路]

4318 Power transmission

4360 As long as Binbin loves Sangsang

4370  0 or 1

4396 More lumber is required  
【POJ】  
[1062 昂贵的聘礼 竟然可以和最短路联系起来](http://poj.org/problem?id=1062)★★          [解题报告](http://blog.csdn.net/liuqiyao_01/article/details/8882783)

1094  Sorting It All OutFloyd 判环+拓扑排序★

1125 Stockbroker Grapevine Floyd★

1135  Domino Effect 最短路,比较有意思★★

1161  Walls 最短路(图太恶心了)★★

1502  MPI Maelstrom Floyd★

1511    Invitation Cards 来回最短路★

1556 The Doors 计算几何+最短路★★

1724  ROADS 带限制的最短路,dis[][]开二维来记录信息(或广搜)★★

1734  Sightseeing trip floyd最小环路径★

1797  Heavy Transportation 二分枚举+最短路★

1847 Tram 简单最短路★

1860 Currency Exchange 货币兑换★

1949  Chores 反向建边,求最长路★★

2139  Six Degrees of Cowvin Bacon Floyd★

2240 Arbitrage 货币兑换★

2253 Frogger 二分+最短路★

2312  坦克大战 spfa最短路本质变形-->广搜★

2387  Til the Cows Come Home 基础最短路★

2394 Checking an Alibi 最短路★

2449 Remmarguts' Date A\*求第K短路★★

2457  Part Acquisition 最短路 (输出路径)★★

2472  106 miles to Chicago 乘积最短路(log一下,乘变加)★★

2502Subway  
2570 Fiber Network floyd  
3013 圣诞树  
3037Skiing  
3072 Robot

3114  Countries in War 强联通+最短路

3160  Father Christmas flymouse 强联通+最长路

3255 Roadblocks

3259  Wormholes （寻找负权回路）

3268 Silver Cow Party

3311  Hie with the Pie floyd+状压

3328 Cliff Climbing  
3439 Server Relocation  
3463 Sightseeing 次短路条数  
3159

3521 Geometric Map 计算几何+最短路

3549 GSM phone 计算几何+最短路

3594 Escort of Dr. Who How

3613  Cow Relays 经过N条边的最短路 // floyd + 二分矩阵

3615 Cow Hurdles  
3621 最优比率环  
3635 full tank?  
3660 传递闭包  
3662 Telephone Lines

4046 Sightseeing

【SGU314】一道神级求前k短路。。。

**差分约束**

【HDU】  
1384 Intervals 基础差分约束★  
1529 Cashier Employment 神级差分约束★★★★

1531 King 差分约束★  
1534 Schedule Problem 差分约束输出一组解★  
3440 House Man 比较好的差分约束★★  
3592 World Exhibition 简单★  
3666 THE MATRIX PROBLEM 中等★★  
4274 Spy's Work [先处理出欧拉序列,然后就是差分约束了...]

【POJ】  
1201 Intervals  
1275 Cashier Employment  
1364 King  
1716 Integer Intervals  
2949 Word Rings  
2983 Is the Information Reliable?  
3159  
3169  
3687

### 二分匹配（学习区）

<https://blog.csdn.net/liuqiyao_01/article/details/8797349>  
【HDU】  
1068Girls and Boys 最大匹配★  
1150Machine Schedule 最小点覆盖★  
1151Air Raid 最小路径覆盖★  
1179Ollivanders 最大匹配★  
1281棋盘游戏 行列匹配+求关键点★★  
149850 years, 50 colors 行列匹配★  
1507Uncle Tom's Inherited Land\* 黑白染色+奇偶匹配(1X2的矩形覆盖)★  
1528Card Game Cheater 最大匹配★  
1845Jimmy’s Assignment 最大匹配(HK算法)★  
2063过山车 最大匹配★  
2119Matrix 行列匹配  
2444The Accomodation of Students 并查集分集合+最大匹配(好题!)★★  
2768Cat vs. Dog 最大独立集★★  
3360National Treasures 黑白染色+最小点覆盖★★  
1045Fire Net 行列匹配变形★★  
1350Taxi Cab Scheme 最小路径覆盖★  
1960Taxi Cab Scheme 最小路径覆盖★  
3118Arbiter 二分匹配本质(好题!)★★★  
3729I'm Telling the Truth最大匹配+输出字典序最大的匹配情况★★  
2389Rain on your Parade 最大匹配(HK算法)★★  
1054Strategic Game 最小点覆盖★  
2819Swap 行列匹配+输出解★★  
1669 Jamie's Contact Groups 二分+多重匹配★★  
3605Escape 多重匹配★  
3861The King’s Problem 强连通+最小路径覆盖★★  
2236无题II 二分+二分匹配★★  
1083Courses 最大匹配★  
1526A Plug for UNIX 最大匹配★  
2458Kindergarten 行列匹配★  
4160Dolls 最大匹配★  
4185Oil Skimming 黑白匹配★  
2413Against Mammoths 二分+二分匹配★★  
3468Treasure Hunting 最短路+二分匹配★★★  
3517 Adopt or not 最大独立集★★★  
3026Chinese Chess 二分匹配必须边★★★  
【POJ】  
1087A Plug for UNIX  
1274 The Perfect Stall  
1469COURSES  
1486 Sorting Slides 二分图的必须边  
1548Robots  
1698Alice's Chance  
1719Shooting Contest  
2060 Taxi Cab Scheme 最小路径覆盖  
2112 Optimal Milking 二分+多重匹配  
2226 Muddy Fields 行列的覆盖  
2239 Selecting Courses  
2289 Jamie's Contact Groups 二分+多重匹配  
2446 Chessboard  
2536 Gopher II  
2584T-Shirt Gumbo  
2594 Treasure Exploration 可相交最小路径覆盖  
2672Hotkeys  
2724 Purifying Machine  
3020 Antenna Placement  
3041 Asteroids 简单行列匹配   
3189Steady Cow Assignment 二分+多重匹配  
3216 Repairing Company  
3343 Against Mammoths  
3692 Kindergarten  
poj2771最大独立集

### 欧拉回路&哈密顿回路（待学习）

<https://blog.csdn.net/liuqiyao_01/article/details/8479339>

<https://blog.csdn.net/liuqiyao_01/article/details/26964205>

### 连通&LCA（学习区）

<https://blog.csdn.net/liuqiyao_01/article/details/8478730>

### 网络流（待学习）

<https://blog.csdn.net/shahdza/article/details/7779537>

### 2-SAT（待学习）

<https://blog.csdn.net/liuqiyao_01/article/details/8797336>

### KM匹配（待学习）

<https://blog.csdn.net/liuqiyao_01/article/details/8797346>

### Dancing Links（待学习）

<https://blog.csdn.net/liuqiyao_01/article/details/8478711>

### 一般图匹配（待学习）

<https://blog.csdn.net/liuqiyao_01/article/details/26963579>

## 线段树

<https://blog.csdn.net/trapper_c/article/details/51919980>

练习区（单点更新-成段更新）

hdu1754

hdu1394

hdu2795

poj2828

poj2886

hdu4288

CodeforcesBeta Round #19 D

poj2481

hdu3950

hdu4521

CodeforcesBeta Round #99 (Div. 1) C

hdu 4605

URAL 1989 Subpalindromes

hdu 4777

hdu1698

poj3468

poj2528

poj1436

poj2991  
uestc1425

uestc1546

CodeforcesRound #136 (Div. 2) D

Uva 12436

CodeforcesRound #169 (Div. 2) E

CodeforcesBeta Round #35 (Div. 2) E

Zoj3299

fzu2105

hdu 4533

URAL 1855

hdu 4578

hdu 4455

hdu 4614

hdu 4747

zoj 3724

cf343D

URAL 1977

学习区（区间合并-扫描线-其他）

hdu3397

hdu2871

hdu1540

CodeforcesBeta Round #43 D

hdu1828& poj 1177（同一题）

hdu1255

hdu 3642

poj2482

poj2464

hdu3255

uva 11983

hdu4052

uestc1525

hdu4419

zoj 3521

zoj 3525

hdu3954

hdu4027

hdu3333

hdu3016

hdu3340

ZOJ3511

UESTC1558

spojGSS21557

poj3162

hdu4358

hdu4267

hdu4417

UVALive4730

CodeforcesRound #163 (Div. 2) E

hdu 4638

hdu 4630

## 字符串算法

### KMP

<https://blog.csdn.net/chenguolinblog/article/details/16857765>

（已完结）

### 字典树

<https://blog.csdn.net/chenguolinblog/article/details/13625389>

### AC自动机

<https://blog.csdn.net/liuqiyao_01/article/details/8798241>

### 后缀数组

## 计算几何（模板库训练）

<https://blog.csdn.net/feizaoSYUACM/article/details/54835327>

<https://blog.csdn.net/chm517/article/details/44892713>

<https://blog.csdn.net/liuqiyao_01/article/details/8750742>

<https://blog.csdn.net/liuqiyao_01/article/details/26964261>

## 金银牌题库

HDU-5859-2016多校10-金牌题

HDU-5864-2016多校10-银牌题

HDU-5865-2016多校10-金牌题

HDU-5866-2016多校10-银牌题

## 温习题库

## 索引

**√（模板）-51nod-1264**

1264 90

**√-51nod-1265-四点共面**

1265 89

**√-51nod-1298-圆与三角形**

1298 87

**√-51nod-1995-三子棋**

1995 105

**√-51nod-2006-飞行员配对**

2006 102

**√-EOJ-110-2018月赛7-数蝌蚪-签到题**

110 100

**√-HDU-1003**

1003 24

**√-HDU-1087**

1087 24

**√-HDU-1159-LCIS模板题**

1159 26

**√-HDU-1358-Period**

**1358** 50

**√-HDU-1503-Advanced Fruits**

1503 25

**√-HDU-1542-Atlantis（矩形面积并）**

1542 65

**√-HDU-1686-Oulipo**

1686 44

**√-HDU-1711-Number Sequence**

1711 41

**√-HDU-2050-折线分割平面**

2050 23

**√-HDU-2087-剪花布条**

2087 46

√**-HDU-2089-不要62**

2089 29

**√-HDU-2602-Bone Collector**

2602 27

**√-HDU-3746-Cyclic Nacklace**

3746 48

**√-HDU-5857-2016多校10-签到**

5857 108

**√-HDU-5858-2016多校10-签到**

5858 94

**√-HDU-5861-2016多校10-铜牌**

5861 70

**√-HDU-5862-2016多校10-铜牌**

5862 67

**√-HDU-5867-2016多校10-签到**

5867 108

**√-HDU-6298-2018多校1-签到题**

6298 82

**√-HDU-6304-2018多校1-铜牌题**

6304 98

**√-HDU-6315-多校2铜牌题**

6315 56

**√-HDU-6330-Visual Cube-签到题**

6330 20

**√-POJ-2406-Power Strings**

2406 52

**√-POJ-2752-Seek the Name,Seek the Fame**

2752 54

**√-POJ-3259-Wormholes**

3259 39

√-POJ-3667-Hotel（线段树区间合并模板题）

3667 62

**√-牛客网2018暑期多校第1场-A（OEIS打表&组合数递推&逆元）** 95

**√-牛客网2018暑期多校第3场-C（可持久化平衡树）** 13

**√-牛客网2018暑期多校第3场-J（简单多边形与圆面积交）** 90

**√-牛客网2018暑期多校第4场-F-Beautiful Garden-签到题** 18

**HDU-1812-Count Teris**

1812 86

**TLE-POJ-2155-Matrix**

2155 70

**未做-hdu-2063**

2063 105

**原理不会-HDU-3923-Invoker**

3923 84

# 十三、暑期集训每日训练安排

## 学习训练实战补题循环队列

字符串（KMP-训练区）

数论-指数循环节题目

DP（背包-训练区）-G

图论（最短路-训练区）

图论（最小生成树-学习区）

图论（拓补排序-学习区）

DP（区间DP-学习区）-Z

线段树（训练区）-HJ

DP（数位DP-训练区）

实战补题区：

----------------------------分割线---------------------------------------------===============================================================================

对于计算几何，主要训练使用模板

对于图论（最短路、网络流、匹配）等建模题目，第一次学习手写，之后训练使用模板

对于线段树、动态规划、字符串算法，全手写

使用模板必须注明使用方法，并收录进入

线段树掌握基本知识点后进入刷题巩固期（题目来源为学习区）

动态规划刷完基本题后进入学习期

同时51nod补充签到题来源、其他知识点的补充

多校提供比赛真题（检验）

**7月 12日**

签到-51nod-1298-1265-1264-1459

铜牌-hdu-5862-5863-5860

专项-线段树-POJ3667-HDU1542-POJ2155

顺序

51nod-1298-1265-1264

51nod-1459

POJ3667-HDU1542-POJ2155

hdu-5862

hdu-5863-5860

**7月 13日**

上午（签到）-下午（线段树专项）-铜牌题

51nod-1265-1264-1459

UVA-567- ACM作业20180502-签到/铜

UVA-762- ACM作业20180502-签到/铜

HDU1542-POJ2155

hdu-5862-5863-5860

**7月 14日**

上午-签到&动态规划专项-下午-多校铜牌题

51nod-1459-签到

UVA-567- ACM作业20180502-签到

UVA-762- ACM作业20180502-签到

HDU-2602- ACM作业20180502-签到

HDU-1176- ACM作业20180502-签到

HDU-1114- ACM作业20180502-签到

hdu-5862-5863-5860-铜

POJ-2155-线段树专项

**7月 15日**

hihocoder编程练习赛68 12点-14点30

hihocoder挑战赛34 18点-20点

HDU-2018-动规专项

HDU-3923-ACM作业20180613--数学专项（Polya定理）

HDU-1114- ACM作业20180502-签到

POJ-3259-ACM作业20180509-签到

HDU-1812-ACM作业20180613-签到

POJ-3169-ACM作业20180509-签到

hdu-3308

HDU-5863-5860-铜

POJ-2155-线段树专项

**7月 16日**

动规专项2-线段树专项2-铜牌2-数学专项1-签到3

hdu 2050-动规专项

CF 429B-动规专项

hdu-3308-线段树专项

HDU-1812-ACM作业20180613-签到

POJ-3169-ACM作业20180509-签到

POJ-3255-ACM作业20180509-签到

HDU-5863-铜

HDU-5860-铜

POJ-2155-线段树专项-二维线段树

HDU-3923-ACM作业20180613--数学专项（Polya定理）

**7月 17日**

CF 429B-动规专项

hdu-3308-线段树专项

HDU-1812-ACM作业20180613-签到

POJ-3169-ACM作业20180509-签到

POJ-3255-ACM作业20180509-签到

HDU-2844-背包

HDU-5860-铜

POJ-2155-线段树专项-二维线段树

HDU-5863-铜-动规

HDU-3923-ACM作业20180613--数学专项（Polya定理）

计蒜客12点-17点 ACM3人训练

**7月20日**

多校补题

牛客1

√-A-卡特兰数&OEIS

<https://blog.csdn.net/wu_tongtong/article/details/78161211>

<https://blog.csdn.net/wlxsq/article/details/50838900>

√-E-dp

<https://www.nowcoder.com/discuss/87183?type=101&order=0&pos=6&page=0>

上午-牛客网补铜牌题-VJ刷题

√-[hdu 1003 Max Sum](http://acm.hdu.edu.cn/showproblem.php?pid=1003)

√-[hdu 1087 Super Jumping!](http://acm.hdu.edu.cn/showproblem.php?pid=1087)

**7月21日**

2018ACM-ICPC 上海大都会赛 热身赛&正赛

**7月22日**

牛客2补题

√-D-签到题-DP

日常训练题

√-[hdu 1503 Advanced Fruits](http://acm.hdu.edu.cn/showproblem.php?pid=1503)

**7月23日**

杭电多校-1（4题-铁铜）

（√谢正平）1001-签到

1002-铜牌

（√叶文豪）-1003-签到

（√谢正平）1004-铜牌

1005-金牌

1006-金牌

（赛后√叶文豪&孙宇洪&OEIS）1007-铜牌

1008-银牌

1009-金牌

1010-金牌

（√孙宇洪）1011-签到

**7月24日**

**7月25日**

√-1001-签到-数论-HDU6298

**杭电多校-2（3题-铜）**

1001-金牌题

1002-金牌题

1003-银牌题

（√谢正平&叶文豪&孙宇洪）1004-签到题-博弈

（WA）1005铜牌题-构造

1006-银牌题

（√叶文豪）1007铜牌题-线段树

1008-金牌题

1009-金牌题

（√谢正平&孙宇洪）1010-签到题

**7月26日**

日常训练

√-[hdu 1159 Common Subsequence](http://acm.hdu.edu.cn/showproblem.php?pid=1159)

多校补题

√-hdu-6312-Game-多校2签到题-博弈

**牛客多校3（68th/513-银）**

（√孙宇洪&谢正平）A-背包-签到题

B-金牌题

（√孙宇洪）C-区间操作模板题splay树-铜牌题

D-金牌题

（√叶文豪）E-字符串KMP循环节模板-铜牌题

F-金牌题

G-金牌题

（√谢正平）H-数论-签到题

I-计算几何（凸包）-银牌题

（√叶文豪）J-二分&计算几何模板（简单多边形与圆面积交&凸多边形面积）-铜牌题

**7月27日**

√-[hdu 2089 不要62](http://acm.hdu.edu.cn/showproblem.php?pid=2089) 简单数位dp

√-hdu 1711-KMP模板题

√-牛客3C-铜牌题-区间操作（splay树模板/rope大法）

**7月28日**

√-hdu 1686

**牛客多校4（1题；3人3题；114th/494；铜）**

(WA谢正平&叶文豪&孙宇洪)A-银牌-dp&扩展欧拉定理

B-金牌题-区间问题&线段树优化

C-银牌题-数学&数位DP

（√谢正平+）D-数学构造-铜牌题

E-金牌题-概率&线段树

（√叶文豪+）F-签到

（√谢正平+）G-铜牌题-枚举

H-金牌题-字符串

I-金牌题

J-银牌题-拓补排序&线段树优化建图

**7月29日**

补题-（未完全）牛客4A-递推&扩展欧拉定理

√-hdu 2087-KMP签到题

√-HDU-3746-KMP铁铜题

**7月30日**

√-hdu 1358-KMP铁铜题（结论题）

**杭电多校3（1题；3人3题-铁）**

A-铜牌

B-金牌

C-铜牌

（√谢正平）D-签到

E-金牌

（√谢正平）F-签到

G-铜牌

H-金牌

I-银牌

J-金牌

K-金牌

（√叶文豪）L-签到

M-银牌

## 7月31日

√-poj 2406-KMP

√-hdu 2203-KMP

√-poj 2752-KMP

## 8月1日

杭电多校4

## 8月2日

牛客多校5

## 8月3日

KMP专场

poj 3080

hdu 2594

hdu 3336

hdu 4300

hdu 1238

hdu 2328

hdu 3374

hdu 2609

fzu 1901

牛客1补题

J-莫队/线段树-铜牌

<https://www.nowcoder.com/discuss/87200?type=101>

D-图同构-铜牌

<https://www.nowcoder.com/discuss/87205?type=101&order=0&pos=3&page=1>

F-拉格朗日插值-银牌

<https://www.nowcoder.com/discuss/87198?type=101&order=0&pos=5&page=1>

牛客2补题

A-签到题-DP

I-铜牌题

J-铜牌题

牛客3补题

A-签到题-背包

E-补模板题-记录

H-签到题-数论

I-银牌

牛客4补题

B-金-区间&线段树

C-银

D-铜

E-金-概率

G-铜

I-金-建模？

J-银

日常训练

POJ-3169-ACM作业20180509-签到

POJ-3255-ACM作业20180509-签到

hdu-6313-多校2铜牌题

[hdu 3709 Balanced Number](http://acm.hdu.edu.cn/showproblem.php?pid=3709) 比较简单

CF 429B-动规专项

hdu-3308-线段树专项

HDU-2844-背包

HDU-5860-铜

POJ-2155-线段树专项-二维线段树

HDU-5863-铜-动规

HDU-3923-ACM作业20180613-Polya定理

HDU-1812-ACM作业20180613-polya定理

多校2签到题-HDU-6318

1002-铜-HDU6299

1004-铜-HDU6301

1008-银-HDU6305

HDU-1599-ACM作业20180516-签到/铜

HDU-3790-ACM作业20180509-签到/铜

HDU-1224-ACM作业20180509-签到/铜

HDU-1429- ACM作业2018404-签到/铜

HDU-1598- ACM作业2018404-签到/铜

HDU-2612- ACM作业20180328-签到/铜

[poj 1141 Brackets Sequence](http://blog.csdn.net/cc_again/article/details/10169643) 括号匹配并输出方案

