

ACM/ICPC Template Library

BJTU_Azwraith,
Beijing Jiao Tong University
2013.10

目录

图论.....	3
一 连通性问题.....	3
1. 有向图强连通分量 Tarjan $O(N)$	3
2. 无向图边双连通分量 Tarjan $O(N)$	3
3. 无向图点双连通分量 Tarjan $O(N)$ 【未验】.....	4
4. 应用：2-sat.....	6
5. 关于无向图连通性.....	7
[点连通度与边连通度].....	7
[双连通图、割点与桥].....	7
[双连通分支].....	7
[求割点与桥].....	7
[求双连通分支].....	7
[构造双连通图].....	8
二 网络流和匹配.....	8
1. 最大流 Dinic.....	8
2. 最小费用最大流 SPFA 增广（稀疏图较快）.....	9
3. 二分图最大匹配 匈牙利算法 $O(NM)$	10
4. 混合图欧拉回路建边.....	11
5. 全局最小割.....	11
三 最短路.....	12
1. Dijkstra 复杂度上限 $O(M \log N)$	12
2. SPFA 估计复杂度 $O(2M)$ 带 SLF 优化.....	12
3. 差分约束系统.....	13
4. 次短路径.....	13
四 生成树.....	13
次小生成树.....	13
五 树的直径.....	14
计算几何.....	14
一 二维几何.....	14
1. 基本操作 【部分未验证】.....	14
2. 凸包 $O(N \log N)$	17
3. 旋转卡壳.....	17
4. 半平面交 $O(N \log N)$	18

5. 最小包围圆 $O(N)$	19
二 三维几何.....	21
1. 基本操作（点和线等）.....	21
2. 凸包 $O(N^2)$	22
3. 凸包切割 $O(N^2)$	24
其他算法专题.....	24
Dancing Links X 精确覆盖.....	24
语言和环境.....	26
一 python 可视化小工具.....	26
二 准备工作.....	27
配终端.....	27

图论

一 连通性问题

1. 有向图强连通分量 Tarjan $O(N)$

注意初始化, insert(int, int) 加入单向边

```

1  const int maxn = ____, maxm = ____;
2  struct edge
3  {
4      int next, tar;
5  }e[maxn];
6  struct vtx
7  {
8      int dfn, low, belong;
9      bool instack;
10 }ver[maxn];
11 int hd[maxn], cnt, ind, sta[maxn], top, bcnt;
12 void insert(int a, int b)
13 {
14     e[cnt].next = hd[a]; hd[a] = cnt; e[cnt].tar = b; cnt++;
15 }
16
17 void tarjanSCC(int u)
18 {
19     ver[u].dfn = ver[u].low = ind++;
20     ver[u].instack = true;
21     sta[top++] = u;
22     for (int i = hd[u]; i != -1; i = e[i].next)
23     {
24         int v = e[i].tar;
25         if (!ver[v].dfn)
26         {
27             tarjanSCC(v);
28             ver[u].low = min(ver[u].low, ver[v].low);
29         }
30         if (ver[v].instack)
31             ver[u].low = min(ver[u].low, ver[v].dfn);
32     }
33     if (ver[u].dfn == ver[u].low)
34     {
35         bcnt++;
36     }

```

```

37     int j = -1;
38     do
39     {
40         j = sta[--top];
41         ver[j].belong = bcnt;
42         ver[j].instack = false;
43     } while (j != u);
44 }
45 }
46 void tarjanInit()
47 {
48     memset(hd, -1, sizeof(hd));
49     memset(ver, 0, sizeof(ver));
50     ind = top = 1;
51     cnt = 0;
52 }

```

2. 无向图边双连通分量 Tarjan $O(N)$

注意初始化

insert(int, int) 加入双向边

tarjan(int n) 调用

点的编号从 1 开始, 双连通分量的编号也从 1 开始

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <queue>
5  #include <stack>
6  #include <ctime>
7  using namespace std;
8  /// 无向图 判断割顶 桥 求边双连通分量 (环)
9  const int maxn = 200010, maxm = 1000010;
10 struct edge
11 {
12     int next, s, t, isbridge;
13 }e[2 * maxm];
14 struct vtx
15 {
16     int dfn, low, belong;
17     bool iscut;
18 }ver[maxn];
19
20 int hd[maxn], ecnt, ind, bcnt;
21
22 /// insert(a, b) 创建双向边

```

