**A**

**C**

**M**

**模**

**板**

**-------lsy**

目录

[赛场注意事项 3](#_Toc374118018)

[1. 测试机器 3](#_Toc374118019)

[1.1 鼠标键盘测试 3](#_Toc374118020)

[1.2 软件测试是否可用 3](#_Toc374118021)

[1.3 内存大小 3](#_Toc374118022)

[2. 基本设定 3](#_Toc374118023)

[3. ASCII 4](#_Toc374118024)

[图论 9](#_Toc374118025)

[1. ShortestPath 9](#_Toc374118026)

[1.1 SPFA 9](#_Toc374118027)

[1.2 Johnson 9](#_Toc374118028)

[2. MinimumSpanningTree 10](#_Toc374118029)

[2.1 Prim 10](#_Toc374118030)

[2.2 Kruskal 10](#_Toc374118031)

[3. LCA 10](#_Toc374118032)

[3.1 倍增算法 10](#_Toc374118033)

[4. 拓扑排序 12](#_Toc374118034)

[4.1 拓扑排序 12](#_Toc374118035)

[4.2 拓扑排序数 13](#_Toc374118036)

[4.3 关键路径 13](#_Toc374118037)

[5. 连通性 15](#_Toc374118038)

[5.1 StrongConnected 15](#_Toc374118039)

[5.2 EdgeConnected 17](#_Toc374118040)

[5.3 PointConnected 18](#_Toc374118041)

[6. 2-SAT 20](#_Toc374118042)

[6.1 判断可行性 20](#_Toc374118043)

[6.2 求任意解 21](#_Toc374118044)

[6.3 求字典序最小的解 23](#_Toc374118045)

[7. 全局最小割 23](#_Toc374118046)

[7.1 全局最小割 23](#_Toc374118047)

[7.2 求最小割边 25](#_Toc374118048)

[8. 匹配问题 28](#_Toc374118049)

[8.1 最大匹配 28](#_Toc374118050)

[8.2相关概念 31](#_Toc374118051)

[8.3 任意图最大匹配(带花树) 31](#_Toc374118052)

[9. 最小环 – 最小环 34](#_Toc374118053)

[10. 欧拉回路 35](#_Toc374118054)

[11. 最大流 35](#_Toc374118055)

[11.1 EK 35](#_Toc374118056)

[11.2 Dinic 36](#_Toc374118057)

[11.3 ISAP 38](#_Toc374118058)

[12. RMQ 39](#_Toc374118059)

[13. 最小树形图 39](#_Toc374118060)

[14. 费用流 41](#_Toc374118061)

[14.1 Spfa 41](#_Toc374118062)

[14.2 ZKW 42](#_Toc374118063)

[15. 树的同构 44](#_Toc374118064)

# 赛场注意事项

## 1. 测试机器

### 1.1 鼠标键盘测试

### 1.2 软件测试是否可用

### 1.3 内存大小

最大数组

栈大小

扩栈测试

## 2. 基本设定

codeblocks

terminal： gnome-terminal -t $TITLE -x 在codeblocks --> setting --> 环境变量

eclipse

Eclipse中默认是输入"."后出现自动提示，用于类成员的自动提示，可是有时候我们希望它能在我们输入类的首字母后就出现自动提示，可以节省大量的输入时间（虽然按alt + /会出现提示，但还是要多按一次按键，太麻烦了）。    从Window -> preferences -> Java -> Editor -> Content assist -> Auto-Activation下，我们可以在"."号后面加入我们需要自动提示的首字幕，比如"ahiz"。    然后我们回到Eclipse的开发环境，输入"a"，提示就出现了。但是我们可以发现，这个Auto-Activation下的输入框里最多只能输入5个字母，也许是Eclipse的开发人员担心我们输入的太多会影响性能，但计算机的性能不用白不用，所以我们要打破这个限制。其实上面都是铺垫，制造一下气氛，以显得我们下面要做的事情很牛似的，其实不然，一切都很简单。嘿嘿 :)

在"."后面随便输入几个字符，比如"abij"，然后回到开发环境，File -> export -> general -> preferences -> 选一个地方保存你的首选项，比如C:"a.epf用任何文本编辑器打开a.epf，查找字符串“abij”，找到以后，替换成“abcdefghijklmnopqrstuvwxyz”，总之就是你想怎样就怎样！！然后回到Eclipse，File -> import -> general -> preferences -> 导入刚才的a.epf文件。此时你会发现输入任何字幕都可以得到自动提示了。爽！！！最后：自动提示弹出的时间最好改成100毫秒以下，这样会比较爽一点，不然你都完事了，自动提示才弹出来:)，不过也要看机器性能。

FileWriter fileWriter=new FileWriter("c:\\Result.txt");

int [] a=new int[]{11112,222,333,444,555,666};

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

fileWriter.write(String.valueOf(a[i])+" ");

}

## 3. ASCII

0 NUL(null) 空字符

1 SOH(start of headline) 标题开始

2 STX (start of text) 正文开始

3 [ETX](http://baike.baidu.com/view/13854.htm)(end of text) 正文结束

4 EOT (end of transmission) 传输结束

5 ENQ (enquiry) 请求

6 ACK (acknowledge) 收到通知

7 BEL (bell) 响铃

8 BS (backspace) 退格

9 HT (horizontal tab) 水平制表符

10 LF (NL line feed, new line) 换行键

11 VT (vertical tab) 垂直制表符

12 FF (NP form feed, new page) 换页键

13 CR (carriage return) 回车键

14 SO (shift out) 不用切换

15 SI (shift in) 启用切换

16 DLE (data link escape) 数据链路转义

17 DC1 (device control 1) 设备控制1

18 DC2 (device control 2) 设备控制2

19 DC3 (device control 3) 设备控制3

20 DC4 (device control 4) 设备控制4

21 NAK (negative acknowledge) 拒绝接收

22 SYN (synchronous idle) 同步空闲

23 ETB (end of trans. block) 传输块结束

24 CAN (cancel) 取消

25 EM (end of medium) 介质中断

26 SUB (substitute) 替补

27 ESC (escape) 换码(溢出)

28 FS (file separator) 文件分割符

29 GS (group separator) 分组符

30 ‑ RS (record separator) 记录分离符

31 ­ US (unit separator) 单元分隔符

32 space 空格

33 !

34 "

35 #

36 $

37 %

38 &

39 '

40 (

41 )

42 \*

43 +

44 ,

45 -

46 .