动规学习区（数位DP、KMP）->区间DP专项训练->树形DP专项训练->背包专项训练

## 8月4日

牛客多校6

## 8月5日

## 8月6日

杭电多校5

# 十四、多校

牛客账号

帐号：15800739489 密码：acm2018

杭电账号

D:\ACM\2018MultiTraining\杭电多校账号.png

**团队分工**

**动态规划**

数位DP-孙宇洪

**图论**

并查集

最短路

最小生成树

拓补排序

二分匹配&网络流

**字符串算法**

KMP-叶文豪

字典树-谢正平

AC自动机

**数据结构**

线段树-叶文豪

**数学**

数论-谢正平

计算几何-叶文豪

## 牛客网-第1场-2018.7.19（周四）

铜牌题

√-打表OEIS&逆元

D-图同构

√-E-dp

F-拉格朗日插值

J-莫队/线段树

银牌题

B-DP

H-斜率优化

I-后缀数组/AC自动机

金牌题

C-矩阵行列式

G-求最小边数的 Dreyfus-Wagner 算法&根斯坦纳树

## ACM-ICPC-上海大都会赛-2018.7.21（周六）

热身赛

A-线段树-铜牌题

√-B-签到

C-数论-铜牌题

D

正赛

A-计算几何（随机？）-铜牌题

B-签到

C-金牌题

√-D-签到

E-金牌题

F-扫描线-铜牌题

G-计算几何-金牌题

H-线段树-金牌题

I-枚举贪心-铜牌题

J-DP-铜牌题

√-K-签到

L-金牌题

## 牛客网-第2场-2018.7.21（周六）

A-签到题

B-银牌题

C-金牌题

D-签到题

E-金牌题

F-金牌题

G-银牌题

H-银牌题

I-铜牌题

J-铜牌题

K-金牌题

## HDU-1-2018.7.23（周一）

1001-签到-数论

1002-铜牌

√-1003-签到

1004-铜牌

1005-金牌

1006-金牌

√-1007-铜牌-OEIS&数学递推

1008-银牌

1009-金牌

1010-金牌

1011-签到

## HDU-2-2018.7.25（周三）

1001-金牌题

1002-金牌题

1003-银牌题

已处理-1004-签到题-博弈

1005铜牌题-构造

1006-银牌题

已处理-1007铜牌题-线段树

1008-金牌题

1009-金牌题

1010-签到题

## 牛客网-第3场-2018.7.26（周四）

A-背包-签到题

B-金牌题

C-区间操作模板题splay树-铜牌题

D-金牌题

E-字符串KMP循环节模板-铜牌题

F-金牌题

G-金牌题

H-数论-签到题

I-计算几何（凸包）-银牌题

已处理-J-二分&计算几何模板（简单多边形与圆面积交&凸多边形面积）-铜牌题

## 牛客网-第4场-2018.7.28（周六）

已处理-A-银牌-dp&扩展欧拉定理

B-金牌题-区间问题&线段树优化

C-银牌题-数学&数位DP

D-数学构造-铜牌题

E-金牌题-概率&线段树

已处理-F-签到

G-铜牌题-枚举

H-金牌题-字符串

I-金牌题

J-银牌题-拓补排序&线段树优化建图

## HDU-3-2018.7.30（周一）

已处理-A-铜牌

B

已处理-C-铜牌

已处理-D-签到

E

已处理-F-签到

G

H

I

J

K

已处理-L-签到

M

## HDU-4-2018.8.1（周三）

## 牛客网-第5场-2018.8.2（周四）

## 牛客网-第6场-2018.8.4（周六）

## HDU-5-2018.8.6（周一）

## HDU-6-2018.8.8（周三）

## HDU-7-2018.8.13（周一）

## HDU-8-2018.8.15（周三）

## HDU-9-2018.8.20（周一）

## HDU-10-2018.8.22（周三）

## 2018中国大学生程序设计竞赛-网络选拔赛-2018.8.25（周六）

# 十五、浙大模板库

打钩为已验证

Zhejiang University 浙江大学ACM 模板

**1几何... 3**

1.1 注意…3 1.2 几何公式... 3 1.3 多边形... 5

1.4 多边形切割... 8 1.5 浮点函数... 9 1.6 面积... 14

1.7 球面... 15 1.8 三角形... 16 1.9 三维几何... 18

1.10 凸包... 26 1.11 网格... 27 1.12 圆... 29

1.13 整数函数... 30

**2、 组合... 32**

2.1 组合公式... 32 2.2 排列组合生成... 33 2.3 生成gray码... 35

2.4 置换(polya) 35 2.5 字典序全排列... 35 2.6 字典序组合... 36

**3、 数据结构... 36**

3.1 并查集... 36 3.2 堆... 37 3.3 线段树... 39

3.4 子段和... 43 3.5 子阵和... 44

**4、 数论... 45**

4.1 阶乘最后非0位... 45 4.2 模线性方程组... 45 4.3 素数... 47

4.4 欧拉函数... 48

**5、 数值计算... 48**

5.1 定积分计算(Romberg)48 5.2 多项式求根(牛顿法)50 5.3周期性方程(追赶法) 51

**6、 图论—NP搜索... 52**

6.1 最大团... 52 6.2 最大团(n<64)(faster) 53

**7、 图论—连通性... 55**

7.1 无向图关键点(dfs邻接阵) 55 7.2 无向图关键边(dfs邻接阵) 56

7.3 无向图的块(bfs邻接阵) 57 7.4 无向图连通分支(dfs/bfs邻接阵) 58

7.5 有向图强连通分支(dfs/bfs邻接阵) 58 7.6 有向图最小点基(邻接阵) 60

**8、 图论—匹配... 60**

8.1 二分图最大匹配(hungary邻接表) 60 8.2 二分图最大匹配(hungary邻接阵) 61

8.3 二分图最大匹配(hungary正向表) 61 8.4二分图最佳匹配(kuhn\_munkras邻接阵)62

8.5 一般图匹配(邻接表) 63 8.6 一般图匹配(邻接阵)64 8.7一般图匹配(正向表)65

**9、 图论—网络流... 66**

9.1 最大流(邻接阵)66 9.2上下界最大流(邻接阵)66 9.3 上下界最小流(邻接阵)67

9.4 最大流无流量(邻接阵)68 9.5 最小费用最大流(邻接阵) 68

**10、图论—应用...69**

10.1 欧拉回路(邻接阵) 69 10.2 树的前序表转化...70 10.3 树的优化算法...71

10.4 拓扑排序(邻接阵) 72 10.5 最佳边割集... 73 10.6 最佳点割集...74

10.7 最小边割集... 75 10.8 最小点割集... 76 10.9 最小路径覆盖..77

**11、 图论—支撑树... 78**

11.1 最小生成树(kruskal邻接表) 78 11.2 最小生成树(kruskal正向表) 79

11.3 最小生成树(prim+binary\_heap邻接表) 81

11.4 最小生成树(prim+binary\_heap正向表) 82

11.5 最小生成树(prim+mapped\_heap邻接表) 83

11.6 最小生成树(prim+mapped\_heap正向表) 84

11.7 最小生成树(prim邻接阵) 85 11.8 最小树形图(邻接阵) 86

**12、 图论—最短路径... 88**

12.1 最短路径(单源bellman\_ford邻接阵) 88

12.2 最短路径(单源dijkstra+bfs邻接表) 88

12.3 最短路径(单源dijkstra+bfs正向表) 89

12.4 最短路径(单源dijkstra+binary\_heap邻接表) 89

12.5 最短路径(单源dijkstra+binary\_heap正向表) 90

12.6 最短路径(单源dijkstra+mapped\_heap邻接表) 91

12.7 最短路径(单源dijkstra+mapped\_heap正向表) 93

12.8 最短路径(单源dijkstra邻接阵) 94

12.9 最短路径(多源floyd\_warshall邻接阵) 94

**13、 应用... 95**

13.1 Joseph问题...95 13.2 N皇后构造解... 96 13.3 布尔母函数... 96

13.4 第k元素... 97 13.5 幻方构造...97 13.6 模式匹配(kmp) 99

13.7 逆序对数... 99 13.8 字符串最小表示..99 13.9最长公共单调子序列..100

13.10 最长子序列..101 13.11 最大子串匹配...102 13.12 最大子段和... 103

13.13 最大子阵和... 103

**14、 其它... 104**

14.1 大数(只能处理正数) 104 14.2 分数... 110 14.3 矩阵... 112

14.4 线性方程组... 114 14.5 线性相关... 115 14.6 日期... 116

**1、几何**

**1.1 注意**

1. 注意舍入方式(0.5的舍入方向);防止输出-0.

2. 几何题注意多测试不对称数据.

3. 整数几何注意xmult和dmult是否会出界; 符点几何注意eps的使用.

4. 避免使用斜率;注意除数是否会为0.

5. 公式一定要化简后再代入.

6. 判断同一个2\*PI域内两角度差应该是

abs(a1-a2)<beta||abs(a1-a2)>pi+pi-beta;

相等应该是

abs(a1-a2)<eps||abs(a1-a2)>pi+pi-eps;

7. 需要的话尽量使用atan2,注意:atan2(0,0)=0,

atan2(1,0)=pi/2,atan2(-1,0)=-pi/2,atan2(0,1)=0,atan2(0,-1)=pi.

8. cross product = |u|\*|v|\*sin(a)

dot product = |u|\*|v|\*cos(a)

9. (P1-P0)x(P2-P0)结果的意义:

正: <P0,P1>在<P0,P2>顺时针(0,pi)内

负: <P0,P1>在<P0,P2>逆时针(0,pi)内

0 : <P0,P1>,<P0,P2>共线,夹角为0或pi

10. 误差限缺省使用1e-8!

**1.2 几何公式**

三角形:

1. 半周长 P=(a+b+c)/2

2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))

3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2

4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)

5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)

6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)

=4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)

=Ptan(A/2)tan(B/2)tan(C/2)

7. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))

四边形:

D1,D2为对角线,M对角线中点连线,A为对角线夹角

1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2

2. S=D1D2sin(A)/2

(以下对圆的内接四边形)

3. ac+bd=D1D2

4. S=sqrt((P-a)(P-b)(P-c)(P-d)),P为半周长

正n边形:

R为外接圆半径,r为内切圆半径

1. 中心角 A=2PI/n

2. 内角 C=(n-2)PI/n

3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)

4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

圆:

1. 弧长 l=rA

2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)

3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2

4. 扇形面积 S1=rl/2=r^2A/2

5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2

棱柱:

1. 体积 V=Ah,A为底面积,h为高

2. 侧面积 S=lp,l为棱长,p为直截面周长

3. 全面积 T=S+2A

棱锥:

1. 体积 V=Ah/3,A为底面积,h为高

(以下对正棱锥)

2. 侧面积 S=lp/2,l为斜高,p为底面周长

3. 全面积 T=S+A

棱台:

1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2为上下底面积,h为高

(以下为正棱台)

2. 侧面积 S=(p1+p2)l/2,p1.p2为上下底面周长,l为斜高

3. 全面积 T=S+A1+A2

圆柱:

1. 侧面积 S=2PIrh

2. 全面积 T=2PIr(h+r)

3. 体积 V=PIr^2h

圆锥:

1. 母线 l=sqrt(h^2+r^2)

2. 侧面积 S=PIrl

3. 全面积 T=PIr(l+r)

4. 体积 V=PIr^2h/3

圆台:

1. 母线 l=sqrt(h^2+(r1-r2)^2)

2. 侧面积 S=PI(r1+r2)l

3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)

4. 体积 V=PI(r1^2+r2^2+r1r2)h/3

球:

1. 全面积 T=4PIr^2

2. 体积 V=4PIr^3/3

球台:

1. 侧面积 S=2PIrh

2. 全面积 T=PI(2rh+r1^2+r2^2)

3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6

球扇形:

1. 全面积 T=PIr(2h+r0),h为球冠高,r0为球冠底面半径

2. 体积 V=2PIr^2h/3

**1.3 多边形**

#include <stdlib.h>

#include <math.h>

#define MAXN 1000

#define offset 10000

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

#define \_sign(x) ((x)>eps?1:((x)<-eps?2:0))

struct point{double x,y;};

struct line{point a,b;};

//

double xmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

//判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线

int is\_convex(int n,point\* p){

int i,s[3]={1,1,1};

for (i=0;i<n&&s[1]|s[2];i++)

s[\_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;

return s[1]|s[2];

}

//判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线

int is\_convex\_v2(int n,point\* p){

int i,s[3]={1,1,1};

for (i=0;i<n&&s[0]&&s[1]|s[2];i++)

s[\_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;

return s[0]&&s[1]|s[2];

}

//判点在凸多边形内或多边形边上,顶点按顺时针或逆时针给出

int inside\_convex(point q,int n,point\* p){

int i,s[3]={1,1,1};

for (i=0;i<n&&s[1]|s[2];i++)

s[\_sign(xmult(p[(i+1)%n],q,p[i]))]=0;

return s[1]|s[2];

}

//判点在凸多边形内,顶点按顺时针或逆时针给出,在多边形边上返回0

int inside\_convex\_v2(point q,int n,point\* p){

int i,s[3]={1,1,1};

for (i=0;i<n&&s[0]&&s[1]|s[2];i++)

s[\_sign(xmult(p[(i+1)%n],q,p[i]))]=0;

return s[0]&&s[1]|s[2];

}

//判点在任意多边形内,顶点按顺时针或逆时针给出

//on\_edge表示点在多边形边上时的返回值,offset为多边形坐标上限

int inside\_polygon(point q,int n,point\* p,int on\_edge=1){

point q2;

int i=0,count;

while (i<n)

for (count=i=0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i++)

if (zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)\*(p[(i+1)%n].x-q.x)<eps&&(p[i].y-q.y)\*(p[(i+1)%n].y-q.y)<eps)

return on\_edge;

else if (zero(xmult(q,q2,p[i])))

break;

else

if(xmult(q,p[i],q2)\*xmult(q,p[(i+1)%n],q2)<-eps&&xmult(p[i],q,p[(i+1)%n])\*xmult(p[i],q2,p[(i+1)%n])<-eps)

count++;

return count&1;

}

inline int opposite\_side(point p1,point p2,point l1,point l2){

return xmult(l1,p1,l2)\*xmult(l1,p2,l2)<-eps;

}

inline int dot\_online\_in(point p,point l1,point l2){

return zero(xmult(p,l1,l2))&&(l1.x-p.x)\*(l2.x-p.x)<eps&&(l1.y-p.y)\*(l2.y-p.y)<eps;

}

//判线段在任意多边形内,顶点按顺时针或逆时针给出,与边界相交返回1

int inside\_polygon(point l1,point l2,int n,point\* p){

point t[MAXN],tt;

int i,j,k=0;

if (!inside\_polygon(l1,n,p)||!inside\_polygon(l2,n,p))

return 0;

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

if (opposite\_side(l1,l2,p[i],p[(i+1)%n])&&opposite\_side(p[i],p[(i+1)%n],l1,l2))

return 0;

else if (dot\_online\_in(l1,p[i],p[(i+1)%n]))

t[k++]=l1;

else if (dot\_online\_in(l2,p[i],p[(i+1)%n]))

t[k++]=l2;

else if (dot\_online\_in(p[i],l1,l2))

t[k++]=p[i];

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

for (j=i+1;j<k;j++){

tt.x=(t[i].x+t[j].x)/2;

tt.y=(t[i].y+t[j].y)/2;

if (!inside\_polygon(tt,n,p))

return 0;

}

return 1;

}

point intersection(line u,line v){

point ret=u.a;

double t=((u.a.x-v.a.x)\*(v.a.y-v.b.y)-(u.a.y-v.a.y)\*(v.a.x-v.b.x))

/((u.a.x-u.b.x)\*(v.a.y-v.b.y)-(u.a.y-u.b.y)\*(v.a.x-v.b.x));

ret.x+=(u.b.x-u.a.x)\*t;

ret.y+=(u.b.y-u.a.y)\*t;

return ret;

}

point barycenter(point a,point b,point c){

line u,v;

u.a.x=(a.x+b.x)/2;

u.a.y=(a.y+b.y)/2;

u.b=c;

v.a.x=(a.x+c.x)/2;

v.a.y=(a.y+c.y)/2;

v.b=b;

return intersection(u,v);

}

//多边形重心

point barycenter(int n,point\* p){

point ret,t;

double t1=0,t2;

int i;

ret.x=ret.y=0;

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

if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){

t=barycenter(p[0],p[i],p[i+1]);

ret.x+=t.x\*t2;

ret.y+=t.y\*t2;

t1+=t2;

}

if (fabs(t1)>eps)

ret.x/=t1,ret.y/=t1;

return ret;

}

**1.4 多边形切割**

//多边形切割

//可用于半平面交

#define MAXN 100

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct point{double x,y;};

//

double xmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

//

int same\_side(point p1,point p2,point l1,point l2){

return xmult(l1,p1,l2)\*xmult(l1,p2,l2)>eps;

}

point intersection(point u1,point u2,point v1,point v2){

point ret=u1;

double t=((u1.x-v1.x)\*(v1.y-v2.y)-(u1.y-v1.y)\*(v1.x-v2.x))

/((u1.x-u2.x)\*(v1.y-v2.y)-(u1.y-u2.y)\*(v1.x-v2.x));

ret.x+=(u2.x-u1.x)\*t;

ret.y+=(u2.y-u1.y)\*t;

return ret;

}

//将多边形沿l1,l2确定的直线切割在side侧切割,保证l1,l2,side不共线

void polygon\_cut(int& n,point\* p,point l1,point l2,point side){

point pp[100];

int m=0,i;

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

if (same\_side(p[i],side,l1,l2))

pp[m++]=p[i];

if (!same\_side(p[i],p[(i+1)%n],l1,l2)&&!(zero(xmult(p[i],l1,l2))&&zero(xmult(p[(i+1)%n],l1,l2))))

pp[m++]=intersection(p[i],p[(i+1)%n],l1,l2);

}

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

if (!i||!zero(pp[i].x-pp[i-1].x)||!zero(pp[i].y-pp[i-1].y))

p[n++]=pp[i];

if (zero(p[n-1].x-p[0].x)&&zero(p[n-1].y-p[0].y))

n--;

if (n<3)

n=0;

}

**1.5 浮点函数**

//浮点几何函数库

#include <math.h>

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct point{double x,y;};

struct line{point a,b;};

//计算cross product (P1-P0)x(P2-P0)

double xmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

//

double xmult(double x1,double y1,double x2,double y2,double x0,double y0){

return (x1-x0)\*(y2-y0)-(x2-x0)\*(y1-y0);

}

//计算dot product (P1-P0).(P2-P0)

double dmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.x-p0.x)+(p1.y-p0.y)\*(p2.y-p0.y);

}

//

double dmult(double x1,double y1,double x2,double y2,double x0,double y0){

return (x1-x0)\*(x2-x0)+(y1-y0)\*(y2-y0);

}

//两点距离

double distance(point p1,point p2){

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

double distance(double x1,double y1,double x2,double y2){

return sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2));

}

//判三点共线

int dots\_inline(point p1,point p2,point p3){

return zero(xmult(p1,p2,p3));

}

int dots\_inline(double x1,double y1,double x2,double y2,double x3,double y3){

return zero(xmult(x1,y1,x2,y2,x3,y3));

}

//判点是否在线段上,包括端点

int dot\_online\_in(point p,line l){

return zero(xmult(p,l.a,l.b))&&(l.a.x-p.x)\*(l.b.x-p.x)<eps&&(l.a.y-p.y)\*(l.b.y-p.y)<eps;

}

int dot\_online\_in(point p,point l1,point l2){

return zero(xmult(p,l1,l2))&&(l1.x-p.x)\*(l2.x-p.x)<eps&&(l1.y-p.y)\*(l2.y-p.y)<eps;

}

int dot\_online\_in(double x,double y,double x1,double y1,double x2,double y2){

return zero(xmult(x,y,x1,y1,x2,y2))&&(x1-x)\*(x2-x)<eps&&(y1-y)\*(y2-y)<eps;

}

//判点是否在线段上,不包括端点

int dot\_online\_ex(point p,line l){

return dot\_online\_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))&&(!zero(p.x-l.b.x)||!zero(p.y-l.b.y));

}

int dot\_online\_ex(point p,point l1,point l2){

return dot\_online\_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y))&&(!zero(p.x-l2.x)||!zero(p.y-l2.y));

}

int dot\_online\_ex(double x,double y,double x1,double y1,double x2,double y2){

return dot\_online\_in(x,y,x1,y1,x2,y2)&&(!zero(x-x1)||!zero(y-y1))&&(!zero(x-x2)||!zero(y-y2));

}

//判两点在线段同侧,点在线段上返回0

int same\_side(point p1,point p2,line l){

return xmult(l.a,p1,l.b)\*xmult(l.a,p2,l.b)>eps;

}

int same\_side(point p1,point p2,point l1,point l2){

return xmult(l1,p1,l2)\*xmult(l1,p2,l2)>eps;

}

//判两点在线段异侧,点在线段上返回0

int opposite\_side(point p1,point p2,line l){

return xmult(l.a,p1,l.b)\*xmult(l.a,p2,l.b)<-eps;

}

int opposite\_side(point p1,point p2,point l1,point l2){

return xmult(l1,p1,l2)\*xmult(l1,p2,l2)<-eps;

}

//判两直线平行

int parallel(line u,line v){

return zero((u.a.x-u.b.x)\*(v.a.y-v.b.y)-(v.a.x-v.b.x)\*(u.a.y-u.b.y));

}

int parallel(point u1,point u2,point v1,point v2){

return zero((u1.x-u2.x)\*(v1.y-v2.y)-(v1.x-v2.x)\*(u1.y-u2.y));

}

//判两直线垂直

int perpendicular(line u,line v){

return zero((u.a.x-u.b.x)\*(v.a.x-v.b.x)+(u.a.y-u.b.y)\*(v.a.y-v.b.y));

}

int perpendicular(point u1,point u2,point v1,point v2){

return zero((u1.x-u2.x)\*(v1.x-v2.x)+(u1.y-u2.y)\*(v1.y-v2.y));

}

//判两线段相交,包括端点和部分重合

int intersect\_in(line u,line v){

if (!dots\_inline(u.a,u.b,v.a)||!dots\_inline(u.a,u.b,v.b))

return !same\_side(u.a,u.b,v)&&!same\_side(v.a,v.b,u);

return dot\_online\_in(u.a,v)||dot\_online\_in(u.b,v)||dot\_online\_in(v.a,u)||dot\_online\_in(v.b,u);

}

int intersect\_in(point u1,point u2,point v1,point v2){

if (!dots\_inline(u1,u2,v1)||!dots\_inline(u1,u2,v2))

return !same\_side(u1,u2,v1,v2)&&!same\_side(v1,v2,u1,u2);

return dot\_online\_in(u1,v1,v2)||dot\_online\_in(u2,v1,v2)||dot\_online\_in(v1,u1,u2)||dot\_online\_in(v2,u1,u2);

}

//判两线段相交,不包括端点和部分重合

int intersect\_ex(line u,line v){

return opposite\_side(u.a,u.b,v)&&opposite\_side(v.a,v.b,u);

}

int intersect\_ex(point u1,point u2,point v1,point v2){

return opposite\_side(u1,u2,v1,v2)&&opposite\_side(v1,v2,u1,u2);

}

//计算两直线交点,注意事先判断直线是否平行!

//线段交点请另外判线段相交(同时还是要判断是否平行!)

point intersection(line u,line v){

point ret=u.a;

double t=((u.a.x-v.a.x)\*(v.a.y-v.b.y)-(u.a.y-v.a.y)\*(v.a.x-v.b.x))

/((u.a.x-u.b.x)\*(v.a.y-v.b.y)-(u.a.y-u.b.y)\*(v.a.x-v.b.x));

ret.x+=(u.b.x-u.a.x)\*t;

ret.y+=(u.b.y-u.a.y)\*t;

return ret;

}

point intersection(point u1,point u2,point v1,point v2){

point ret=u1;

double t=((u1.x-v1.x)\*(v1.y-v2.y)-(u1.y-v1.y)\*(v1.x-v2.x))

/((u1.x-u2.x)\*(v1.y-v2.y)-(u1.y-u2.y)\*(v1.x-v2.x));

ret.x+=(u2.x-u1.x)\*t;

ret.y+=(u2.y-u1.y)\*t;

return ret;

}

//点到直线上的最近点

point ptoline(point p,line l){

point t=p;

t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;

return intersection(p,t,l.a,l.b);

}

point ptoline(point p,point l1,point l2){

point t=p;

t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;

return intersection(p,t,l1,l2);

}

//点到直线距离

double disptoline(point p,line l){

return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);

}

double disptoline(point p,point l1,point l2){

return fabs(xmult(p,l1,l2))/distance(l1,l2);

}

double disptoline(double x,double y,double x1,double y1,double x2,double y2){

return fabs(xmult(x,y,x1,y1,x2,y2))/distance(x1,y1,x2,y2);

}

//点到线段上的最近点

point ptoseg(point p,line l){

point t=p;

t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;

if (xmult(l.a,t,p)\*xmult(l.b,t,p)>eps)

return distance(p,l.a)<distance(p,l.b)?l.a:l.b;

return intersection(p,t,l.a,l.b);

}

point ptoseg(point p,point l1,point l2){

point t=p;

t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;

if (xmult(l1,t,p)\*xmult(l2,t,p)>eps)

return distance(p,l1)<distance(p,l2)?l1:l2;

return intersection(p,t,l1,l2);

}

//点到线段距离

double disptoseg(point p,line l){

point t=p;

t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;

if (xmult(l.a,t,p)\*xmult(l.b,t,p)>eps)

return distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance(p,l.b);

return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);

}

double disptoseg(point p,point l1,point l2){

point t=p;

t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;

if (xmult(l1,t,p)\*xmult(l2,t,p)>eps)

return distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,l2);

return fabs(xmult(p,l1,l2))/distance(l1,l2);

}

//矢量V以P为顶点逆时针旋转angle并放大scale倍

point rotate(point v,point p,double angle,double scale){

point ret=p;

v.x-=p.x,v.y-=p.y;

p.x=scale\*cos(angle);

p.y=scale\*sin(angle);

ret.x+=v.x\*p.x-v.y\*p.y;

ret.y+=v.x\*p.y+v.y\*p.x;

return ret;

}

**1.6 面积**

#include <math.h>

struct point{double x,y;};

//计算cross product (P1-P0)x(P2-P0)

double xmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

double xmult(double x1,double y1,double x2,double y2,double x0,double y0){

return (x1-x0)\*(y2-y0)-(x2-x0)\*(y1-y0);

}

//计算三角形面积,输入三顶点

double area\_triangle(point p1,point p2,point p3){

return fabs(xmult(p1,p2,p3))/2;

}

double area\_triangle(double x1,double y1,double x2,double y2,double x3,double y3){

return fabs(xmult(x1,y1,x2,y2,x3,y3))/2;

}

//计算三角形面积,输入三边长

double area\_triangle(double a,double b,double c){

double s=(a+b+c)/2;

return sqrt(s\*(s-a)\*(s-b)\*(s-c));

}

//计算多边形面积,顶点按顺时针或逆时针给出

double area\_polygon(int n,point\* p){

double s1=0,s2=0;

int i;

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

s1+=p[(i+1)%n].y\*p[i].x,s2+=p[(i+1)%n].y\*p[(i+2)%n].x;

return fabs(s1-s2)/2;

}

**1.7 球面**

#include <math.h>

const double pi=acos(-1);

//计算圆心角lat表示纬度,-90<=w<=90,lng表示经度

//返回两点所在大圆劣弧对应圆心角,0<=angle<=pi

double angle(double lng1,double lat1,double lng2,double lat2){

double dlng=fabs(lng1-lng2)\*pi/180;

while (dlng>=pi+pi)

dlng-=pi+pi;

if (dlng>pi)

dlng=pi+pi-dlng;

lat1\*=pi/180,lat2\*=pi/180;

return acos(cos(lat1)\*cos(lat2)\*cos(dlng)+sin(lat1)\*sin(lat2));

}

//计算距离,r为球半径

double line\_dist(double r,double lng1,double lat1,double lng2,double lat2){

double dlng=fabs(lng1-lng2)\*pi/180;

while (dlng>=pi+pi)

dlng-=pi+pi;

if (dlng>pi)

dlng=pi+pi-dlng;

lat1\*=pi/180,lat2\*=pi/180;

return r\*sqrt(2-2\*(cos(lat1)\*cos(lat2)\*cos(dlng)+sin(lat1)\*sin(lat2)));

}

//计算球面距离,r为球半径

inline double sphere\_dist(double r,double lng1,double lat1,double lng2,double lat2){

return r\*angle(lng1,lat1,lng2,lat2);