```

4 void insert(int a, int b)
5 {
6     e[ecnt].next = hd[a]; hd[a] = ecnt; e[ecnt].s = a; e[ecnt].t
7 = b; ecnt++;
8     e[ecnt].next = hd[b]; hd[b] = ecnt; e[ecnt].s = b; e[ecnt].t
9 = a; ecnt++;
10 }
11
12 stack<int> sta;
13 /// 第一次调用时 fa = -1
14 void dfs(int u, int fa, int lastedge)
15 {
16     ver[u].dfn = ver[u].low = ind++;
17     int child = 0;
18     for (int i = hd[u]; i != -1; i = e[i].next)
19     {
20         int v = e[i].t;
21
22         if (!ver[v].dfn)
23         {
24             sta.push(v);
25             child++;
26             dfs(v, u, i);
27             ver[u].low = min(ver[u].low, ver[v].low);
28             if (ver[v].low >= ver[u].dfn)
29                 ver[u].iscut = true;
30             if (ver[v].low > ver[u].dfn)
31             {
32                 e[i].isbridge = true;
33                 ++bcnt;
34                 while (sta.top() != v)
35                 {
36                     ver[sta.top()].belong = bcnt;
37                     sta.pop();
38                 }
39                 ver[sta.top()].belong = bcnt;
40                 sta.pop();
41             }
42         }
43         else if (ver[v].dfn < ver[u].dfn && lastedge != (i^1))
44             ver[u].low = min(ver[u].low, ver[v].dfn);
45     }
46     if (fa < 0 && child == 1) ver[u].iscut = 0;
47 }
48
49 void tarjan(int n)
50 {
51     for (int i = 1; i <= n; i++) if (!ver[i].dfn)
52     {

```

```

53         sta.push(i);
54         dfs(1, -1, -1);
55         if (!sta.empty())
56         {
57             bcnt++;
58             while (!sta.empty())
59             {
60                 ver[sta.top()].belong = bcnt;
61                 sta.pop();
62             }
63         }
64     }
65 }
66
67 void init()
68 {
69     memset(hd, -1, sizeof(hd));
70     memset(ver, 0, sizeof(ver));
71     memset(e, 0, sizeof(e));
72     ind = 1;
73     ecnt = bcnt = 0;
74 }
75

```

3. 无向图点双连通分量 Tarjan $O(N)$ 【未验】

```

1 #include <cstdio>
2 #include <cstdlib>
3 #include <cstring>
4 #include <cmath>
5 #include <map>
6 #include <set>
7 #include <stack>
8 #include <vector>
9 #include <string>
10 #include <iostream>
11 #include <algorithm>
12
13 using namespace std;
14
15 typedef long long LL;
16 typedef vector<int> VI;
17 typedef pair<int,int> PII;
18
19 #define MP make_pair
20 #define PB push_back

```

```

2  #define eps 1e-8
3  #define inf 0x3f3f3f3f
4
5  #define Maxm 200010
6  #define Maxn 10010
7
8  struct edge {
9      int u, v, next;
10 } e[Maxm];
11
12 int cnt, head[Maxn], n, m, block[Maxm];
13 void addedge(int u, int v) {
14     e[cnt].u = u, e[cnt].v = v, e[cnt].next = head[u], head[u] = cnt
15 ++ ;
16     e[cnt].u = v, e[cnt].v = u, e[cnt].next = head[v], head[v] = cnt
17 ++ ;
18 }
19 void Init() {
20     int u, v;
21     memset(head, -1, sizeof(head));
22     cnt = 0;
23     for (int i = 0; i < m; i ++ ) {
24         scanf("%d%d", &u, &v);
25         addedge(u, v);
26     }
27 }
28 int deep, dfn[Maxn], low[Maxn], n1, iscut[Maxn];
29 stack<int> sta;
30 void tarjan(int u, int pre) {
31     int v, child = 0;
32     dfn[u] = low[u] = ++ deep;
33     for (int i = head[u]; i != -1; i = e[i].next) {
34         if (i == pre) continue;
35         v = e[i].v;
36         if (dfn[v] < dfn[u]) {
37             sta.push(i);
38             if (!dfn[v]) {
39                 child ++ ;
40                 tarjan(v, i ^ 1);
41                 low[u] = min(low[u], low[v]);
42                 if (low[v] >= dfn[u]) {
43                     while (sta.top() != i) {
44                         block[sta.top() / 2] = n1;
45                         sta.pop();
46                     }
47                     block[i / 2] = n1 ++ ;
48                     sta.pop();
49                     iscut[u] = 1;
50                 }

```