47 /

48 0

49 1

50 2

51 3

52 4

53 5

54 6

55 7

56 8

57 9

58 :

59 ;

60 <

61 =

62 >

63 ?

64 @

65 A

66 B

67 C

68 D

69 E

70 F

71 G

72 H

73 I

74 J

75 K

76 L

77 M

78 N

79 O

80 P

81 Q

82 R

83 S

84 T

85 U

86 V

87 W

88 X

89 Y

90 Z

91 [

92 \

93 ]

94 ^

95 \_

96 `

97 a

98 b

99 c

100 d

101 e

102 f

103 g

104 h

105 i

106 j

107 k

108 l

109 m

110 n

111 o

112 p

113 q

114 r

115 s

116 t

117 u

118 v

119 w

120 x

121 y

122 z

123 {

124 |

125 }

126 ~

127 DEL（delete 删除）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bin | Dec | Hex | 缩写/字符 | 解释 | |
| 0000 0000 | 0 | 00 | NUL(null) | 空字符 | |
| 0000 0001 | 1 | 01 | SOH(start of headline) | 标题开始 | |
| 0000 0010 | 2 | 02 | STX (start of text) | 正文开始 | |
| 0000 0011 | 3 | 03 | ETX (end of text) | 正文结束 | |
| 0000 0100 | 4 | 04 | EOT (end of transmission) | 传输结束 | |
| 0000 0101 | 5 | 05 | ENQ (enquiry) | 请求 | |
| 0000 0110 | 6 | 06 | ACK (acknowledge) | 收到通知 | |
| 0000 0111 | 7 | 07 | BEL (bell) | 响铃 | |
| 0000 1000 | 8 | 08 | BS (backspace) | 退格 | |
| 0000 1001 | 9 | 09 | HT (horizontal tab) | 水平制表符 | |
| 0000 1010 | 10 | 0A | LF (NL line feed, new line) | 换行键 | |
| 0000 1011 | 11 | 0B | VT (vertical tab) | 垂直制表符 | |
| 0000 1100 | 12 | 0C | FF (NP form feed, new page) | 换页键 | |
| 0000 1101 | 13 | 0D | CR (carriage return) | 回车键 | |
| 0000 1110 | 14 | 0E | SO (shift out) | 不用切换 | |
| 0000 1111 | 15 | 0F | SI (shift in) | 启用切换 | |
| 0001 0000 | 16 | 10 | DLE (data link escape) | 数据链路转义 | |
| 0001 0001 | 17 | 11 | DC1 (device control 1) | 设备控制1 | |
| 0001 0010 | 18 | 12 | DC2 (device control 2) | 设备控制2 | |
| 0001 0011 | 19 | 13 | DC3 (device control 3) | 设备控制3 | |
| 0001 0100 | 20 | 14 | DC4 (device control 4) | 设备控制4 | |
| 0001 0101 | 21 | 15 | NAK (negative acknowledge) | 拒绝接收 | |
| 0001 0110 | 22 | 16 | SYN (synchronous idle) | 同步空闲 | |
| 0001 0111 | 23 | 17 | ETB (end of trans. block) | 传输块结束 | |
| 0001 1000 | 24 | 18 | CAN (cancel) | 取消 | |
| 0001 1001 | 25 | 19 | EM (end of medium) | 介质中断 | |
| 0001 1010 | 26 | 1A | SUB (substitute) | 替补 | |
| 0001 1011 | 27 | 1B | ESC (escape) | 换码(溢出) | |
| 0001 1100 | 28 | 1C | FS (file separator) | 文件分割符 | |
| 0001 1101 | 29 | 1D | GS (group separator) | 分组符 | |
| 0001 1110 | 30 | 1E | RS (record separator) | 记录分离符 | |
| 0001 1111 | 31 | 1F | US (unit separator) | 单元分隔符 | |
| 0010 0000 | 32 | 20 | (space) | 空格 | |
| 0010 0001 | 33 | 21 | ! |  | |
| 0010 0010 | 34 | 22 | " |  | |
| 0010 0011 | 35 | 23 | # |  | |
| 0010 0100 | 36 | 24 | $ |  | |
| 0010 0101 | 37 | 25 | % |  | |
| 0010 0110 | 38 | 26 | & |  | |
| 0010 0111 | 39 | 27 | ' |  | |
| 0010 1000 | 40 | 28 | ( |  | |
| 0010 1001 | 41 | 29 | ) |  | |
| 0010 1010 | 42 | 2A | \* |  | |
| 0010 1011 | 43 | 2B | + |  | |
| 0010 1100 | 44 | 2C | , |  | |
| 0010 1101 | 45 | 2D | - |  | |
| 0010 1110 | 46 | 2E | . |  | |
| 00101111 | 47 | 2F | / |  | |
| 00110000 | 48 | 30 | 0 |  | |
| 00110001 | 49 | 31 | 1 |  |  |
| 00110010 | 50 | 32 | 2 |  |  |
| 00110011 | 51 | 33 | 3 |  |  |
| 00110100 | 52 | 34 | 4 |  |  |
| 00110101 | 53 | 35 | 5 |  |  |
| 00110110 | 54 | 36 | 6 |  |  |
| 00110111 | 55 | 37 | 7 |  |  |
| 00111000 | 56 | 38 | 8 |  |  |
| 00111001 | 57 | 39 | 9 |  |  |
| 00111010 | 58 | 3A | : |  |  |
| 00111011 | 59 | 3B | ; |  |  |
| 00111100 | 60 | 3C | < |  |  |
| 00111101 | 61 | 3D | = |  |  |
| 00111110 | 62 | 3E | > |  |  |
| 00111111 | 63 | 3F | ? |  |  |
| 01000000 | 64 | 40 | @ |  |  |
| 01000001 | 65 | 41 | A |  |  |
| 01000010 | 66 | 42 | B |  |  |
| 01000011 | 67 | 43 | C |  |  |
| 01000100 | 68 | 44 | D |  |  |
| 01000101 | 69 | 45 | E |  |  |
| 01000110 | 70 | 46 | F |  |  |
| 01000111 | 71 | 47 | G |  |  |
| 01001000 | 72 | 48 | H |  |  |
| 01001001 | 73 | 49 | I |  |  |
| 01001010 | 74 | 4A | J |  |  |
| 01001011 | 75 | 4B | K |  |  |
| 01001100 | 76 | 4C | L |  |  |
| 01001101 | 77 | 4D | M |  |  |
| 01001110 | 78 | 4E | N |  |  |
| 01001111 | 79 | 4F | O |  |  |
| 01010000 | 80 | 50 | P |  |  |
| 01010001 | 81 | 51 | Q |  |  |
| 01010010 | 82 | 52 | R |  |  |
| 01010011 | 83 | 53 | S |  |  |
| 01010100 | 84 | 54 | T |  |  |
| 01010101 | 85 | 55 | U |  |  |
| 01010110 | 86 | 56 | V |  |  |
| 01010111 | 87 | 57 | W |  |  |
| 01011000 | 88 | 58 | X |  |  |
| 01011001 | 89 | 59 | Y |  |  |
| 01011010 | 90 | 5A | Z |  |  |
| 01011011 | 91 | 5B | [ |  |  |
| 01011100 | 92 | 5C | \ |  |  |
| 01011101 | 93 | 5D | ] |  |  |
| 01011110 | 94 | 5E | ^ |  |  |
| 01011111 | 95 | 5F | \_ |  |  |
| 01100000 | 96 | 60 | ` |  |  |
| 01100001 | 97 | 61 | a |  |  |
| 01100010 | 98 | 62 | b |  |  |
| 01100011 | 99 | 63 | c |  |  |
| 01100100 | 100 | 64 | d |  |  |
| 01100101 | 101 | 65 | e |  |  |
| 01100110 | 102 | 66 | f |  |  |
| 01100111 | 103 | 67 | g |  |  |
| 01101000 | 104 | 68 | h |  |  |
| 01101001 | 105 | 69 | i |  |  |
| 01101010 | 106 | 6A | j |  |  |
| 01101011 | 107 | 6B | k |  |  |
| 01101100 | 108 | 6C | l |  |  |
| 01101101 | 109 | 6D | m |  |  |
| 01101110 | 110 | 6E | n |  |  |
| 01101111 | 111 | 6F | o |  |  |
| 01110000 | 112 | 70 | p |  |  |
| 01110001 | 113 | 71 | q |  |  |
| 01110010 | 114 | 72 | r |  |  |
| 01110011 | 115 | 73 | s |  |  |
| 01110100 | 116 | 74 | t |  |  |
| 01110101 | 117 | 75 | u |  |  |
| 01110110 | 118 | 76 | v |  |  |
| 01110111 | 119 | 77 | w |  |  |
| 01111000 | 120 | 78 | x |  |  |
| 01111001 | 121 | 79 | y |  |  |
| 01111010 | 122 | 7A | z |  |  |
| 01111011 | 123 | 7B | { |  |  |
| 01111100 | 124 | 7C | | |  |  |
| 01111101 | 125 | 7D | } |  |  |
| 01111110 | 126 | 7E | ~ |  |  |
| 01111111 | 127 | 7F | DEL (delete) | 删除 |  |