}

**1.8 三角形**

#include <math.h>

struct point{double x,y;};

struct line{point a,b;};

double distance(point p1,point p2){

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

point intersection(line u,line v){

point ret=u.a;

double t=((u.a.x-v.a.x)\*(v.a.y-v.b.y)-(u.a.y-v.a.y)\*(v.a.x-v.b.x))

/((u.a.x-u.b.x)\*(v.a.y-v.b.y)-(u.a.y-u.b.y)\*(v.a.x-v.b.x));

ret.x+=(u.b.x-u.a.x)\*t;

ret.y+=(u.b.y-u.a.y)\*t;

return ret;

}

//外心

point circumcenter(point a,point b,point c){

line u,v;

u.a.x=(a.x+b.x)/2;

u.a.y=(a.y+b.y)/2;

u.b.x=u.a.x-a.y+b.y;

u.b.y=u.a.y+a.x-b.x;

v.a.x=(a.x+c.x)/2;

v.a.y=(a.y+c.y)/2;

v.b.x=v.a.x-a.y+c.y;

v.b.y=v.a.y+a.x-c.x;

return intersection(u,v);

}

//内心

point incenter(point a,point b,point c){

line u,v;

double m,n;

u.a=a;

m=atan2(b.y-a.y,b.x-a.x);

n=atan2(c.y-a.y,c.x-a.x);

u.b.x=u.a.x+cos((m+n)/2);

u.b.y=u.a.y+sin((m+n)/2);

v.a=b;

m=atan2(a.y-b.y,a.x-b.x);

n=atan2(c.y-b.y,c.x-b.x);

v.b.x=v.a.x+cos((m+n)/2);

v.b.y=v.a.y+sin((m+n)/2);

return intersection(u,v);

}

//垂心

point perpencenter(point a,point b,point c){

line u,v;

u.a=c;

u.b.x=u.a.x-a.y+b.y;

u.b.y=u.a.y+a.x-b.x;

v.a=b;

v.b.x=v.a.x-a.y+c.y;

v.b.y=v.a.y+a.x-c.x;

return intersection(u,v);

}

//重心

//到三角形三顶点距离的平方和最小的点

//三角形内到三边距离之积最大的点

point barycenter(point a,point b,point c){

line u,v;

u.a.x=(a.x+b.x)/2;

u.a.y=(a.y+b.y)/2;

u.b=c;

v.a.x=(a.x+c.x)/2;

v.a.y=(a.y+c.y)/2;

v.b=b;

return intersection(u,v);

}

//费马点

//到三角形三顶点距离之和最小的点

point fermentpoint(point a,point b,point c){

point u,v;

double step=fabs(a.x)+fabs(a.y)+fabs(b.x)+fabs(b.y)+fabs(c.x)+fabs(c.y);

int i,j,k;

u.x=(a.x+b.x+c.x)/3;

u.y=(a.y+b.y+c.y)/3;

while (step>1e-10)

for (k=0;k<10;step/=2,k++)

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

for (j=-1;j<=1;j++){

v.x=u.x+step\*i;

v.y=u.y+step\*j;

if (distance(u,a)+distance(u,b)+distance(u,c)>distance(v,a)+distance(v,b)+distance(v,c))

u=v;

}

return u;

}

**1.9 三维几何**

//三维几何函数库

#include <math.h>

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct point3{double x,y,z;};

struct line3{point3 a,b;};

struct plane3{point3 a,b,c;};

//计算cross product U x V

point3 xmult(point3 u,point3 v){

point3 ret;

ret.x=u.y\*v.z-v.y\*u.z;

ret.y=u.z\*v.x-u.x\*v.z;

ret.z=u.x\*v.y-u.y\*v.x;

return ret;

}

//计算dot product U . V

double dmult(point3 u,point3 v){

return u.x\*v.x+u.y\*v.y+u.z\*v.z;

}

//矢量差 U - V

point3 subt(point3 u,point3 v){

point3 ret;

ret.x=u.x-v.x;

ret.y=u.y-v.y;

ret.z=u.z-v.z;

return ret;

}

//取平面法向量

point3 pvec(plane3 s){

return xmult(subt(s.a,s.b),subt(s.b,s.c));

}

point3 pvec(point3 s1,point3 s2,point3 s3){

return xmult(subt(s1,s2),subt(s2,s3));

}

//两点距离,单参数取向量大小

double distance(point3 p1,point3 p2){

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y)+(p1.z-p2.z)\*(p1.z-p2.z));

}

//向量大小

double vlen(point3 p){

return sqrt(p.x\*p.x+p.y\*p.y+p.z\*p.z);

}

//判三点共线

int dots\_inline(point3 p1,point3 p2,point3 p3){

return vlen(xmult(subt(p1,p2),subt(p2,p3)))<eps;

}

//判四点共面

int dots\_onplane(point3 a,point3 b,point3 c,point3 d){

return zero(dmult(pvec(a,b,c),subt(d,a)));

}

//判点是否在线段上,包括端点和共线

int dot\_online\_in(point3 p,line3 l){

return zero(vlen(xmult(subt(p,l.a),subt(p,l.b))))&&(l.a.x-p.x)\*(l.b.x-p.x)<eps&&

(l.a.y-p.y)\*(l.b.y-p.y)<eps&&(l.a.z-p.z)\*(l.b.z-p.z)<eps;

}

int dot\_online\_in(point3 p,point3 l1,point3 l2){

return zero(vlen(xmult(subt(p,l1),subt(p,l2))))&&(l1.x-p.x)\*(l2.x-p.x)<eps&&

(l1.y-p.y)\*(l2.y-p.y)<eps&&(l1.z-p.z)\*(l2.z-p.z)<eps;

}

//判点是否在线段上,不包括端点

int dot\_online\_ex(point3 p,line3 l){

return dot\_online\_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y)||!zero(p.z-l.a.z))&&

(!zero(p.x-l.b.x)||!zero(p.y-l.b.y)||!zero(p.z-l.b.z));

}

int dot\_online\_ex(point3 p,point3 l1,point3 l2){

return dot\_online\_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y)||!zero(p.z-l1.z))&&

(!zero(p.x-l2.x)||!zero(p.y-l2.y)||!zero(p.z-l2.z));

}

//判点是否在空间三角形上,包括边界,三点共线无意义

int dot\_inplane\_in(point3 p,plane3 s){

return zero(vlen(xmult(subt(s.a,s.b),subt(s.a,s.c)))-vlen(xmult(subt(p,s.a),subt(p,s.b)))-

vlen(xmult(subt(p,s.b),subt(p,s.c)))-vlen(xmult(subt(p,s.c),subt(p,s.a))));

}

int dot\_inplane\_in(point3 p,point3 s1,point3 s2,point3 s3){

return zero(vlen(xmult(subt(s1,s2),subt(s1,s3)))-vlen(xmult(subt(p,s1),subt(p,s2)))-

vlen(xmult(subt(p,s2),subt(p,s3)))-vlen(xmult(subt(p,s3),subt(p,s1))));

}

//判点是否在空间三角形上,不包括边界,三点共线无意义

int dot\_inplane\_ex(point3 p,plane3 s){

return dot\_inplane\_in(p,s)&&vlen(xmult(subt(p,s.a),subt(p,s.b)))>eps&& vlen(xmult(subt(p,s.b),subt(p,s.c)))>eps&&vlen(xmult(subt(p,s.c),subt(p,s.a)))>eps;

}

int dot\_inplane\_ex(point3 p,point3 s1,point3 s2,point3 s3){

return dot\_inplane\_in(p,s1,s2,s3)&&vlen(xmult(subt(p,s1),subt(p,s2)))>eps&& vlen(xmult(subt(p,s2),subt(p,s3)))>eps&&vlen(xmult(subt(p,s3),subt(p,s1)))>eps;

}

//判两点在线段同侧,点在线段上返回0,不共面无意义

int same\_side(point3 p1,point3 p2,line3 l){

return dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))>eps;

}

int same\_side(point3 p1,point3 p2,point3 l1,point3 l2){

return dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))>eps;

}

//判两点在线段异侧,点在线段上返回0,不共面无意义

int opposite\_side(point3 p1,point3 p2,line3 l){

return dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))<-eps;

}

int opposite\_side(point3 p1,point3 p2,point3 l1,point3 l2){

return dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))<-eps;

}

//判两点在平面同侧,点在平面上返回0

int same\_side(point3 p1,point3 p2,plane3 s){

return dmult(pvec(s),subt(p1,s.a))\*dmult(pvec(s),subt(p2,s.a))>eps;

}

int same\_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){

return dmult(pvec(s1,s2,s3),subt(p1,s1))\*dmult(pvec(s1,s2,s3),subt(p2,s1))>eps;

}

//判两点在平面异侧,点在平面上返回0

int opposite\_side(point3 p1,point3 p2,plane3 s){

return dmult(pvec(s),subt(p1,s.a))\*dmult(pvec(s),subt(p2,s.a))<-eps;

}

int opposite\_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){

return dmult(pvec(s1,s2,s3),subt(p1,s1))\*dmult(pvec(s1,s2,s3),subt(p2,s1))<-eps;

}

//判两直线平行

int parallel(line3 u,line3 v){

return vlen(xmult(subt(u.a,u.b),subt(v.a,v.b)))<eps;

}

int parallel(point3 u1,point3 u2,point3 v1,point3 v2){

return vlen(xmult(subt(u1,u2),subt(v1,v2)))<eps;

}

//判两平面平行

int parallel(plane3 u,plane3 v){

return vlen(xmult(pvec(u),pvec(v)))<eps;

}

int parallel(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){

return vlen(xmult(pvec(u1,u2,u3),pvec(v1,v2,v3)))<eps;

}

//判直线与平面平行

int parallel(line3 l,plane3 s){

return zero(dmult(subt(l.a,l.b),pvec(s)));

}

int parallel(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){

return zero(dmult(subt(l1,l2),pvec(s1,s2,s3)));

}

//判两直线垂直

int perpendicular(line3 u,line3 v){

return zero(dmult(subt(u.a,u.b),subt(v.a,v.b)));

}

int perpendicular(point3 u1,point3 u2,point3 v1,point3 v2){

return zero(dmult(subt(u1,u2),subt(v1,v2)));

}

//判两平面垂直

int perpendicular(plane3 u,plane3 v){

return zero(dmult(pvec(u),pvec(v)));

}

int perpendicular(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){

return zero(dmult(pvec(u1,u2,u3),pvec(v1,v2,v3)));

}

//判直线与平面平行

int perpendicular(line3 l,plane3 s){

return vlen(xmult(subt(l.a,l.b),pvec(s)))<eps;

}

int perpendicular(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){

return vlen(xmult(subt(l1,l2),pvec(s1,s2,s3)))<eps;

}

//判两线段相交,包括端点和部分重合

int intersect\_in(line3 u,line3 v){

if (!dots\_onplane(u.a,u.b,v.a,v.b))

return 0;

if (!dots\_inline(u.a,u.b,v.a)||!dots\_inline(u.a,u.b,v.b))

return !same\_side(u.a,u.b,v)&&!same\_side(v.a,v.b,u);

return dot\_online\_in(u.a,v)||dot\_online\_in(u.b,v)||dot\_online\_in(v.a,u)||dot\_online\_in(v.b,u);

}

int intersect\_in(point3 u1,point3 u2,point3 v1,point3 v2){

if (!dots\_onplane(u1,u2,v1,v2))

return 0;

if (!dots\_inline(u1,u2,v1)||!dots\_inline(u1,u2,v2))

return !same\_side(u1,u2,v1,v2)&&!same\_side(v1,v2,u1,u2);

return dot\_online\_in(u1,v1,v2)||dot\_online\_in(u2,v1,v2)||dot\_online\_in(v1,u1,u2)||dot\_online\_in(v2,u1,u2);

}

//判两线段相交,不包括端点和部分重合

int intersect\_ex(line3 u,line3 v){

return dots\_onplane(u.a,u.b,v.a,v.b)&&opposite\_side(u.a,u.b,v)&&opposite\_side(v.a,v.b,u);

}

int intersect\_ex(point3 u1,point3 u2,point3 v1,point3 v2){

return dots\_onplane(u1,u2,v1,v2)&&opposite\_side(u1,u2,v1,v2)&&opposite\_side(v1,v2,u1,u2);

}

//判线段与空间三角形相交,包括交于边界和(部分)包含

int intersect\_in(line3 l,plane3 s){

return !same\_side(l.a,l.b,s)&&!same\_side(s.a,s.b,l.a,l.b,s.c)&&

!same\_side(s.b,s.c,l.a,l.b,s.a)&&!same\_side(s.c,s.a,l.a,l.b,s.b);

}

int intersect\_in(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){

return !same\_side(l1,l2,s1,s2,s3)&&!same\_side(s1,s2,l1,l2,s3)&&

!same\_side(s2,s3,l1,l2,s1)&&!same\_side(s3,s1,l1,l2,s2);

}

//判线段与空间三角形相交,不包括交于边界和(部分)包含

int intersect\_ex(line3 l,plane3 s){

return opposite\_side(l.a,l.b,s)&&opposite\_side(s.a,s.b,l.a,l.b,s.c)&&

opposite\_side(s.b,s.c,l.a,l.b,s.a)&&opposite\_side(s.c,s.a,l.a,l.b,s.b);

}

int intersect\_ex(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){

return opposite\_side(l1,l2,s1,s2,s3)&&opposite\_side(s1,s2,l1,l2,s3)&&

opposite\_side(s2,s3,l1,l2,s1)&&opposite\_side(s3,s1,l1,l2,s2);

}

//计算两直线交点,注意事先判断直线是否共面和平行!

//线段交点请另外判线段相交(同时还是要判断是否平行!)

point3 intersection(line3 u,line3 v){

point3 ret=u.a;

double t=((u.a.x-v.a.x)\*(v.a.y-v.b.y)-(u.a.y-v.a.y)\*(v.a.x-v.b.x))

/((u.a.x-u.b.x)\*(v.a.y-v.b.y)-(u.a.y-u.b.y)\*(v.a.x-v.b.x));

ret.x+=(u.b.x-u.a.x)\*t;

ret.y+=(u.b.y-u.a.y)\*t;

ret.z+=(u.b.z-u.a.z)\*t;

return ret;

}

point3 intersection(point3 u1,point3 u2,point3 v1,point3 v2){

point3 ret=u1;

double t=((u1.x-v1.x)\*(v1.y-v2.y)-(u1.y-v1.y)\*(v1.x-v2.x))

/((u1.x-u2.x)\*(v1.y-v2.y)-(u1.y-u2.y)\*(v1.x-v2.x));

ret.x+=(u2.x-u1.x)\*t;

ret.y+=(u2.y-u1.y)\*t;

ret.z+=(u2.z-u1.z)\*t;

return ret;

}

//计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!

//线段和空间三角形交点请另外判断

point3 intersection(line3 l,plane3 s){

point3 ret=pvec(s);

double t=(ret.x\*(s.a.x-l.a.x)+ret.y\*(s.a.y-l.a.y)+ret.z\*(s.a.z-l.a.z))/

(ret.x\*(l.b.x-l.a.x)+ret.y\*(l.b.y-l.a.y)+ret.z\*(l.b.z-l.a.z));

ret.x=l.a.x+(l.b.x-l.a.x)\*t;

ret.y=l.a.y+(l.b.y-l.a.y)\*t;

ret.z=l.a.z+(l.b.z-l.a.z)\*t;

return ret;

}

point3 intersection(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){

point3 ret=pvec(s1,s2,s3);

double t=(ret.x\*(s1.x-l1.x)+ret.y\*(s1.y-l1.y)+ret.z\*(s1.z-l1.z))/

(ret.x\*(l2.x-l1.x)+ret.y\*(l2.y-l1.y)+ret.z\*(l2.z-l1.z));

ret.x=l1.x+(l2.x-l1.x)\*t;

ret.y=l1.y+(l2.y-l1.y)\*t;

ret.z=l1.z+(l2.z-l1.z)\*t;

return ret;

}

//计算两平面交线,注意事先判断是否平行,并保证三点不共线!

line3 intersection(plane3 u,plane3 v){

line3 ret;

ret.a=parallel(v.a,v.b,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):intersection(v.a,v.b,u.a,u.b,u.c);

ret.b=parallel(v.c,v.a,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):intersection(v.c,v.a,u.a,u.b,u.c);

return ret;

}

line3 intersection(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){

line3 ret;

ret.a=parallel(v1,v2,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v1,v2,u1,u2,u3);

ret.b=parallel(v3,v1,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v3,v1,u1,u2,u3);

return ret;

}

//点到直线距离

double ptoline(point3 p,line3 l){

return vlen(xmult(subt(p,l.a),subt(l.b,l.a)))/distance(l.a,l.b);

}

double ptoline(point3 p,point3 l1,point3 l2){

return vlen(xmult(subt(p,l1),subt(l2,l1)))/distance(l1,l2);

}

//点到平面距离

double ptoplane(point3 p,plane3 s){

return fabs(dmult(pvec(s),subt(p,s.a)))/vlen(pvec(s));

}

double ptoplane(point3 p,point3 s1,point3 s2,point3 s3){

return fabs(dmult(pvec(s1,s2,s3),subt(p,s1)))/vlen(pvec(s1,s2,s3));

}

//直线到直线距离

double linetoline(line3 u,line3 v){

point3 n=xmult(subt(u.a,u.b),subt(v.a,v.b));

return fabs(dmult(subt(u.a,v.a),n))/vlen(n);

}

double linetoline(point3 u1,point3 u2,point3 v1,point3 v2){

point3 n=xmult(subt(u1,u2),subt(v1,v2));

return fabs(dmult(subt(u1,v1),n))/vlen(n);

}

//两直线夹角cos值

double angle\_cos(line3 u,line3 v){

return dmult(subt(u.a,u.b),subt(v.a,v.b))/vlen(subt(u.a,u.b))/vlen(subt(v.a,v.b));

}

double angle\_cos(point3 u1,point3 u2,point3 v1,point3 v2){

return dmult(subt(u1,u2),subt(v1,v2))/vlen(subt(u1,u2))/vlen(subt(v1,v2));

}

//两平面夹角cos值

double angle\_cos(plane3 u,plane3 v){

return dmult(pvec(u),pvec(v))/vlen(pvec(u))/vlen(pvec(v));

}

double angle\_cos(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){

return dmult(pvec(u1,u2,u3),pvec(v1,v2,v3))/vlen(pvec(u1,u2,u3))/vlen(pvec(v1,v2,v3));

}

//直线平面夹角sin值

double angle\_sin(line3 l,plane3 s){

return dmult(subt(l.a,l.b),pvec(s))/vlen(subt(l.a,l.b))/vlen(pvec(s));

}

double angle\_sin(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){

return dmult(subt(l1,l2),pvec(s1,s2,s3))/vlen(subt(l1,l2))/vlen(pvec(s1,s2,s3));

}

**1.10 凸包**

#include <stdlib.h>

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct point{double x,y;};

//计算cross product (P1-P0)x(P2-P0)

double xmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

//graham算法顺时针构造包含所有共线点的凸包,O(nlogn)

point p1,p2;

int graham\_cp(const void\* a,const void\* b){

double ret=xmult(\*((point\*)a),\*((point\*)b),p1);

return zero(ret)?(xmult(\*((point\*)a),\*((point\*)b),p2)>0?1:-1):(ret>0?1:-1);

}

void \_graham(int n,point\* p,int& s,point\* ch){

int i,k=0;

for (p1=p2=p[0],i=1;i<n;p2.x+=p[i].x,p2.y+=p[i].y,i++)

if (p1.y-p[i].y>eps||(zero(p1.y-p[i].y)&&p1.x>p[i].x))

p1=p[k=i];

p2.x/=n,p2.y/=n;

p[k]=p[0],p[0]=p1;

qsort(p+1,n-1,sizeof(point),graham\_cp);

for (ch[0]=p[0],ch[1]=p[1],ch[2]=p[2],s=i=3;i<n;ch[s++]=p[i++])

for (;s>2&&xmult(ch[s-2],p[i],ch[s-1])<-eps;s--);

}

//构造凸包接口函数,传入原始点集大小n,点集p(p原有顺序被打乱!)

//返回凸包大小,凸包的点在convex中

//参数maxsize为1包含共线点,为0不包含共线点,缺省为1

//参数clockwise为1顺时针构造,为0逆时针构造,缺省为1

//在输入仅有若干共线点时算法不稳定,可能有此类情况请另行处理!

//不能去掉点集中重合的点

int graham(int n,point\* p,point\* convex,int maxsize=1,int dir=1){

point\* temp=new point[n];

int s,i;

\_graham(n,p,s,temp);

for (convex[0]=temp[0],n=1,i=(dir?1:(s-1));dir?(i<s):i;i+=(dir?1:-1))

if (maxsize||!zero(xmult(temp[i-1],temp[i],temp[(i+1)%s])))

convex[n++]=temp[i];

delete []temp;

return n;

}

**1.11 网格**

#define abs(x) ((x)>0?(x):-(x))

struct point{int x,y;};

int gcd(int a,int b){

return b?gcd(b,a%b):a;

}

//多边形上的网格点个数

int grid\_onedge(int n,point\* p){

int i,ret=0;

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

ret+=gcd(abs(p[i].x-p[(i+1)%n].x),abs(p[i].y-p[(i+1)%n].y));

return ret;

}

//多边形内的网格点个数

int grid\_inside(int n,point\* p){

int i,ret=0;

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

ret+=p[(i+1)%n].y\*(p[i].x-p[(i+2)%n].x);

return (abs(ret)-grid\_onedge(n,p))/2+1;

}

**1.12 圆**

#include <math.h>

#define eps 1e-8

struct point{double x,y;};

double xmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

double distance(point p1,point p2){

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

double disptoline(point p,point l1,point l2){

return fabs(xmult(p,l1,l2))/distance(l1,l2);

}

point intersection(point u1,point u2,point v1,point v2){

point ret=u1;

double t=((u1.x-v1.x)\*(v1.y-v2.y)-(u1.y-v1.y)\*(v1.x-v2.x))

/((u1.x-u2.x)\*(v1.y-v2.y)-(u1.y-u2.y)\*(v1.x-v2.x));

ret.x+=(u2.x-u1.x)\*t;

ret.y+=(u2.y-u1.y)\*t;

return ret;

}

//判直线和圆相交,包括相切

int intersect\_line\_circle(point c,double r,point l1,point l2){

return disptoline(c,l1,l2)<r+eps;

}

//判线段和圆相交,包括端点和相切

int intersect\_seg\_circle(point c,double r,point l1,point l2){

double t1=distance(c,l1)-r,t2=distance(c,l2)-r;

point t=c;

if (t1<eps||t2<eps)

return t1>-eps||t2>-eps;

t.x+=l1.y-l2.y;

t.y+=l2.x-l1.x;

return xmult(l1,c,t)\*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;

}

//判圆和圆相交,包括相切

int intersect\_circle\_circle(point c1,double r1,point c2,double r2){

return distance(c1,c2)<r1+r2+eps&&distance(c1,c2)>fabs(r1-r2)-eps;

}

//计算圆上到点p最近点,如p与圆心重合,返回p本身

point dot\_to\_circle(point c,double r,point p){

point u,v;

if (distance(p,c)<eps)

return p;

u.x=c.x+r\*fabs(c.x-p.x)/distance(c,p);

u.y=c.y+r\*fabs(c.y-p.y)/distance(c,p)\*((c.x-p.x)\*(c.y-p.y)<0?-1:1);

v.x=c.x-r\*fabs(c.x-p.x)/distance(c,p);

v.y=c.y-r\*fabs(c.y-p.y)/distance(c,p)\*((c.x-p.x)\*(c.y-p.y)<0?-1:1);

return distance(u,p)<distance(v,p)?u:v;

}

//计算直线与圆的交点,保证直线与圆有交点

//计算线段与圆的交点可用这个函数后判点是否在线段上

void intersection\_line\_circle(point c,double r,point l1,point l2,point& p1,point& p2){

point p=c;

double t;

p.x+=l1.y-l2.y;

p.y+=l2.x-l1.x;

p=intersection(p,c,l1,l2);

t=sqrt(r\*r-distance(p,c)\*distance(p,c))/distance(l1,l2);

p1.x=p.x+(l2.x-l1.x)\*t;

p1.y=p.y+(l2.y-l1.y)\*t;

p2.x=p.x-(l2.x-l1.x)\*t;

p2.y=p.y-(l2.y-l1.y)\*t;

}

//计算圆与圆的交点,保证圆与圆有交点,圆心不重合

void intersection\_circle\_circle(point c1,double r1,point c2,double r2,point& p1,point& p2){

point u,v;

double t;

t=(1+(r1\*r1-r2\*r2)/distance(c1,c2)/distance(c1,c2))/2;

u.x=c1.x+(c2.x-c1.x)\*t;

u.y=c1.y+(c2.y-c1.y)\*t;

v.x=u.x+c1.y-c2.y;

v.y=u.y-c1.x+c2.x;

intersection\_line\_circle(c1,r1,u,v,p1,p2);

}

**1.13 整数函数**

//整数几何函数库

//注意某些情况下整数运算会出界!

