字符串模板

[图论 3](#_Toc401907965)

[一、 基础图 3](#_Toc401907966)

[二、 生成树 7](#_Toc401907967)

[三、 最短路 10](#_Toc401907968)

[四、 欧拉回路 11](#_Toc401907969)

[五、 RMQ 13](#_Toc401907970)

[六、 最小环&最大环 14](#_Toc401907971)

[七、 拓扑排序 15](#_Toc401907972)

[八、 LCA 20](#_Toc401907973)

[九、 连通性 23](#_Toc401907974)

[十、 2 SAT 28](#_Toc401907975)

[十一、 匹配问题 29](#_Toc401907976)

[十二、 最大流 35](#_Toc401907977)

[十三、 费用流 39](#_Toc401907978)

[十四、 全局最小割 47](#_Toc401907979)

[十五、 网络流拓展 51](#_Toc401907980)

[十六、 最小树形图(有向图的最小生成树) 53](#_Toc401907981)

[十七、 树的Hash与同构 55](#_Toc401907982)

[十八、 差分约束 55](#_Toc401907983)

[十九、 最大团 55](#_Toc401907984)

[最大团 55](#_Toc401907985)

[字符串 56](#_Toc401907986)

[一、 Hash 56](#_Toc401907987)

[一般字符串Hash 56](#_Toc401907988)

[字符串区间Hash 56](#_Toc401907989)

[二、 KMP 57](#_Toc401907990)

[函数版 57](#_Toc401907991)

[KMP类 57](#_Toc401907992)

[三、 拓展KMP 58](#_Toc401907993)

[函数版 58](#_Toc401907994)

[四、 Manacher 59](#_Toc401907995)

[Manacher模板 59](#_Toc401907996)

[五、 AC自动机 61](#_Toc401907997)

[普通匹配 61](#_Toc401907998)

[AC自动机+DP 62](#_Toc401907999)

[六、 后缀数组 64](#_Toc401908000)

[论文模板(O(nlogn)) 64](#_Toc401908001)

[七、 字符串最小表示法 65](#_Toc401908002)

[朴素最小表示法 65](#_Toc401908003)

[判定两串是否循环同构 65](#_Toc401908004)

# **图论**

## 基础图

图struct

struct GRAPH {

struct node {

int u, v, next;

}adj[Maxm];

int tot, last[Maxn];

void init(int n) {

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

tot = 0;

}

void adde(int u, int v) {

adj[tot].u = u; adj[tot].v = v;

adj[tot].next = last[u]; last[u] = tot++;

}

}e;

n条边的带一个环树森林

struct CirGraph {

struct node {

//onCir==0 不在环上, ==1在环上

int u, v, l, onCir, next;

} adj[Maxm];

int tot, last[Maxn];

//vis==0未遍历, ==1遍历不是环上, ==2在环上

int top, vis[Maxn], sta[Maxn];

void adde(int u, int v, int l) {

adj[tot].u = u; adj[tot].v = v; adj[tot].l = l; adj[tot].onCir = 0;

adj[tot].next = last[u]; last[u] = tot++;

}

void init(int n) {

tot = 0;

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

}

void dfs(int u, int from) {

int j, v;

vis[u] = 1;

for(j = last[u]; j != -1; j = adj[j].next) {

if(j == from) continue;

v = adj[j].v;

if(!vis[v]) {

sta[++top] = j;

dfs(v, j ^ 1);

top--;

}

else if(vis[v] == 1 && vis[u] == 1){

int tp = top, jj;

sta[++tp] = j;

while(tp) {

jj = sta[tp--];

adj[jj].onCir = adj[jj ^ 1].onCir = 1;

vis[adj[jj].u] = 2;

if(adj[jj].u == v) break;

}

}

}

}

void ready(int n) {

int i, j;

for(i = 0; i <= n; i++) vis[i] = 0;

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

if(!vis[i]) {

top = 0;

dfs(i, -1);

}

}

}

}gg;

完整版

set<LL> Hset;

int S[Maxn<<1|1], T[Maxn];

int next[Maxn<<1|1], is[Maxn];

void getnext(int T[], int LT, int next[]) {

int i, j;

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

for (i = 1, j = 0; i < LT; ) {

while (j != -1 && T[i] != T[j]) j = next[j];

i++; j++;

next[i] = j;

}

}

void KMP (int S[], int LS, int T[], int LT, int next[]) {

int i, j;

for(i = 0; i < LS; i++) is[i] = 0;

for (i = 0, j = 0; i < LS; i++) {

while (j != -1 && S[i] != T[j]) j = next[j];

j++;

if (j == LT) {

is[i - LT + 1] = 1;

j = next[j];

}

}

}

//init初始化

//adde加边(注意单向边和双向边)

//ready()连通块, 环, 求树hash, 求环可旋转位置

struct CirGraph {

struct node {

//onCir==0 不在环上, ==1在环上

int u, v, l, onCir, next;

} adj[Maxm];

int tot, last[Maxn];

//vis==0未遍历, ==1遍历不是环上, ==2在环上

int top, vis[Maxn], sta[Maxn];

int dep[Maxn];

LL H[Maxn];

LL h[Maxn];

vector<int> ok[Maxn];//(每个环可旋转长度)

vector<int> belong[Maxn];//(每个环结点集合)

int circnt;//环/树个数(1~circnt)

void adde(int u, int v, int l) {

adj[tot].u = u; adj[tot].v = v; adj[tot].l = l; adj[tot].onCir = 0;

adj[tot].next = last[u]; last[u] = tot++;

}

void init(int n) {

tot = 0;

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

}

void dfs(int u, int from) {

int j, v;

vis[u] = 1;

for(j = last[u]; j != -1; j = adj[j].next) {

if(j == from) continue;

v = adj[j].v;

if(!vis[v]) {

sta[++top] = j;

dfs(v, j ^ 1);

top--;

}

else if(vis[v] == 1 && vis[u] == 1){

int tp = top, jj;

sta[++tp] = j;

while(tp) {

jj = sta[tp--];

adj[jj].onCir = adj[jj ^ 1].onCir = 1;

vis[adj[jj].u] = 2;

belong[circnt].PB(adj[jj].u);

if(adj[jj].u == v) break;

}

}

}

}

void getDep(int u, int from) {

int j, v;

dep[u] = 1;

for(j = last[u]; j != -1; j = adj[j].next) {

if(j == from || adj[j].onCir) continue;

v = adj[j].v;

getDep(v, j ^ 1);

cmax(dep[u], dep[v] + 1);

}

}

void Hash(int u, int from) {

int j, v;

LL s = h[dep[u]];

for(j = last[u]; j != -1; j = adj[j].next) {

if(j == from || adj[j].onCir) continue;

v = adj[j].v;

Hash(v, j ^ 1);

s = (s + h[dep[u]] \* H[v]) % MOD;

}

H[u] = s;

}

void ready(int n) {

int i, j;

circnt = 0;

for(i = 0; i <= n; i++) vis[i] = 0;

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

if(!vis[i]) {

top = 0;

circnt++;

belong[circnt].resize(0);

dfs(i, -1);

}

}

// for(i = 0; i < tot; i++) printf("%d %d --> %d %d\n", i, adj[i].u, adj[i].v, adj[i].onCir);

Hset.clear();

srand(time(NULL));

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

LL x = ((rand() << 15) + rand()) % MOD;

while(Hset.find(x) != Hset.ED) x = ((rand() << 15) + rand()) % MOD;

Hset.insert(x);

h[i] = x;

}

for(i = 1; i <= n; i++) if(vis[i] == 2) {

getDep(i, -1);

Hash(i, -1);

}

// for(i = 1; i <= n; i++) cout << "hash " << i <<" " << H[i] << endl;

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

ok[i].resize(0);

int L = belong[i].SZ;

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

S[j + L] = S[j] = T[j] = H[belong[i][j]];

}

getnext(T, L, next);

KMP(S, L << 1, T, L, next);

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

if(is[j]) ok[i].PB(j);

}

// cout << "OK ";for(j = 0; j < ok[i].SZ; j++) cout << ok[i][j] << " "; cout << endl;

}

}

}gg;

## 生成树

Prim

Kruskal

Borůvka算法

次小生成树

最小瓶颈生成树

直径最短生成树

度数最多的点, 度数最少的生成树

两点之间路径, 权重总和最小的生成树

有两种权重, 权重比值最小的生成树(最优比例生成树)

边不重叠, 权重最小的k棵树

重叠边增加额外权重, 权重最小的k棵树

权重最小的生成子树, 刚好k个点

权重最小的生成子树, 刚好是给定的K个点

度限制最小生成树

动态最小生成树

有向图最小生成树(最小树形图)

任意两点间路径的瓶颈, 形成两点间最小割的无向生成树

dfs树

bfs树

最短路径树

斯坦纳树

生成树计数

Matrix-Tree定理(Kirchhoff矩阵-树定理)

Kirchhoff矩阵C[G]=D[G]-A[G] (度数矩阵-邻接矩阵)

求C[G]任意一个n-1阶主子式的行列式(取出第i行和第i列)

//spoj 104 Highways

//朴素求解, ans是double用printf("%0.0f\n",ans)输出成int

//返回是否有解,若有解ans\*=a[i][i]中

#define eps 1e-10

#define fabs(x) ((x)>eps?(x):-(x))

#define ZERO(x) (fabs(x) > eps ? 0 : 1)

int gauss\_tpivot (int n, double a[][Maxn]) {

int i, j, k, p;

double maxp, t;

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

if(ZERO(a[k][k])) {

for(i = k + 1; i < n; i++) if(!ZERO(a[i][i])) break;

if(i >= n) return 0;

for(j = k; j < n; j++) {

swap(a[k][j], a[i][j]);

}

}

maxp = a[k][k];

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];

}

}

return 1;

}

int main() {

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

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

a[i][j] = b[i][j] = 0;

}

}

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

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

if(u == v) continue;

u--; v--;

a[u][u]++; a[v][v]++;

b[u][v] = b[v][u] = 1;

}

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

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

a[i][j] = a[i][j] - b[i][j];

}

}

double ans = 0;

if(gauss\_tpivot(n - 1, a)) {

ans = 1;

for(i = 0; i < n - 1; i++) ans \*= a[i][i];

}

else ans = 0;

printf("%0.0f\n", ans);

}

//hdu 4305 MOD = 10007

//答案取模, 每一步需要取模处理

LL gauss\_tpivot (int n, LL a[][Maxn]) {

LL ret = 1;

int i, j, k;

LL p , q;

tot = 0;

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

if(!a[k][k]) {

for(i = k + 1; i < n; i++) if(a[i][k]) break;

if(i >= n) return 0;

for(j = k; j < n; j++) {

swap(a[k][j], a[i][j]);

}

ret \*= -1;

}

if(a[k][k] < 0) {

for(j = 0; j < n; j++) a[k][j] \*= -1;

}

p = a[k][k];

ret = ret \* p % MOD;

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

q = a[i][k] \* inv[a[k][k]] % MOD;

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

a[i][j] = (a[i][j] - a[k][j] \* q) % MOD;

if(a[i][j] < 0) a[i][j] += MOD;

}

}

}

return ret % MOD;

}

## 最短路

floyd

dijkstra

spfa

普通入队列方式可以被cha

deque判当前节点和队首节点可以被三角形数据cha

priority\_queue优化(本质dijkstra靠谱!)