#### 5.DLX

/\* dancing link

\* 精确覆盖问题

\* 可以添加迭代加深优化：

\* 1）枚举深度h；

\* 2）若当前深度+predeep > h return false；

\*

int predeep() {

bool vis[Maxm];

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

int ret = 0;

for (Node \*p = head->R; p != head; p = p->R)

if (!vis[p->col]) {

ret ++ ;

vis[p->col] ++ ;

for (Node \*q = p->D; q != p; q = p->D)

for (Node \*r = q->R; r != q; r = r->R)

vis[r->col] = true;

}

return ret;

}

\* \*/

#define Maxn 1010

#define Maxm 1010

struct Node {

Node \*L, \*R, \*U, \*D;

int col, row;

} \*head, \*row[Maxn], \*col[Maxm], node[Maxn \* Maxm];

int colsum[Maxm], cnt;

void init(int mat[][Maxm], int n, int m) {

cnt = 0;

memset(colsum, 0, sizeof(colsum));

head = &node[cnt ++ ];

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

row[i] = &node[cnt ++ ];

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

col[j] = &node[cnt ++ ];

head->D = row[1], row[1]->U = head;

head->R = col[1], col[1]->L = head;

head->U = row[n], row[n]->D = head;

head->L = col[m], col[m]->R = head;

head->row = head->col = 0;

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

if (i != n) row[i]->D = row[i + 1];

if (i != 1) row[i]->U = row[i - 1];

row[i]->L = row[i]->R = row[i];

row[i]->row = i, row[i]->col = 0;

}

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

if (i != m) col[i]->R = col[i + 1];

if (i != 1) col[i]->L = col[i - 1];

col[i]->U = col[i]->D = col[i];

col[i]->col = i, col[i]->row = 0;

}

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

for (int j = m; j > 0; j -- )

if (mat[i][j]) {

Node \*p = &node[cnt ++ ];

p->R = row[i]->R, row[i]->R->L = p;

p->L = row[i], row[i]->R = p;

p->D = col[j]->D, col[j]->D->U = p;

p->U = col[j], col[j]->D = p;

p->row = i;

p->col = j;

colsum[j] ++ ;

}

}

/\*多重覆盖只需删除列，无需对应行删除

void remove(Node \*c) {

for (Node \*p = c->D; p != c; p = p->D) {

p->L->R = p->R;

p->R->L = p->L;

}

}

\*/

void remove(Node \*c) {

c->L->R = c->R;

c->R->L = c->L;

for (Node \*p = c->D; p != c; p = p->D) {

for (Node \*q = p->R; q != p; q = q->R) {

q->U->D = q->D;

q->D->U = q->U;

colsum[q->col] -- ;

}

}

}

void resume(Node \*c) {

for (Node \*p = c->U; p != c; p = p->U) {

for (Node \*q = p->L; q != p; q = q->L) {

q->U->D = q;

q->D->U = q;

colsum[q->col] ++ ;

}

}

col[c->col]->L->R = col[c->col];

col[c->col]->R->L = col[c->col];

}

int ans[Maxm];

int dfs(int deep) {

if (head->R == head) return deep;

Node \*p, \*q = head->R;

for (p = head->R; p != head; p = p->R)

if (colsum[p->col] < colsum[q->col])

q = p;

remove(q);

for (p = q->D; p != q; p = p->D) {

for (Node\* r = p->R; r != p; r = r->R)

if (r->col != 0)

remove(col[r->col]);

/\*--------可修改区域-----------\*/

ans[deep] = p->row;

/\*-----------------------------\*/

int sta = dfs(deep + 1);

if (sta != -1) return sta;

for (Node\* r = p->L; r != p; r = r->L)

if (r->col != 0)

resume(col[r->col]);