#define sign(a) ((a)>0?1:(((a)<0?-1:0)))

struct point{int x,y;};

struct line{point a,b;};

//计算cross product (P1-P0)x(P2-P0)

int xmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

int xmult(int x1,int y1,int x2,int y2,int x0,int y0){

return (x1-x0)\*(y2-y0)-(x2-x0)\*(y1-y0);

}

//计算dot product (P1-P0).(P2-P0)

int dmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.x-p0.x)+(p1.y-p0.y)\*(p2.y-p0.y);

}

int dmult(int x1,int y1,int x2,int y2,int x0,int y0){

return (x1-x0)\*(x2-x0)+(y1-y0)\*(y2-y0);

}

//判三点共线

int dots\_inline(point p1,point p2,point p3){

return !xmult(p1,p2,p3);

}

int dots\_inline(int x1,int y1,int x2,int y2,int x3,int y3){

return !xmult(x1,y1,x2,y2,x3,y3);

}

//判点是否在线段上,包括端点和部分重合

int dot\_online\_in(point p,line l){

return !xmult(p,l.a,l.b)&&(l.a.x-p.x)\*(l.b.x-p.x)<=0&&(l.a.y-p.y)\*(l.b.y-p.y)<=0

;

}

int dot\_online\_in(point p,point l1,point l2){

return !xmult(p,l1,l2)&&(l1.x-p.x)\*(l2.x-p.x)<=0&&(l1.y-p.y)\*(l2.y-p.y)<=0;

}

//

int dot\_online\_in(int x,int y,int x1,int y1,int x2,int y2){

return !xmult(x,y,x1,y1,x2,y2)&&(x1-x)\*(x2-x)<=0&&(y1-y)\*(y2-y)<=0;

}

//判点是否在线段上,不包括端点

int dot\_online\_ex(point p,line l){

return dot\_online\_in(p,l)&&(p.x!=l.a.x||p.y!=l.a.y)&&(p.x!=l.b.x||p.y!=l.b.y);

}

int dot\_online\_ex(point p,point l1,point l2){

return dot\_online\_in(p,l1,l2)&&(p.x!=l1.x||p.y!=l1.y)&&(p.x!=l2.x||p.y!=l2.y);

}

int dot\_online\_ex(int x,int y,int x1,int y1,int x2,int y2){

return dot\_online\_in(x,y,x1,y1,x2,y2)&&(x!=x1||y!=y1)&&(x!=x2||y!=y2);

}

//判两点在直线同侧,点在直线上返回0

int same\_side(point p1,point p2,line l){

return sign(xmult(l.a,p1,l.b))\*xmult(l.a,p2,l.b)>0;

}

int same\_side(point p1,point p2,point l1,point l2){

return sign(xmult(l1,p1,l2))\*xmult(l1,p2,l2)>0;

}

//判两点在直线异侧,点在直线上返回0

int opposite\_side(point p1,point p2,line l){

return sign(xmult(l.a,p1,l.b))\*xmult(l.a,p2,l.b)<0;

}

int opposite\_side(point p1,point p2,point l1,point l2){

return sign(xmult(l1,p1,l2))\*xmult(l1,p2,l2)<0;

}

//判两直线平行

int parallel(line u,line v){

return (u.a.x-u.b.x)\*(v.a.y-v.b.y)==(v.a.x-v.b.x)\*(u.a.y-u.b.y);

}

int parallel(point u1,point u2,point v1,point v2){

return (u1.x-u2.x)\*(v1.y-v2.y)==(v1.x-v2.x)\*(u1.y-u2.y);

}

//判两直线垂直

int perpendicular(line u,line v){

return (u.a.x-u.b.x)\*(v.a.x-v.b.x)==-(u.a.y-u.b.y)\*(v.a.y-v.b.y);

}

int perpendicular(point u1,point u2,point v1,point v2){

return (u1.x-u2.x)\*(v1.x-v2.x)==-(u1.y-u2.y)\*(v1.y-v2.y);

}

//判两线段相交,包括端点和部分重合

int intersect\_in(line u,line v){

if (!dots\_inline(u.a,u.b,v.a)||!dots\_inline(u.a,u.b,v.b))

return !same\_side(u.a,u.b,v)&&!same\_side(v.a,v.b,u);

return dot\_online\_in(u.a,v)||dot\_online\_in(u.b,v)||dot\_online\_in(v.a,u)||dot\_online\_in(v.b,u);

}

int intersect\_in(point u1,point u2,point v1,point v2){

if (!dots\_inline(u1,u2,v1)||!dots\_inline(u1,u2,v2))

return !same\_side(u1,u2,v1,v2)&&!same\_side(v1,v2,u1,u2);

return dot\_online\_in(u1,v1,v2)||dot\_online\_in(u2,v1,v2)||dot\_online\_in(v1,u1,u2)||dot\_online\_in(v2,u1,u2);

}

//判两线段相交,不包括端点和部分重合

int intersect\_ex(line u,line v){

return opposite\_side(u.a,u.b,v)&&opposite\_side(v.a,v.b,u);

}

int intersect\_ex(point u1,point u2,point v1,point v2){

return opposite\_side(u1,u2,v1,v2)&&opposite\_side(v1,v2,u1,u2);

}

**2、组合**

**2.1 组合公式**

1. C(m,n)=C(m,m-n)

2. C(m,n)=C(m-1,n)+C(m-1,n-1)

derangement D(n) = n!(1 - 1/1! + 1/2! - 1/3! + ... + (-1)^n/n!)

= (n-1)(D(n-2) - D(n-1))

Q(n) = D(n) + D(n-1)

求和公式,k = 1..n

1. sum( k ) = n(n+1)/2

2. sum( 2k-1 ) = n^2

3. sum( k^2 ) = n(n+1)(2n+1)/6

4. sum( (2k-1)^2 ) = n(4n^2-1)/3

5. sum( k^3 ) = (n(n+1)/2)^2

6. sum( (2k-1)^3 ) = n^2(2n^2-1)

7. sum( k^4 ) = n(n+1)(2n+1)(3n^2+3n-1)/30

8. sum( k^5 ) = n^2(n+1)^2(2n^2+2n-1)/12

9. sum( k(k+1) ) = n(n+1)(n+2)/3

10. sum( k(k+1)(k+2) ) = n(n+1)(n+2)(n+3)/4

12. sum( k(k+1)(k+2)(k+3) ) = n(n+1)(n+2)(n+3)(n+4)/5

**2.2 排列组合生成**

//gen\_perm产生字典序排列P(n,m)

//gen\_comb产生字典序组合C(n,m)

//gen\_perm\_swap产生相邻位对换全排列P(n,n)

//产生元素用1..n表示

//dummy为产生后调用的函数,传入a[]和n,a[0]..a[n-1]为一次产生的结果

#define MAXN 100

int count;

#include <iostream.h>

void dummy(int\* a,int n){

int i;

cout<<count++<<": ";

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

cout<<a[i]<<' ';

cout<<a[n-1]<<endl;

}

void \_gen\_perm(int\* a,int n,int m,int l,int\* temp,int\* tag){

int i;

if (l==m)

dummy(temp,m);

else

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

if (!tag[i]){

temp[l]=a[i],tag[i]=1;

\_gen\_perm(a,n,m,l+1,temp,tag);

tag[i]=0;

}

}

void gen\_perm(int n,int m){

int a[MAXN],temp[MAXN],tag[MAXN]={0},i;

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

a[i]=i+1;

\_gen\_perm(a,n,m,0,temp,tag);

}

void \_gen\_comb(int\* a,int s,int e,int m,int& count,int\* temp){

int i;

if (!m)

dummy(temp,count);

else

for (i=s;i<=e-m+1;i++){

temp[count++]=a[i];

\_gen\_comb(a,i+1,e,m-1,count,temp);

count--;

}

}

void gen\_comb(int n,int m){

int a[MAXN],temp[MAXN],count=0,i;

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

a[i]=i+1;

\_gen\_comb(a,0,n-1,m,count,temp);

}

void \_gen\_perm\_swap(int\* a,int n,int l,int\* pos,int\* dir){

int i,p1,p2,t;

if (l==n)

dummy(a,n);

else{

\_gen\_perm\_swap(a,n,l+1,pos,dir);

for (i=0;i<l;i++){

p2=(p1=pos[l])+dir[l];

t=a[p1],a[p1]=a[p2],a[p2]=t;

pos[a[p1]-1]=p1,pos[a[p2]-1]=p2;

\_gen\_perm\_swap(a,n,l+1,pos,dir);

}

dir[l]=-dir[l];

}

}

void gen\_perm\_swap(int n){

int a[MAXN],pos[MAXN],dir[MAXN],i;

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

a[i]=i+1,pos[i]=i,dir[i]=-1;

\_gen\_perm\_swap(a,n,0,pos,dir);

}

**2.3 生成gray码**

//生成reflected gray code

//每次调用gray取得下一个码

//000...000是第一个码,100...000是最后一个码

void gray(int n,int \*code){

int t=0,i;

for (i=0;i<n;t+=code[i++]);

if (t&1)

for (n--;!code[n];n--);

code[n-1]=1-code[n-1];

}

**2.4 置换(polya)**

//求置换的循环节,polya原理

//perm[0..n-1]为0..n-1的一个置换(排列)

//返回置换最小周期,num返回循环节个数

#define MAXN 1000

int gcd(int a,int b){

return b?gcd(b,a%b):a;

}

int polya(int\* perm,int n,int& num){

int i,j,p,v[MAXN]={0},ret=1;

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

if (!v[i]){

for (num++,j=0,p=i;!v[p=perm[p]];j++)

v[p]=1;

ret\*=j/gcd(ret,j);

}

return ret;

}

**2.5 字典序全排列**

//字典序全排列与序号的转换

int perm2num(int n,int \*p){

int i,j,ret=0,k=1;

for (i=n-2;i>=0;k\*=n-(i--))

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

if (p[j]<p[i])

ret+=k;

return ret;

}

void num2perm(int n,int \*p,int t){

int i,j;

for (i=n-1;i>=0;i--)

p[i]=t%(n-i),t/=n-i;

for (i=n-1;i;i--)

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

if (p[j]<=p[i])

p[i]++;

}

**2.6 字典序组合**

//字典序组合与序号的转换

//comb为组合数C(n,m),必要时换成大数,注意处理C(n,m)=0|n<m

int comb(int n,int m){

int ret=1,i;

m=m<(n-m)?m:(n-m);

for (i=n-m+1;i<=n;ret\*=(i++));

for (i=1;i<=m;ret/=(i++));

return m<0?0:ret;

}

int comb2num(int n,int m,int \*c){

int ret=comb(n,m),i;

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

ret-=comb(n-c[i],m-i);

return ret;

}

void num2comb(int n,int m,int\* c,int t){

int i,j=1,k;

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

for (;t>(k=comb(n-j,m-i-1));t-=k,j++);

}

**3、 数据结构**

**3.1 并查集**

//带路径压缩的并查集,用于动态维护查询等价类

//图论算法中动态判点集连通常用

//维护和查询复杂度略大于O(1)

//集合元素取值1..MAXN-1(注意0不能用!),默认不等价

#include <string.h>

#define MAXN 100000

#define \_ufind\_run(x) for(;p[t=x];x=p[x],p[t]=(p[x]?p[x]:x))

#define \_run\_both \_ufind\_run(i);\_ufind\_run(j)

struct ufind{

int p[MAXN],t;

void init(){memset(p,0,sizeof(p));}

void set\_friend(int i,int j){\_run\_both;p[i]=(i==j?0:j);}

int is\_friend(int i,int j){\_run\_both;return i==j&&i;}

};

//带路径压缩的并查集扩展形式

//用于动态维护查询friend-enemy型等价类

//维护和查询复杂度略大于O(1)

//集合元素取值1..MAXN-1(注意0不能用!),默认无关

#include <string.h>

#define MAXN 100000

#define sig(x) ((x)>0?1:-1)

#define abs(x) ((x)>0?(x):-(x))

#define \_ufind\_run(x) for(;p[t=abs(x)];x=sig(x)\*p[abs(x)],p[t]=sig(p[t])\*(p[abs(x)]?p[abs(x)]:abs(p[t])))

#define \_run\_both \_ufind\_run(i);\_ufind\_run(j)

#define \_set\_side(x) p[abs(i)]=sig(i)\*(abs(i)==abs(j)?0:(x)\*j)

#define \_judge\_side(x) (i==(x)\*j&&i)

struct ufind{

int p[MAXN],t;

void init(){memset(p,0,sizeof(p));}

int set\_friend(int i,int j){\_run\_both;\_set\_side(1);return !\_judge\_side(-1);}

int set\_enemy(int i,int j){\_run\_both;\_set\_side(-1);return !\_judge\_side(1);}

int is\_friend(int i,int j){\_run\_both;return \_judge\_side(1);}

int is\_enemy(int i,int j){\_run\_both;return \_judge\_side(-1);}

};

**3.2 堆**

//二分堆(binary)

//可插入,获取并删除最小(最大)元素,复杂度均O(logn)

//可更改元素类型,修改比较符号或换成比较函数

#define MAXN 10000

#define \_cp(a,b) ((a)<(b))

typedef int elem\_t;

struct heap{

elem\_t h[MAXN];

int n,p,c;

void init(){n=0;}

void ins(elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(elem\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

//映射二分堆(mapped)

//可插入,获取并删除任意元素,复杂度均O(logn)

//插入时提供一个索引值,删除时按该索引删除,获取并删除最小元素时一起获得该索引

//索引值范围0..MAXN-1,不能重复,不负责维护索引的唯一性,不在此返回请另外映射

//主要用于图论算法,该索引值可以是节点的下标

//可更改元素类型,修改比较符号或换成比较函数

#define MAXN 10000

#define \_cp(a,b) ((a)<(b))

typedef int elem\_t;

struct heap{

elem\_t h[MAXN];

int ind[MAXN],map[MAXN],n,p,c;

void init(){n=0;}

void ins(int i,elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

h[map[ind[p]=i]=p]=e;

}

int del(int i,elem\_t& e){

i=map[i];if (i<1||i>n) return 0;

for (e=h[p=i];p>1;h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

for (c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

int delmin(int& i,elem\_t& e){

if (n<1) return 0;i=ind[1];

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

};

**3.3 线段树**

线段树应用：

求面积:

1) 坐标离散化

2) 垂直边按x坐标排序

3) 从左往右用线段树处理垂直边

累计每个离散x区间长度和线段树长度的乘积

求周长:

1) 坐标离散化

2) 垂直边按x坐标排序, 第二关键字为入边优于出边

3) 从左往右用线段树处理垂直边

在每个离散点上先加入所有入边, 累计线段树长度变化值

再删除所有出边, 累计线段树长度变化值

4) 水平边按y坐标排序, 第二关键字为入边优于出边

5) 从上往下用线段树处理水平边

在每个离散点上先加入所有入边, 累计线段树长度变化值

再删除所有出边, 累计线段树长度变化值

//线段树

//可以处理加入边和删除边不同的情况

//inc\_seg和dec\_seg用于加入边

//seg\_len求长度

//t传根节点(一律为1)

//l0,r0传树的节点范围(一律为1..t)

//l,r传线段(端点)

#define MAXN 10000

struct segtree{

int n,cnt[MAXN],len[MAXN];

segtree(int t):n(t){

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

cnt[i]=len[i]=0;

};

void update(int t,int l,int r);

void inc\_seg(int t,int l0,int r0,int l,int r);

void dec\_seg(int t,int l0,int r0,int l,int r);

int seg\_len(int t,int l0,int r0,int l,int r);

};

int length(int l,int r){

return r-l;

}

void segtree::update(int t,int l,int r){

if (cnt[t]||r-l==1)

len[t]=length(l,r);

else

len[t]=len[t+t]+len[t+t+1];

}

void segtree::inc\_seg(int t,int l0,int r0,int l,int r){

if (l0==l&&r0==r)

cnt[t]++;

else{

int m0=(l0+r0)>>1;

if (l<m0)

inc\_seg(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

inc\_seg(t+t+1,m0,r0,m0>l?m0:l,r);

if (cnt[t+t]&&cnt[t+t+1]){

cnt[t+t]--;

update(t+t,l0,m0);

cnt[t+t+1]--;

update(t+t+1,m0,r0);

cnt[t]++;

}

}

update(t,l0,r0);

}

void segtree::dec\_seg(int t,int l0,int r0,int l,int r){

if (l0==l&&r0==r)

cnt[t]--;

else if (cnt[t]){

cnt[t]--;

if (l>l0)

inc\_seg(t,l0,r0,l0,l);

if (r<r0)

inc\_seg(t,l0,r0,r,r0);

}

else{

int m0=(l0+r0)>>1;

if (l<m0)

dec\_seg(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

dec\_seg(t+t+1,m0,r0,m0>l?m0:l,r);

}

update(t,l0,r0);

}

int segtree::seg\_len(int t,int l0,int r0,int l,int r){

if (cnt[t]||(l0==l&&r0==r))

return len[t];

else{

int m0=(l0+r0)>>1,ret=0;

if (l<m0)

ret+=seg\_len(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

ret+=seg\_len(t+t+1,m0,r0,m0>l?m0:l,r);

return ret;

}

}

//线段树扩展

//可以计算长度和线段数

//可以处理加入边和删除边不同的情况

//inc\_seg和dec\_seg用于加入边

//seg\_len求长度,seg\_cut求线段数

//t传根节点(一律为1)

//l0,r0传树的节点范围(一律为1..t)

//l,r传线段(端点)

#define MAXN 10000

struct segtree{

int n,cnt[MAXN],len[MAXN],cut[MAXN],bl[MAXN],br[MAXN];

segtree(int t):n(t){

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

cnt[i]=len[i]=cut[i]=bl[i]=br[i]=0;

};

void update(int t,int l,int r);

void inc\_seg(int t,int l0,int r0,int l,int r);

void dec\_seg(int t,int l0,int r0,int l,int r);

int seg\_len(int t,int l0,int r0,int l,int r);

int seg\_cut(int t,int l0,int r0,int l,int r);

};

int length(int l,int r){

return r-l;

}

void segtree::update(int t,int l,int r){

if (cnt[t]||r-l==1)

len[t]=length(l,r),cut[t]=bl[t]=br[t]=1;

else{

len[t]=len[t+t]+len[t+t+1];

cut[t]=cut[t+t]+cut[t+t+1];

if (br[t+t]&&bl[t+t+1])

cut[t]--;

bl[t]=bl[t+t],br[t]=br[t+t+1];

}

}

void segtree::inc\_seg(int t,int l0,int r0,int l,int r){

if (l0==l&&r0==r)

cnt[t]++;

else{

int m0=(l0+r0)>>1;

if (l<m0)

inc\_seg(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

inc\_seg(t+t+1,m0,r0,m0>l?m0:l,r);

if (cnt[t+t]&&cnt[t+t+1]){

cnt[t+t]--;

update(t+t,l0,m0);

cnt[t+t+1]--;

update(t+t+1,m0,r0);

cnt[t]++;

}

}

update(t,l0,r0);

}

void segtree::dec\_seg(int t,int l0,int r0,int l,int r){

if (l0==l&&r0==r)

cnt[t]--;

else if (cnt[t]){

cnt[t]--;

if (l>l0)

inc\_seg(t,l0,r0,l0,l);

if (r<r0)

inc\_seg(t,l0,r0,r,r0);

}

else{

int m0=(l0+r0)>>1;

if (l<m0)

dec\_seg(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

dec\_seg(t+t+1,m0,r0,m0>l?m0:l,r);

}

update(t,l0,r0);

}

int segtree::seg\_len(int t,int l0,int r0,int l,int r){

if (cnt[t]||(l0==l&&r0==r))

return len[t];

else{

int m0=(l0+r0)>>1,ret=0;

if (l<m0)

ret+=seg\_len(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

ret+=seg\_len(t+t+1,m0,r0,m0>l?m0:l,r);

return ret;

}

}

int segtree::seg\_cut(int t,int l0,int r0,int l,int r){

if (cnt[t])

return 1;

if (l0==l&&r0==r)

return cut[t];

else{

int m0=(l0+r0)>>1,ret=0;

if (l<m0)

ret+=seg\_cut(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

ret+=seg\_cut(t+t+1,m0,r0,m0>l?m0:l,r);

if (l<m0&&r>m0&&br[t+t]&&bl[t+t+1])

ret--;

return ret;

}

}

**3.4 子段和**

//求sum{[0..n-1]}

//维护和查询复杂度均为O(logn)

//用于动态求子段和,数组内容保存在sum.a[]中

//可以改成其他数据类型

#include <string.h>

#define lowbit(x) ((x)&((x)^((x)-1)))

#define MAXN 10000

typedef int elem\_t;

struct sum{

elem\_t a[MAXN],c[MAXN],ret;

int n;

void init(int i){memset(a,0,sizeof(a));memset(c,0,sizeof(c));n=i;}

void update(int i,elem\_t v){for (v-=a[i],a[i++]+=v;i<=n;c[i-1]+=v,i+=lowbit(i));}

elem\_t query(int i){for (ret=0;i;ret+=c[i-1],i^=lowbit(i));return ret;}

};

**3.5 子阵和**

//求sum{a[0..m-1][0..n-1]}

//维护和查询复杂度均为O(logm\*logn)

//用于动态求子阵和,数组内容保存在sum.a[][]中

//可以改成其他数据类型

#include <string.h>

#define lowbit(x) ((x)&((x)^((x)-1)))

#define MAXN 100

typedef int elem\_t;

struct sum{

elem\_t a[MAXN][MAXN],c[MAXN][MAXN],ret;

int m,n,t;

void init(int i,int j)

{memset(a,0,sizeof(a));memset(c,0,sizeof(c));m=i,n=j;}

void update(int i,int j,elem\_t v){

for (v-=a[i][j],a[i++][j++]+=v,t=j;i<=m;i+=lowbit(i))

for (j=t;j<=n;c[i-1][j-1]+=v,j+=lowbit(j));

}

elem\_t query(int i,int j){

for (ret=0,t=j;i;i^=lowbit(i))

for (j=t;j;ret+=c[i-1][j-1],j^=lowbit(j));

return ret;

}

};

**4、数论**

**4.1 阶乘最后非0位**

//求阶乘最后非零位,复杂度O(nlogn)

//返回该位,n以字符串方式传入

#include <string.h>

#define MAXN 10000

int lastdigit(char\* buf){

const int mod[20]={1,1,2,6,4,2,2,4,2,8,4,4,8,4,6,8,8,6,8,2};

int len=strlen(buf),a[MAXN],i,c,ret=1;

if (len==1)

return mod[buf[0]-'0'];

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

a[i]=buf[len-1-i]-'0';

for (;len;len-=!a[len-1]){

ret=ret\*mod[a[1]%2\*10+a[0]]%5;

for (c=0,i=len-1;i>=0;i--)

c=c\*10+a[i],a[i]=c/5,c%=5;

}

return ret+ret%2\*5;

}

**4.2 模线性方程组**

#ifdef WIN32

typedef \_\_int64 i64;

#else

typedef long long i64;

#endif

//扩展Euclid求解gcd(a,b)=ax+by

int ext\_gcd(int a,int b,int& x,int& y){

int t,ret;

if (!b){

x=1,y=0;

return a;

}

ret=ext\_gcd(b,a%b,x,y);

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

return ret;

}

//计算m^a, O(loga), 本身没什么用, 注意这个按位处理的方法 :-P

int exponent(int m,int a){

int ret=1;

for (;a;a>>=1,m\*=m)

if (a&1)

ret\*=m;

return ret;

}

//计算幂取模a^b mod n, O(logb)

int modular\_exponent(int a,int b,int n){ //a^b mod n

int ret=1;

for (;b;b>>=1,a=(int)((i64)a)\*a%n)

if (b&1)

ret=(int)((i64)ret)\*a%n;

return ret;

}

//求解模线性方程ax=b (mod n)

//返回解的个数,解保存在sol[]中

//要求n>0,解的范围0..n-1

int modular\_linear(int a,int b,int n,int\* sol){

int d,e,x,y,i;

d=ext\_gcd(a,n,x,y);

if (b%d)

return 0;

e=(x\*(b/d)%n+n)%n;

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

sol[i]=(e+i\*(n/d))%n;

return d;

}

//求解模线性方程组(中国余数定理)

// x = b[0] (mod w[0])

// x = b[1] (mod w[1])

// ...