```

51     }
52     else low[u] = min(low[u], low[v]);
53 }
54 }
55 if (pre < 0 && child == 1) iscut[u] = -1;
56 }
57 VI lin[Maxn * 3];
58 int fa[Maxn * 3], sum[Maxn * 3], h[Maxn * 3];
59 bool vis[Maxn * 3], color[Maxn * 3];
60 void dfs(int t) {
61     vis[t] = true;
62     if (color[t])
63         sum[t] ++ ;
64     for (int i = 0; i < lin[t].size(); i ++ )
65         if (!vis[lin[t][i]]) {
66             fa[lin[t][i]] = t;
67             sum[lin[t][i]] = sum[t];
68             h[lin[t][i]] = h[t] + 1;
69             dfs(lin[t][i]);
70         }
71 }
72 void solve() {
73     memset(dfn, 0, sizeof(dfn));
74     memset(low, 0, sizeof(low));
75     memset(iscut, -1, sizeof(iscut));
76     memset(color, 0, sizeof(color));
77     memset(sum, 0, sizeof(sum));
78     memset(h, 0, sizeof(h));
79     n1 = 0;
80     for (int i = 1; i <= n; i ++ )
81         if (!dfn[i]) {
82             deep = 0;
83             while (!sta.empty()) sta.pop();
84             tarjan(i, -1);
85         }
86     for (int i = 1; i <= n; i ++ )
87         if (iscut[i] == 1) {
88             color[n1] = true;
89             iscut[i] = n1 ++ ;
90         }
91     for (int i = 0; i < n1; i ++ )
92         lin[i].clear();
93     for (int i = 0; i < cnt / 2; i ++ )
94         if (iscut[e[i * 2].u] == -1 && iscut[e[i * 2].v] == -1)
95             continue;
96         else {
97             int u, v;
98             if (iscut[e[i * 2].u] != -1) {
99                 u = iscut[e[i * 2].u];

```

```

100         v = block[i];
101         lin[u].PB(v);
102         lin[v].PB(u);
103     }
104     if (iscut[e[i * 2].v] != -1){
105         u = iscut[e[i * 2].v];
106         v = block[i];
107         lin[u].PB(v);
108         lin[v].PB(u);
109     }
110 }
111 memset(vis, 0, sizeof(vis));
112 for (int i = 0; i < n1; i++)
113     if (!vis[i]) {
114         fa[i] = -1;
115         dfs(i);
116     }
117 }
118 int anc[Maxn * 3][20];
119 void lca_prepare() {
120     memset(anc, -1, sizeof(anc));
121     for (int i = 0; i < n1; i++)
122         anc[i][0] = fa[i];
123     for (int j = 1; j < 20; j++) {
124         bool flag = false;
125         for (int i = 0; i < n1; i++) {
126             if (anc[i][j - 1] != -1)
127                 anc[i][j] = anc[anc[i][j - 1]][j - 1];
128             if (anc[i][j] != -1)
129                 flag = true;
130         }
131         if (!flag) break;
132     }
133 }
134 int query_lca(int u, int v) {
135     int dif = abs(h[u] - h[v]), i = 0;
136     if (h[u] < h[v]) swap(u, v);
137     while (dif) {
138         if (dif & 1) u = anc[u][i];
139         i++;
140         dif >>= 1;
141     }
142     for (i = 19; i >= 0; i--)
143         if (anc[u][i] == anc[v][i]) continue;
144     else u = anc[u][i], v = anc[v][i];
145     if (u == v) return u;
146     else return anc[u][0];
147 }
148 void query() {

```