K短路

练手: POJ2449 UVA10740

提高: SGU145 SGU314

最短路+A\*

f[x] = g[x] + h[x]

其中h[x]以从t到x的最短路

g[x]为从s到x的当前路径长度

YEN算法

MPS算法

最短路树+回溯

带环K短路

无环K短路

不重复k短路

费用流~.~

求字典序最小的路径

无重复标号(如给定边长度, 输出最短路径上经过的结点的最小字典序):hdu1385

有重复标号(如边长度都为d, 每条边给定标号, 输出最短路径上边标号的最小字典序)http://codeforces.com/gym/100084 (I题)

有重复标号(如给定边长度和边标号, 输出最短路径上每条边给定的标号的最小字典序)???

## 欧拉回路

判断无向图欧拉路和欧拉回路的条件

判断有向图欧拉路和欧拉回路的条件

注意图的连通性的影响

fleury算法(和一般dfs的差别???)

磁鼓欧拉回路(HDU 2894)

n个触头, 建图以(1<<(n-1))个结点, 建1<<n条有向边, 从最大结点出发跑欧拉回路

混合欧拉路欧拉回路的判断-->网络流模型

(相关题目：pku1637，zju1992，hdu3472, hdu1956)

给出一张混合图（有有向边，也有无向边），判断是否存在欧拉回路。

首先是对图中的无向边随意定一个方向，然后统计每个点的入度（indeg）和出度（outdeg），如果（indeg - outdeg）是奇数的话，一定不存在欧拉回路；

如果所有点的入度和出度之差都是偶数，那么就开始网络流构图：

1，对于有向边，舍弃；对于无向边，就按照最开始指定的方向建权值为 1 的边；

2，对于入度小于出度的点，从源点连一条到它的边，权值为（outdeg - indeg）/2；出度小于入度的点，连一条它到汇点的权值为（indeg - outdeg）/2 的边；

构图完成，如果满流（求出的最大流值 == 和汇点所有连边的权值之和），那么存在欧拉回路，否则不存在。

建图基础:

//outdeg出度, indeg入度, !w双向边

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

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

outdeg[u]++;

indeg[v]++;

if(!w) {

adde(u, v, 1);

}

}

int flag = true;

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

if(abs(indeg[i] - outdeg[i]) & 1) {

flag = false;

break;

}

}

if(flag) {

S = 0; T = n + 1; sums = sumt = 0;

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

if(outdeg[i] > indeg[i]) {

adde(S, i, (outdeg[i] - indeg[i]) / 2);

sums += (outdeg[i] - indeg[i]) / 2;

}

if(indeg[i] > outdeg[i]) {

adde(i, T, (indeg[i] - outdeg[i]) / 2);

sumt += (indeg[i] - outdeg[i]) / 2;

}

}

int flow = dinic(S, T, T + 1);

if(flow != sums) flag = false;

}

## RMQ

RMQ实现模板

验题 POJ3264

struct RMQ {

int n;

int st[20][Maxn];

void init(int v[], int L) {

int i, j, k;

n = L;

for(i = 0; i <= n; i++) st[0][i] = v[i];

for(j = 1, k = 0; (1<<j) <= n; j++, k++) {

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

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

}

}

}

int query(int l, int r) {

int k = log2(r - l + 1);

// int k = 31 - \_\_builtin\_clz(r - l + 1);

// int k = kk[r - l + 1];

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

}

}rmq1, rmq2;

## 最小环&最大环

最小环

验题: POJ1734

//floyd算法

//g[][] 边

//f[][] 最短路

//pre[][] 记录路径

int MinCircle() {

int i, j, k, p;

int mc = MOD;

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

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

f[i][j] = g[i][j];

if(g[i][j] < MOD && i != j) {

pre[i][j] = j; //记录初始长度为1的路径

}

else {

pre[i][j] = -1;

}

}

}

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

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

for(j = i + 1; j < k; ++j) {

int tp = g[k][i] + f[i][j] + g[j][k];

if(tp < mc) { //更新最小环和路径

mc = tp;

for(p = i, len = 0; p != -1; p = pre[p][j], len++) {

path[len] = p;

}

path[len++] = k;

}

}

}

for(i = 1; i <= n; ++i) { //普通floyd过程

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

if(f[i][j] > f[i][k] + f[k][j]) {

f[i][j] = f[i][k] + f[k][j];

pre[i][j] = pre[i][k];

}

}

}

}

return mc;

}

未验题:

//http://www.cnblogs.com/Yz81128/archive/2012/08/15/2640940.html

//dijkstra

//求一条边的两段的结点之间最短路(不包含这条边)和这条边之和

//O(m\*m\*logn)

最大环(??? 可做)

poj2240

//这一题, 货币兑换, 看看是否存在能使自己增值的方案, 感官上并不能算最大环问题, 而更贴切的看法是转换为最长路, 由BellmanFord思想判断存在>1的环的问题, 不过可以通过floyd的\*\*松弛操作\*\*同样解决掉

## 拓扑排序

基础知识

一、拓扑排序

什么是拓扑排序？简单地说，由某个集合上的一个偏序得到该集合上的一个全序，这个操作称之为拓扑排序。

AVO网（Activity On Vertex Network），也称作顶点表示活动的网，就是用顶点表示活动，用弧表示活动间的优先关系的有向图。

拓扑排序是对AOV网构造一个线性序列，使所有的优先关系在此序列中得以体现，换句话说，就是在AVO网中，若Vi是Vj的前驱，那么在拓扑序列中，Vi必须排在Vj之前。

拓扑排序的步骤如下：

1、在有向图中选取一个没有前驱的顶点，并输出；

2、从图中删除该顶点和所有以它为尾的弧；

3、重复1和2，直到全部顶点均被输出为止。

（注：弧尾指弧的起始点，弧头是指弧的终点）

一个有向无环图的拓扑序列是不唯一的。

二、关键路径

与AVO网相对应的是AOE网（Activity On Edge）即边表示活动的网。

所谓AOE网，也称作“用表示活动的网”，就是用顶点表示事件，用弧表示活动，弧上的权值表示该活动的持续时间的有向图。

关键路径：是指AOE网上，完成工程的最短时间是从开始点到完成点的最长路径长度，路径长度最长的路径叫做关键路径。

分析关键路径的目的是辨别哪些是关键活动，以便争取提高关键活动的工效，缩短整个工期。

事件Vi的最早发生时间Ve(i)中从顶点V到Vi的最长路径长度。

活动Ai的最迟开始时间V(i)是该活动的终点所表示的事件最迟发生时间与该活动的所需时间之差。

一个活动的最迟开始时间减去最早开始时间之差定义为完成该活动的时间余量，而最迟开始时间等于最早开始时间的活动叫做关键活动。

求关键路径的算法描述：

(1) 输入e条弧<j,k>，建立AOE网的存储结构。

(2) 从源点v1出发，令ve(1)=0,求 ve(j) 2<=j<=n。

(3) 从汇点vn出发，令vl(n)=ve(n),求 vl(i) 1<=i<=n-1。

(4) 根据各顶点的ve和vl值，求每条弧s（活动）的最早开始时间e(s)和最晚开始时间l(s)，其中e(s)=l(s)的为关键活动。

求关键路径的算法分析

（1） 求关键路径必须在拓扑排序的前提下进行，有环图不能求关键路径；

（2） 只有缩短关键活动的工期才有可能缩短工期；

（3） 若一个关键活动不在所有的关键路径上，减少它并不能减少工期；

（4） 只有在不改变关键路径的前提下，缩短关键活动才能缩短整个工期。

求任意拓扑排序或判断是否可以拓扑排序

队列实现

dfs实现

记录各点完成访问的时刻(完成时间),用DFS遍历一次整个图,得出各结点的完成时间,然后按完成时间倒序排列就得到了图的拓扑序列

/\* 拓扑排序 O(e)

\* 确保是有向无环图！

\* 结果逆序存放在sta中！

\* \*/

VI ve[Maxn];

bool vis[Maxn];

int sta[Maxn], top;

void dfsTopo(int u) {

vis[u] = true;

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

if (!vis[ve[u][i]])

dfsTopo(ve[u][i]);

sta[top ++ ] = u;

}

void Toposort(int n) {

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

top = 0;

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

if (!vis[i]) dfsTopo(i);

}

字典序最小解

一般直接将队列改为优先队列即可(要求无重复标号!!!)