}

resume(q);

return -1;

}

//#pragma comment(linker,"/STACK:102400000,102400000")

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <cmath>

#include <cctype>

#include <iostream>

#include <sstream>

#include <algorithm>

#include <string>

#include <vector>

#include <set>

#include <map>

#include <stack>

#include <queue>

#define N 10100

#define M 200100

#define LEN 100100

#define INF (1 << 30)

#define eps 1e-10

#define PB push\_back

#define A first

#define B second

using namespace std;

typedef long long LL;

# 图论

## 1. ShortestPath

### 1.1 SPFA

void spfa(int s) {

priority\_Que <pairII> Q;

memset(dist, CLRINF, sizeof(dist));

dist[s] = 0;

Q.push(pairII(- dist[s], s));

while (!Q.empty()) {

int u = Q.top().B, d = - Q.top().A;

Q.pop();

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

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

int v = edge[j].v, temp = d + edge[j].w;

if (temp < dist[v]) {

dist[v] = temp;

Q.push(pairII( - dist[v], v));

}

}

}

}

bool dfsSpfa(int u) // 可用于判负权环，注意初始化 dis 和 vis 数组

{

visit[u] = true;

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

if (dist[u] + edge[j].w < dist[edge[j].v])

{

int v = edge[j].v, temp = dist[u] + edge[j].w;

dist[v] = temp;

if (visit[v]) return true;

if (dfsSpfa(v)) return true;

}

visit[u] = false;

return false;

}

### 1.2 Johnson

算法步骤简述：

1.计算图G加入新结点后的图G'，加入的新结点0到所有原结点之间距离为0，同时形成新的边集E'；

2.使用Bellman-Ford算法处理G'，并形成0结点到各结点的最小距离d。

3.如果Bellman-Ford算法检测出有负权回路则提示FALSE并退出，否则继续。

4.对所有G'中的顶点v,根据0结点到v的最小距离，将h(v)设置为这个值。

5.对所有的边w(u,v)，权值更新为w(u,v)+h(u)-h(v)

6.对图G中所有结点运行Dijkstra算法计算与其他顶点最短距离d'[u][v]

（此处假定G和w集合是分开存储的。直接使用G'也可以，因为0结点对其他结点是不可达的，但这显然浪费了计算时间。如果权值信息存在G'中，可以对G'进行操作，只不过跳过了0结点的处理）

7.原图G中最短距离d[u][v] = d'[u][v] +h(v)-h(u)

## 2. MinimumSpanningTree

### 2.1 Prim

### 2.2 Kruskal

## 3. LCA

### 3.1 倍增算法

struct node

{

int u, v, w, next;

} edge[2\*M];

int tot, last[N];

int length[N], depth[N], fa[STEP][N];

int n, m;

void addedge(int x, int y)

{

edge[tot].u = x; edge[tot].v = y; edge[tot].next = last[x]; last[x] = tot++;

}

void init()

{

memset(length, 0, sizeof(length));

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

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

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

tot = 0;

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

int x, y, z;

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

{

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

addedge(x, y);

addedge(y, x);

}

}

void dfs(int u)

{

int v;

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

{

v = edge[j].v;

if(fa[0][v] == 0)

{

fa[0][v] = u;

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

dfs(v);

}

}

}

int getLCA(int x, int y)

{

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

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

for(int i = STEP - 1; i >= 0; i--)

if((1 << i) & dif)

{

dif -= (1 << i); x = fa[i][x];

}

for(int i = STEP - 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 solve()

{

fa[0][1] = 1;

depth[1] = 0;

length[1] = 0;

dfs(1);

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

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

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

int x, y, z;

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

{

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

z = getLCA(x, y);

printf("LCA(%d, %d)=%d\n", x, y, z);

}

}

## 4. 拓扑排序

### 4.1 拓扑排序

1. 朴素算法：

将入度为0的点加入队列， 再依次取出队列中的元素，把其出边依次遍历，并且将边指向的节点入度减一，同时将入度为0的点加入队列。

queue < int > Q;

memset(idegree, 0, sizeof(idegree));

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

tot = 0;

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

{

int x, y;

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

addedge(x, y);

idegree[y]++;

}

int now = 0;

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

if(idegree[i] == 0)

Q.push(i);

while(!Q.empty())

{

int u = Q.top();

Q.pop();

now++;

printf("%d", u);

if(now < n) putchar(' ');

else putchar('\n');

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

{

int v = edge[j].v;

idegree[v]--;

if(idegree[v] == 0)

Q.push(v);

}

}

2. 求字典序最小的方案：

当各点标号均互不相同时，直接将队列改为优先队列；

当存在重复标号时？？？？

3. 基于DFS的拓扑排序（很短很好写）

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);

}

### 4.2 拓扑排序数

求树的拓扑排序数：

dp[root] = num[root] ! / (num[i] , i in tree[root])

### 4.3 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）

/\*

\* 求出early 和 latest

\* idegree odegree 出入度

\* 中间有重建图， 所以保存了一条边的u 和 v

\*/

memset(idegree, 0, sizeof(idegree));

memset(odegree, 0, sizeof(odegree));

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

tot = 0;

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

{

int x, y, z;

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

addedge(x, y, z);

idegree[y]++;

odegree[x]++;

}

memset(early, 0, sizeof(early));

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

if(!idegree[i])

Q.push(i);

while(!Q.empty())

{

int u = Q.front();

Q.pop();

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

{

int v = edge[j].v;

idegree[v]--;

if(!idegree[v])

Q.push(v);

int temp = early[u] + edge[j].len;

early[v] = max(early[v], temp);

}

}

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

tot = 0;

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

addedge(edge[i].v, edge[i].u, edge[i].len);

memset(latest, CLRINF, sizeof(latest));

cout << latest[0] << endl;

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

if(odegree[i] == 0)

{

latest[i] = early[i];

Q.push(i);

}

while(!Q.empty())

{

int u = Q.front();

Q.pop();

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

{

int v = edge[j].v;

odegree[v]--;

if(!odegree[v])

Q.push(v);

int temp = latest[u] - edge[j].len;

latest[v] = min(latest[v], temp);

}

}

## 5. 连通性

### 5.1 StrongConnected

/\*

\*TarjanSCC\_adj\_list

\*/

struct edgenode

{

int u, v, next;

} edge[2\*M];

int tot, last[N];

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

stack <int> sta;

void addedge(int x, int y)

{

edge[tot].u = x; edge[tot].v = y; edge[tot].next = last[x]; last[x] = tot++;