// x = b[k-1] (mod w[k-1])

//要求w[i]>0,w[i]与w[j]互质,解的范围1..n,n=w[0]\*w[1]\*...\*w[k-1]

int modular\_linear\_system(int b[],int w[],int k){

int d,x,y,a=0,m,n=1,i;

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

n\*=w[i];

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

m=n/w[i];

d=ext\_gcd(w[i],m,x,y);

a=(a+y\*m\*b[i])%n;

}

return (a+n)%n;

}

**4.3 素数**

//用素数表判定素数,先调用initprime

int plist[10000],pcount=0;

int prime(int n){

int i;

if ((n!=2&&!(n%2))||(n!=3&&!(n%3))||(n!=5&&!(n%5))||(n!=7&&!(n%7)))

return 0;

for (i=0;plist[i]\*plist[i]<=n;i++)

if (!(n%plist[i]))

return 0;

return n>1;

}

void initprime(){

int i;

for (plist[pcount++]=2,i=3;i<50000;i++)

if (prime(i))

plist[pcount++]=i;

}

//miller rabin

//判断自然数n是否为素数

//time越高失败概率越低,一般取10到50

#include <stdlib.h>

#ifdef WIN32

typedef \_\_int64 i64;

#else

typedef long long i64;

#endif

int modular\_exponent(int a,int b,int n){ //a^b mod n

int ret;

for (;b;b>>=1,a=(int)((i64)a)\*a%n)

if (b&1)

ret=(int)((i64)ret)\*a%n;

return ret;

}

// Carmicheal number: 561,41041,825265,321197185

int miller\_rabin(int n,int time=10){

if (n==1||(n!=2&&!(n%2))||(n!=3&&!(n%3))||(n!=5&&!(n%5))||(n!=7&&!(n%7)))

return 0;

while (time--)

if (modular\_exponent(((rand()&0x7fff<<16)+rand()&0x7fff+rand()&0x7fff)%(n-1)+1,n-1,n)!=1)

return 0;

return 1;

}

**4.4 欧拉函数**

int gcd(int a,int b){

return b?gcd(b,a%b):a;

}

inline int lcm(int a,int b){

return a/gcd(a,b)\*b;

}

//求1..n-1中与n互质的数的个数

int eular(int n){

int ret=1,i;

for (i=2;i\*i<=n;i++)

if (n%i==0){

n/=i,ret\*=i-1;

while (n%i==0)

n/=i,ret\*=i;

}

if (n>1)

ret\*=n-1;

return ret;

}

**5、数值计算**

**5.1 定积分计算(Romberg)**

/\* Romberg求定积分

输入：积分区间[a,b]，被积函数f(x,y,z)

输出：积分结果

f(x,y,z)示例：

double f0( double x, double l, double t )

{

return sqrt(1.0+l\*l\*t\*t\*cos(t\*x)\*cos(t\*x));

}

\*/

double Integral(double a, double b, double (\*f)(double x, double y, double z), double eps,

double l, double t)

double Romberg (double a, double b, double (\*f)(double x, double y, double z), double eps,

double l, double t)

{

#define MAX\_N 1000

int i, j, temp2, min;

double h, R[2][MAX\_N], temp4;

for (i=0; i<MAX\_N; i++) {

R[0][i] = 0.0;

R[1][i] = 0.0;

}

h = b-a;

min = (int)(log(h\*10.0)/log(2.0)); //h should be at most 0.1

R[0][0] = ((\*f)(a, l, t)+(\*f)(b, l, t))\*h\*0.50;

i = 1;

temp2 = 1;

while (i<MAX\_N){

i++;

R[1][0] = 0.0;

for (j=1; j<=temp2; j++)

R[1][0] += (\*f)(a+h\*((double)j-0.50), l, t);

R[1][0] = (R[0][0] + h\*R[1][0])\*0.50;

temp4 = 4.0;

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

R[1][j] = R[1][j-1] + (R[1][j-1]-R[0][j-1])/(temp4-1.0);

temp4 \*= 4.0;

}

if ((fabs(R[1][i-1]-R[0][i-2])<eps)&&(i>min))

return R[1][i-1];

h \*= 0.50;

temp2 \*= 2;

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

R[0][j] = R[1][j];

}

return R[1][MAX\_N-1];

}

double Integral(double a, double b, double (\*f)(double x, double y, double z), double eps,

double l, double t)

{

#define pi 3.1415926535897932

int n;

double R, p, res;

n = (int)(floor)(b \* t \* 0.50 / pi);

p = 2.0 \* pi / t;

res = b - (double)n \* p;

if (n)

R = Romberg (a, p, f0, eps/(double)n, l, t);

R = R \* (double)n + Romberg( 0.0, res, f0, eps, l, t );

return R/100.0;

}

**5.2 多项式求根(牛顿法)**

/\* 牛顿法解多项式的根

输入：多项式系数c[]，多项式度数n，求在[a,b]间的根

输出：根

要求保证[a,b]间有根

\*/

double fabs( double x )

{

return (x<0)? -x : x;

}

double f(int m, double c[], double x)

{

int i;

double p = c[m];

for (i=m; i>0; i--)

p = p\*x + c[i-1];

return p;

}

int newton(double x0, double \*r,

double c[], double cp[], int n,

double a, double b, double eps)

{

int MAX\_ITERATION = 1000;

int i = 1;

double x1, x2, fp, eps2 = eps/10.0;

x1 = x0;

while (i < MAX\_ITERATION) {

x2 = f(n, c, x1);

fp = f(n-1, cp, x1);

if ((fabs(fp)<0.000000001) && (fabs(x2)>1.0))

return 0;

x2 = x1 - x2/fp;

if (fabs(x1-x2)<eps2) {

if (x2<a || x2>b)

return 0;

\*r = x2;

return 1;

}

x1 = x2;

i++;

}

return 0;

}

double Polynomial\_Root(double c[], int n, double a, double b, double eps)

{

double \*cp;

int i;

double root;

cp = (double \*)calloc(n, sizeof(double));

for (i=n-1; i>=0; i--) {

cp[i] = (i+1)\*c[i+1];

}

if (a>b) {

root = a; a = b; b = root;

}

if ((!newton(a, &root, c, cp, n, a, b, eps)) &&

(!newton(b, &root, c, cp, n, a, b, eps)))

newton((a+b)\*0.5, &root, c, cp, n, a, b, eps);

free(cp);

if (fabs(root)<eps)

return fabs(root);

else

return root;

}

**5.3 周期性方程(追赶法)**

/\* 追赶法解周期性方程

周期性方程定义：| a1 b1 c1 ... | = x1

| a2 b2 c2 ... | = x2

| ... | \* X = ...

| cn-1 ... an-1 bn-1 | = xn-1

| bn cn an | = xn

输入：a[],b[],c[],x[]

输出：求解结果X在x[]中

\*/

void run()

{

c[0] /= b[0]; a[0] /= b[0]; x[0] /= b[0];

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

double temp = b[i] - a[i] \* c[i - 1];

c[i] /= temp;

x[i] = (x[i] - a[i] \* x[i - 1]) / temp;

a[i] = -a[i] \* a[i - 1] / temp;

}

a[N - 2] = -a[N - 2] - c[N - 2];

for (int i = N - 3; i >= 0; i --) {

a[i] = -a[i] - c[i] \* a[i + 1];

x[i] -= c[i] \* x[i + 1];

}

x[N - 1] -= (c[N - 1] \* x[0] + a[N - 1] \* x[N - 2]);

x[N - 1] /= (c[N - 1] \* a[0] + a[N - 1] \* a[N - 2] + b[N - 1]);

for (int i = N - 2; i >= 0; i --)

x[i] += a[i] \* x[N - 1];

}

**6、图论—NP搜索**

**6.1 最大团**

//最大团

//返回最大团大小和一个方案,传入图的大小n和邻接阵mat

//mat[i][j]为布尔量

#define MAXN 60

void clique(int n, int\* u, int mat[][MAXN], int size, int& max, int& bb, int\* res, int\* rr, int\* c) {

int i, j, vn, v[MAXN];

if (n) {

if (size + c[u[0]] <= max) return;

for (i = 0; i < n + size - max && i < n; ++ i) {

for (j = i + 1, vn = 0; j < n; ++ j)

if (mat[u[i]][u[j]])

v[vn ++] = u[j];

rr[size] = u[i];

clique(vn, v, mat, size + 1, max, bb, res, rr, c);

if (bb) return;

}

} else if (size > max) {

max = size;

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

res[i] = rr[i];

bb = 1;

}

}

int maxclique(int n, int mat[][MAXN], int \*ret) {

int max = 0, bb, c[MAXN], i, j;

int vn, v[MAXN], rr[MAXN];

for (c[i = n - 1] = 0; i >= 0; -- i) {

for (vn = 0, j = i + 1; j < n; ++ j)

if (mat[i][j])

v[vn ++] = j;

bb = 0;

rr[0] = i;

clique(vn, v, mat, 1, max, bb, ret, rr, c);

c[i] = max;

}

return max;

}

**6.2 最大团(n<64)(faster)**

/\*\*

\* WishingBone's ACM/ICPC Routine Library

\*

\* maximum clique solver

\*/

#include <vector>

using std::vector;

// clique solver calculates both size and consitution of maximum clique

// uses bit operation to accelerate searching

// graph size limit is 63, the graph should be undirected

// can optimize to calculate on each component, and sort on vertex degrees

// can be used to solve maximum independent set

class clique {

public:

static const long long ONE = 1;

static const long long MASK = (1 << 21) - 1;

char\* bits;

int n, size, cmax[63];

long long mask[63], cons;

// initiate lookup table

clique() {

bits = new char[1 << 21];

bits[0] = 0;

for (int i = 1; i < 1 << 21; ++i) bits[i] = bits[i >> 1] + (i & 1);

}

~clique() {

delete bits;

}

// search routine

bool search(int step, int size, long long more, long long con);

// solve maximum clique and return size

int sizeClique(vector<vector<int> >& mat);

// solve maximum clique and return constitution

vector<int> consClique(vector<vector<int> >& mat);

};

// search routine

// step is node id, size is current solution, more is available mask, cons is

constitution mask

bool clique::search(int step, int size, long long more, long long cons) {

if (step >= n) {

// a new solution reached

this->size = size;

this->cons = cons;

return true;

}

long long now = ONE << step;

if ((now & more) > 0) {

long long next = more & mask[step];

if (size + bits[next & MASK] + bits[(next >> 21) & MASK] + bits[next >>

42] >= this->size

&& size + cmax[step] > this->size) {

// the current node is in the clique

if (search(step + 1, size + 1, next, cons | now)) return true;

}

}

long long next = more & ~now;

if (size + bits[next & MASK] + bits[(next >> 21) & MASK] + bits[next >> 42]

> this->size) {

// the current node is not in the clique

if (search(step + 1, size, next, cons)) return true;

}

return false;

}

// solve maximum clique and return size

int clique::sizeClique(vector<vector<int> >& mat) {

n = mat.size();

// generate mask vectors

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

mask[i] = 0;

for (int j = 0; j < n; ++j) if (mat[i][j] > 0) mask[i] |= ONE << j;

}

size = 0;

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

search(i + 1, 1, mask[i], ONE << i);

cmax[i] = size;

}

return size;

}

// solve maximum clique and return constitution

// calls sizeClique and restore cons

vector<int> clique::consClique(vector<vector<int> >& mat) {

sizeClique(mat);

vector<int> ret;

for (int i = 0; i < n; ++i) if ((cons & (ONE << i)) > 0) ret.push\_back(i);

return ret;

}

**7、图论—连通性**

**7.1 无向图关键点(dfs邻接阵)**

//无向图的关键点,dfs邻接阵形式,O(n^2)

//返回关键点个数,key[]返回点集

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 110

void search(int n,int mat[][MAXN],int\* dfn,int\* low,int now,int& ret,int\* key,int& cnt,int root,int& rd,int\* bb){

int i;

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

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

if (mat[now][i]){

if (!dfn[i]){

search(n,mat,dfn,low,i,ret,key,cnt,root,rd,bb);

if (low[i]<low[now])

low[now]=low[i];

if (low[i]>=dfn[now]){

if (now!=root&&!bb[now])

key[ret++]=now,bb[now]=1;

else if(now==root)

rd++;

}

}

else if (dfn[i]<low[now])

low[now]=dfn[i];

}

}

int key\_vertex(int n,int mat[][MAXN],int\* key){

int ret=0,i,cnt,rd,dfn[MAXN],low[MAXN],bb[MAXN];

for (i=0;i<n;dfn[i++]=bb[i]=0);

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

if (!dfn[i]){

rd=0;

search(n,mat,dfn,low,i,ret,key,cnt,i,rd,bb);

if (rd>1&&!bb[i])

key[ret++]=i,bb[i]=1;

}

return ret;

}

**7.2 无向图关键边(dfs邻接阵)**

//无向图的关键边,dfs邻接阵形式,O(n^2)

//返回关键边条数,key[][2]返回边集

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

void search(int n,int mat[][MAXN],int\* dfn,int\* low,int now,int& cnt,int key[][2]){

int i;

for (low[now]=dfn[now],i=0;i<n;i++)

if (mat[now][i]){

if (!dfn[i]){

dfn[i]=dfn[now]+1;

search(n,mat,dfn,low,i,cnt,key);

if (low[i]>dfn[now])

key[cnt][0]=i,key[cnt++][1]=now;

if (low[i]<low[now])

low[now]=low[i];

}

else if (dfn[i]<dfn[now]-1&&dfn[i]<low[now])

low[now]=lev[i];

}

}

int key\_edge(int n,int mat[][MAXN],int key[][2]){

int ret=0,i,dfn[MAXN],low[MAXN];

for (i=0;i<n;dfn[i++]=0);

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

if (!dfn[i])

dfn[i]=1,bridge(n,mat,dfn,low,i,ret,key);

return ret;

}

**7.3 无向图的块(bfs邻接阵)**

//无向图的块,dfs邻接阵形式,O(n^2)

//每产生一个块调用dummy

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

#include <iostream.h>

void dummy(int n,int\* a){

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

cout<<a[i]<<' ';

cout<<endl;

}

void search(int n,int mat[][MAXN],int\* dfn,int\* low,int now,int& cnt,int\* st,int& sp){

int i,m,a[MAXN];

dfn[st[sp++]=now]=low[now]=++cnt;

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

if (mat[now][i]){

if (!dfn[i]){

search(n,mat,dfn,low,i,cnt,st,sp);

if (low[i]<low[now])

low[now]=low[i];

if (low[i]>=dfn[now]){

for (st[sp]=-1,a[0]=now,m=1;st[sp]!=i;a[m++]=st[--sp]);

dummy(m,a);

}

}

else if (dfn[i]<low[now])

low[now]=dfn[i];

}

}

void block(int n,int mat[][MAXN]){

int i,cnt,dfn[MAXN],low[MAXN],st[MAXN],sp=0;

for (i=0;i<n;dfn[i++]=0);

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

if (!dfn[i])

search(n,mat,dfn,low,i,cnt,st,sp);

}

**7.4 无向图连通分支(dfs/bfs邻接阵)**

//无向图连通分支,dfs邻接阵形式,O(n^2)

//返回分支数,id返回1..分支数的值

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

void floodfill(int n,int mat[][MAXN],int\* id,int now,int tag){

int i;

for (id[now]=tag,i=0;i<n;i++)

if (!id[i]&&mat[now][i])

floodfill(n,mat,id,i,tag);

}

int find\_components(int n,int mat[][MAXN],int\* id){

int ret,i;

for (i=0;i<n;id[i++]=0);

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

if (!id[i])

floodfill(n,mat,id,i,++ret);

return ret;

}

//无向图连通分支,bfs邻接阵形式,O(n^2)

//返回分支数,id返回1..分支数的值

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

int find\_components(int n,int mat[][MAXN],int\* id){

int ret,k,i,j,m;

for (k=0;k<n;id[k++]=0);

for (ret=k=0;k<n;k++)

if (!id[k])

for (id[k]=-1,ret++,m=1;m;)

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

if (id[i]==-1)

for (m++,id[i]=ret,j=0;j<n;j++)

if (!id[j]&&mat[i][j])

id[j]=-1;

return ret;

}

**7.5 有向图强连通分支(dfs/bfs邻接阵)**

//有向图强连通分支,dfs邻接阵形式,O(n^2)

//返回分支数,id返回1..分支数的值

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

void search(int n,int mat[][MAXN],int\* dfn,int\* low,int now,int& cnt,int& tag,int\* id,int\* st,int& sp){

int i,j;

dfn[st[sp++]=now]=low[now]=++cnt;

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

if (mat[now][i]){

if (!dfn[i]){

ssearch(n,mat,dfn,low,i,cnt,tag,id,st,sp);

if (low[i]<low[now])

low[now]=low[i];

}

else if (dfn[i]<dfn[now]){

for (j=0;j<sp&&st[j]!=i;j++);

if (j<cnt&&dfn[i]<low[now])

low[now]=dfn[i];

}

}

if (low[now]==dfn[now])

for (tag++;st[sp]!=now;id[st[--sp]]=tag);

}

int find\_components(int n,int mat[][MAXN],int\* id){

int ret=0,i,cnt,sp,st[MAXN],dfn[MAXN],low[MAXN];

for (i=0;i<n;dfn[i++]=0);

for (sp=cnt=i=0;i<n;i++)

if (!dfn[i])

search(n,mat,dfn,low,i,cnt,ret,id,st,sp);

return ret;

}

//有向图强连通分支,bfs邻接阵形式,O(n^2)

//返回分支数,id返回1..分支数的值

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

int find\_components(int n,int mat[][MAXN],int\* id){

int ret=0,a[MAXN],b[MAXN],c[MAXN],d[MAXN],i,j,k,t;

for (k=0;k<n;id[k++]=0);

for (k=0;k<n;k++)

if (!id[k]){

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

a[i]=b[i]=c[i]=d[i]=0;

a[k]=b[k]=1;

for (t=1;t;)

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

if (a[i]&&!c[i])

for (c[i]=t=1,j=0;j<n;j++)

if (mat[i][j]&&!a[j])

a[j]=1;

if (b[i]&&!d[i])

for (d[i]=t=1,j=0;j<n;j++)

if (mat[j][i]&&!b[j])

b[j]=1;

}

for (ret++,i=0;i<n;i++)

if (a[i]&b[i])

id[i]=ret;

}

return ret;

}

**7.6 有向图最小点基(邻接阵)**

//有向图最小点基,邻接阵形式,O(n^2)

//返回电集大小和点集

//传入图的大小n和邻接阵mat,不相邻点边权0

//需要调用强连通分支

#define MAXN 100

int base\_vertex(int n,int mat[][MAXN],int\* sets){

int ret=0,id[MAXN],v[MAXN],i,j;

j=find\_components(n,mat,id);

for (i=0;i<j;v[i++]=1);

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

for (j=0;j<n;j++)

if (id[i]!=id[j]&&mat[i][j])

v[id[j]-1]=0;

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

if (v[id[i]-1])

v[id[sets[ret++]=i]-1]=0;

return ret;

}

**8、图论—匹配**

**8.1 二分图最大匹配(hungary邻接表)**

//二分图最大匹配,hungary算法,邻接表形式,复杂度O(m\*e)

//返回最大匹配数,传入二分图大小m,n和邻接表list(只需一边)

//match1,match2返回一个最大匹配,未匹配顶点match值为-1

#include <string.h>

#define MAXN 310

#define \_clr(x) memset(x,0xff,sizeof(int)\*MAXN)

struct edge\_t{

int from,to;

edge\_t\* next;

};

int hungary(int m,int n,edge\_t\* list[],int\* match1,int\* match2){

int s[MAXN],t[MAXN],p,q,ret=0,i,j,k;edge\_t\* e;

for (\_clr(match1),\_clr(match2),i=0;i<m;ret+=(match1[i++]>=0))

for (\_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)

for (e=list[k=s[p]];e&&match1[i]<0;e=e->next)

if (t[j=e->to]<0){

s[++q]=match2[j],t[j]=k;

if (s[q]<0)

for (p=j;p>=0;j=p)

match2[j]=k=t[j],p=match1[k],match1[k]=j;

}

return ret;

}

**8.2 二分图最大匹配(hungary邻接阵)**

//二分图最大匹配,hungary算法,邻接阵形式,复杂度O(m\*m\*n)

//返回最大匹配数,传入二分图大小m,n和邻接阵mat,非零元素表示有边

//match1,match2返回一个最大匹配,未匹配顶点match值为-1

#include <string.h>

#define MAXN 310

#define \_clr(x) memset(x,0xff,sizeof(int)\*MAXN)

int hungary(int m,int n,int mat[][MAXN],int\* match1,int\* match2){

int s[MAXN],t[MAXN],p,q,ret=0,i,j,k;

for (\_clr(match1),\_clr(match2),i=0;i<m;ret+=(match1[i++]>=0))

for (\_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)

for (k=s[p],j=0;j<n&&match1[i]<0;j++)

if (mat[k][j]&&t[j]<0){

s[++q]=match2[j],t[j]=k;

if (s[q]<0)

for (p=j;p>=0;j=p)

match2[j]=k=t[j],p=match1[k],match1[k]=j;

}

return ret;

}

**8.3 二分图最大匹配(hungary正向表)**

//二分图最大匹配,hungary算法,正向表形式,复杂度O(m\*e)

//返回最大匹配数,传入二分图大小m,n和正向表list,buf(只需一边)

//match1,match2返回一个最大匹配,未匹配顶点match值为-1

#include <string.h>

#define MAXN 310

#define \_clr(x) memset(x,0xff,sizeof(int)\*MAXN)

int hungary(int m,int n,int\* list,int\* buf,int\* match1,int\* match2){

int s[MAXN],t[MAXN],p,q,ret=0,i,j,k,l;

for (\_clr(match1),\_clr(match2),i=0;i<m;ret+=(match1[i++]>=0))

for (\_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)

for (l=list[k=s[p]];l<list[k+1]&&match1[i]<0;l++)

if (t[j=buf[l]]<0){

s[++q]=match2[j],t[j]=k;

if (s[q]<0)

for (p=j;p>=0;j=p)

match2[j]=k=t[j],p=match1[k],match1[k]=j;

}

return ret;

}

**8.4二分图最佳匹配(kuhn\_munkras邻接阵)**

//二分图最佳匹配,kuhn munkras算法,邻接阵形式,复杂度O(m\*m\*n)

//返回最佳匹配值,传入二分图大小m,n和邻接阵mat,表示权值

//match1,match2返回一个最佳匹配,未匹配顶点match值为-1

//一定注意m<=n,否则循环无法终止

//最小权匹配可将权值取相反数

#include <string.h>

#define MAXN 310

#define inf 1000000000

#define \_clr(x) memset(x,0xff,sizeof(int)\*n)

int kuhn\_munkras(int m,int n,int mat[][MAXN],int\* match1,int\* match2){

int s[MAXN],t[MAXN],l1[MAXN],l2[MAXN],p,q,ret=0,i,j,k;

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

for (l1[i]=-inf,j=0;j<n;j++)

l1[i]=mat[i][j]>l1[i]?mat[i][j]:l1[i];

for (i=0;i<n;l2[i++]=0);

for (\_clr(match1),\_clr(match2),i=0;i<m;i++){

for (\_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)

for (k=s[p],j=0;j<n&&match1[i]<0;j++)

if (l1[k]+l2[j]==mat[k][j]&&t[j]<0){

s[++q]=match2[j],t[j]=k;

if (s[q]<0)

for (p=j;p>=0;j=p)

match2[j]=k=t[j],p=match1[k],match1[k]=j;

}

if (match1[i]<0){

for (i--,p=inf,k=0;k<=q;k++)

for (j=0;j<n;j++)

if (t[j]<0&&l1[s[k]]+l2[j]-mat[s[k]][j]<p)

p=l1[s[k]]+l2[j]-mat[s[k]][j];

for (j=0;j<n;l2[j]+=t[j]<0?0:p,j++);

for (k=0;k<=q;l1[s[k++]]-=p);

}

}

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

ret+=mat[i][match1[i]];

return ret;

}

**8.5 一般图匹配(邻接表)**

//一般图最大匹配,邻接表形式,复杂度O(n\*e)

//返回匹配顶点对数,match返回匹配,未匹配顶点match值为-1

//传入图的顶点数n和邻接表list

#define MAXN 100

struct edge\_t{

int from,to;

edge\_t\* next;

};

int aug(int n,edge\_t\* list[],int\* match,int\* v,int now){

int t,ret=0;edge\_t\* e;

v[now]=1;

for (e=list[now];e;e=e->next)

if (!v[t=e->to]){

if (match[t]<0)

match[now]=t,match[t]=now,ret=1;

else{

v[t]=1;

if (aug(n,list,match,v,match[t]))

match[now]=t,match[t]=now,ret=1;

v[t]=0;

}

if (ret)

break;

}

v[now]=0;

return ret;