```

149     int q, u, v;
150     scanf("%d", &q);
151     for (int i = 0; i < q; i++) {
152         scanf("%d%d", &u, &v);
153         u = block[u - 1];
154         v = block[v - 1];
155         int ans = sum[u] + sum[v];
156         int lca = query_lca(u, v);
157         if (fa[lca] != -1) ans -= sum[fa[lca]];
158         ans -= sum[lca];
159         printf("%d\n", ans);
160     }
161 }
162 int main() {
163     while (scanf("%d%d", &n, &m) != EOF) {
164         if (n == 0 && m == 0) break;
165         Init();
166         solve();
167         lca_prepare();
168         query();
169     }
170     return 0;
171 }

```

4. 应用：2-sat

```

1 bool twoSat(int n)
2 {
3     for (int i = 0; i < 2 * n; i++)
4         if (!ver[i].dfn) tarjanSCC(i);
5     for (int i = 0; i < n; i++)
6         if (ver[i * 2].belong == ver[i * 2 + 1].belong)
7             return false;
8     return true;
9 }
10
11 // 用法参考下面的 main() 函数
12 // 建图前, 先 tarjanInit() 建图时用 insert(u, v) u-->v
13 // 对立点为 i 与 (i ^ 1)
14 int main()
15 {
16     int m;
17     while (~scanf("%d", &m))
18     {
19         tarjanInit();

```

```

19     for (int i = 0; i < m; i++)
20     {
21         int a1, a2, c1, c2;
22         scanf("%d%d%d%d", &a1, &a2, &c1, &c2);
23         insert(a1 + c1 * n, a2 + (1 - c2) * n); // x1 --> ~x2
24         insert(a2 + c2 * n, a1 + (1 - c1) * n); // x2 --> ~x1
25     }
26     if (twoSat(n))
27         printf("YES\n");
28     else
29         printf("NO\n");
30 }
31 return 0;
32 }

```

5. 关于无向图连通性

[点连通度与边连通度]

在一个无向连通图中，如果有一个顶点集合，删除这个顶点集合，以及这个集合中所有顶点相关联的边以后，原图变成多个连通块，就称这个点集为**割点集合**。一个图的**点连通度**的定义为，最小割点集合中的顶点数。

类似的，如果有一个边集合，删除这个边集合以后，原图变成多个连通块，就称这个点集为**割边集合**。一个图的**边连通度**的定义为，最小割边集合中的边数。

[双连通图、割点与桥]

如果一个无向连通图的点连通度大于 1，则称该图是**点双连通的**(point biconnected)，简称**双连通**或**重连通**。一个图有割点，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为**割点**(cut point)，又叫**关节点**(articulation point)。

如果一个无向连通图的边连通度大于 1，则称该图是**边双连通的**(edge biconnected)，简称**双连通**或**重连通**。一个图有桥，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为**桥**(bridge)，又叫**关节边**

(articulation edge)。

可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，均既可指点双连通，又可指边双连通。

[双连通分支]

在图 G 的所有子图 G' 中，如果 G' 是双连通的，则称 G' 为**双连通子图**。如果一个双连通子图 G' 它不是任何一个双连通子图的真子集，则 G' 为**极大双连通子图**。**双连通分支**(biconnected component)，或**重连通分支**，就是图的极大双连通子图。特殊的，点双连通分支又叫做**块**。

[求割点与桥]

该算法是 R.Tarjan 发明的。对图深度优先搜索，定义 $DFS(u)$ 为 u 在搜索树（以下简称为树）中被遍历到的次序号。定义 $Low(u)$ 为 u 或 u 的子树中能通过非父子边追溯到的最早的节点，即 DFS 序号最小的节点。根据定义，则有：

$Low(u) = \min \{ DFS(u) \mid DFS(v) \text{ 为后向边(返祖边) 等价于 } DFS(v) < DFS(u) \text{ 且 } v \text{ 不为 } u \text{ 的父亲节点 } Low(v) \mid (u,v) \text{ 为树枝边(父子边)} \}$

一个顶点 u 是割点，当且仅当满足(1)或(2) (1) u 为树根，且 u 有多于一个子树。(2) u 不为树根，且满足存在 (u,v) 为树枝边(或称父子边，即 u 为 v 在搜索树中的父亲)，使得 $DFS(u) \leq Low(v)$ 。

一条无向边 (u,v) 是桥，当且仅当 (u,v) 为树枝边，且满足 $DFS(u) < Low(v)$ 。

[求双连通分支]

下面要分开讨论点双连通分支与边双连通分支的求法。

对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储当前双连通分支，在搜索图时，每找到一条树枝边或后向边(非横叉边)，就把这条边加入栈中。如果遇到某时满足

DFS(u) \leq Low(v), 说明 u 是一个割点, 同时把边从栈顶一个个取出, 直到遇到了边 (u, v), 取出的这些边与其关联的点, 组成一个点双连通分支。割点可以属于多个点双连通分支, 其余点和每条边只属于且属于一个点双连通分支。

对于边双连通分支, 求法更为简单。只需在求出所有的桥以后, 把桥边删除, 原图变成了多个连通块, 则每个连通块就是一个边双连通分支。桥不属于任何一个边双连通分支, 其余的边和每个顶点都属于且只属于一个边双连通分支。

【构造双连通图】

一个有桥的连通图, 如何把它通过加边变成边双连通图? 方法为首先求出所有的桥, 然后删除这些桥边, 剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点, 再把桥边加回来, 最后的这个图一定是一棵树, 边连通度为 1。

统计出树中度为 1 的节点的个数, 即为叶节点的个数, 记为 leaf。则至少在树上添加 (leaf+1)/2 条边, 就能使树达到边二连通, 所以至少添加的边数就是 (leaf+1)/2。具体方法为, 首先把两个最近公共祖先最远的两个叶节点之间连接一条边, 这样可以把这两个点到祖先的路径上所有点收缩到一起, 因为一个形成的环一定是双连通的。然后再找两个最近公共祖先最远的两个叶节点, 这样一对一对找完, 恰好是 (leaf+1)/2 次, 把所有点收缩到了一起。

二 网络流和匹配

1. 最大流 Dinic