HDU1285

区分字典序最小和标号小的尽量靠前 特殊题目要求: POJ3687

int n, m;

struct node {

int u, v, next;

}e[Maxm];

int tot, last[Maxn];

priority\_queue<int > Q;

vector<int> ans, out;

int deg[Maxn];

void adde(int u, int v) {

e[tot].u = u; e[tot].v = v; e[tot].next = last[u]; last[u] = tot++;

}

int main() {

int i, j, u, v, te;

scanf("%d", &te);

while(te--) {

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

tot = 0;

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

last[i] = -1;

deg[i] = 0;

}

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

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

adde(v, u);

deg[u]++;

}

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

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

if(deg[i] == 0) Q.push(i);

}

ans.clear();

out.clear();

while(!Q.empty()) {

u = Q.top();

Q.pop();

ans.PB(u);

for(j = last[u]; j != -1; j = e[j].next) {

v = e[j].v;

deg[v]--;

if(deg[v] == 0) Q.push(v);

}

}

if(ans.SZ < n) {

printf("-1\n");

}

else {

reverse(ans.BG, ans.ED);

out = ans;

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

out[ans[i] - 1] = i + 1;

}

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

if(i) printf(" ");

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

}

printf("\n");

}

}

return 0;

}

/\*

2

5 4

5 1

4 2

1 3

2 3

10 5

4 1

8 1

7 8

4 1

2 8

ans:

2 4 5 3 1 逆向建图

5 1 6 2 7 8 3 4 9 10 没有判重边的话就输出 -1

\*/

有重复标号字典序最小???

有向图拓扑排序计数(状压dp)

验题: ZOJ1346 HDU4917

// dp[status] 当前图中结点01表示, 初始化只剩一个点dp[1<<j]=1

// deg[] 结点度数

// g[][] vector存边

// group[] 将当前块结点映射到group的0~now位置

// now当前块节点数

// 各个连通块求dp值, 乘上组合数C[n][now], 没处理一块n-=now

int doit(int status) {

if(dp[status] != -1) return dp[status];

LL sum = 0;

int i, j, u, v;

for(i = 0; i < now; i++) {

u = group[i];

if(deg[u] == 0) {

deg[u] = -1;

for(j = 0; j < g[u].SZ; j++) {

v = g[u][j];

deg[v]--;

}

sum += doit(status ^ (1<<i));

for(j = 0; j < g[u].SZ; j++) {

v = g[u][j];

deg[v]++;

}

deg[u] = 0;

}

}

return (dp[status] = sum % MOD);

}

树拓扑排序计数

计算公式

n!/node[i] (1 < i <= n)

n表示树的总节点数, node[i] 表示以i为根的子树的节点数

关键路径

求关键路径：

1. 关键节点：

正向拓扑序， 求每个点最早到达时间early[i] = max(early[i], early[j] + edge[k]);

利用汇点的early值，反向拓扑求每个点迟到达时间late[i] = min(late[i], late[j] - edge[k]);

关键节点early == late

PS：

最早开始时间可以理解为是必须等到之前的任务完成才能做

最迟开始时间可以理解为是必须为后面的任务留出足够的时间

2. 关键路径 ：

源点到汇点的最长路（可能有多条）

## LCA

倍增算法

//改bfs

验题: POJ1330,

//LCA O(nlogn)

//加边之前使用initLCA()初始化数组

//调用solveLCA()初始化LCA

//调用getLCA(x,y)返回x和y的LCA

struct node {

int u, v, l, next;

}e[Maxm];

int tot, last[Maxn];

void adde(int u, int v) {

e[tot].u = u; e[tot].v = v;

e[tot].next = last[u]; last[u] = tot++;

}

int depth[Maxn], fa[16][Maxn];

int n, m;

void initLCA() {

int i, j;

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

depth[i] = -1;

last[i] = -1;

for(j = 0; j < 16; j++) fa[j][i] = -1;

}

tot = 0;

}

void dfsLCA(int u) {

int j, v;

for(j = last[u]; j != -1; j = e[j].next) {

v = e[j].v;

if(depth[v] == -1) {

fa[0][v] = u;

depth[v] = depth[u] + 1;

dfsLCA(v);

}

}

}

int getLCA (int x, int y) {

int i, dif = abs(depth[x] - depth[y]);

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

for (i = 20 - 1; i >= 0; i--) {

if ((1 << i) & dif) {

dif -= (1 << i);

x = fa[i][x];

}

}

for (i = 16 - 1; i >= 0; i--) {

if (fa[i][x] != fa[i][y]) {

x = fa[i][x];

y = fa[i][y];

}

}

if (x == y) return x;

else return fa[0][x];

}

void solveLCA() {

int i, j, root = 1;

for(i = 0; i <= n; i++) depth[i] = -1;

fa[0][root] = root;

depth[root] = 0;

dfsLCA(root);

for (i = 1; i < 16; i++)

for(j = 0; j <= n; j++)

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

}

Tarjan

验题: HDU2586

//Tarjan离线LCA

//e存放树边

//q存放query

struct GRAPH {

struct node {

int u, v, n, next;

}adj[Maxm];

int tot, last[Maxn];

void init(int n) {

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

tot = 0;

}

void adde(int u, int v, int n) {

adj[tot].u = u; adj[tot].v = v; adj[tot].n = n;

adj[tot].next = last[u]; last[u] = tot++;

}

}e, q;

int fa[Maxn], lca[Maxn], ans[Maxn];

int visit[Maxn];

int getfa(int x) {

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

else return (fa[x] = getfa(fa[x]));

}

void tarjanLCA(int u) {

int i, j, v, f;

fa[u] = u;

visit[u] = 1;

for(j = e.last[u]; j != -1; j = e.adj[j].next) {

v = e.adj[j].v;

if(!visit[v]) {

tarjanLCA(v);

fa[v] = u;

}

}

for(j = q.last[u]; j != -1; j = q.adj[j].next) {

v = q.adj[j].v;

if(visit[v]) {

lca[q.adj[j].n] = f = getfa(v);

ans[q.adj[j].n] = dist[v] + dist[u] - 2 \* dist[f];

}

}

}

LCA 转 RMQ

0: (1)

/ \

1: (2) (7)

/ \ \

2: (3) (4) (8)

/ \

3: (5) (6)

step 1: dfs遍历树, 依次记录每次到达的点, 以及每个点的深度得到序列:

结点访问顺序是: 1 2 3 2 4 5 4 6 4 2 1 7 8 7 1 //共2n-1个值

结点对应深度是: 0 1 2 1 2 3 2 3 2 1 0 1 2 1 0

step 2: 利用ST计算任意区间最小深度的点的ID

step 3: 对于每次查询, 查询u第一次出现位置到v第一次出现位置区间的最小值

## 连通性

强联通分量(Tarjan)

/\*

\* SCC 使用图struct

\*/

struct GRAPH {

struct node {

int u, v, next;

}adj[Maxm];

int tot, last[Maxn];

void init(int n) {

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

tot = 0;

}

void adde(int u, int v) {

adj[tot].u = u; adj[tot].v = v;

adj[tot].next = last[u]; last[u] = tot++;

}

}e;

int dfn[Maxn], low[Maxn], belong[Maxn], instack[Maxn], ncnt, nindex;

stack<int> sta;

void Tarjan(int u) {

int j, v;

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

sta.push(u);

instack[u] = 1;

for(j = e.last[u]; ~j; j = e.adj[j].next) {

v = e.adj[j].v;

if(-1 == dfn[v]) {

Tarjan(v);

if(low[u] > low[v]) low[u] = low[v];

}

else if(instack[v] && dfn[v] < low[u]) {

low[u] = dfn[v];

}

}

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

do {

v = sta.top();

sta.pop();

instack[v] = 0;

belong[v] = ncnt;

}while(u != v);

ncnt++;

}

}

void solve(){

int i, flag = 0;

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

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

memset(instack, 0, sizeof(instack));

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

ncnt = 1;

nindex = 1;

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

if(-1 == dfn[i]) {

Tarjan(i);

flag++;

}

}

///\*

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

// printf("%d %d\n", i, belong[i]);

//\*/

}

边双连通(Tarjan)

/\*

\*EBCC

\*/

struct GRAPH {

struct node {

int u, v, next;

}adj[Maxm];

int tot, last[Maxn];

void init(int n) {

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

tot = 0;

}

void adde(int u, int v) {

adj[tot].u = u; adj[tot].v = v;

adj[tot].next = last[u]; last[u] = tot++;

}

}e;

int n, m;

int dfn[Maxn], low[Maxn], belong[Maxn], instack[Maxn], nindex, ncnt;

stack<int> sta;

void Tarjan(int u, int from) {

int j, v;

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

instack[u] = 1;

sta.push(u);

for(j = e.last[u]; ~j; j = e.adj[j].next) {

if(j == from) continue;

v = e.adj[j].v;

if(-1 == dfn[v]) {

Tarjan(u, j ^ 1);

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

}

else if(instack[v] && dfn[v] < low[u]) {

low[u] = dfn[v];

}

}

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

do {

v = sta.top();

sta.pop();

instack[v] = 0;

belong[v] = ncnt;

} while(v != u);

ncnt++;

}

}

void solve() {

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

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

memset(instack, 0, sizeof(instack));

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

ncnt = 1;

nindex = 1;

int flag = 0;

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

if(-1 == dfn[i]) {

Tarjan(i, -1);

flag++;

}

///\*

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

cout << i << " " << belong[i] <<endl;

//\*/

}

点双连通(Tarjan+重建图)

#define STEP 20

struct GRAPH {

struct node {

int u, v, next;

}adj[Maxm];

int tot, last[Maxn];

void init(int n) {

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

tot = 0;

}

void adde(int u, int v) {

// cout << "adde " << u << " " << v << endl;

adj[tot].u = u; adj[tot].v = v;

adj[tot].next = last[u]; last[u] = tot++;

}

}e, e1;

int dfn[Maxn], low[Maxn], iscut[Maxn], belong[Maxm], color[Maxm];

int lcnt[Maxm];

int nindex, ncnt;

stack<int> sta;

int n, m, q;

void Tarjan(int u, int from) {

int v, child = 0;

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

for(int j = e.last[u]; j != -1; j = e.adj[j].next) {

if(j == from) continue;

v = e.adj[j].v;

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

sta.push(j);

if(!dfn[v]) {

child ++;

Tarjan(v, j ^ 1);

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

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

ncnt++;

while(sta.top() != j) {

belong[sta.top() / 2] = ncnt;

sta.pop();