}

int graph\_match(int n,edge\_t\* list[],int\* match){

int v[MAXN],i,j;

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

v[i]=0,match[i]=-1;

for (i=0,j=n;i<n&&j>=2;)

if (match[i]<0&&aug(n,list,match,v,i))

i=0,j-=2;

else

i++;

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

j+=(match[i]>=0);

return j/2;

}

**8.6 一般图匹配(邻接阵)**

//一般图最大匹配,邻接阵形式,复杂度O(n^3)

//返回匹配顶点对数,match返回匹配,未匹配顶点match值为-1

//传入图的顶点数n和邻接阵mat

#define MAXN 100

int aug(int n,int mat[][MAXN],int\* match,int\* v,int now){

int i,ret=0;

v[now]=1;

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

if (!v[i]&&mat[now][i]){

if (match[i]<0)

match[now]=i,match[i]=now,ret=1;

else{

v[i]=1;

if (aug(n,mat,match,v,match[i]))

match[now]=i,match[i]=now,ret=1;

v[i]=0;

}

if (ret)

break;

}

v[now]=0;

return ret;

}

int graph\_match(int n,int mat[][MAXN],int\* match){

int v[MAXN],i,j;

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

v[i]=0,match[i]=-1;

for (i=0,j=n;i<n&&j>=2;)

if (match[i]<0&&aug(n,mat,match,v,i))

i=0,j-=2;

else

i++;

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

j+=(match[i]>=0);

return j/2;

}

**8.7 一般图匹配(正向表)**

//一般图最大匹配,正向表形式,复杂度O(n\*e)

//返回匹配顶点对数,match返回匹配,未匹配顶点match值为-1

//传入图的顶点数n和正向表list,buf

#define MAXN 100

int aug(int n,int\* list,int\* buf,int\* match,int\* v,int now){

int i,t,ret=0;

v[now]=1;

for (i=list[now];i<list[now+1];i++)

if (!v[t=buf[i]]){

if (match[t]<0)

match[now]=t,match[t]=now,ret=1;

else{

v[t]=1;

if (aug(n,list,buf,match,v,match[t]))

match[now]=t,match[t]=now,ret=1;

v[t]=0;

}

if (ret)

break;

}

v[now]=0;

return ret;

}

int graph\_match(int n,int\* list,int\* buf,int\* match){

int v[MAXN],i,j;

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

v[i]=0,match[i]=-1;

for (i=0,j=n;i<n&&j>=2;)

if (match[i]<0&&aug(n,list,buf,match,v,i))

i=0,j-=2;

else

i++;

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

j+=(match[i]>=0);

return j/2;

}

**9、图论—网络流**

**9.1 最大流(邻接阵)**

//求网络最大流,邻接阵形式

//返回最大流量,flow返回每条边的流量

//传入网络节点数n,容量mat,源点source,汇点sink

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink,int flow[][MAXN]){

int pre[MAXN],que[MAXN],d[MAXN],p,q,t,i,j;

if (source==sink) return inf;

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

for (j=0;j<n;flow[i][j++]=0);

for (;;){

for (i=0;i<n;pre[i++]=0);

pre[t=source]=source+1,d[t]=inf;

for (p=q=0;p<=q&&!pre[sink];t=que[p++])

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

if (!pre[i]&&j=mat[t][i]-flow[t][i])

pre[que[q++]=i]=t+1,d[i]=d[t]<j?d[t]:j;

else if (!pre[i]&&j=flow[i][t])

pre[que[q++]=i]=-t-1,d[i]=d[t]<j?d[t]:j;

if (!pre[sink]) break;

for (i=sink;i!=source;)

if (pre[i]>0)

flow[pre[i]-1][i]+=d[sink],i=pre[i]-1;

else

flow[i][-pre[i]-1]-=d[sink],i=-pre[i]-1;

}

for (j=i=0;i<n;j+=flow[source][i++]);

return j;

}

**9.2 上下界最大流(邻接阵)**

//求上下界网络最大流,邻接阵形式

//返回最大流量,-1表示无可行流,flow返回每条边的流量

//传入网络节点数n,容量mat,流量下界bf,源点source,汇点sink

//MAXN应比最大结点数多2,无可行流返回-1时mat未复原!

#define MAXN 100

#define inf 1000000000

int limit\_max\_flow(int n,int mat[][MAXN],int bf[][MAXN],int source,int sink,int flow[][MAXN]){

int i,j,sk,ks;

if (source==sink) return inf;

for (mat[n][n+1]=mat[n+1][n]=mat[n][n]=mat[n+1][n+1]=i=0;i<n;i++)

for (mat[n][i]=mat[i][n]=mat[n+1][i]=mat[i][n+1]=j=0;j<n;j++)

mat[i][j]-=bf[i][j],mat[n][i]+=bf[j][i],mat[i][n+1]+=bf[i][j];

sk=mat[source][sink],ks=mat[sink][source],mat[source][sink]=mat[sink][source]=inf;

for (i=0;i<n+2;i++)

for (j=0;j<n+2;flow[i][j++]=0);

\_max\_flow(n+2,mat,n,n+1,flow);

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

if (flow[n][i]<mat[n][i]) return -1;

flow[source][sink]=flow[sink][source]=0,mat[source][sink]=sk,mat[sink][source]=ks;

\_max\_flow(n,mat,source,sink,flow);

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

for (j=0;j<n;j++)

mat[i][j]+=bf[i][j],flow[i][j]+=bf[i][j];

for (j=i=0;i<n;j+=flow[source][i++]);

return j;

}

**9.3 上下界最小流(邻接阵)**

//求上下界网络最小流,邻接阵形式

//返回最大流量,-1表示无可行流,flow返回每条边的流量

//传入网络节点数n,容量mat,流量下界bf,源点source,汇点sink

//MAXN应比最大结点数多2,无可行流返回-1时mat未复原!

#define MAXN 100

#define inf 1000000000

int limit\_min\_flow(int n,int mat[][MAXN],int bf[][MAXN],int source,int sink,int flow[][MAXN]){

int i,j,sk,ks;

if (source==sink) return inf;

for (mat[n][n+1]=mat[n+1][n]=mat[n][n]=mat[n+1][n+1]=i=0;i<n;i++)

for (mat[n][i]=mat[i][n]=mat[n+1][i]=mat[i][n+1]=j=0;j<n;j++)

mat[i][j]-=bf[i][j],mat[n][i]+=bf[j][i],mat[i][n+1]+=bf[i][j];

sk=mat[source][sink],ks=mat[sink][source],mat[source][sink]=mat[sink][source]=inf;

for (i=0;i<n+2;i++)

for (j=0;j<n+2;flow[i][j++]=0);

\_max\_flow(n+2,mat,n,n+1,flow);

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

if (flow[n][i]<mat[n][i]) return -1;

flow[source][sink]=flow[sink][source]=0,mat[source][sink]=sk,mat[sink][source]=ks;

\_max\_flow(n,mat,sink,source,flow);

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

for (j=0;j<n;j++)

mat[i][j]+=bf[i][j],flow[i][j]+=bf[i][j];

for (j=i=0;i<n;j+=flow[source][i++]);

return j;

}

**9.4 最大流无流量(邻接阵)**

//求网络最大流,邻接阵形式

//返回最大流量

//传入网络节点数n,容量mat,源点source,汇点sink

//注意mat矩阵被修改

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink){

int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;

for (;;){

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

v[i]=c[i]=0;

for (c[source]=inf;;){

for (j=-1,i=0;i<n;i++)

if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))

j=i;

if (j<0) return ret;

if (j==sink) break;

for (v[j]=1,i=0;i<n;i++)

if (mat[j][i]>c[i]&&c[j]>c[i])

c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;

}

for (ret+=j=c[i=sink];i!=source;i=p[i])

mat[p[i]][i]-=j,mat[i][p[i]]+=j;

}

}

**9.5 最小费用最大流(邻接阵)**

//求网络最小费用最大流,邻接阵形式

//返回最大流量,flow返回每条边的流量,netcost返回总费用

//传入网络节点数n,容量mat,单位费用cost,源点source,汇点sink

#define MAXN 100

#define inf 1000000000

int min\_cost\_max\_flow(int n,int mat[][MAXN],int cost[][MAXN],int source,int sink,int flow[][MAXN],int& netcost){

int pre[MAXN],min[MAXN],d[MAXN],i,j,t,tag;

if (source==sink) return inf;

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

for (j=0;j<n;flow[i][j++]=0);

for (netcost=0;;){

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

pre[i]=0,min[i]=inf;

for (pre[source]=source+1,min[source]=0,d[source]=inf,tag=1;tag;)

for (tag=t=0;t<n;t++)

if (d[t])

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

if (j=mat[t][i]-flow[t][i]&&min[t]+cost[t][i]<min[i])

tag=1,min[i]=min[t]+cost[t][i],pre[i]=t+1,d[i]=d[t]<j?d[t]:j;

else if (j=flow[i][t]&&min[t]<inf&&min[t]-cost[i][t]<min[i])

tag=1,min[i]=min[t]-cost[i][t],pre[i]=-t-1,d[i]=d[t]<j?d[t]:j;

if (!pre[sink]) break;

for (netcost+=min[sink]\*d[i=sink];i!=source;)

if (pre[i]>0)

flow[pre[i]-1][i]+=d[sink],i=pre[i]-1;

else

flow[i][-pre[i]-1]-=d[sink],i=-pre[i]-1;

}

for (j=i=0;i<n;j+=flow[source][i++]);

return j;

}

**10、 图论—应用**

**10.1 欧拉回路(邻接阵)**

//求欧拉回路或欧拉路,邻接阵形式,复杂度O(n^2)

//返回路径长度,path返回路径(有向图时得到的是反向路径)

//传入图的大小n和邻接阵mat,不相邻点边权0

//可以有自环与重边,分为无向图和有向图

#define MAXN 100

void find\_path\_u(int n,int mat[][MAXN],int now,int& step,int\* path){

int i;

for (i=n-1;i>=0;i--)

while (mat[now][i]){

mat[now][i]--,mat[i][now]--;

find\_path\_u(n,mat,i,step,path);

}

path[step++]=now;

}

void find\_path\_d(int n,int mat[][MAXN],int now,int& step,int\* path){

int i;

for (i=n-1;i>=0;i--)

while (mat[now][i]){

mat[now][i]--;

find\_path\_d(n,mat,i,step,path);

}

path[step++]=now;

}

int euclid\_path(int n,int mat[][MAXN],int start,int\* path){

int ret=0;

find\_path\_u(n,mat,start,ret,path);

// find\_path\_d(n,mat,start,ret,path);

return ret;

}

**10.2 树的前序表转化**

//将用边表示的树转化为前序表示的树

//传入节点数n和邻接表list[],邻接表必须是双向的,会在函数中释放

//pre[]返回前序表,map[]返回前序表中的节点到原来节点的映射

#define MAXN 10000

struct node{

int to;

node\* next;

};

void prenode(int n,node\* list[],int\* pre,int\* map,int\* v,int now,int last,int& id){

node\* t;

int p=id++;

for (v[map[p]=now]=1,pre[p]=last;list[now];){

t=list[now],list[now]=t->next;

if (!v[t->to])

prenode(n,list,pre,map,v,t->to,p,id);

}

}

void makepre(int n,node\* list[],int\* pre,int\* map){

int v[MAXN],id=0,i;

for (i=0;i<n;v[i++]=0);

prenode(n,list,pre,map,v,0,-1,id);

}

**10.3 树的优化算法**

//最大顶点独立集

int max\_node\_independent(int n,int\* pre,int\* set){

int c[MAXN],i,ret=0;

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

c[i]=set[i]=0;

for (i=n-1;i>=0;i--)

if (!c[i]){

set[i]=1;

if (pre[i]!=-1)

c[pre[i]]=1;

ret++;

}

return ret;

}

//最大边独立集

int max\_edge\_independent(int n,int\* pre,int\* set){

int c[MAXN],i,ret=0;

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

c[i]=set[i]=0;

for (i=n-1;i>=0;i--)

if (!c[i]&&pre[i]!=-1&&!c[pre[i]]){

set[i]=1;

c[pre[i]]=1;

ret++;

}

return ret;

}

//最小顶点覆盖集

int min\_node\_cover(int n,int\* pre,int\* set){

int c[MAXN],i,ret=0;

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

c[i]=set[i]=0;

for (i=n-1;i>=0;i--)

if (!c[i]&&pre[i]!=-1&&!c[pre[i]]){

set[i]=1;

c[pre[i]]=1;

ret++;

}

return ret;

}

//最小顶点支配集

int min\_node\_dominant(int n,int\* pre,int\* set){

int c[MAXN],i,ret=0;

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

c[i]=set[i]=0;

for (i=n-1;i>=0;i--)

if (!c[i]&&(pre[i]==-1||!set[pre[i]])){

if (pre[i]!=-1){

set[pre[i]]=1;

c[pre[i]]=1;

if (pre[pre[i]]!=-1)

c[pre[pre[i]]]=1;

}

else

set[i]=1;

ret++;

}

return ret;

}

**10.4 拓扑排序(邻接阵)**

//拓扑排序,邻接阵形式,复杂度O(n^2)

//如果无法完成排序,返回0,否则返回1,ret返回有序点列

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

int toposort(int n,int mat[][MAXN],int\* ret){

int d[MAXN],i,j,k;

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

for (d[i]=j=0;j<n;d[i]+=mat[j++][i]);

for (k=0;k<n;ret[k++]=i){

for (i=0;d[i]&&i<n;i++);

if (i==n)

return 0;

for (d[i]=-1,j=0;j<n;j++)

d[j]-=mat[i][j];

}

return 1;

}

**10.5 最佳边割集**

//最佳边割集

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink){

int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;

for (;;){

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

v[i]=c[i]=0;

for (c[source]=inf;;){

for (j=-1,i=0;i<n;i++)

if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))

j=i;

if (j<0) return ret;

if (j==sink) break;

for (v[j]=1,i=0;i<n;i++)

if (mat[j][i]>c[i]&&c[j]>c[i])

c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;

}

for (ret+=j=c[i=sink];i!=source;i=p[i])

mat[p[i]][i]-=j,mat[i][p[i]]+=j;

}

}

int best\_edge\_cut(int n,int mat[][MAXN],int source,int sink,int set[][2],int& mincost){

int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,l,ret=0,last;

if (source==sink)

return -1;

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

for (j=0;j<n;j++)

m0[i][j]=mat[i][j];

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

for (j=0;j<n;j++)

m[i][j]=m0[i][j];

mincost=last=max\_flow(n,m,source,sink);

for (k=0;k<n&&last;k++)

for (l=0;l<n&&last;l++)

if (m0[k][l]){

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

for (j=0;j<n+n;j++)

m[i][j]=m0[i][j];

m[k][l]=0;

if (max\_flow(n,m,source,sink)==last-mat[k][l]){

set[ret][0]=k;

set[ret++][1]=l;

m0[k][l]=0;

last-=mat[k][l];

}

}

return ret;

}

**10.6 最佳点割集**

//最佳顶点割集

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink){

int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;

for (;;){

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

v[i]=c[i]=0;

for (c[source]=inf;;){

for (j=-1,i=0;i<n;i++)

if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))

j=i;

if (j<0) return ret;

if (j==sink) break;

for (v[j]=1,i=0;i<n;i++)

if (mat[j][i]>c[i]&&c[j]>c[i])

c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;

}

for (ret+=j=c[i=sink];i!=source;i=p[i])

mat[p[i]][i]-=j,mat[i][p[i]]+=j;

}

}

int best\_vertex\_cut(int n,int mat[][MAXN],int\* cost,int source,int sink,int\* set,int& mincost){

int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,ret=0,last;

if (source==sink||mat[source][sink])

return -1;

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

for (j=0;j<n+n;j++)

m0[i][j]=0;

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

for (j=0;j<n;j++)

if (mat[i][j])

m0[i][n+j]=inf;

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

m0[n+i][i]=cost[i];

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

for (j=0;j<n+n;j++)

m[i][j]=m0[i][j];

mincost=last=max\_flow(n+n,m,source,n+sink);

for (k=0;k<n&&last;k++)

if (k!=source&&k!=sink){

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

for (j=0;j<n+n;j++)

m[i][j]=m0[i][j];

m[n+k][k]=0;

if (max\_flow(n+n,m,source,n+sink)==last-cost[k]){

set[ret++]=k;

m0[n+k][k]=0;

last-=cost[k];

}

}

return ret;

}

**10.7 最小边割集**

//最小边割集

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink){

int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;

for (;;){

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

v[i]=c[i]=0;

for (c[source]=inf;;){

for (j=-1,i=0;i<n;i++)

if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))

j=i;

if (j<0) return ret;

if (j==sink) break;

for (v[j]=1,i=0;i<n;i++)

if (mat[j][i]>c[i]&&c[j]>c[i])

c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;

}

for (ret+=j=c[i=sink];i!=source;i=p[i])

mat[p[i]][i]-=j,mat[i][p[i]]+=j;

}

}

int min\_edge\_cut(int n,int mat[][MAXN],int source,int sink,int set[][2]){

int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,l,ret=0,last;

if (source==sink)

return -1;

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

for (j=0;j<n;j++)

m0[i][j]=(mat[i][j]!=0);

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

for (j=0;j<n;j++)

m[i][j]=m0[i][j];

last=max\_flow(n,m,source,sink);

for (k=0;k<n&&last;k++)

for (l=0;l<n&&last;l++)

if (m0[k][l]){

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

for (j=0;j<n+n;j++)

m[i][j]=m0[i][j];

m[k][l]=0;

if (max\_flow(n,m,source,sink)<last){

set[ret][0]=k;

set[ret++][1]=l;

m0[k][l]=0;

last--;

}

}

return ret;

}

**10.8 最小点割集**

//最小顶点割集

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink){

int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;

for (;;){

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

v[i]=c[i]=0;

for (c[source]=inf;;){

for (j=-1,i=0;i<n;i++)

if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))

j=i;

if (j<0) return ret;

if (j==sink) break;

for (v[j]=1,i=0;i<n;i++)

if (mat[j][i]>c[i]&&c[j]>c[i])

c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;

}

for (ret+=j=c[i=sink];i!=source;i=p[i])

mat[p[i]][i]-=j,mat[i][p[i]]+=j;

}

}

int min\_vertex\_cut(int n,int mat[][MAXN],int source,int sink,int\* set){

int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,ret=0,last;

if (source==sink||mat[source][sink])

return -1;

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

for (j=0;j<n+n;j++)

m0[i][j]=0;

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

for (j=0;j<n;j++)

if (mat[i][j])

m0[i][n+j]=inf;

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

m0[n+i][i]=1;

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

for (j=0;j<n+n;j++)

m[i][j]=m0[i][j];

last=max\_flow(n+n,m,source,n+sink);

for (k=0;k<n&&last;k++)

if (k!=source&&k!=sink){

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

for (j=0;j<n+n;j++)

m[i][j]=m0[i][j];

m[n+k][k]=0;

if (max\_flow(n+n,m,source,n+sink)<last){

set[ret++]=k;

m0[n+k][k]=0;

last--;

}

}

return ret;

}

**10.9 最小路径覆盖**

//最小路径覆盖,O(n^3)

//求解最小的路径覆盖图中所有点,有向图无向图均适用

//注意此问题等价二分图最大匹配,可以用邻接表或正向表减小复杂度

//返回最小路径条数,pre返回前指针(起点-1),next返回后指针(终点-1)

#include <string.h>

#define MAXN 310

#define \_clr(x) memset(x,0xff,sizeof(int)\*n)

int hungary(int n,int mat[][MAXN],int\* match1,int\* match2){

int s[MAXN],t[MAXN],p,q,ret=0,i,j,k;

for (\_clr(match1),\_clr(match2),i=0;i<n;ret+=(match1[i++]>=0))

for (\_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)

for (k=s[p],j=0;j<n&&match1[i]<0;j++)

if (mat[k][j]&&t[j]<0){

s[++q]=match2[j],t[j]=k;

if (s[q]<0)

for (p=j;p>=0;j=p)

match2[j]=k=t[j],p=match1[k],match1[k]=j;

}

return ret;

}

inline int path\_cover(int n,int mat[][MAXN],int\* pre,int\* next){

return n-hungary(n,mat,next,pre);

}

**11、 图论—支撑树**

**11.1 最小生成树(kruskal邻接表)**

//无向图最小生成树,kruskal算法,邻接表形式,复杂度O(mlogm)

//返回最小生成树的长度,传入图的大小n和邻接表list

//可更改边权的类型,edge[][2]返回树的构造,用边集表示

//如果图不连通,则对各连通分支构造最小生成树,返回总长度

#include <string.h>

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int from,to;

elem\_t len;

edge\_t\* next;

};

#define \_ufind\_run(x) for(;p[t=x];x=p[x],p[t]=(p[x]?p[x]:x))

#define \_run\_both \_ufind\_run(i);\_ufind\_run(j)

struct ufind{

int p[MAXN],t;

void init(){memset(p,0,sizeof(p));}

void set\_friend(int i,int j){\_run\_both;p[i]=(i==j?0:j);}

int is\_friend(int i,int j){\_run\_both;return i==j&&i;}

};

#define \_cp(a,b) ((a).len<(b).len)

struct heap\_t{int a,b;elem\_t len;};

struct minheap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

elem\_t kruskal(int n,edge\_t\* list[],int edge[][2]){

ufind u;minheap h;

edge\_t\* t;heap\_t e;

elem\_t ret=0;int i,m=0;

u.init(),h.init();

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

for (t=list[i];t;t=t->next)

if (i<t->to)

e.a=i,e.b=t->to,e.len=t->len,h.ins(e);

while (m<n-1&&h.del(e))

if (!u.is\_friend(e.a+1,e.b+1))

edge[m][0]=e.a,edge[m][1]=e.b,ret+=e.len,u.set\_friend(e.a+1,e.b+1);

return ret;

}

**11.2 最小生成树(kruskal正向表)**

//无向图最小生成树,kruskal算法,正向表形式,复杂度O(mlogm)

//返回最小生成树的长度,传入图的大小n和正向表list,buf

//可更改边权的类型,edge[][2]返回树的构造,用边集表示

//如果图不连通,则对各连通分支构造最小生成树,返回总长度

#include <string.h>

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int to;

elem\_t len;

};

#define \_ufind\_run(x) for(;p[t=x];x=p[x],p[t]=(p[x]?p[x]:x))

#define \_run\_both \_ufind\_run(i);\_ufind\_run(j)

struct ufind{

int p[MAXN],t;

void init(){memset(p,0,sizeof(p));}

void set\_friend(int i,int j){\_run\_both;p[i]=(i==j?0:j);}

int is\_friend(int i,int j){\_run\_both;return i==j&&i;}

};

#define \_cp(a,b) ((a).len<(b).len)

struct heap\_t{int a,b;elem\_t len;};

struct minheap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

elem\_t kruskal(int n,int\* list,edge\_t\* buf,int edge[][2]){

ufind u;minheap h;

heap\_t e;elem\_t ret=0;

int i,j,m=0;

u.init(),h.init();

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

for (j=list[i];j<list[i+1];j++)

if (i<buf[j].to)

e.a=i,e.b=buf[j].to,e.len=buf[j].len,h.ins(e);

while (m<n-1&&h.del(e))

if (!u.is\_friend(e.a+1,e.b+1))

edge[m][0]=e.a,edge[m][1]=e.b,ret+=e.len,u.set\_friend(e.a+1,e.b+1);

return ret;

}

**11.3 最小生成树(prim+binary\_heap邻接表)**

//无向图最小生成树,prim算法+二分堆,邻接表形式,复杂度O(mlogm)

//返回最小生成树的长度,传入图的大小n和邻接表list

//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre值为-1

//必须保证图的连通的!

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int from,to;

elem\_t len;

edge\_t\* next;

};

#define \_cp(a,b) ((a).d<(b).d)

struct heap\_t{elem\_t d;int v;};

struct heap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

elem\_t prim(int n,edge\_t\* list[],int\* pre){

heap h;

elem\_t min[MAXN],ret=0;

edge\_t\* t;heap\_t e;

int v[MAXN],i;

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

min[i]=inf,v[i]=0,pre[i]=-1;

h.init();e.v=0,e.d=0,h.ins(e);

while (h.del(e))

if (!v[e.v])

for (v[e.v]=1,ret+=e.d,t=list[e.v];t;t=t->next)

if (!v[t->to]&&t->len<min[t->to])

pre[t->to]=t->from,min[e.v=t->to]=e.d=t->len,h.ins(e);

return ret;

}

**11.4 最小生成树(prim+binary\_heap正向表)**

//无向图最小生成树,prim算法+二分堆,正向表形式,复杂度O(mlogm)

//返回最小生成树的长度,传入图的大小n和正向表list,buf

//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre值为-1

//必须保证图的连通的!