```
1 typedef ____ Ftype; // 流的数据类型
2 const int maxn = ____, maxm = ____;
3 const Ftype inf = ____;
4
5 // 使用之前必须 init(n)
```

```
6 struct FlowNetwork
7 {
8     struct edge
9     {
10         int s, t, nxt;
11         Ftype c, f;
12     } e[maxm];
13
14     int n, m, s, t;
15     int hd[maxn], cur[maxn], dep[maxn];
16     Ftype max_flow;
17     bool vis[maxn];
18     void init(int n)
19     {
20         this->n = n;
21         memset(hd, -1, sizeof hd);
22         m = max_flow = 0;
23     }
24     void AddEdge(int from, int to, Ftype cap)
25     {
26         e[m] = (edge){from, to, hd[from], cap, 0}; hd[from] = m; m++;
27         e[m] = (edge){to, from, hd[to], 0, 0}; hd[to] = m; m++;
28     }
29     bool BFS()
30     {
31         memset(vis, 0, sizeof(vis));
32         queue<int> Q;
33         Q.push(s);
34         dep[s] = 0, vis[s] = 1;
35         while (!Q.empty())
36         {
37             int x = Q.front();
38             Q.pop();
39             for (int i = hd[x]; i != -1; i = e[i].nxt)
40             {
41                 if (!vis[e[i].t] && e[i].c > e[i].f)
42                 {
43                     vis[e[i].t] = 1;
44                     dep[e[i].t] = dep[x] + 1;
45                     Q.push(e[i].t);
46                 }
47             }
48         }
49         return vis[t];
50     }
51
52     int DFS(int x, Ftype a)
53     {
54         if (x == t || a == 0) return a;
```