}

belong[j / 2] = ncnt;

sta.pop();

iscut[u] = 1;

}

}

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

}

}

if(from < 0 && child == 1)iscut[u] = -1;//child

// if(child

}

set<PII> S;

void newAdde(int u, int v) {

if(u > v) swap(u, v);

PII ss = MP(u, v);

if(S.find(ss) == S.ED) {

S.insert(ss);

e1.adde(u, v);

e1.adde(v, u);

}

}

int visit[Maxm];

void gao(int u) {

int j, v;

for(j = e1.last[u]; j != -1; j = e1.adj[j].next) {

v = e1.adj[j].v;

if(visit[v] == -1) {

visit[v] = 1;

gao(v);

}

}

}

void buildGraph() {

int i, j, u, v, x, y;

e1.init(ncnt + 10);

S.clear();

for(j = 0; j < e.tot; j += 2) {

u = e.adj[j].u; v = e.adj[j].v;

if(iscut[u] == -1 && iscut[v] == -1) continue;

else {

if(iscut[u] != -1){

x = iscut[u];

y = belong[j / 2];

newAdde(x, y);

}

if(iscut[v] != -1){

x = iscut[v];

y = belong[j / 2];

newAdde(x, y);

}

}

}

for(i = 0; i <= ncnt; i++) visit[i] = -1;

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

if(visit[i] == -1) {

visit[i] = 1;

gao(i);

e1.adde(0, i);

e1.adde(i, 0);

}

}

}

void solve() {

int i, j;

memset(dfn, 0, sizeof(dfn));

memset(low, 0, sizeof(low));

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

memset(color, 0, sizeof(color));

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

ncnt = nindex = 0;

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

if(!dfn[i]) {

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

Tarjan(i, -1);

}

}

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

if(iscut[i] == 1) {

color[++ncnt] = 1;

iscut[i] = ncnt;

}

}

buildGraph();

}

## 2 SAT

## 匹配问题

匈牙利 + 贪心优化

// O(n\*m)

// 可以处理n=3000规模

struct enode {

int v, next;

}e[Maxm];

int last[Maxn], tot;

int n, m, match = 0;

int mx[Maxn], my[Maxn], dx[Maxn], dy[Maxn], dis, visit[Maxn];

int preHungary()

{

int res = 0, u, v;

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

{

u = i;

for(int j = last[i]; -1 != j; j = e[j].next)

{

v = e[j].v;

if(-1 == my[v])

{

mx[u] = v;

my[v] = u;

res++;

break;

}

}

}

return res;

}

bool dfs(int u)

{

int v;

for(int j = last[u]; -1 != j; j = e[j].next)

{

v = e[j].v;

if(visit[v]) continue;

visit[v] = true;

if(-1 == my[v] || dfs(my[v]))

{

my[v] = u;

mx[u] = v;

return true;

}

}

return false;

}

int solve()

{

match = 0;

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

for(int j = n; j < n + m; j++) my[j] = -1;

match += preHungary();

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

if(-1 == mx[i])

{

for(int j = n; j < n + m; j++) visit[j] = false;

if(dfs(i)) match++;

}

return match;

}

HK (O(E\*N^0.5))

验题: HDU 2389 spoj 4206(n=50000,m=150000)

/\*

\* HK O(E\*V^0.5) 最大匹配

\* 左半图 0~n-1

\* 右半图 n~n+m-1

\*/

struct edgenode {

int v, next;

}e[Maxm];

int last[Maxn], tot;

int n, m, match = 0;

int mx[Maxn], my[Maxn], dx[Maxn], dy[Maxn], dis, visit[Maxn];

queue<int> Q;

int ux[Maxn], uy[Maxn], px[Maxn], py[Maxn], pv[Maxn];

void adde(int u, int v) {

e[tot].v = v; e[tot].next = last[u]; last[u] = tot++;

}

bool searchPath()

{

int i, j;

dis = MOD;

for(i = 0; i < n; i++) dx[i] = -1;

for(j = n; j < n + m; j++) dy[j] = -1;

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

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

if(-1 == mx[i]) Q.push(i);

int u, v;

while(!Q.empty())

{

u = Q.front(); Q.pop();

if(dx[u] > dis) break;

for(int j = last[u]; j != -1; j = e[j].next)

{

v = e[j].v;

if(-1 != dy[v]) continue;

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

if(-1 == my[v])

dis = dy[v];

else

{

dx[my[v]] = dy[v] + 1;

Q.push(my[v]);

}

}

}

return dis != MOD;

}

bool dfs(int u)

{

int v;

for(int j = last[u]; j != -1; j = e[j].next)

{

v = e[j].v;

if(visit[v] || dx[u] + 1 != dy[v]) continue;

if(dy[v] == dis && my[v] != -1) continue;

visit[v] = true;

if(-1 == my[v] || dfs(my[v]))

{

my[v] = u;

mx[u] = v;

return true;

}

}

return false;

}

int solve()

{

int i, j;

match = 0;

for(i = 0; i < n; i++) mx[i] = -1;

for(j = n; j < n + m; j++) my[j] = -1;

// match += preHungary();

while(searchPath())

{

for(j = n; j < n + m; j++) visit[j] = false;

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

if(-1 == mx[i] && dfs(i)) match++;

}

return match;

}

稳定婚姻问题

延迟认可算法(Gale-Shapley算法)

int mx[Maxn], my[Maxm]; //x匹配的y, y匹配的x

//yorder表示在x眼中y的顺序, 0~m-1为喜爱度递减的y的编号

//xorder表示在y眼中x的顺序, 0~n-1为编号0~n-1的x的重要度

//越重要, 值越小

int yorder[Maxn][Maxm], xorder[Maxm][Maxn];

int cur[Maxn];

int n, m;

queue<int> que;

void GaleShapley() {

int i, j, v;

for(i = 0; i <= n; i++) mx[i] = -1,cur[i] = 0;//初始化

for(j = 0; j <= m; j++) my[j] = -1;

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

for(i = 0; i < n; i++) que.push(i);//将x加入队列

while(!que.empty()) {//x还有没找到朋友的

i = que.front(); que.pop();

if(cur[i] >= m) continue;

v = yorder[i][cur[i]++];

if(my[v] == -1) { //y没有匹配

mx[i] = v; my[v] = i;

}

else if(xorder[v][i] < xorder[v][my[v]]) {//i比之前的好

mx[my[v]] = -1;

que.push(my[v]);

my[v] = i; mx[i] = v;

}

else {//i比以前的差, i找下一个y

que.push(i);

}

}

}

带花树

模板题 验题: ZOJ 3316

int n, head, tail, Start, Finish;

int match[Maxn]; //表示哪个点匹配了哪个点

int father[Maxn]; //这个是增广路径的father

int base[Maxn]; //该点属于哪朵花

int Q[Maxn];

bool mark[Maxn], mp[Maxn][Maxn], InBlossom[Maxn], inqueue[Maxn]; //mp[][]邻接关系

void BlossomContract(int x, int y) {

memset(mark, false, sizeof(mark));

memset(InBlossom, false, sizeof(InBlossom));

#define pre father[match[i]]

int lca, i;

for( i = x; i; i = pre) {

i = base[i];

mark[i] = true;

}

for(i = y; i; i = pre) {

i = base[i]; //寻找lca之旅……一定要注意i=base[i]

if(mark[i]) {

lca = i;

break;

}

}

for(i = x; base[i] != lca; i = pre) {

if(base[pre] != lca) father[pre] = match[i]; //对于BFS树中的父边是匹配边的点，

//father向后跳

InBlossom[base[i]] = true;

InBlossom[base[match[i]]] = true;

}

for(i = y; base[i] != lca; i = pre) {

if(base[pre] != lca) father[pre] = match[i]; // 同理

InBlossom[base[i]] = true;

InBlossom[base[match[i]]] = true;

}

#undef pre

if(base[x] != lca) father[x] = y; //注意不能从lca这个奇环的关键点跳回来

if(base[y] != lca) father[y] = x;

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

if(InBlossom[base[i]]) {

base[i] = lca;

if(!inqueue[i]) {

Q[tail++] = i;

inqueue[i] = true; //要注意如果本来连向BFS树中父结点的边是非匹配边的点，

//可能是没有入队的

}

}

}

}

void Change() {

int x, y, z;

z = Finish;

while(z) {

y = father[z];

x = match[y];

match[y] = z;

match[z] = y;

z = x;

}

}

void FindAugmentPath() {

int i;

memset(father, 0, sizeof(father));

memset(inqueue, false, sizeof(inqueue));

for(i = 1; i <= n; i++) base[i] = i;

head = tail = 0;

Q[tail++] = Start;

inqueue[Start] = 1;

while(head < tail) {

int x = Q[head++];

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

if(mp[x][y] && base[x] != base[y] && match[x] != y) { //无意义的边

if(Start == y || match[y] && father[match[y]]) { //精髓地用father表示该点是否

BlossomContract(x, y);

}

else if(!father[y]) {

father[y] = x;

if(match[y]) {

Q[tail++] = match[y];

inqueue[match[y]] = true;

}

else {

Finish = y;

Change();

return;

}

}

}

}

}

}

void Edmonds() {

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

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

if(match[Start] == 0) {

FindAugmentPath();

}

}

}

void output() {

memset(mark, false, sizeof(mark));

int i, cnt = 0;

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

if(match[i]) cnt++; //计算N个点中匹配好的点数

}

// printf("%d\n", cnt); //输出匹配关系

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

// if(!mark[i] && match[i]) {

// mark[i] = true;

// mark[match[i]] = true;

// printf("%d %d\n", i, match[i]);

// }

// }

if(cnt < n) {

printf("NO\n");

}

else {

printf("YES\n");

}

}

一般图最小权匹配

可将求一般图最小权完美匹配的问题转化为求最大权完美匹配的问题，在杜端甫编著的《运筹图论》和龚劬编的《图论与网络最优化算法》中均介绍了求一般图最大权完美匹配的算法，可参考。

## 最大流

EK

//EK O(n\*m^2)

//每次找最短路, 直接调整流量

Dinic

//dinic O(m\*n^2)

const int MOD = 1000000007;