}

void init() //输入数据

{

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

tot = 0;

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

{

int x, y;

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

addedge(x, y);

}

}

void Tarjan(int u) //Tarjan

{

dfn[u] = low[u] = nindex++; //给dfn和low赋初始值

sta.push(u); //将节点压栈

instack[u] = 1;

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

{

int v = edge[j].v;

if(!dfn[v]) //未遍历，即未在栈内的节点

{

Tarjan(v);

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

low[u] = low[v];

}

else if(instack[v] && dfn[v] < low[u]) //已经入栈的节点

low[u] = low[v];

}

if(dfn[u] == low[u]) //弹出root及之上的节点， 为一个强连通分量

{

//并且标记或者染色

int i;

do

{

i = sta.top();

instack[i] = 0;

belong[i] = ncnt;

sta.pop();

}

while(i != u);

ncnt++;

}

}

void solve()

{

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

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

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

memset(belong, 0, sizeof(belong));

ncnt = 1;

nindex = 1;

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

if(0 == dfn[i])

{

Tarjan(i);

}

///\*

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

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

//\*/

}

### 5.2 EdgeConnected

/\*

\*TarjanEBCC

\*/

struct edgenode

{

int u, v, next, n;

} edge[2\*M];

int last[N], tot;

int n, m;

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

stack <int> sta;

void addedge(int x, int y)

{

edge[tot].u = x; edge[tot].v = y; edge[tot].next = last[x]; last[x] = tot++;

}

void init()

{

tot = 0;

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

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

int x, y;

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

{

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

addedge(x, y);

addedge(y, x);

}

}

void Tarjan(int u, int from)

{

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

instack[u] = 1;

sta.push(u);

int v;

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

{

if(j == from) continue;

v = edge[j].v;

if(0 == dfn[v])

{

Tarjan(v, j ^ 1);

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

low[u] = low[v];

}

else if(instack[v])

{

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

low[u] = dfn[v];

}

}

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

{

ncnt++;

do

{

v = sta.top();

instack[v] = 0;

sta.pop();

belong[v] = ncnt;

}

while(v != u);

}

}

void solve()

{

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

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

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

memset(belong, 0, sizeof(belong));

ncnt = 0;

nindex = 0;

int flag = 0;

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

if(0 == belong[i])

{

Tarjan(i, -1);

flag++;

}

///\*

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

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

//\*/

}

### 5.3 PointConnected

/\*

\*TarjanPBCC

\*/

struct edge

{

int u, v, w, next;

} edge[2\*M];

int last[N], tot;

int n, m, belong[2\*M];

int dfn[N], low[N], ncnt, nindex, iscut[N];

stack<int> sta;

vecI ne[N \* 3]; //新图

bool color[N \* 3];

void init()

{

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

tot = 0;

}

void addedge(int u, int v)

{

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

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

}

void Tarjan(int u, int from)

{

int v, child = 0;

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

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

{

if (j == from) continue;

v = edge[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]) //割点判断

{

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;

}

void solve()

{

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

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

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

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

ncnt = nindex = 0;

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

if (!dfn[i])

{

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

Tarjan(i, -1);

}

/\*建立新图\*/

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

if (iscut[i] == 1)

{

color[++ncnt] = true;

iscut[i] = ncnt;

}

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

ne[i].clear();

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

if (iscut[edge[i \* 2].u] == -1

&& iscut[edge[i \* 2].v] == -1) continue;

else

{

int u, v;

if (iscut[edge[i \* 2].u] != -1)

{

u = iscut[edge[i \* 2].u];

v = belong[i];

ne[u].PB(v);

ne[v].PB(u);

}

if (iscut[edge[i \* 2].v] != -1)

{

u = iscut[edge[i \* 2].v];

v = belong[i];

ne[u].PB(v);

ne[v].PB(u);

}

}

}

## 6. 2-SAT

### 6.1 判断可行性

建图，求强连通，判断任意对象的两部是否属于同一强连通：

### 6.2 求任意解

方法是，对原图求一次强连通分量，然后看每组中的两个点是否属于同一个强连通分量，如果存在这种情况，那么无解

然后对于缩点后的图G'，我们将G'中所有边转置。进行拓扑排序。  
   对于缩点后的所有点，我们先预处理求出所有冲突顶点。例如缩点后Ai所在强连通分支的ID  
   为id[ Ai] ，同理~Ai在 id[ ~Ai ],所以冲突顶点  
    conflict[ id[Ai] ]=conflict[ id[~Ai] ];  
同理conflict[ id[~Ai] ]=conflict[ id[Ai] ];

    设缩点后有Nscc个点。  
    然后对拓扑序进行染色，初始化所有点color均为未着色  
    顺序遍历得到的拓扑序列，对于未着色的点x，将x染成红色，同时将所有与x矛盾的点conflic[x]染成  
蓝色。

2-sat的一组解就等价于所有缩点后点颜色为红色的点，也就是color[ id[i] ]=RED的所有点

缩点  
　　Tarjan算法缩点，将所有的边反过来（ 为什么？后面有嗯 ）  
判可行  
　　枚举集合的两个元素，看其是否处于不同的块内，若否的话则给出不可行信息  
记录矛盾  
　　这里所说的矛盾不是题中给出的两个人之间有仇恨，那样的边是实际存在，我们这里说的矛盾是指若两个块分别含有两个对立节点，也就是说一个集合的两个元素分布在了两个不同的块中，那么这两个块就是矛盾的，即不可能同时被选择，这样一种矛盾关系是不存在于边中的，是不依赖于输入的数据的，我们要找到与一个块对立的块，并把它们保存下来。  
拓扑排序  
　　将缩点后的图进行拓扑排序（排的是块而不是节点）  
构造方案  
　　按照拓扑序列的顺序，依次访问所有块，若一个块未被标记，将其标记为“选择”，不传递“选择”标记，将被选块的对立块标记为“不选择”，将其“不选择”标记沿着边正向传递。（ 不是逆着边么？哼，图已经被反过来了，你到底有没有认真看呐！囧 ）

/\*2-sat\*/

#define Maxn 5010

vector<VI> e, ne;

int dfn[Maxn], low[Maxn], block[Maxn], sta[Maxn], top, tsp, N;

int sta1[Maxn], rid[Maxn], top1; //rid 新旧图映射，sta1, top1 拓扑排序，输出路径

/\*初始化 top = tsp = N = 0 和 block为 -1, dfn 为 0\*/

void tarjan(int u, int root) {

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

sta[top ++ ] = u;