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int to;

elem\_t len;

};

#define \_cp(a,b) ((a).d<(b).d)

struct heap\_t{elem\_t d;int v;};

struct heap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

elem\_t prim(int n,int\* list,edge\_t\* buf,int\* pre){

heap h;heap\_t e;

elem\_t min[MAXN],ret=0;

int v[MAXN],i,j;

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

min[i]=inf,v[i]=0,pre[i]=-1;

h.init();e.v=0,e.d=0,h.ins(e);

while (h.del(e))

if (!v[i=e.v])

for (v[i]=1,ret+=e.d,j=list[i];j<list[i+1];j++)

if (!v[buf[j].to]&&buf[j].len<min[buf[j].to])

pre[buf[j].to]=i,min[e.v=buf[j].to]=e.d=buf[j].len,h.ins(e);

return ret;

}

**11.5 最小生成树(prim+mapped\_heap邻接表)**

//无向图最小生成树,prim算法+映射二分堆,邻接表形式,复杂度O(mlogn)

//返回最小生成树的长度,传入图的大小n和邻接表list

//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre值为-1

//必须保证图的连通的!

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int from,to;

elem\_t len;

edge\_t\* next;

};

#define \_cp(a,b) ((a)<(b))

struct heap{

elem\_t h[MAXN+1];

int ind[MAXN+1],map[MAXN+1],n,p,c;

void init(){n=0;}

void ins(int i,elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

h[map[ind[p]=i]=p]=e;

}

int del(int i,elem\_t& e){

i=map[i];if (i<1||i>n) return 0;

for (e=h[p=i];p>1;h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

for (c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

int delmin(int& i,elem\_t& e){

if (n<1) return 0;i=ind[1];

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

};

elem\_t prim(int n,edge\_t\* list[],int\* pre){

heap h;

elem\_t min[MAXN],ret=0,e;

edge\_t\* t;

int v[MAXN],i;

for (h.init(),i=0;i<n;i++)

min[i]=(i?inf:0),v[i]=0,pre[i]=-1,h.ins(i,min[i]);

while (h.delmin(i,e))

for (v[i]=1,ret+=e,t=list[i];t;t=t->next)

if (!v[t->to]&&t->len<min[t->to])

pre[t->to]=t->from,h.del(t->to,e),h.ins(t->to,min[t->to]=t->len);

return ret;

}

**11.6 最小生成树(prim+mapped\_heap正向表)**

//无向图最小生成树,prim算法+映射二分堆,正向表形式,复杂度O(mlogn)

//返回最小生成树的长度,传入图的大小n和正向表list,buf

//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre值为-1

//必须保证图的连通的!

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int to;

elem\_t len;

};

#define \_cp(a,b) ((a)<(b))

struct heap{

elem\_t h[MAXN+1];

int ind[MAXN+1],map[MAXN+1],n,p,c;

void init(){n=0;}

void ins(int i,elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

h[map[ind[p]=i]=p]=e;

}

int del(int i,elem\_t& e){

i=map[i];if (i<1||i>n) return 0;

for (e=h[p=i];p>1;h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

for (c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

int delmin(int& i,elem\_t& e){

if (n<1) return 0;i=ind[1];

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

};

elem\_t prim(int n,int\* list,edge\_t\* buf,int\* pre){

heap h;

elem\_t min[MAXN],ret=0,e;

int v[MAXN],i,j;

for (h.init(),i=0;i<n;i++)

min[i]=(i?inf:0),v[i]=0,pre[i]=-1,h.ins(i,min[i]);

while (h.delmin(i,e))

for (v[i]=1,ret+=e,j=list[i];j<list[i+1];j++)

if (!v[buf[j].to]&&buf[j].len<min[buf[j].to])

pre[buf[j].to]=i,h.del(buf[j].to,e),h.ins(buf[j].to,min[buf[j].to]=buf[j].len);

return ret;

}

**11.7 最小生成树(prim邻接阵)**

//无向图最小生成树,prim算法,邻接阵形式,复杂度O(n^2)

//返回最小生成树的长度,传入图的大小n和邻接阵mat,不相邻点边权inf

//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre值为-1

//必须保证图的连通的!

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

elem\_t prim(int n,elem\_t mat[][MAXN],int\* pre){

elem\_t min[MAXN],ret=0;

int v[MAXN],i,j,k;

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

min[i]=inf,v[i]=0,pre[i]=-1;

for (min[j=0]=0;j<n;j++){

for (k=-1,i=0;i<n;i++)

if (!v[i]&&(k==-1||min[i]<min[k]))

k=i;

for (v[k]=1,ret+=min[k],i=0;i<n;i++)

if (!v[i]&&mat[k][i]<min[i])

min[i]=mat[pre[i]=k][i];

}

return ret;

}

**11.8 最小树形图(邻接阵)**

//多源最小树形图,edmonds算法,邻接阵形式,复杂度O(n^3)

//返回最小生成树的长度,构造失败返回负值

//传入图的大小n和邻接阵mat,不相邻点边权inf

//可更改边权的类型,pre[]返回树的构造,用父结点表示

//传入时pre[]数组清零,用-1标出源点

#include <string.h>

#define MAXN 120

#define inf 1000000000

typedef int elem\_t;

elem\_t edmonds(int n,elem\_t mat[][MAXN\*2],int\* pre){

elem\_t ret=0;

int c[MAXN\*2][MAXN\*2],l[MAXN\*2],p[MAXN\*2],m=n,t,i,j,k;

for (i=0;i<n;l[i]=i,i++);

do{

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

for (t=m,i=0;i<m;c[i][i]=1,i++);

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

if (l[i]==i&&pre[i]!=-1){

for (j=0;j<m;j++)

if (l[j]==j&&i!=j&&mat[j][i]<inf&&(p[i]==-1||mat[j][i]<mat[p[i]][i]))

p[i]=j;

if ((pre[i]=p[i])==-1)

return -1;

if (c[i][p[i]]){

for (j=0;j<=m;mat[j][m]=mat[m][j]=inf,j++);

for (k=i;l[k]!=m;l[k]=m,k=p[k])

for (j=0;j<m;j++)

if (l[j]==j){

if (mat[j][k]-mat[p[k]][k]<mat[j][m])

mat[j][m]=mat[j][k]-mat[p[k]][k];

if (mat[k][j]<mat[m][j])

mat[m][j]=mat[k][j];

}

c[m][m]=1,l[m]=m,m++;

}

for (j=0;j<m;j++)

if (c[i][j])

for (k=p[i];k!=-1&&l[k]==k;c[k][j]=1,k=p[k]);

}

}

while (t<m);

for (;m-->n;pre[k]=pre[m])

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

if (l[i]==m){

for (j=0;j<m;j++)

if (pre[j]==m&&mat[i][j]==mat[m][j])

pre[j]=i;

if (mat[pre[m]][m]==mat[pre[m]][i]-mat[pre[i]][i])

k=i;

}

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

if (pre[i]!=-1)

ret+=mat[pre[i]][i];

return ret;

}

**12、 图论—最短路径**

**12.1 最短路径(单源bellman\_ford邻接阵)**

//单源最短路径,bellman\_ford算法,邻接阵形式,复杂度O(n^3)

//求出源s到所有点的最短路经,传入图的大小n和邻接阵mat

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,路权可为负,若图包含负环则求解失败,返回0

//优化:先删去负边使用dijkstra求出上界,加速迭代过程

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

int bellman\_ford(int n,elem\_t mat[][MAXN],int s,elem\_t\* min,int\* pre){

int v[MAXN],i,j,k,tag;

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

min[i]=inf,v[i]=0,pre[i]=-1;

for (min[s]=0,j=0;j<n;j++){

for (k=-1,i=0;i<n;i++)

if (!v[i]&&(k==-1||min[i]<min[k]))

k=i;

for (v[k]=1,i=0;i<n;i++)

if (!v[i]&&mat[k][i]>=0&&min[k]+mat[k][i]<min[i])

min[i]=min[k]+mat[pre[i]=k][i];

}

for (tag=1,j=0;tag&&j<=n;j++)

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

for (k=0;k<n;k++)

if (min[k]+mat[k][i]<min[i])

min[i]=min[k]+mat[pre[i]=k][i],tag=1;

return j<=n;

}

**12.2 最短路径(单源dijkstra+bfs邻接表)**

//单源最短路径,用于路权相等的情况,dijkstra优化为bfs,邻接表形式,复杂度O(m)

//求出源s到所有点的最短路经,传入图的大小n和邻接表list,边权值len

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负且相等!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

struct edge\_t{

int from,to;

edge\_t\* next;

};

void dijkstra(int n,edge\_t\* list[],elem\_t len,int s,elem\_t\* min,int\* pre){

edge\_t\* t;

int i,que[MAXN],f=0,r=0,p=1,l=1;

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

min[i]=inf;

min[que[0]=s]=0,pre[s]=-1;

for (;r<=f;l++,r=f+1,f=p-1)

for (i=r;i<=f;i++)

for (t=list[que[i]];t;t=t->next)

if (min[t->to]==inf)

min[que[p++]=t->to]=len\*l,pre[t->to]=que[i];

}

**12.3 最短路径(单源dijkstra+bfs正向表)**

//单源最短路径,用于路权相等的情况,dijkstra优化为bfs,正向表形式,复杂度O(m)

//求出源s到所有点的最短路经,传入图的大小n和正向表list,buf,边权值len

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负且相等!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

void dijkstra(int n,int\* list,int\* buf,elem\_t len,int s,elem\_t\* min,int\* pre){

int i,que[MAXN],f=0,r=0,p=1,l=1,t;

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

min[i]=inf;

min[que[0]=s]=0,pre[s]=-1;

for (;r<=f;l++,r=f+1,f=p-1)

for (i=r;i<=f;i++)

for (t=list[que[i]];t<list[que[i]+1];t++)

if (min[buf[t]]==inf)

min[que[p++]=buf[t]]=len\*l,pre[buf[t]]=que[i];

}

**12.4 最短路径(单源dijkstra+binary\_heap邻接表)**

//单源最短路径,dijkstra算法+二分堆,邻接表形式,复杂度O(mlogm)

//求出源s到所有点的最短路经,传入图的大小n和邻接表list

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

struct edge\_t{

int from,to;

elem\_t len;

edge\_t\* next;

};

#define \_cp(a,b) ((a).d<(b).d)

struct heap\_t{elem\_t d;int v;};

struct heap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

void dijkstra(int n,edge\_t\* list[],int s,elem\_t\* min,int\* pre){

heap h;

edge\_t\* t;heap\_t e;

int v[MAXN],i;

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

min[i]=inf,v[i]=0,pre[i]=-1;

h.init();min[e.v=s]=e.d=0,h.ins(e);

while (h.del(e))

if (!v[e.v])

for (v[e.v]=1,t=list[e.v];t;t=t->next)

if (!v[t->to]&&min[t->from]+t->len<min[t->to])

pre[t->to]=t->from,min[e.v=t->to]=e.d=min[t->from]+t->len,h.ins(e);

}

**12.5 最短路径(单源dijkstra+binary\_heap正向表)**

//单源最短路径,dijkstra算法+二分堆,正向表形式,复杂度O(mlogm)

//求出源s到所有点的最短路经,传入图的大小n和正向表list,buf

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

struct edge\_t{

int to;

elem\_t len;

};

#define \_cp(a,b) ((a).d<(b).d)

struct heap\_t{elem\_t d;int v;};

struct heap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

void dijkstra(int n,int\* list,edge\_t\* buf,int s,elem\_t\* min,int\* pre){

heap h;heap\_t e;

int v[MAXN],i,t,f;

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

min[i]=inf,v[i]=0,pre[i]=-1;

h.init();min[e.v=s]=e.d=0,h.ins(e);

while (h.del(e))

if (!v[e.v])

for (v[f=e.v]=1,t=list[f];t<list[f+1];t++)

if (!v[buf[t].to]&&min[f]+buf[t].len<min[buf[t].to])

pre[buf[t].to]=f,min[e.v=buf[t].to]=e.d=min[f]+buf[t].len,h.ins(e);

}

**12.6 最短路径(单源dijkstra+mapped\_heap邻接表)**

//单源最短路径,dijkstra算法+映射二分堆,邻接表形式,复杂度O(mlogn)

//求出源s到所有点的最短路经,传入图的大小n和邻接表list

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

struct edge\_t{

int from,to;

elem\_t len;

edge\_t\* next;

};

#define \_cp(a,b) ((a)<(b))

struct heap{

elem\_t h[MAXN+1];

int ind[MAXN+1],map[MAXN+1],n,p,c;

void init(){n=0;}

void ins(int i,elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

h[map[ind[p]=i]=p]=e;

}

int del(int i,elem\_t& e){

i=map[i];if (i<1||i>n) return 0;

for (e=h[p=i];p>1;h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

for (c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

int delmin(int& i,elem\_t& e){

if (n<1) return 0;i=ind[1];

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

};

void dijkstra(int n,edge\_t\* list[],int s,elem\_t\* min,int\* pre){

heap h;

edge\_t\* t;elem\_t e;

int v[MAXN],i;

for (h.init(),i=0;i<n;i++)

min[i]=((i==s)?0:inf),v[i]=0,pre[i]=-1,h.ins(i,min[i]);

while (h.delmin(i,e))

for (v[i]=1,t=list[i];t;t=t->next)

if (!v[t->to]&&min[i]+t->len<min[t->to])

pre[t->to]=i,h.del(t->to,e),min[t->to]=e=min[i]+t->len,h.ins(t->to,e);

}

**12.7 最短路径(单源dijkstra+mapped\_heap正向表)**

//单源最短路径,dijkstra算法+映射二分堆,正向表形式,复杂度O(mlogn)

//求出源s到所有点的最短路经,传入图的大小n和正向表list,buf

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

struct edge\_t{

int to;

elem\_t len;

};

#define \_cp(a,b) ((a)<(b))

struct heap{

elem\_t h[MAXN+1];

int ind[MAXN+1],map[MAXN+1],n,p,c;

void init(){n=0;}

void ins(int i,elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

h[map[ind[p]=i]=p]=e;

}

int del(int i,elem\_t& e){

i=map[i];if (i<1||i>n) return 0;

for (e=h[p=i];p>1;h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

for (c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

int delmin(int& i,elem\_t& e){

if (n<1) return 0;i=ind[1];

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

};

void dijkstra(int n,int\* list,edge\_t\* buf,int s,elem\_t\* min,int\* pre){

heap h;elem\_t e;

int v[MAXN],i,t;

for (h.init(),i=0;i<n;i++)

min[i]=((i==s)?0:inf),v[i]=0,pre[i]=-1,h.ins(i,min[i]);

while (h.delmin(i,e))

for (v[i]=1,t=list[i];t<list[i+1];t++)

if (!v[buf[t].to]&&min[i]+buf[t].len<min[buf[t].to])

pre[buf[t].to]=i,h.del(buf[t].to,e),min[buf[t].to]=e=min[i]+buf[t].len,h.ins(buf[t].to,e);

}

**12.8 最短路径(单源dijkstra邻接阵)**

//单源最短路径,dijkstra算法,邻接阵形式,复杂度O(n^2)

//求出源s到所有点的最短路经,传入图的顶点数n,(有向)邻接矩阵mat

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

void dijkstra(int n,elem\_t mat[][MAXN],int s,elem\_t\* min,int\* pre){

int v[MAXN],i,j,k;

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

min[i]=inf,v[i]=0,pre[i]=-1;

for (min[s]=0,j=0;j<n;j++){

for (k=-1,i=0;i<n;i++)

if (!v[i]&&(k==-1||min[i]<min[k]))

k=i;

for (v[k]=1,i=0;i<n;i++)

if (!v[i]&&min[k]+mat[k][i]<min[i])

min[i]=min[k]+mat[pre[i]=k][i];

}

}

**12.9 最短路径(多源floyd\_warshall邻接阵)**

//多源最短路径,floyd\_warshall算法,复杂度O(n^3)

//求出所有点对之间的最短路经,传入图的大小和邻接阵

//返回各点间最短距离min[]和路径pre[],pre[i][j]记录i到j最短路径上j的父结点

//可更改路权类型,路权必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

void floyd\_warshall(int n,elem\_t mat[][MAXN],elem\_t min[][MAXN],int pre[][MAXN]){

int i,j,k;

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

for (j=0;j<n;j++)

min[i][j]=mat[i][j],pre[i][j]=(i==j)?-1:i;

for (k=0;k<n;k++)

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

for (j=0;j<n;j++)

if (min[i][k]+min[k][j]<min[i][j])

min[i][j]=min[i][k]+min[k][j],pre[i][j]=pre[k][j];

}

**13、 应用**

**13.1 Joseph问题**

// Joseph's Problem

// input: n,m -- the number of persons, the inteval between persons

// output: -- return the reference of last person

int josephus0(int n, int m)

{

if (n == 2) return (m%2) ? 2 : 1;

int v = (m+josephus0(n-1,m)) % n;

if (v == 0) v = n;

return v;

}

int josephus(int n, int m)

{

if (m == 1) return n;

if (n == 1) return 1;

if (m >=n) return josephus0(n,m);

int l = (n/m)\*m;

int j = josephus(n - (n/m), m);

if (j <= n-l) return l+j;

j -= n-l;

int t = (j/(m-1))\*m;

if ((j % (m-1)) == 0) return t-1;

return t + (j % (m-1));

}

**13.2 N皇后构造解**

//N皇后构造解,n>=4

void even1(int n,int \*p){

int i;

for (i=1;i<=n/2;i++)

p[i-1]=2\*i;

for (i=n/2+1;i<=n;i++)

p[i-1]=2\*i-n-1;

}

void even2(int n,int \*p){

int i;

for (i=1;i<=n/2;i++)

p[i-1]=(2\*i+n/2-3)%n+1;

for (i=n/2+1;i<=n;i++)

p[i-1]=n-(2\*(n-i+1)+n/2-3)%n;

}

void generate(int,int\*);

void odd(int n,int \*p){

generate(n-1,p),p[n-1]=n;

}

void generate(int n,int \*p){

if (n&1)

odd(n,p);

else if (n%6!=2)

even1(n,p);

else

even2(n,p);

}

**13.3 布尔母函数**

//布尔母函数

//判m[]个价值为w[]的货币能否构成value

//适合m[]较大w[]较小的情况

//返回布尔量

//传入货币种数n,个数m[],价值w[]和目标值value

#define MAXV 100000

int genfunc(int n,int\* m,int\* w,int value){

int i,j,k,c;

char r[MAXV];

for (r[0]=i=1;i<=value;r[i++]=0);

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

for (j=0;j<w[i];j++){

c=m[i]\*r[k=j];

while ((k+=w[i])<=value)

if (r[k])

c=m[i];

else if (c)

r[k]=1,c--;

if (r[value])

return 1;

}

}

return 0;

}

**13.4 第k元素**

//取第k个元素,k=0..n-1

//平均复杂度O(n)

//注意a[]中的顺序被改变

#define \_cp(a,b) ((a)<(b))

typedef int elem\_t;

elem\_t kth\_element(int n,elem\_t\* a,int k){

elem\_t t,key;

int l=0,r=n-1,i,j;

while (l<r){

for (key=a[((i=l-1)+(j=r+1))>>1];i<j;){

for (j--;\_cp(key,a[j]);j--);

for (i++;\_cp(a[i],key);i++);

if (i<j) t=a[i],a[i]=a[j],a[j]=t;

}

if (k>j) l=j+1;

else r=j;

}

return a[k];

}

**13.5 幻方构造**

//幻方构造(l!=2)

#define MAXN 100

void dllb(int l,int si,int sj,int sn,int d[][MAXN]){

int n,i=0,j=l/2;

for (n=1;n<=l\*l;n++){

d[i+si][j+sj]=n+sn;

if (n%l){

i=(i)?(i-1):(l-1);

j=(j==l-1)?0:(j+1);

}

else

i=(i==l-1)?0:(i+1);

}

}

void magic\_odd(int l,int d[][MAXN]){

dllb(l,0,0,0,d);

}

void magic\_4k(int l,int d[][MAXN]){

int i,j;

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

for (j=0;j<l;j++)

d[i][j]=((i%4==0||i%4==3)&&(j%4==0||j%4==3)||(i%4==1||i%4==2)&&(j%4==1||j%4==2))?(l\*l-(i\*l+j)):(i\*l+j+1);

}

void magic\_other(int l,int d[][MAXN]){

int i,j,t;

dllb(l/2,0,0,0,d);

dllb(l/2,l/2,l/2,l\*l/4,d);

dllb(l/2,0,l/2,l\*l/2,d);

dllb(l/2,l/2,0,l\*l/4\*3,d);

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

for (j=0;j<l/4;j++)

if (i!=l/4||j)

t=d[i][j],d[i][j]=d[i+l/2][j],d[i+l/2][j]=t;

t=d[l/4][l/4],d[l/4][l/4]=d[l/4+l/2][l/4],d[l/4+l/2][l/4]=t;

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

for (j=l-l/4+1;j<l;j++)

t=d[i][j],d[i][j]=d[i+l/2][j],d[i+l/2][j]=t;

}

void generate(int l,int d[][MAXN]){

if (l%2)

magic\_odd(l,d);

else if (l%4==0)

magic\_4k(l,d);

else

magic\_other(l,d);

}

**13.6 模式匹配(kmp)**

//模式匹配,kmp算法,复杂度O(m+n)

//返回匹配位置,-1表示匹配失败,传入匹配串和模式串和长度

//可更改元素类型,更换匹配函数

#define MAXN 10000

#define \_match(a,b) ((a)==(b))

typedef char elem\_t;

int pat\_match(int ls,elem\_t\* str,int lp,elem\_t\* pat){

int fail[MAXN]={-1},i=0,j;

for (j=1;j<lp;j++){

for (i=fail[j-1];i>=0&&!\_match(pat[i+1],pat[j]);i=fail[i]);

fail[j]=(\_match(pat[i+1],pat[j])?i+1:-1);

}

for (i=j=0;i<ls&&j<lp;i++)

if (\_match(str[i],pat[j]))

j++;

else if (j)

j=fail[j-1]+1,i--;

return j==lp?(i-lp):-1;

}

**13.7 逆序对数**

//序列逆序对数,复杂度O(nlogn)

//传入序列长度和内容,返回逆序对数

//可更改元素类型和比较函数

#include <string.h>

#define MAXN 1000000

#define \_cp(a,b) ((a)<=(b))

typedef int elem\_t;

elem\_t \_tmp[MAXN];

int inv(int n,elem\_t\* a){

int l=n>>1,r=n-l,i,j;

int ret=(r>1?(inv(l,a)+inv(r,a+l)):0);

for (i=j=0;i<=l;\_tmp[i+j]=a[i],i++)

for (ret+=j;j<r&&(i==l||!\_cp(a[i],a[l+j]));\_tmp[i+j]=a[l+j],j++);

memcpy(a,\_tmp,sizeof(elem\_t)\*n);

return ret;

}

**13.8 字符串最小表示**

/\* 求字符串的最小表示

输入：字符串

返回：字符串最小表示的首字母位置(0...size-1)

\*/

template <class T>

int MinString(vector <T> &str)

{

int i, j, k;

vector <T> ss(str.size() << 1);

for (i = 0; i < str.size(); i ++) ss[i] = ss[i + str.size()] = str[i];

for (i = k = 0, j = 1; k < str.size() && i < str.size() && j < str.size(); ) {

for (k = 0; k < str.size() && ss[i + k] == ss[j + k]; k ++);

if (k < str.size()) {

if (ss[i + k] > ss[j + k])

i += k + 1;

else j += k + 1;

if (i == j) j ++;

}

}

return i < j ? i : j;

}

**13.9 最长公共单调子序列**

// 最长公共递增子序列， 时间复杂度O(n^2 \* logn)，空间 O(n^2)

//\* n为a的大小, m为b的大小

//\* 结果在ans中

// "define \_cp(a,b) ((a)<(b))"求解最长严格递增序列

#define MAXN 1000

#define \_cp(a,b) ((a)<(b))

typedef int elem\_t;

elem\_t DP[MAXN][MAXN];

int num[MAXN], p[1<<20];

int LIS(int n, elem\_t \*a, int m, elem\_t \*b, elem\_t \*ans){

int i, j, l, r, k;

DP[0][0] = 0;

num[0] = (b[0] == a[0]);

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

num[i] = (b[i] == a[0]) || num[i-1];

DP[i][0] = 0;

}

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

if(b[0] == a[i] && !num[0]) {

num[0] = 1;

DP[0][0] = i<<10;

}

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

for(k=((l=0)+(r=num[j-1]-1))>>1; l<=r; k=(l+r)>>1)

if(\_cp(a[DP[j-1][k]>>10], a[i]))

l=k+1;

else

r=k-1;

if(l < num[j-1] && i == (DP[j-1][l]>>10) ){

if(l >= num[j]) DP[j][num[j]++] = DP[j-1][l];

else DP[j][l] = \_cp(a[DP[j][l]>>10],a[i]) ? DP[j][l] : DP[j-1][l];

}

if(b[j] == a[i]){

for(k=((l=0)+(r=num[j]-1))>>1; l<=r; k=(l+r)>>1)

if(\_cp(a[DP[j][k]>>10], a[i]))

l=k+1;

else

r=k-1;

DP[j][l] = (i<<10) + j;

num[j] += (l>=num[j]);

p[DP[j][l]] = l ? DP[j][l-1] : -1;

}

}

}

for (k=DP[m-1][i=num[m-1]-1];i>=0;ans[i--]=a[k>>10],k=p[k]);

return num[m-1];

}

**13.10 最长子序列**

//最长单调子序列,复杂度O(nlogn)

//注意最小序列覆盖和最长序列的对应关系,例如

//"define \_cp(a,b) ((a)>(b))"求解最长严格递减序列,则

//"define \_cp(a,b) (!((a)>(b)))"求解最小严格递减序列覆盖

//可更改元素类型和比较函数

#define MAXN 10000

#define \_cp(a,b) ((a)>(b))

typedef int elem\_t;

int subseq(int n,elem\_t\* a){

int b[MAXN],i,l,r,m,ret=0;

for (i=0;i<n;b[l]=i++,ret+=(l>ret))

for (m=((l=1)+(r=ret))>>1;l<=r;m=(l+r)>>1)

if (\_cp(a[b[m]],a[i]))

l=m+1;

else

r=m-1;

return ret;

}

int subseq(int n,elem\_t\* a,elem\_t\* ans){

int b[MAXN],p[MAXN],i,l,r,m,ret=0;

for (i=0;i<n;p[b[l]=i++]=b[l-1],ret+=(l>ret))

for (m=((l=1)+(r=ret))>>1;l<=r;m=(l+r)>>1)

if (\_cp(a[b[m]],a[i]))

l=m+1;

else

r=m-1;

for (m=b[i=ret];i;ans[--i]=a[m],m=p[m]);

return ret;

}

**13.11 最大子串匹配**

//最大子串匹配,复杂度O(mn)

//返回最大匹配值,传入两个串和串的长度,重载返回一个最大匹配

//注意做字符串匹配是串末的'\0'没有置!