#define Maxn 10000

#define Maxm 100000

struct node {

int u, v, c, next;

}e[Maxm];

int tot, last[Maxn];

int cur[Maxn], dist[Maxn], que[Maxn], sta[Maxn], top, head, tail;

void adde(int u, int v, int c, int c1) {

e[tot].u = u; e[tot].v = v; e[tot].c = c; e[tot].next = last[u]; last[u] = tot++;

e[tot].u = v; e[tot].v = u; e[tot].c = c1; e[tot].next = last[v]; last[v] = tot++;

}

bool bfs(int s, int t, int n) {

int i, j, u, v;

for(i = 0; i < n; i++) dist[i] = MOD;

dist[s] = 0;

head = tail = 0;

que[tail++] = s;

while(head < tail) {

u = que[head++];

for(j = last[u]; j != -1; j = e[j].next) {

if(e[j].c == 0) continue;

v = e[j].v;

if(dist[v] > dist[u] + 1) {

dist[v] = dist[u] + 1;

if(v == t) return true;

que[tail++] = v;

}

}

}

return false;

}

int dinic(int s, int t, int n) {

int i, j, u, v;

int maxflow = 0;

while(bfs(s, t, n)) {

for(i = 0; i < n; i++) cur[i] = last[i];

u = s; top = 0;

while(cur[s] != -1) {

if(u == t) {

int tp = MOD;

for(i = top - 1; i >= 0; i--) {

tp = min(tp, e[sta[i]].c);

}

maxflow += tp;

for(i = top - 1; i >= 0; i--) {

e[sta[i]].c -= tp;

e[sta[i] ^ 1].c += tp;

if(e[sta[i]].c == 0) top = i;

}

u = e[sta[top]].u;

}

else if(cur[u] != -1 && e[cur[u]].c > 0 && dist[u] + 1 == dist[e[cur[u]].v]) {

sta[top++] = cur[u];

u = e[cur[u]].v;

}

else {

while(u != s && cur[u] == -1) {

u = e[sta[--top]].u;

}

cur[u] = e[cur[u]].next;

}

}

}

return maxflow;

}

\*\*“有提交就会有奇迹”的dinic\*\*

dinic是有技巧的，以下是我喜欢的19行版dinic (by 雁过留声)

//这是数组以及结构定义，不计算行数

int level[NMax+2],queue[NMax+2];

long long Min(long long a,long long b){

return a>b?b:a;

}

struct edge{

long long e,f;

edge \*next,\*opt;

}\*E[NMax];

//正文

int mkLevel(){

for (int i=0;i<N;i++)level[i]=-1;

int bot,x;

level[queue[0]=0]=0;bot=1;

for (int top=0;top<bot;top++){x=queue[top];

for (edge \*p=E[x];p;p=p->next)if (p->f && level[p->e]==-1)

level[queue[bot++]=p->e]=level[x]+1;

}

return level[N-1]!=-1;

}

long long extend(int x,long long alpha){

if (x==N-1)return alpha;

long long r,t;r=0;

for (edge \*p=E[x];p;p=p->next)if (p->f && level[p->e]==level[x]+1)

t=extend(p->e,Min(p->f,alpha-r)),p->f-=t,p->opt->f+=t,r+=t;

if (!r)level[x]=-1;

return r;

}

void Dinic(){while (mkLevel())extend(0,(long long)1<<60);}

ISAP

//ISAP O(m\*n^2)

struct node {

int u, v, c, next;

}e[Maxm];

int tot, last[Maxn];

void adde(int u, int v, int c, int c1) {

e[tot].u = u; e[tot].v = v; e[tot].c = c; e[tot].next = last[u]; last[u] = tot++;

e[tot].u = v; e[tot].v = u; e[tot].c = c1; e[tot].next = last[v]; last[v] = tot++;

}

int dist[Maxn], cur[Maxn], gap[Maxn], pre[Maxn];

int ISAP(int s, int t, int n) {

int i, j, u, v, det;

int maxflow = 0;

memset(dist, 0, sizeof(dist[0]) \* (n + 3));

memset(gap, 0, sizeof(gap[0]) \* (n + 3));

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

cur[i] = last[i];

u = s;

gap[0] = n;

pre[s] = -1;

while (dist[s] <= n) {

bool flag = false;

for (j = cur[u]; j != -1; j = e[j].next) {

v = e[j].v;

if (e[j].c > 0 && dist[u] == dist[v] + 1) {

flag = true;

pre[v] = u;

cur[u] = j;

u = v;

break;

}

}

if (flag) {

if (u == t) {

int det = MOD;

for (i = u; i != s; i = pre[i])

det = min(det, e[cur[pre[i]]].c);

for (i = u; i != s; i = pre[i]) {

e[cur[pre[i]]].c -= det;

e[cur[pre[i]] ^ 1].c += det;

}

maxflow += det;

u = s;

}

}

else {

int mind = n;

for (j = last[u]; j != -1; j = e[j].next ) {

v = e[j].v;

if (e[j].c > 0 && dist[v] < mind) {

mind = dist[v];

cur[u] = j;

}

}

if (( -- gap[dist[u]]) == 0) break;

gap[dist[u] = mind + 1] ++ ;

if (u != s) u = pre[u];

}

}

return maxflow;

}

平面图最小割转最短路

## 费用流

spfa

验题: 3680

struct node {

int u, v, c, w, next;

} e[Maxm];

int tot, last[Maxn];

int dist[Maxn], pre[Maxn];

bool visit[Maxn];

queue<int> Q;

#define MOD 0x3f3f3f3f

void adde (int u, int v, int c, int w) {

e[tot].u = u; e[tot].v = v; e[tot].c = c; e[tot].w = w; e[tot].next = last[u]; last[u] = tot++;

e[tot].u = v; e[tot].v = u; e[tot].c = 0; e[tot].w = -w; e[tot].next = last[v]; last[v] = tot++;

}

bool spfa (int s, int t, int n) {

memset (dist, 0x3f, sizeof (dist[0]) \* (n + 3) );

memset (visit, 0, sizeof (visit[0]) \* (n + 3) );

memset (pre, -1, sizeof (pre[0]) \* (n + 3) );

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

Q.push (s);

visit[s] = true;

dist[s] = 0;

pre[s] = -1;

while (!Q.empty() ) {

int u = Q.front();

visit[u] =false;

Q.pop();

for (int j = last[u]; j != -1; j = e[j].next)

if (e[j].c > 0 && dist[u] + e[j].w < dist[e[j].v]) {

dist[e[j].v] = dist[u] + e[j].w;

pre[e[j].v] = j;

if (!visit[e[j].v]) {

Q.push (e[j].v);

visit[e[j].v] = true;

}

}

}

if (dist[t] == MOD) return false;

else return true;

}

int ChangeFlow (int t) {

int det = MOD, u = t;

while (~pre[u]) {

u = pre[u];

det = min (det, e[u].c);

u = e[u].u;

}

u = t;

while (~pre[u]) {

u = pre[u];

e[u].c -= det;

e[u ^ 1].c += det;

u = e[u].u;

}

return det;

}

int MinCostFlow (int s, int t, int n) {

int mincost, maxflow;

mincost = maxflow = 0;

while (spfa (s, t, n) ) {

int det = ChangeFlow (t);

mincost += det \* dist[t];

maxflow += det;

}

return mincost;

}

zkw

验题: 3680

朴素zkw(存在bug)

//没有优化, 但是存在找到第一条可行流而计算不到费用的情况

//不可以用了!!!

#define MOD 0x3f3f3f3f

int flow, cost, value;

int dist[Maxn], visit[Maxn], src, des;

deque<int> Q;

void adde(int u, int v, int c, int w) {

e[tot].u = u; e[tot].v = v; e[tot].c = c; e[tot].w = w; e[tot].next = last[u]; last[u] = tot++;

e[tot].u = v; e[tot].v = u; e[tot].c = 0; e[tot].w = -w; e[tot].next = last[v]; last[v] = tot++;

}

int Aug(int u, int m) {

if(u == des) {

cost += value \* m;

flow += m;

return m;

}

visit[u] = true;

int l = m;

int j, v, c, w;

for(j = last[u]; j != -1; j = e[j].next) {

v = e[j].v; c = e[j].c; w = e[j].w;

if(c && !w && !visit[v]) {

int del = Aug(v, l < c ? l : c);

e[j].c -= del; e[j ^ 1].c += del; l -= del;

if(!l) return m;

}

}

return m - l;

}

bool Modlabel(int src, int des, int n) {

int i, j, u, v, c, w, del;

del = MOD;

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

if(visit[i]) {

for(j = last[i]; j != -1; j = e[j].next) {

if(e[j].c && !visit[e[j].v] && e[j].w < del) del = e[j].w;

}

}

}

if(del == MOD) return false;

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

if(visit[i]) {

for(j = last[i]; j != -1; j = e[j].next) {

e[j].w -= del; e[j ^ 1].w += del;

}

}

}

value += dist[des];

return true;

}

void zkw(int src, int des, int n) {

value = cost = flow = 0;

do {

do {

memset(visit, 0, sizeof(visit[0]) \* (n + 3));

}while(Aug(src, MOD));

}while(Modlabel(src, des, n));

}

zkw+deque

#define MOD 0x3f3f3f3f

int flow, cost, value;

int dist[Maxn], visit[Maxn], src, des;

deque<int> Q;

void adde(int u, int v, int c, int w) {

e[tot].u = u; e[tot].v = v; e[tot].c = c; e[tot].w = w; e[tot].next = last[u]; last[u] = tot++;

e[tot].u = v; e[tot].v = u; e[tot].c = 0; e[tot].w = -w; e[tot].next = last[v]; last[v] = tot++;

}

int Aug(int u, int m) {

if(u == des) {

cost += value \* m;

flow += m;

return m;

}

visit[u] = true;

int l = m;

int j, v, c, w;

for(j = last[u]; j != -1; j = e[j].next) {

v = e[j].v; c = e[j].c; w = e[j].w;

if(c && !w && !visit[v]) {

int del = Aug(v, l < c ? l : c);

e[j].c -= del; e[j ^ 1].c += del; l -= del;

if(!l) return m;