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

int v = e[u][i];

if (!dfn[v]) tarjan(v, root);

if (dfn[v] < dfn[root]) continue;

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

}

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

int v;

do {

v = sta[ -- top];

block[v] = N;

} while (v != u);

rid[N] = u;

sta1[top1 ++ ] = N;

N ++ ;

}

}

int color[Maxn];

void colorBlue(int u) {

color[u] = 2;

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

int v = ne[u][i];

if (!color[v])

colorBlue(v);

}

}

bool twoSat(int n) {

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

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

top1 = top = N = tsp = 0;

for (int i = 0; i < n \* 2; i ++ )

if (!dfn[i])

tarjan(i, i);

for (int i = 0; i < n \* 2; i += 2 )

if (block[i] == block[i ^ 1])

return 0;

//return 1;

/\*建立新图\*/

ne.clear();

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

VI p;

ne.PB(p);

}

for (int i = 0; i < n \* 2; i ++ )

for (int j = 0; j < e[i].size(); j ++ )

if (block[i] != block[e[i][j]])

ne[block[e[i][j]]].PB(block[i]); //反向建图

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

for (int i = 0; i < top1; i ++ ) {

int x = sta1[i];

if (!color[x]) {

color[x] = 1;

x = block[rid[x] ^ 1];

colorBlue(x);

}

}

return 1;

}

### 6.3 求字典序最小的解

也就是说，沿着一条路径，如果一个点被选择了，那么这条路径以后的点都将被选择，那么，出现不可行的情况就是，存在一个党派两个党员都被选择了，那么，我们只需要枚举一下就可以了，数据不大，答案总是可以出来的。  
　　这么一个简单的算法，放在第一个说，看来你已经知道它不是重点了，但是，若题目要求的是字典序最小的方案的话，那就只能选择这个算法了。

## 7. 全局最小割

### 7.1 全局最小割

int combine[N];

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

int S, T, minCut, k;

int top, sta[N];

int maxi;

VECTOR\_I parta, partb;

VECTOR\_I belong[N];

int Search(int n)

{

int vis[N];

int wet[N];

clr(vis,0);

clr(wet,0);

int minCut = 0;

int temp = -1;

S = -1, T = -1;

int top = 0;

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

{

int maxi = -INF;

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

{

int u = node[j];

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

{

temp = u;

maxi = wet[u];

}

}

sta[top++] = temp;

vis[temp] = true;

if(i == n - 1)

minCut = maxi;

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

{

int u = node[j];

if(!combine[u] && !vis[u])

{

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

}

}

}

S = sta[top - 2];

T = sta[top - 1];

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

return minCut;

}

int StoerWagner(int n)

{

int ans = INF;

clr(combine,0);

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

node[i] = i;

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

{

k = n - i + 1;

int cur = Search(k);

if(cur < ans)

{

ans = cur;

}

if(ans == 0) return ans;

combine[T] = true;

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

{

if(j == S) continue;

if(!combine[j])

{

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

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

}

}

}

return ans;

}

int main()

{

//freopen("input.txt","r",stdin);

int n, m;

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

{

clr(edge, 0);

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;

}

### 7.2 求最小割边

/\*

\*StoerWagner

\*hdu 4654

\*/

int combine[N];

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

int S, T, minCut, k;

int top, sta[N];

int maxi;

VECTOR\_I parta, partb;

VECTOR\_I belong[N];

int Search(int n)

{

int vis[N];

int wet[N];

clr(vis,0);

clr(wet,0);

int minCut = 0;

int temp = -1;

S = -1, T = -1;

int top = 0;

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

{

int maxi = -INF;

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

{

int u = node[j];

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

{

temp = u;

maxi = wet[u];

// printf("temp %d maxi %d\n", temp, maxi);

}

}

sta[top++] = temp;

vis[temp] = true;

if(i == n - 1)

minCut = maxi;

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

{

int u = node[j];

if(!combine[u] && !vis[u])

{

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

}

}

}

S = sta[top - 2];

T = sta[top - 1];

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

return minCut;

}

int StoerWagner(VECTOR\_I & t)

{

int k;

int ans = INF, n = t.size(), vis[N];

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

{

belong[i].clear();

belong[i].push\_back(i);

}

clr(combine,0);

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

node[i] = i;

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

{

k = n - i + 1;

int cur = Search(k);

if(cur < ans)

{

ans = cur;

//cout << "ans" << ans << endl;

clr(vis, 0);

for(int j = 0; j < belong[T].size(); j++)

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

}

//if(ans == 0) return ans;

combine[T] = true;

for(int j = 0; j < belong[T].size(); j++)

belong[S].push\_back(belong[T][j]);

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

{

if(j == S) continue;

if(!combine[j])

{

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

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

}

}

}

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

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

if(vis[i]) parta.push\_back(t[i]);

else partb.push\_back(t[i]);

return ans;

}

int dfs(VECTOR\_I & t)

{

int n = t.size();

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

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

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

int cut = StoerWagner(t);

if(cut >= k) return 1;

VECTOR\_I a(parta), b(partb);

return dfs(a) + dfs(b);

}

int main()

{

//freopen("input.txt","r",stdin);

int n, m;

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

{

clr(g, 0);

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

{

int u, v, z;

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

u--; v--;

g[u][v] += 1;

g[v][u] += 1;

}

VECTOR\_I a;

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

a.push\_back(i);

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

}

return 0;

}

## 8. 匹配问题

### 8.1 最大匹配

/\*

\* 匈牙利算法 DFS

\* 临接表

\*/

struct edgenode

{

int v, next;

}edge[M];

int last[N], tot;

int n, match = 0;

int matchx[N], matchy[N];

int visit[N];

int preHungary()

{

int res = 0, u, v;

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

{

u = i;

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

{

v = edge[j].v;

if(-1 == matchy[v])

{

matchx[u] = v;

matchy[v] = u;

res++;

break;

}

}

}

return res;

}

bool dfs(int u)

{

int v;

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

{

v = edge[j].v;

if(visit[v]) continue;

visit[v] = true;

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

{

matchy[v] = u;

matchx[u] = v;

return true;

}

}

return false;

}

void solve()

{

match = 0;

clr(matchx, -1); clr(matchy, -1);

match += preHungary();

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

if(-1 == matchx[i])

{

clr(visit, false);

if(dfs(i)) match++;