//可更改元素类型,更换匹配函数和匹配价值函数

#include <string.h>

#define MAXN 100

#define max(a,b) ((a)>(b)?(a):(b))

#define \_match(a,b) ((a)==(b))

#define \_value(a,b) 1

typedef char elem\_t;

int str\_match(int m,elem\_t\* a,int n,elem\_t\* b){

int match[MAXN+1][MAXN+1],i,j;

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

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

for (j=0;j<n;j++)

match[i+1][j+1]=max(max(match[i][j+1],match[i+1][j]),

(match[i][j]+\_value(a[i],b[i]))\*\_match(a[i],b[j]));

return match[m][n];

}

int str\_match(int m,elem\_t\* a,int n,elem\_t\* b,elem\_t\* ret){

int match[MAXN+1][MAXN+1],last[MAXN+1][MAXN+1],i,j,t;

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

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

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

match[i+1][j+1]=(match[i][j+1]>match[i+1][j]?match[i][j+1]:match[i+1][j]);

last[i+1][j+1]=(match[i][j+1]>match[i+1][j]?3:1);

if ((t=(match[i][j]+\_value(a[i],b[i]))\*\_match(a[i],b[j]))>match[i+1][j+1])

match[i+1][j+1]=t,last[i+1][j+1]=2;

}

for (;match[i][j];i-=(last[t=i][j]>1),j-=(last[t][j]<3))

ret[match[i][j]-1]=(last[i][j]<3?a[i-1]:b[j-1]);

return match[m][n];

}

**13.12 最大子段和**

//求最大子段和,复杂度O(n)

//传入串长n和内容list[]

//返回最大子段和,重载返回子段位置(maxsum=list[start]+...+list[end])

//可更改元素类型

typedef int elem\_t;

elem\_t maxsum(int n,elem\_t\* list){

elem\_t ret,sum=0;

int i;

for (ret=list[i=0];i<n;i++)

sum=(sum>0?sum:0)+list[i],ret=(sum>ret?sum:ret);

return ret;

}

elem\_t maxsum(int n,elem\_t\* list,int& start,int& end){

elem\_t ret,sum=0;

int s,i;

for (ret=list[start=end=s=i=0];i<n;i++,s=(sum>0?s:i))

if ((sum=(sum>0?sum:0)+list[i])>ret)

ret=sum,start=s,end=i;

return ret;

}

**13.13 最大子阵和**

//求最大子阵和,复杂度O(n^3)

//传入阵的大小m,n和内容mat[][]

//返回最大子阵和,重载返回子阵位置(maxsum=list[s1][s2]+...+list[e1][e2])

//可更改元素类型

#define MAXN 100

typedef int elem\_t;

elem\_t maxsum(int m,int n,elem\_t mat[][MAXN]){

elem\_t matsum[MAXN][MAXN+1],ret,sum;

int i,j,k;

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

for (matsum[i][j=0]=0;j<n;j++)

matsum[i][j+1]=matsum[i][j]+mat[i][j];

for (ret=mat[0][j=0];j<n;j++)

for (k=j;k<n;k++)

for (sum=0,i=0;i<m;i++)

sum=(sum>0?sum:0)+matsum[i][k+1]-matsum[i][j],ret=(sum>ret?sum:ret);

return ret;

}

elem\_t maxsum(int m,int n,elem\_t mat[][MAXN],int& s1,int& s2,int& e1,int& e2){

elem\_t matsum[MAXN][MAXN+1],ret,sum;

int i,j,k,s;

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

for (matsum[i][j=0]=0;j<n;j++)

matsum[i][j+1]=matsum[i][j]+mat[i][j];

for (ret=mat[s1=e1=0][s2=e2=j=0];j<n;j++)

for (k=j;k<n;k++)

for (sum=0,s=i=0;i<m;i++,s=(sum>0?s:i))

if ((sum=(sum>0?sum:0)+matsum[i][k+1]-matsum[i][j])>ret)

ret=sum,s1=s,s2=i,e1=j,e2=k;

return ret;

}

**14、 其它**

**14.1 大数(只能处理正数)**

#include <iostream.h>

#include <string.h>

#define DIGIT 4

#define DEPTH 10000

#define MAX 100

typedef int bignum\_t[MAX+1];

int read(bignum\_t a,istream& is=cin){

char buf[MAX\*DIGIT+1],ch;

int i,j;

memset((void\*)a,0,sizeof(bignum\_t));

if (!(is>>buf)) return 0;

for (a[0]=strlen(buf),i=a[0]/2-1;i>=0;i--)

ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;

for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf);j<a[0]\*DIGIT;buf[j++]='0');

for (i=1;i<=a[0];i++)

for (a[i]=0,j=0;j<DIGIT;j++)

a[i]=a[i]\*10+buf[i\*DIGIT-1-j]-'0';

for (;!a[a[0]]&&a[0]>1;a[0]--);

return 1;

}

void write(const bignum\_t a,ostream& os=cout){

int i,j;

for (os<<a[i=a[0]],i--;i;i--)

for (j=DEPTH/10;j;j/=10)

os<<a[i]/j%10;

}

int comp(const bignum\_t a,const bignum\_t b){

int i;

if (a[0]!=b[0])

return a[0]-b[0];

for (i=a[0];i;i--)

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

return a[i]-b[i];

return 0;

}

int comp(const bignum\_t a,const int b){

int c[12]={1};

for (c[1]=b;c[c[0]]>=DEPTH;c[c[0]+1]=c[c[0]]/DEPTH,c[c[0]]%=DEPTH,c[0]++);

return comp(a,c);

}

int comp(const bignum\_t a,const int c,const int d,const bignum\_t b){

int i,t=0,O=-DEPTH\*2;

if (b[0]-a[0]<d&&c)

return 1;

for (i=b[0];i>d;i--){

t=t\*DEPTH+a[i-d]\*c-b[i];

if (t>0) return 1;

if (t<O) return 0;

}

for (i=d;i;i--){

t=t\*DEPTH-b[i];

if (t>0) return 1;

if (t<O) return 0;

}

return t>0;

}

void add(bignum\_t a,const bignum\_t b){

int i;

for (i=1;i<=b[0];i++)

if ((a[i]+=b[i])>=DEPTH)

a[i]-=DEPTH,a[i+1]++;

if (b[0]>=a[0])

a[0]=b[0];

else

for (;a[i]>=DEPTH&&i<a[0];a[i]-=DEPTH,i++,a[i]++);

a[0]+=(a[a[0]+1]>0);

}

void add(bignum\_t a,const int b){

int i=1;

for (a[1]+=b;a[i]>=DEPTH&&i<a[0];a[i+1]+=a[i]/DEPTH,a[i]%=DEPTH,i++);

for (;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);

}

void sub(bignum\_t a,const bignum\_t b){

int i;

for (i=1;i<=b[0];i++)

if ((a[i]-=b[i])<0)

a[i+1]--,a[i]+=DEPTH;

for (;a[i]<0;a[i]+=DEPTH,i++,a[i]--);

for (;!a[a[0]]&&a[0]>1;a[0]--);

}

void sub(bignum\_t a,const int b){

int i=1;

for (a[1]-=b;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH\*DEPTH,i++);

for (;!a[a[0]]&&a[0]>1;a[0]--);

}

void sub(bignum\_t a,const bignum\_t b,const int c,const int d){

int i,O=b[0]+d;

for (i=1+d;i<=O;i++)

if ((a[i]-=b[i-d]\*c)<0)

a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH\*DEPTH;

for (;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH\*DEPTH,i++);

for (;!a[a[0]]&&a[0]>1;a[0]--);

}

void mul(bignum\_t c,const bignum\_t a,const bignum\_t b){

int i,j;

memset((void\*)c,0,sizeof(bignum\_t));

for (c[0]=a[0]+b[0]-1,i=1;i<=a[0];i++)

for (j=1;j<=b[0];j++)

if ((c[i+j-1]+=a[i]\*b[j])>=DEPTH)

c[i+j]+=c[i+j-1]/DEPTH,c[i+j-1]%=DEPTH;

for (c[0]+=(c[c[0]+1]>0);!c[c[0]]&&c[0]>1;c[0]--);

}

void mul(bignum\_t a,const int b){

int i;

for (a[1]\*=b,i=2;i<=a[0];i++){

a[i]\*=b;

if (a[i-1]>=DEPTH)

a[i]+=a[i-1]/DEPTH,a[i-1]%=DEPTH;

}

for (;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);

for (;!a[a[0]]&&a[0]>1;a[0]--);

}

void mul(bignum\_t b,const bignum\_t a,const int c,const int d){

int i;

memset((void\*)b,0,sizeof(bignum\_t));

for (b[0]=a[0]+d,i=d+1;i<=b[0];i++)

if ((b[i]+=a[i-d]\*c)>=DEPTH)

b[i+1]+=b[i]/DEPTH,b[i]%=DEPTH;

for (;b[b[0]+1];b[0]++,b[b[0]+1]=b[b[0]]/DEPTH,b[b[0]]%=DEPTH);

for (;!b[b[0]]&&b[0]>1;b[0]--);

}

void div(bignum\_t c,bignum\_t a,const bignum\_t b){

int h,l,m,i;

memset((void\*)c,0,sizeof(bignum\_t));

c[0]=(b[0]<a[0]+1)?(a[0]-b[0]+2):1;

for (i=c[0];i;sub(a,b,c[i]=m,i-1),i--)

for (h=DEPTH-1,l=0,m=(h+l+1)>>1;h>l;m=(h+l+1)>>1)

if (comp(b,m,i-1,a)) h=m-1;

else l=m;

for (;!c[c[0]]&&c[0]>1;c[0]--);

c[0]=c[0]>1?c[0]:1;

}

void div(bignum\_t a,const int b,int& c){

int i;

for (c=0,i=a[0];i;c=c\*DEPTH+a[i],a[i]=c/b,c%=b,i--);

for (;!a[a[0]]&&a[0]>1;a[0]--);

}

void sqrt(bignum\_t b,bignum\_t a){

int h,l,m,i;

memset((void\*)b,0,sizeof(bignum\_t));

for (i=b[0]=(a[0]+1)>>1;i;sub(a,b,m,i-1),b[i]+=m,i--)

for (h=DEPTH-1,l=0,b[i]=m=(h+l+1)>>1;h>l;b[i]=m=(h+l+1)>>1)

if (comp(b,m,i-1,a)) h=m-1;

else l=m;

for (;!b[b[0]]&&b[0]>1;b[0]--);

for (i=1;i<=b[0];b[i++]>>=1);

}

int length(const bignum\_t a){

int t,ret;

for (ret=(a[0]-1)\*DIGIT,t=a[a[0]];t;t/=10,ret++);

return ret>0?ret:1;

}

int digit(const bignum\_t a,const int b){

int i,ret;

for (ret=a[(b-1)/DIGIT+1],i=(b-1)%DIGIT;i;ret/=10,i--);

return ret%10;

}

int zeronum(const bignum\_t a){

int ret,t;

for (ret=0;!a[ret+1];ret++);

for (t=a[ret+1],ret\*=DIGIT;!(t%10);t/=10,ret++);

return ret;

}

void comp(int\* a,const int l,const int h,const int d){

int i,j,t;

for (i=l;i<=h;i++)

for (t=i,j=2;t>1;j++)

while (!(t%j))

a[j]+=d,t/=j;

}

void convert(int\* a,const int h,bignum\_t b){

int i,j,t=1;

memset(b,0,sizeof(bignum\_t));

for (b[0]=b[1]=1,i=2;i<=h;i++)

if (a[i])

for (j=a[i];j;t\*=i,j--)

if (t\*i>DEPTH)

mul(b,t),t=1;

mul(b,t);

}

void combination(bignum\_t a,int m,int n){

int\* t=new int[m+1];

memset((void\*)t,0,sizeof(int)\*(m+1));

comp(t,n+1,m,1);

comp(t,2,m-n,-1);

convert(t,m,a);

delete []t;

}

void permutation(bignum\_t a,int m,int n){

int i,t=1;

memset(a,0,sizeof(bignum\_t));

a[0]=a[1]=1;

for (i=m-n+1;i<=m;t\*=i++)

if (t\*i>DEPTH)

mul(a,t),t=1;

mul(a,t);

}

#define SGN(x) ((x)>0?1:((x)<0?-1:0))

#define ABS(x) ((x)>0?(x):-(x))

int read(bignum\_t a,int &sgn,istream& is=cin){

char str[MAX\*DIGIT+2],ch,\*buf;

int i,j;

memset((void\*)a,0,sizeof(bignum\_t));

if (!(is>>str)) return 0;

buf=str,sgn=1;

if (\*buf=='-') sgn=-1,buf++;

for (a[0]=strlen(buf),i=a[0]/2-1;i>=0;i--)

ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;

for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf);j<a[0]\*DIGIT;buf[j++]='0');

for (i=1;i<=a[0];i++)

for (a[i]=0,j=0;j<DIGIT;j++)

a[i]=a[i]\*10+buf[i\*DIGIT-1-j]-'0';

for (;!a[a[0]]&&a[0]>1;a[0]--);

if (a[0]==1&&!a[1]) sgn=0;

return 1;

}

**14.2 分数**

struct frac{

int num,den;

};

double fabs(double x){

return x>0?x:-x;

}

int gcd(int a,int b){

int t;

if (a<0)

a=-a;

if (b<0)

b=-b;

if (!b)

return a;

while (t=a%b)

a=b,b=t;

return b;

}

void simplify(frac& f){

int t;

if (t=gcd(f.num,f.den))

f.num/=t,f.den/=t;

else

f.den=1;

}

frac f(int n,int d,int s=1){

frac ret;

if (d<0)

ret.num=-n,ret.den=-d;

else

ret.num=n,ret.den=d;

if (s)

simplify(ret);

return ret;

}

frac convert(double x){

frac ret;

for (ret.den=1;fabs(x-int(x))>1e-10;ret.den\*=10,x\*=10);

ret.num=(int)x;

simplify(ret);

return ret;

}

int fraqcmp(frac a,frac b){

int g1=gcd(a.den,b.den),g2=gcd(a.num,b.num);

if (!g1||!g2)

return 0;

return b.den/g1\*(a.num/g2)-a.den/g1\*(b.num/g2);

}

frac add(frac a,frac b){

int g1=gcd(a.den,b.den),g2,t;

if (!g1)

return f(1,0,0);

t=b.den/g1\*a.num+a.den/g1\*b.num;

g2=gcd(g1,t);

return f(t/g2,a.den/g1\*(b.den/g2),0);

}

frac sub(frac a,frac b){

return add(a,f(-b.num,b.den,0));

}

frac mul(frac a,frac b){

int t1=gcd(a.den,b.num),t2=gcd(a.num,b.den);

if (!t1||!t2)

return f(1,1,0);

return f(a.num/t2\*(b.num/t1),a.den/t1\*(b.den/t2),0);

}

frac div(frac a,frac b){

return mul(a,f(b.den,b.num,0));

}

**14.3 矩阵**

define MAXN 100

#define fabs(x) ((x)>0?(x):-(x))

#define zero(x) (fabs(x)<1e-10)

struct mat{

int n,m;

double data[MAXN][MAXN];

};

int mul(mat& c,const mat& a,const mat& b){

int i,j,k;

if (a.m!=b.n)

return 0;

c.n=a.n,c.m=b.m;

for (i=0;i<c.n;i++)

for (j=0;j<c.m;j++)

for (c.data[i][j]=k=0;k<a.m;k++)

c.data[i][j]+=a.data[i][k]\*b.data[k][j];

return 1;

}

int inv(mat& a){

int i,j,k,is[MAXN],js[MAXN];

double t;

if (a.n!=a.m)

return 0;

for (k=0;k<a.n;k++){

for (t=0,i=k;i<a.n;i++)

for (j=k;j<a.n;j++)

if (fabs(a.data[i][j])>t)

t=fabs(a.data[is[k]=i][js[k]=j]);

if (zero(t))

return 0;

if (is[k]!=k)

for (j=0;j<a.n;j++)

t=a.data[k][j],a.data[k][j]=a.data[is[k]][j],a.data[is[k]][j]=t;

if (js[k]!=k)

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

t=a.data[i][k],a.data[i][k]=a.data[i][js[k]],a.data[i][js[k]]=t;

a.data[k][k]=1/a.data[k][k];

for (j=0;j<a.n;j++)

if (j!=k)

a.data[k][j]\*=a.data[k][k];

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

if (i!=k)

for (j=0;j<a.n;j++)

if (j!=k)

a.data[i][j]-=a.data[i][k]\*a.data[k][j];

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

if (i!=k)

a.data[i][k]\*=-a.data[k][k];

}

for (k=a.n-1;k>=0;k--){

for (j=0;j<a.n;j++)

if (js[k]!=k)

t=a.data[k][j],a.data[k][j]=a.data[js[k]][j],a.data[js[k]][j]=t;

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

if (is[k]!=k)

t=a.data[i][k],a.data[i][k]=a.data[i][is[k]],a.data[i][is[k]]=t;

}

return 1;

}

double det(const mat& a){

int i,j,k,sign=0;

double b[MAXN][MAXN],ret=1,t;

if (a.n!=a.m)

return 0;

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

for (j=0;j<a.m;j++)

b[i][j]=a.data[i][j];

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

if (zero(b[i][i])){

for (j=i+1;j<a.n;j++)

if (!zero(b[j][i]))

break;

if (j==a.n)

return 0;

for (k=i;k<a.n;k++)

t=b[i][k],b[i][k]=b[j][k],b[j][k]=t;

sign++;

}

ret\*=b[i][i];

for (k=i+1;k<a.n;k++)

b[i][k]/=b[i][i];

for (j=i+1;j<a.n;j++)

for (k=i+1;k<a.n;k++)

b[j][k]-=b[j][i]\*b[i][k];

}

if (sign&1)

ret=-ret;

return ret;

}

**14.4 线性方程组**

#define MAXN 100

#define fabs(x) ((x)>0?(x):-(x))

#define eps 1e-10

//列主元gauss消去求解a[][]x[]=b[]

//返回是否有唯一解,若有解在b[]中

int gauss\_cpivot(int n,double a[][MAXN],double b[]){

int i,j,k,row;

double maxp,t;

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

for (maxp=0,i=k;i<n;i++)

if (fabs(a[i][k])>fabs(maxp))

maxp=a[row=i][k];

if (fabs(maxp)<eps)

return 0;

if (row!=k){

for (j=k;j<n;j++)

t=a[k][j],a[k][j]=a[row][j],a[row][j]=t;

t=b[k],b[k]=b[row],b[row]=t;

}

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

a[k][j]/=maxp;

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

a[i][j]-=a[i][k]\*a[k][j];

}

b[k]/=maxp;

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

b[i]-=b[k]\*a[i][k];

}

for (i=n-1;i>=0;i--)

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

b[i]-=a[i][j]\*b[j];

return 1;

}

//全主元gauss消去解a[][]x[]=b[]

//返回是否有唯一解,若有解在b[]中

int gauss\_tpivot(int n,double a[][MAXN],double b[]){

int i,j,k,row,col,index[MAXN];

double maxp,t;

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

index[i]=i;

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

for (maxp=0,i=k;i<n;i++)

for (j=k;j<n;j++)

if (fabs(a[i][j])>fabs(maxp))

maxp=a[row=i][col=j];

if (fabs(maxp)<eps)

return 0;

if (col!=k){

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

t=a[i][col],a[i][col]=a[i][k],a[i][k]=t;

j=index[col],index[col]=index[k],index[k]=j;

}

if (row!=k){

for (j=k;j<n;j++)

t=a[k][j],a[k][j]=a[row][j],a[row][j]=t;

t=b[k],b[k]=b[row],b[row]=t;

}

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

a[k][j]/=maxp;

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

a[i][j]-=a[i][k]\*a[k][j];

}

b[k]/=maxp;

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

b[i]-=b[k]\*a[i][k];

}

for (i=n-1;i>=0;i--)

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

b[i]-=a[i][j]\*b[j];

for (k=0;k<n;k++)

a[0][index[k]]=b[k];

for (k=0;k<n;k++)

b[k]=a[0][k];

return 1;

}

**14.5 线性相关**

//判线性相关(正交化)

//传入m个n维向量

#include <math.h>

#define MAXN 100

#define eps 1e-10

int linear\_dependent(int m,int n,double vec[][MAXN]){

double ort[MAXN][MAXN],e;

int i,j,k;

if (m>n)

return 1;

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

for (j=0;j<n;j++)

ort[i][j]=vec[i][j];

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

for (e=j=0;j<n;j++)

e+=ort[i][j]\*ort[k][j];

for (j=0;j<n;j++)

ort[i][j]-=e\*ort[k][j];

for (e=j=0;j<n;j++)

e+=ort[i][j]\*ort[i][j];

if (fabs(e=sqrt(e))<eps)

return 1;

for (j=0;j<n;j++)

ort[i][j]/=e;

}

}

return 0;

}

**14.6 日期**

//日期函数

int days[12]={31,28,31,30,31,30,31,31,30,31,30,31};

struct date{

int year,month,day;

};

//判闰年

inline int leap(int year){

return (year%4==0&&year%100!=0)||year%400==0;

}

//判合法性

inline int legal(date a){

if (a.month<0||a.month>12)

return 0;

if (a.month==2)

return a.day>0&&a.day<=28+leap(a.year);

return a.day>0&&a.day<=days[a.month-1];

}

//比较日期大小

inline int datecmp(date a,date b){

if (a.year!=b.year)

return a.year-b.year;

if (a.month!=b.month)

return a.month-b.month;

return a.day-b.day;

}

//返回指定日期是星期几

int weekday(date a){

int tm=a.month>=3?(a.month-2):(a.month+10);

int ty=a.month>=3?a.year:(a.year-1);

return (ty+ty/4-ty/100+ty/400+(int)(2.6\*tm-0.2)+a.day)%7;

}

//日期转天数偏移

int date2int(date a){

int ret=a.year\*365+(a.year-1)/4-(a.year-1)/100+(a.year-1)/400,i;

days[1]+=leap(a.year);

for (i=0;i<a.month-1;ret+=days[i++]);

days[1]=28;

return ret+a.day;

}

//天数偏移转日期

date int2date(int a){

date ret;

ret.year=a/146097\*400;

for (a%=146097;a>=365+leap(ret.year);a-=365+leap(ret.year),ret.year++);

days[1]+=leap(ret.year);

for (ret.month=1;a>=days[ret.month-1];a-=days[ret.month-1],ret.month++);

days[1]=28;

ret.day=a+1;

return ret;

}