}

}

return m - l;

}

bool Modlabel(int src, int des, int n) {

int i, j, u, v, c, w, del;

memset(dist, 0x3f, sizeof(dist[0])\*(n + 3));

dist[src] = 0;

while(!Q.empty()) Q.pop\_back();

Q.push\_back(src);

while(!Q.empty()) {

u = Q.front(); Q.pop\_front();

for(j = last[u]; j != -1; j = e[j].next) {

v = e[j].v; c = e[j].c; w = e[j].w;

if(c && (del = dist[u] + w) < dist[v]) {

dist[v] = del;

if(Q.empty() || del <= dist[Q.front()]) {

Q.push\_front(v);

}

else {

Q.push\_back(v);

}

}

}

}

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

for(j = last[i]; j != -1; j = e[j].next) {

e[j].w -= dist[e[j].v] - dist[i];

}

}

value += dist[des];

return dist[des] < MOD;

}

void zkw(int src, int des, int n) {

value = cost = flow = 0;

while(Modlabel(src, des, n)){

do {

memset(visit, 0, sizeof(visit[0]) \* (n + 3));

}while(Aug(src, MOD));

}

}

zkw+priority\_queue

#define MOD 0x3f3f3f3f

int flow, cost, value;

int dist[Maxn], visit[Maxn], src, des;

priority\_queue<pair<int, int> > Q;

void adde(int u, int v, int c, int w) {

e[tot].u = u; e[tot].v = v; e[tot].c = c; e[tot].w = w; e[tot].next = last[u]; last[u] = tot++;

e[tot].u = v; e[tot].v = u; e[tot].c = 0; e[tot].w = -w; e[tot].next = last[v]; last[v] = tot++;

}

int Aug(int u, int m) {

if(u == des) {

cost += value \* m;

flow += m;

return m;

}

visit[u] = true;

int l = m;

int j, v, c, w;

for(j = last[u]; j != -1; j = e[j].next) {

v = e[j].v; c = e[j].c; w = e[j].w;

if(c && !w && !visit[v]) {

int del = Aug(v, l < c ? l : c);

e[j].c -= del; e[j ^ 1].c += del; l -= del;

if(!l) return m;

}

}

return m - l;

}

bool Modlabel(int src, int des, int n) {

int i, j, u, v, c, w, del, d;

memset(dist, 0x3f, sizeof(dist[0])\*(n + 3));

dist[src] = 0;

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

Q.push(MP(0, src));

while(!Q.empty()) {

u = Q.top().BB;

d = -Q.top().AA;

Q.pop();

if(d != dist[u]) continue;

for(j = last[u]; j != -1; j = e[j].next) {

v = e[j].v; c = e[j].c; w = e[j].w;

if(c && (del = d + w) < dist[v]) {

dist[v] = del;

Q.push(MP(-del, v));

}

}

}

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

for(j = last[i]; j != -1; j = e[j].next) {

e[j].w -= dist[e[j].v] - dist[i];

}

}

value += dist[des];

return dist[des] < MOD;

}

void zkw(int src, int des, int n) {

value = cost = flow = 0;

while(Modlabel(src, des, n)){

do {

memset(visit, 0, sizeof(visit[0]) \* (n + 3));

}while(Aug(src, MOD));

}

}

zkw+queue

//zkw queue实现

#define MOD 0x3f3f3f3f

int flow, cost, value;

int dist[Maxn], visit[Maxn], src, des;

queue<int> Q;

void adde(int u, int v, int c, int w) {

e[tot].u = u; e[tot].v = v; e[tot].c = c; e[tot].w = w; e[tot].next = last[u]; last[u] = tot++;

e[tot].u = v; e[tot].v = u; e[tot].c = 0; e[tot].w = -w; e[tot].next = last[v]; last[v] = tot++;

}

int Aug(int u, int m) {

if(u == des) {

cost += value \* m;

flow += m;

return m;

}

visit[u] = true;

int l = m;

int j, v, c, w;

for(j = last[u]; j != -1; j = e[j].next) {

v = e[j].v; c = e[j].c; w = e[j].w;

if(c && !w && !visit[v]) {

int del = Aug(v, l < c ? l : c);

e[j].c -= del; e[j ^ 1].c += del; l -= del;

if(!l) return m;

}

}

return m - l;

}

bool Modlabel(int src, int des, int n) {

int i, j, u, v, c, w, del;

memset(dist, 0x3f, sizeof(dist[0])\*(n + 3));

dist[src] = 0;

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

Q.push(src);

while(!Q.empty()) {

u = Q.front(); Q.pop();

for(j = last[u]; j != -1; j = e[j].next) {

v = e[j].v; c = e[j].c; w = e[j].w;

if(c && (del = dist[u] + w) < dist[v]) {

dist[v] = del;

Q.push(v);

}

}

}

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

for(j = last[i]; j != -1; j = e[j].next) {

e[j].w -= dist[e[j].v] - dist[i];

}

}

value += dist[des];

return dist[des] < MOD;

}

void zkw(int src, int des, int n) {

value = cost = flow = 0;

while(Modlabel(src, des, n)) {

do {

memset(visit, 0, sizeof(visit[0]) \* (n + 3));

} while(Aug(src, MOD));

}

}

对于poj3680效率比较:

nocow上zkw板块的作者代码可以达到360ms

最快是 zkw + deque 860ms

其次是 zkw + priority\_queue 1250ms

然后是 zkw + queue 2094ms

对于普通spfa 2407ms

另外, 手写que需要注意, 一个点可能入队不止一次.

## 全局最小割

StoerWagner(求全局最小割)

验题:POJ2914

//朴素最小割

//注意节点下标0~n-1

int comb[Maxn];

int edge[Maxn][Maxn], g[Maxn][Maxn], node[Maxn];

int S, T, minCut;

int top, sta[Maxn];

int maxi;

vector<int> parta, partb;

vector<int> belong[Maxn];

int Search (int n) {

int i, j, u;

int vis[Maxn], wet[Maxn];

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

memset(wet, 0, sizeof(wet));

int minCut = 0, temp = -1, top = 0;

int maxi;

S = -1, T = -1;

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

maxi = -MOD;

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

u = node[j];

if (!comb[u] && !vis[u] && wet[u] > maxi) {

temp = u;

maxi = wet[u];

}

}

sta[top++] = temp;

vis[temp] = true;

if (i == n - 1)

minCut = maxi;

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

u = node[j];

if (!comb[u] && !vis[u]) {

wet[u] += edge[temp][u];

}

}

}

S = sta[top - 2];

T = sta[top - 1];

for (i = 0; i < top; i++) node[i] = sta[i];

return minCut;

}

int StoerWagner (int n) {

int ans = MOD, i, j, cur;

memset(comb, 0, sizeof(comb));

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

node[i] = i;

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

k = n - i + 1;

cur = Search (k);

if (cur < ans) {

ans = cur;

}

if (ans == 0) return ans;

comb[T] = true;

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

if (j == S) continue;

if (!comb[j]) {

edge[S][j] += edge[T][j];

edge[j][S] += edge[j][T];

}

}

}

return ans;

}

int n, m;

int main() {

int i, j, u, v, w;

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

memset(edge, 0, sizeof(edge));

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

int u, v, z;

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

edge[u][v] += z;

edge[v][u] += z;

}

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

}

return 0;

}

K连通块计数

HDU 4654

int n, m, k;

//K连通块计数

//注意节点下标0~n-1

int comb[Maxn];

int edge[Maxn][Maxn], g[Maxn][Maxn], node[Maxn];

int S, T, minCut;

int top, sta[Maxn];

int maxi;

vector<int> parta, partb;

vector<int> belong[Maxn];

int Search (int n) {

int i, j, u;

int vis[Maxn], wet[Maxn];

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

memset(wet, 0, sizeof(wet));

int minCut = 0, temp = -1, top = 0;

int maxi;

S = -1, T = -1;

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

maxi = -MOD;

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

u = node[j];

if (!comb[u] && !vis[u] && wet[u] > maxi) {

temp = u;

maxi = wet[u];

}

}

sta[top++] = temp;

vis[temp] = true;

if (i == n - 1)

minCut = maxi;

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

u = node[j];

if (!comb[u] && !vis[u]) {

wet[u] += edge[temp][u];

}

}

}

S = sta[top - 2];

T = sta[top - 1];

for (i = 0; i < top; i++) node[i] = sta[i];

return minCut;

}

int StoerWagner (vector<int> & li) {

int ans = MOD, i, j, k,cur, n = li.SZ, u, v;

int used[Maxn];

memset(comb, 0, sizeof(comb));

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

node[i] = i;

belong[i].clear();

belong[i].PB(i);

}

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

k = n - i + 1;

cur = Search(k);

if (cur < ans) {

ans = cur;

for(j = 0; j < n; j++) used[j] = 0;

for(j = 0; j < belong[T].SZ; j++) {

used[belong[T][j]] = 1;

}

}

for(j = 0; j < belong[T].SZ; j++) belong[S].PB(belong[T][j]);

if (ans == 0) break;

comb[T] = true;

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

if (j == S) continue;

if (!comb[j]) {

edge[S][j] += edge[T][j];

edge[j][S] += edge[j][T];

}

}

}

parta.clear(); partb.clear();

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

if(used[j]) parta.PB(li[j]);

else partb.PB(li[j]);

}

return ans;

}

int dfs(vector<int> &li) {

int n = li.SZ, i, j;

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

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

edge[i][j] = g[li[i]][li[j]];

}

}

int cur = StoerWagner(li);

if(cur >= k) return 1;

vector<int> a(parta), b(partb);

return dfs(a) + dfs(b);

}

int main() {

int i, j, u, v, w;

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

memset(g, 0, sizeof(g));

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

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

u--; v--;

g[u][v] += 1;

g[v][u] += 1;

}

vector<int> li;

for(i = 0; i < n; i++) li.PB(i);

printf ("%d\n", dfs(li));

}

return 0;

}

## 网络流拓展

常见建图方式

B[u,v]表示(u,v)流量的下限，C[u,v]表示(u,v)流量的上限, F[u,v]表示(u,v)的流量,

g[u,v]表示F[u,v]-B[u,v] 显然 0<=g[u,v]<=C[u,v]-B[u,v]

1.无源汇的可行流 :

我们要想办法转换为有源汇的最大流问题.