}

}

/\*

\* 匈牙利算法 DFS

\* 临接矩阵

\*/

vecI edge[N];

int n, match = 0;

int matchx[N], matchy[N];

int visit[N];

int preHungary()

{

int res = 0, u, v;

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

{

u = i;

for(int j = 0; j < edge[u].size(); j++)

{

v = edge[u][j];

if(-1 == matchy[v])

{

matchx[u] = v;

matchy[v] = u;

res++;

break;

}

}

}

return res;

}

bool dfs(int u)

{

int v;

for(int j = 0; j < edge[u].size(); j++)

{

v = edge[u][j];

if(visit[v]) continue;

visit[v] = true;

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

{

matchy[v] = u;

matchx[u] = v;

return true;

}

}

return false;

}

void solve()

{

match = 0;

clr(matchx, -1); clr(matchy, -1);

match += preHungary();

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

if(-1 == matchx[i])

{

clr(visit, false);

if(dfs(i)) match++;

}

}

### 8.2相关概念

1. 最大匹配

1. dfs

2. bfs

3. Hopcroft-Karp

4. 利用贪心构造最初解再利用匈牙利增广路调整

5. 最大流

2. 最优完备匹配

1. KM( O(n^4)

2. KM + (O(n^3)) PS: 好像没有这么低的复杂度

3. 费用流

3. 稳定婚姻问题

4. 多重匹配

建立最大流模型解决

5.多重最优匹配

建立费用流模型解决

6. 最小独立集 = 顶点数 - 最大匹配数

7. 最小点覆盖 = 最大匹配数

8. 最小路径覆盖 = 顶点数 - 最大匹配数

(PS: 此处注意, 最小路径覆盖是针对一般图而言, 那么将一个点拆开成为i和i'建立二分图)

### 8.3 任意图最大匹配(带花树)

#define N 1100

#define M 200100

#define LEN 100100

#define INF (1 << 30)

typedef long long LL;

int n, head, tail, Start, Finish;

int match[N]; //表示哪个点匹配了哪个点

int Father[N]; //这个就是增广路的father……但是用起来太精髓了

int base[N]; //该点属于哪朵花

int Q[N];

bool mark[N], map[N][N], InBlossom[N], in\_Queue[N];

void init()

{

int x,y;

scanf("%d",&n);

while (scanf("%d%d",&x,&y)!=EOF)

map[x][y]=map[y][x]=1;

}

void BlossomContract(int x,int y)

{

clr(mark, false);

clr(InBlossom,false);

#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 (!in\_Queue[i])

{

Q[++tail]=i;

in\_Queue[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()

{

clr(father, 0);

clr(in\_Queue, false);

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

head=0;

tail=1;

Q[1]=Start;

in\_Queue[Start]=1;

while (head!=tail)

{

int x=Q[++head];

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

if (map[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];

in\_Queue[match[y]]=true;

}

else

{

Finish=y;

Change();

return;

}

}

}

}

void Edmonds()

{

clr(match, 0);

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

if (match[Start]==0)

FindAugmentPath();

}

void output()

{

clr(mark, false);

int cnt=0;

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

if (match[i]) cnt++;

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

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]);

}

}

int main()

{

// freopen("input.txt","r",stdin);

init();

Edmonds();

output();

return 0;

}

## 9. 最小环 – 最小环

g[i][j]=i,j之间的边长

dist:=g;

for k:=1 to n do

begin

for i:=1 to k-1 do //求最大环不要这部分循环, 普通的floyd就可以

for j:=i+1 to k-1 do //最后最大环为max(dist[i][i]

answer:=min(answer,dist[i][j]+g[i][k]+g[k][j]);//

for i:=1 to n do

for j:=1 to n do

dist[i][j]:=min(dist[i][j],dist[i][k]+dist[k][j]);

end;

## 10. 欧拉回路

/\*欧拉回路, 有向图\*/

VI ve[Maxn];

int cur[Maxn];

stack<int> eulerianWalk(int u) { //返回欧拉回路的逆序

stack<int> sta, ret;

sta.push(u);

cur[u] = 0;

while (!sta.empty()) {

u = sta.top();

sta.pop();

while (cur[u] < ve[u].size()) {

sta.push(u);

u = ve[u][cur[u] ++ ];

}

ret.push(u);

}

return ret;

}

## 11. 最大流

### 11.1 EK

int c[N][N], pre[N], que[N][2], n, m, S, T;

bool visit[N];

bool Extend(int s, int t, int &r)

{

int idx = s, l;

l = r = 0;

memset(visit, 0, sizeof(visit));

que[r][1] = INF;

que[r ++ ][0] = idx;

pre[idx] = idx;

visit[idx] = true;

while (l < r)

{

idx = que[l][0];

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

if (!visit[i] && c[idx][i] > 0)

{

que[r][1] = min(que[l][1], c[idx][i]);

que[r][0] = i;

pre[i] = idx;

if (i == t) return true;

r ++ ;

visit[i] = true;

}

l ++ ;

}

return false;

}

void Fill(int s, int t, int w)

{

int u, v = t;

while (v != s)

{

u = pre[v];

c[u][v] -= w;

c[v][u] += w;

v = u;

}

}

int EK(int s, int t)

{

int tail = 0, flow = 0;

while (Extend(s, t, tail))

{

flow += que[tail][1];

Fill(s, t, que[tail][1]);

tail = 0;

}

return flow;

}

### 11.2 Dinic

/\*Dinic\*/

#define M 200005

#define N 50005

#define INF 0x3f3f3f3f

struct edge {

int u, v, c, next;

} edge[M];

int last[N], tot;

void init() {

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

tot = 0;

}

void addedge(int s, int t, int v){

edge[tot].u = s, edge[tot].v = t, edge[tot].c = v;

edge[tot].next = last[s], last[s] = tot ++ ;

edge[tot].u = t, edge[tot].v = s, edge[tot].c = 0;

edge[tot].next = last[t], last[t] = tot ++ ;

}

int dist[N], cur[N], sta[N], que[N], pre[N];

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

int front = 0, tail = 0;

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

dist[s] = 0;

que[tail ++ ] = s;

while (front < tail) {

for (int i = last[que[front ++ ]]; i != -1; i = edge[i].next)

if (edge[i].c > 0 && dist[edge[i].v] == -1){

dist[edge[i].v] = dist[edge[i].u] + 1;

if (edge[i].v == t)

return 1;

que[tail ++ ] = edge[i].v;

}

}

return 0;

}

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

int maxflow = 0, tp;