```

55     Ftype flow = 0, f;
56     for (; cur[x] != -1; cur[x] = e[cur[x]].nxt)
57     {
58         int i = cur[x];
59         if (dep[x] + 1 == dep[e[i].t])
60         && (f = DFS(e[i].t, min(a, e[i].c - e[i].f))) > 0)
61         {
62             e[i].f += f;
63             e[i^1].f -= f;
64             flow += f;
65             a -= f;
66             if (a == 0) break;
67         }
68     }
69     return flow;
70 }
71
72 vector<int> minCut() // 最小割边集
73 {
74     BFS();
75     vector<int> ans;
76     for (int i = 0; i < m; i += 2)
77         if (vis[e[i].s] && !vis[e[i].t]) ans.push_back(i);
78     return ans;
79 }
80
81 Ftype maxFlow(int s, int t) // 最大流
82 {
83     this->s = s; this->t = t;
84     Ftype flow = 0;
85
86     while (BFS())
87     {
88         memcpy(cur, hd, sizeof hd);
89         flow += DFS(s, inf);
90     }
91     return max_flow += flow;
92 }
93 void reset() // 重置 (不删边)
94 {
95     max_flow = 0;
96     for (int i = 0; i < m; i++) e[i].f = 0;
97 }
98 };

```

2. 最小费用最大流 SPFA 增广 (稀疏图较快)

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <cstdlib>
5  #include <algorithm>
6  #include <deque>
7  using namespace std;
8
9  const int maxn = _____, maxm = _____;
10 typedef _____ Ftype;
11 Ftype inf = _____;
12 struct edge
13 {
14     int from, to;
15     Ftype cap, flow, cost;
16     int next;
17 };
18
19 struct MCMF
20 {
21     int n, m, s, t, hd[maxn], inq[maxn], p[maxn];
22     edge e[maxm];
23     Ftype a[maxn], d[maxn];
24
25     void init(int n)
26     {
27         this->n = n;
28         memset(hd, -1, sizeof hd);
29         m = 0;
30     }
31     void addEdge(int from, int to, Ftype cap, Ftype cost)
32     {
33         e[m] = (edge){from, to, cap, 0, cost, hd[from]}; hd[from] =
34         m; m++;
35         e[m] = (edge){to, from, 0, 0, -cost, hd[to]}; hd[to] = m; m+
36         ++;
37     }
38
39     bool SPFA(int s, int t, Ftype &flow, Ftype &cost)
40     {
41         for (int i = 0; i < n; i++) d[i] = inf;
42         memset(inq, 0, sizeof inq);
43         d[s] = 0, inq[s] = 1, p[s] = 0, a[s] = inf;
44         deque<int> Q;
45         Q.push_back(s);
46         while (!Q.empty())

```

```

5      {
6          int u = Q.front(); Q.pop_front();
7          inq[u] = false;
8          for (int i = hd[u]; i != -1; i = e[i].next)
9              {
10                 if (e[i].cap > e[i].flow && d[e[i].to] > d[u] +
11 e[i].cost)
12                     {
13                         d[e[i].to] = d[u] + e[i].cost;
14                         p[e[i].to] = i;
15                         a[e[i].to] = min(a[u], e[i].cap - e[i].flow);
16                         if (!inq[e[i].to])
17                             {
18                                 if (d[e[i].to] <= d[u])
19 Q.push_front(e[i].to);
20                                 else Q.push_back(e[i].to);
21                                 inq[e[i].to] = true;
22                             }
23                     }
24             }
25         }
26         if (d[t] == inf) return false;
27         flow += a[t];
28         cost += d[t] * a[t];
29         int u = t;
30         while (u != s)
31             {
32                 e[p[u]].flow += a[t];
33                 e[p[u]^1].flow -= a[t];
34                 u = e[p[u]].from;
35             }
36         return true;
37     }
38     Ftype minCost(int s, int t)
39     {
40         Ftype flow = 0, cost = 0;
41         while (SPFA(s, t, flow, cost))
42             ;
43         return cost;
44     }
45 };