考虑流量都为g[,]且容量为C[,]-B[,]的网络，貌似有点接近最后的转换方式了，

为了不忽略B[,]这一条件，我们把g[,]最后强制加上B[,].

但会发现一个致命漏洞，加上后就未必满足流量平衡了！

对于这个有两种办法解决..

一种方法是 添加附加源汇S,T 对于某点 u, 设 M(u)=sigma(B[i,u])-sigma(B[u,j]) ，

则根据流量平衡条件有 M(u)同时等于 sigma(g[u,j])-sigma(g[i,u])

若M(u)<0，即sigma(g[u,j]) < sigma(g[i,u]) 进入u的流量比从u 出去的多，

所以 u -> T 连容量为 -(sigma(B[i,u])-sigma(B[u,j]) ) 的边

同理. M(u)>0时，即 S->u 连容量为 sigma(B[i,u])-sigma(B[u,j]) 的边.

然后再 对于任意边(i,u)/(u,j) 连一条 C[u,v]-B[u,v]的边.

这样 只需对新的网络求一遍最大流即可. 若出附加源点的边都满流即是存在可行流，反之不然.

满流的必要条件是显然的. 不满流不能保证加上B[,]后流量平衡. 前面都白费了.

另一种方法相对简单.其实类似，本质相同.

仍添加附加源汇S,T 对于某边 (u,v) 在新网络中连边

S->v 容量 B[u,v] , u->T 容量 B[u,v] ， u->v 容量 C[u,v]-B[u,v]

可以这样理解，边S->v : 求的时候直接从S流过来的流量值B[u,v], 与最终解中边(u,v)强制加上的从 u流过来的流量B[u,v]，对v点的流量平衡条件的影响 实质等价.

边u->T同理.

最后，一样也是求一下新网络的最大流，判断从附加源点的边，是否都满流即可.

具体的解？根据最前面提出的强制转换方式，边(u,v)的最终解中的实际流量即为g[u,v]+B[u,v]

为什么这种方法只适用于无源汇上下界可行流？

本质上是因为S,T并不满足流量平衡，而上述的方法都是考虑到每点的流量平衡而建的. 但有些时候貌似还是可以出正确解. 至于有没有什么解决方法，下次再想想吧~【标记下】

例题 ZOJ 2314 / SGU 194 Reactor Cooling http://acm.sgu.ru/problem.php?contest=0&problem=194

2.有源汇的上下界可行流

从汇点到源点连一条上限为INF，下限为0的边. 按照 1.无源汇的上下界可行流 一样做即可.

改成无源汇后，求的可行流是类似环的，流量即T->S边上的流量. 这样做显然使S,T也变得流量平衡了.

3.有源汇的上下界最大流

方法一 ： 2.有源汇上下界可行流中，从汇点到源点的边改为连一条上限为INF，下限为x的边.

因为显然x>ans即MIN(T->S )> MAX(S->T) ,会使求新网络的无源汇可行流无解的（S,T流量怎样都不能平衡）

而x<=ans会有解.

所以满足二分性质，二分x，最大的x使得新网络有解的即是所求答案原图最大流.

方法二：从汇点T到源点S连一条上限为INF，下限为0的边，变成无源汇的网络. 照求无源汇可行流的方法(如1)，建附加源点S'与汇点T'，求一遍S'->T‘的最大流. 再把从汇点T到源点S的这条边拆掉 . 求一次从S 到T 的最大流即可. （关于S',T'的边好像可以不拆？）（这样一定满足流量平衡？）表示这方法我也没有怎么理解.

4.有源汇的上下界最小流

方法一： 2.有源汇上下界可行流中，从汇点到源点的边改为连一条上限为x，下限为0的边.

与3同理，二分上限，最小的x使新网络无源汇可行流有解，即是所求答案原图最小流.

方法二: 照求无源汇可行流的方法(如1)，建附加源点S'与汇点T'，求一遍S'->T‘的最大流. 但是注意这一遍不加汇点到源点的这条边，即不使之改为无源汇的网络去求解. 求完后，再加上那条汇点到源点上限INF的边. 因为这条边下限为0，所以S',T'无影响. 再直接求一遍S'->T'的最大流. 若S’出去的边全满流，T->S边上的流量即为答案原图最小流，否则若不全满流即无解.

和求3.有源汇的上下界最大流过程相反，感性理解是:

首先明确，我们的方法是通过加边转化成对任一点都有流量平衡的无源汇的网络，进行求解.

即最终解只能是加上边后，求的无源汇可行流，即T->S这边上的流量. 不改成无源汇的直接求的解是未必正确的，在（1）中已经提到.

然后，因为第一遍做的时候并无这条边，所以S->T的流量在第一遍做的时候都已经尽力往其他边流了. 于是加上T->S这条边后，都是些剩余的流不到其他边的流量. 从而达到尽可能减少T->S这边上的流量的效果，即减小了最终答案.

感觉上第一遍做的既然是不改成无源汇直接求的，应该是错误的？

这里不是错误的. 首先我们的解都是按照第二遍所求的而定，其次这里这样做本质是延迟对T->S这条边的增流.

## 最小树形图(有向图的最小生成树)

基础知识

hdu4009 hdu2121 poj3164 UVa11865

本题为不是固定根的最小树形图，我们可以虚拟出一根来，然后在把这个根跟每个点相连，相连的点可以设为无穷大，或者设为所有边和大一点，比如为r，然后就可以利用最小树形图进行计算了，计算出的结果减去r,如果比r还大就可以认为通过这个虚拟节点我们连过原图中两个点，即原图是不连通的，我们就可以认为不存在最小树形图。关于输出最小根也挺简单，在找最小入弧时，如果这条弧的起点是虚拟根，那么这条弧的终点就是要求的根

朱刘算法

验题:POJ3164 HDU2121

/\*

\* 最小树形图（根固定） O(VE)

\* 有向图最小生成树

\* 根不固定，添加一个根节点与所有点连无穷大的边！

\* 如果求出比2\*MOD大, 则不连通; 求根, 则求和虚拟根相连的结点

\* 根据pre的信息能构造出这棵树！

\* 注意结点必须从0~n-1, 因为要考虑重新标号建图的统一

\* mytype 根据实际情况确定

\*/

typedef int mytype;

int visit[Maxn], pre[Maxn], belong[Maxn], ROOT;

mytype inv[Maxn];

mytype dirtree(int n, int m, int root) {

mytype sum = 0;

int i, j, k, u, v;

while (1) {

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

inv[i] = MOD;

pre[i] = -1;

belong[i] = -1;

visit[i] = -1;

}

inv[root] = 0;

for (i = 0; i < m; i++) {//除原点外，找每个点的最小入边

u = e[i].u; v = e[i].v;

if (u != v) {

if (e[i].w < inv[v]) {

inv[v] = e[i].w;

pre[v] = u;

if(u == root) ROOT = i; //记录根所在的边

//输出根时利用ROOT-m计算是原图哪个结点

}

}

}

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

if (inv[i] == MOD) return -1;

}

int num = 0;

for (i = 0; i < n; i++) { //找圈，收缩圈

if (visit[i] == -1) {

j = i;

for(j = i; j != -1 && visit[j] == -1 && j != root; j = pre[j]) {

visit[j] = i;

}

if (j != -1 && visit[j] == i) {

for (k = pre[j]; k != j; k = pre[k]) {

belong[k] = num;

}

belong[j] = num ++ ;

}

}

sum += inv[i];

}

if (num == 0) return sum;

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

if (belong[i] == -1) {

belong[i] = num ++ ;

}

}

for (i = 0; i < m; i++) { //重新构图

e[i].w = e[i].w - inv[e[i].v];

e[i].v = belong[e[i].v];

e[i].u = belong[e[i].u];

}

n = num;

root = belong[root];

}

}

## 树的Hash与同构

树Hash判定树同构

int h[11000];

char str1[3100], str2[3100];

char \*p;

int Hash (int j) {

int sum = h[j + 5000];//这里的j是记录的节点度

//这个巧妙的循环,把子节点的hash值都加给了父节点,作为父节点的hash值

//由于树节点顺序不确定, 因此是子树hash值\*根值的累加

while (\*p && \*p++ == '0') {

sum = (sum + h[j] \* Hash (j + 1) ) % 19001;

}

return (sum \* sum) % 19001;

}

inline void init() {

//为每一个树根节点给定一个随机权值

for (int i = 0; i < 10000; i++)

h[i] = (rand() %19901);

}

int main() {

int T;

scanf ("%d", &T);

init();

while (T--) {

scanf ("%s%s", str1, str2);

p = str1;

int a = Hash (1);

p = str2;

int b = Hash (1);

if (a == b) {

puts ("same");

} else {

puts ("different");

}

}

return 0;

}

## 差分约束

## 最大团

## 最大团

极大团计数

# **字符串**

## **Hash**

### 一般字符串Hash

LL strhash (char \*str) {

int i;

LL c, ret = 0;

for(i= 0, ret = 0; str[i]; i++) {

c = str[i] - 'a';

ret = ((ret << 5) + ret) ^ c;

}

return ret;

}

17,37, 79, 163, 331,

673,1361,2729,5471,10949,

21911,43853, 87719, 175447, 350899,

701819,1403641,2807303,5614657, 11229331,

22458671, 44917381, 89834777, 179669557, 359339171,

718678369, 1437356741, 2147483647

### 字符串区间Hash

//prePow预处理MOD^x的值, 可以在程序最开始做!

prePow[0] = 1;

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

prePow[i] = prePow[i - 1] \* MOD;

}

//存储在H数组中, 利用溢出进行Hash

void preHash(int len, char str[], LL H[]) {

int i;

H[0] = str[0];

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

H[i] = (H[i - 1] \* MOD + str[i]);