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

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

cur[i] = last[i];

int u = s, tail = 0;

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

if(u == t){

tp = INF;

for (int i = tail - 1; i >= 0; i -- )

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

maxflow += tp;

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

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

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

if (edge[sta[i]].c == 0)

tail = i;

}

u = edge[sta[tail]].u;

}

else if (cur[u] != -1 && edge[cur[u]].c > 0

&& dist[u] + 1 == dist[edge[cur[u]].v]) {

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

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

}

else {

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

u = edge[sta[ -- tail]].u;

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

}

}

}

return maxflow;

}

### 11.3 ISAP

/\*SAP\*/

#define M 200005

#define N 50005

#define INF 0x7fffffff

struct node {

int u, v, c, next;

} edge[M];

int last[N], tot;

void init() {

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

tot = 0;

}

void addedge(int s, int t, int v){

edge[tot].u = s, edge[tot].v = t, edge[tot].c = v;

edge[tot].next = last[s], last[s] = tot ++ ;

edge[tot].u = t, edge[tot].v = s, edge[tot].c = v;

edge[tot].next = last[t], last[t] = tot ++ ;

}

int dis[N], cur[N], gap[N], pre[N];

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

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

memset(cur, 0, sizeof(cur));

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

cur[i] = last[i];

memset(gap, 0, sizeof(gap));

int v = s, maxflow = 0;

gap[0] = n;

while (dis[s] <= n) {

bool flag = false;

for (int i = cur[v]; i != -1; i = edge[i].next)

if (edge[i].c > 0 && dis[v] == dis[edge[i].v] + 1) {

flag = true;

pre[edge[i].v] = v;

cur[v] = i;

v = edge[i].v;

break;

}

if (flag) {

if (v == t) {

int det = INF;

for (int i = v; i != s; i = pre[i])

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

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

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

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

}

maxflow += det;

v = s;

}

}

else {

int mind = n;

for (int i = last[v]; i != -1; i = edge[i].next )

if (edge[i].c > 0 && dis[edge[i].v] < mind) {

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

cur[v] = i;

}

if (( -- gap[dis[v]]) == 0) break;

gap[dis[v] = mind + 1] ++ ;

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

}

}

return maxflow;

}

## 12. RMQ

void ST(){

e[ 0 ] = 1;

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

e[i] = e[i - 1] << 1;

for(int i = 1; i <= n; i++) //边界[1, n]

st[i][0] = height[i];

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

for(int i = 1; i + e[t] - 1 <= n; i++){

int m = i + e[t - 1];

st[i][t] = min(st[i][t - 1], st[m][t - 1]); //最小值

}

}

}

int rmq(int a,int b){

int k = (int)(log(double(b-a+1)) / log(2.0));

return min(st[a][k], st[b - e[k] + 1][k]);

}

## 13. 最小树形图

1.7 最小树形图

/\* 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！

\* 根据pre的信息能构造出这棵树！

\* \*/

int vis[N], pre[N], belong[N], in[N];

int dirtree(int n, int root) {

int sum = 0;

while (1) {

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

in[i] = inf;

pre[i] = -1;

belong[i] = -1;

vis[i] = -1;

}

in[root] = 0;

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

if (e[i].u != e[i].v) {

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

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

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

}

}

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

if (in[i] == inf) return -1;

int num = 0;

for (int i = 0; i < n; i ++ ) { //找圈，收缩圈

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

int j = i;

while (j != -1 && vis[j] == -1 && j != root) {

vis[j] = i;

j = pre[j];

}

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

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

belong[k] = num;

belong[j] = num ++ ;

}

}

sum += in[i];

}

if (num == 0) return sum;

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

if (belong[i] == -1)

belong[i] = num ++ ;

for (int i = 0; i < cnt; i ++ ) { //重新构图

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

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

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

}

n = num;

root = belong[root];

}

}

## 14. 费用流

### 14.1 Spfa

/\*

注意！！！INF 和dist[t]的比较来判断是否有增广路,

\*/

struct edgenode

{

int u, v, next, c, w;

edgenode(int u,int v,int next,int c,int w):u(u),v(v),next(next),c(c),w(w) {}

edgenode() {}

} edge[2\*M];

int tot, last[N];

int dist[N], pre[N];

bool visit[N];

int n, m, S, T, all;

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

{

memset(dist, CLRINF, sizeof(dist[0]) \* (n + 13));

memset(visit, 0, sizeof(visit[0]) \* (n + 13));

memset(pre, -1, sizeof(pre[0]) \* (n + 13));

queue<int> q;

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

q.push(s);

vis[s] = true;

dist[s] = 0;

pre[s] = -1;

while (!q.empty())

{

int u = q.front();

vis[u] =false;

q.pop();

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

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

{

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

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

if (!visit[edge[j].v])

{

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

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

}

}

}

if (dist[t] == FLOWINF) return false;/\*HINT注意INF和dist[t]的初始值的关系\*/

else return true;

}

int ChangeFlow(int t)

{

int det = INF, u = t;

while (~pre[u])

{

u = pre[u];

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

u = edge[u].u;

}

u = t;

while (~pre[u])

{

u = pre[u];

edge[u].c -= det;

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

u = edge[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;

}

### 14.2 ZKW

struct node

{

int u, v, c, w, next;

}edge[M];

int tot, last[N];

int dist[N],pre[N];

bool visit[N];

int n, m, src, des;

int flow, cost, value;

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

{

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

edge[tot].u = v; edge[tot].v = u; edge[tot].c = 0; edge[tot].w = -w; edge[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;

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

{

int v = edge[j].v, c = edge[j].c, w = edge[j].w;

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

{

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

edge[j].c -= delta;

edge[j ^ 1].c += delta;

l -= delta;

if(!l) return m;

}

}

return(m - l);

}

bool ModLabel(int src, int des)

{

int d = INF;

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

if(visit[i])

{

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

{

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

}

}

if(d == INF) return false;

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

if(visit[i])

{

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

{

edge[j].w -= d;

edge[j^1].w += d;

}

}

value += d;

return true;

}

void MinCostMaxFlow(int src, int des)

{

flow = cost = value = 0;

int xx = 0;

do

{

//cout << xx ++ << endl;

do

{

memset(visit, 0, sizeof(visit));

}while(Aug(src, INF));

}while(ModLabel(src, des));

}

## 15. 树的同构

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

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

{

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

}

return (sum \* sum) % 19001;

}

inline void init()

{

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;

}

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.