}

}

//求区间[l,r]的Hash值

LL HASH(int l, int r, LL H[], LL prePow[]) {[l,r], base 0

LL ret = 0;

if(l) ret = H[l - 1] \* prePow[r - l + 1];

return (H[r] - ret);

}

## KMP

### 函数版

char S[Maxn], T[Maxn];

int next[Maxn], is[Maxn];

void getnext(char T[], int LT, int next[]) {

int i, j;

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

for (i = 1, j = 0; i < LT; ) {

while (j != -1 && T[i] != T[j]) j = next[j];

i++; j++;

next[i] = j;

}

}

void KMP (char S[], int LS, char T[], int LT, int next[]) {

int i, j;

for(i = 0; i < LS; i++) is[i] = 0;

for (i = 0, j = 0; i < LS; i++) {

while (j != -1 && S[i] != T[j]) j = next[j];

j++;

if (j == LT) {

is[i - LT + 1] = 1;

j = next[j];

}

}

}

### KMP类

struct KMP {//调用init(s(待匹配串),t(模式串));适用多次KMP

char S[Maxn], T[Maxn];

int next[Maxn], is[Maxn];

int LT, LS;

void init(char s[], char t[]) {

int i;

LT = strlen(t);

LS = strlen(s);

for(i = 0; i <= LT; i++) T[i] = t[i];

for(i = 0; i <= LS; i++) S[i] = s[i];

}

void getnext() {

int i, j;

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

for (i = 1, j = 0; i < LT; ) {

while (j != -1 && T[i] != T[j]) j = next[j];

i++; j++;

next[i] = j;

}

}

void kmp() {

int i, j;

for(i = 0; i < LS; i++) is[i] = 0;

for (i = 0, j = 0; i < LS; i++) {

while (j != -1 && S[i] != T[j]) j = next[j];

j++;

if (j == LT) {

is[i - LT + 1] = 1;

j = next[j];

}

}

}

}kmp;

## 拓展KMP

### 函数版

char S[Maxn], T[Maxn];

int next[Maxn], B[Maxn];

void preExKmp(char T[], int LT, int next[]) {

int i, ind = 0, k = 1;

next[0] = LT;

while(ind + 1 < LT && T[ind + 1] == T[ind]) ind++;

next[1] = ind;

for(i = 2; i < LT; i++) {

if(i <= k + next[k] - 1 && next[i - k] + i < k + next[k])

next[i] = next[i - k];

else {

ind = max(0, k + next[k] - i);

while(ind + i < LT && T[ind + i] == T[ind]) ind++;

next[i] = ind; k = i;

}

}

}

void exKmp(char S[], int LS, char T[], int LT, int next[], int B[]) {

int i, ind = 0, k = 0;

preExKmp(T, LT, next);

while(ind < LS && ind < LT && T[ind] == S[ind]) ind++;

B[0] = ind;

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

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

if((i - 1) + L < p)

B[i] = L;

else {

ind = max(0, p - i + 1);

while(ind + i < LS && ind < LT && S[ind + i] == T[ind]) ind++;

B[i] = ind;

k = i;

}

}

}

## Manacher

### Manacher模板

//HDU\_3068

//s为原串, str为插入$和#的串

//读入s后, 调用init(s, str, len),

//最后调用Manacher(str,p,len)

//求解遍历p数组求最大值, 注意输出ans-1

//最长回文子串对应原串T中的位置：l = (i - p[i])/2; r = (i + p[i])/2 - 2;

int len, p[Maxn];

char s[Maxn], str[Maxn];

void init(char s[], char str[], int& len) {

int i, j, k;

str[0] = '$';

str[1] = '#';

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

str[i \* 2 + 2] = s[i];

str[i \* 2 + 3] = '#';

}

len = len \* 2 + 2;

s[len] = 0;

}

void Manacher (char str[], int p[], int len) {

int i;

int mx = 0;

int id;

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

str[i] = 0; //没有这一句有问题。。就过不了ural1297，比如数据：ababa aba

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

if ( mx > i )

p[i] = min ( p[2 \* id - i], p[id] + id - i );

else

p[i] = 1;

for (; str[i + p[i]] == str[i - p[i]]; p[i]++)

;

if ( p[i] + i > mx ) {

mx = p[i] + i;

id = i;

}

}

}

int main() {

int i, ans;

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

len = strlen (s);

init(s, str, len);

Manacher(str, p, len);

ans = 0;

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

if (p[i] > ans)

ans = p[i];

printf ("%d\n", ans - 1);

}

return 0;

}

## AC自动机

### 普通匹配

struct node {

struct node \* fail;

struct node \* next[10];

bool is;

int lab;

}Tree[Maxn], \* root, \* que[Maxn];

int tot, head, tail;

node \* newNode() {

node \* p = &Tree[tot++];

p->is = false;

p->lab = 0;

p->fail = NULL;

for(int i = 0; i < 10; i++) p->next[i] = NULL;

return p;

}

void Insert(char str[], int n) {

node \* p = root;

for(int i = 0, k; str[i]; i++) {

k = str[i] - '0';

if(p->next[k] == NULL) {

p->next[k] = newNode();

}

p = p->next[k];

}

p->is = true;

p->lab = n;

}

void buildAC() {

head = tail = 0;

int i;

node \* p, \* q;

root->fail = root;

que[tail++] = root;

while(head < tail) {

p = que[head++]; q = p->fail;

for(i = 0; i < 10; i++) {

if(p->next[i] != NULL) {

if(p == root) p->next[i]->fail = root;

else {

p->next[i]->fail = q->next[i];

p->next[i]->is |= q->next[i]->is;

}

que[tail++] = p->next[i];

}

else {

if(p == root) p->next[i] = root;

else p->next[i] = q->next[i];

}

}

}

}

void query(char str[]) {

node \* p , \* q;

p = root;

for(int i = 0, k; str[i]; i++) {

k = str[i]-'0';

p = p->next[k];

if(p->is) {

q = p;

while(q->is) {

cnt[q->lab]++;

q = q->fail;

}

}

}

}

### AC自动机+DP

struct node {

struct node \* next[26];

struct node \* fail;

int is;

int v;

}A[Maxn], \*root, \*que[Maxn];

int head, tail;

char str[Maxn];

int tot, ans, n;

node \* newNode() {

node \* p = &A[tot++];

p->fail = NULL;

p->is = false;

p->v = -MOD;

for(int i = 0; i <26; i++) p->next[i] = NULL;

return p;

}

void Insert(node \* root, char str[], int l, int r) {

int i, k;

for(i = l; i < r; i++) {

k = str[i] - 'a';

if(root->next[k] == NULL) root->next[k] = newNode();

root = root->next[k];

}

root->is = 1;

}

void buildAC() {

int k;

head = tail = 0;

que[tail++] = root;

root->fail = root;

node \* p, \* q;

while(head < tail) {

p = que[head++]; q = p->fail;

for(k = 0; k < 26; k++) {

if(p->next[k] != NULL) {

if(p == root) p->next[k]->fail = root;

else {

p->next[k]->fail = q->next[k];

p->next[k]->is |= q->next[k]->is;

}

que[tail++] = p->next[k];

}

else {

if(p == root) p->next[k] = root;

else p->next[k] = q->next[k];

}

}

}

}

void query(int n, int l, int r, int v0) {

int i, k, v;

node \* p, \* q;

p = root;

v = v0;

for(i = l; i < r; i++) {

k = str[i] - 'a';

p = p->next[k];

if(p->is) {

q = p;

while(q->is) {

cmax(v, v0 + q->v);

q = q->fail;

}

}

}

cmax(p->v, v);

cmax(ans, v);

}

## 后缀数组

### 论文模板(O(nlogn))

论文模板, 使用时注意num[]有效位为0~n-1, 但是需要将num[n]=0, 否则RE;另外, 对于模板的处理将空串也处理了, 作为rank最小的串, 因此有效串为0~n共, n-1个, 在调用da()函数时, 需要调用da(num, n + 1, m); 对于sa[], rank[]和height[]数组都将空串考虑在内, 作为rank最小的后缀!

//O(nlogn)

//调用da(num, len+1, m);//m为字符个数略大

int len;

int num[Maxn]; //待处理的串

int sa[Maxn], rank[Maxn], height[Maxn]; //sa[1~n]value(0~n-1); rank[0..n-1]value(1..n); height[2..n]

int wa[Maxn], wb[Maxn], wv[Maxn], wd[Maxn];

int cmp(int \*r, int a, int b, int x) {

return r[a] == r[b] && r[a + x] == r[b + x];

}

void da(int \*r, int n, int m) { // 倍增算法 r为待匹配数组 n为总长度+1 m为字符范围

int i, j, k, p, \*x = wa, \*y = wb, \*t;

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

for(i = 0; i < n; i++) wd[x[i] = r[i]]++;

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

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

for(j = 1, p = 1; p < n; j <<= 1, m = p) {

for(p = 0, i = n - j; i < n; i++) y[p++] = i;

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

for(i = 0; i < n; i++) wv[i] = x[y[i]];

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

for(i = 0; i < n; i++) wd[wv[i]]++;

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

for(i = n - 1; i >= 0; i--) sa[--wd[wv[i]]] = y[i];

for(t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++) {

x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;

}

}

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

for(i = 0; i < n - 1; height[rank[i++]] = k) {

for(k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++);

}

}

## 字符串最小表示法

### 朴素最小表示法

ZOJ2006 ZOJ1729 HDU 3374

int MinRep (char S[], int L) {

int i = 0, j = 1, k = 0, t;

while (i < L && j < L && k < L) { //找不到比它还小的或者完全匹配

t = S[ (i + k) % L] - S[ (j + k) % L];

// t = s[(i + k) >= L ? i + k - L : i + k]

// - s[(j + k) >= L ? j + k - L : j + k];

if (t == 0)

k++;//相等的话,检测长度加1

else {

//大于的话,s[i]为首的肯定不是最小表示,最大表示就改<

if (t > 0) i += k + 1;

else j += k + 1;

if (i == j) j++;

k = 0;

}

}

return min (i, j);

}

### 判定两串是否循环同构