```

```

3 #include <cstring>
4 const int maxn = 1010, maxm = 510*510;
5 struct edge
6 {
7     int s, t, next;
8 } e[maxm];
9 struct vtx
10 {
11     int used, match;
12 } ver[maxn];
13 int hd[maxn], ecnt;
14 bool maxMatchDFS(int v)
15 {
16     for (int i = hd[v]; i != -1; i = e[i].next) if(!ver[u].used)
17     {
18         int u = e[i].t, w = ver[e[i].t].match;
19         ver[u].used = true;
20         if (w == -1 || (!ver[w].used && maxMatchDFS(w)))
21             {
22                 ver[v].match = u, ver[u].match = v;
23                 return true;
24             }
25     }
26     return false;
27 }
28 int maxMatch(int n)
29 {
30     int match = 0;
31     for (int i = 1; i <= n; i++)
32     {
33         for (int j = n+1; j <= (n<<1); j++) ver[j].used = 0;
34         if (maxMatchDFS(i)) match++;
35     }
36     return match;
37 }
38 void init(int n)
39 {
40     memset(hd, -1, sizeof hd);
41     ecnt = 0;
42     for (int i = 1; i <= (n<<1); i++)
43         ver[i].match = -1, ver[i].used = false;
44 }
45 void addEdge(int a, int b)
46 {
47     e[ecnt].s = a, e[ecnt].t = b; e[ecnt].next = hd[a];
48     hd[a] = ecnt++;
49 }

```

3. 二分图最大匹配 匈牙利算法 $O(NM)$

```

1 #include <iostream>
2 #include <cstdio>

```

4. 混合图欧拉回路建边

基础知识

欧拉回路是图 G 中的一个回路，经过每条边有且仅一次，称该回路为欧拉回路。具有欧拉回路的图称为欧拉图，简称 E 图。

无向图中存在欧拉回路的条件：每个点的度数均为偶数。

有向图中存在欧拉回路的条件：每个点的入度=出度。

欧拉路径比欧拉回路要求少一点：

无向图中存在欧拉路径的条件：每个点的度数均为偶数或者有且仅有 2 个度数为奇数的点。

有向图中存在欧拉路径的条件：除了 2 个点外，其余的点入度=出度，且在这 2 个点中，一个点的入度比出度大 1，另一个出度比入度大 1。

欧拉路径的输出：经典的套圈算法。

下面来重点讲讲混合图的欧拉回路问题。

混合图就是边集中有有向边和无向边同时存在。这时候需要用网络流建模求解。

建模：

把该图的无向边随便定向，计算每个点的入度和出度。如果有某个点出入度之差为奇数，那么肯定不存在欧拉回路。因为欧拉回路要求每点入度 = 出度，也就是总度数为偶数，存在奇数度点必不能有欧拉回路。

好了，现在每个点入度和出度之差均为偶数。那么将这个偶数除以 2，得 x 。也就是说，对于每一个点，只要将 x 条边改变方向（入 > 出就是变入，出 > 入就是变出），就能保证出 = 入。如果每个点都是出 = 入，那么很明显，该图就存在欧拉回路。

现在的问题就变成了：我该改变哪些边，可以让每个点出 = 入？构造网络流模型。

首先，有向边是不能改变方向的，要之无用，删。一开始不是把无向边定向了吗？定的是什么向，就把网络构建成什么样，边长容量上限 1。另新建 s 和 t 。对于入 > 出的点 u ，连接边(u, t)、容量为 x ，对于出 > 入的点 v ，连接边(s, v)，容量为 x （注意对不同的点 x 不同）。

之后，察看从 S 发出的所有边是否满流。有就是能有欧拉回路，没有就是没有。

欧拉回路是哪个？察看流值分配，将所有流量非 0（上限是 1，流值不是 0 就是 1）的边反向，就能得到每点入度 = 出度的欧拉图。

由于是满流，所以每个入 > 出的点，都有 x 条边进来，将这些进来的边反向，OK，入 = 出了。对于出 > 入的点亦然。那么，没和 s 、 t 连接的点怎么办？和 s 连接的条件是出 > 入，和 t 连接的条件是入 > 出，那么这个既没和 s 也没和 t 连接的点，自然早在开始就已经满足入 = 出了。那么在网络流过程中，这些点属于“中间点”。我们知道中间点流量不允许有累积的，这样，进去多少就出来多少，反向之后，自然仍保持平衡。

所以，就这样，混合图欧拉回路问题，解了。

例：HDU3472

题意：给出一些单词，其中有些单词反转之后也是有意义的单词，问是否能将所有单词首尾相连，每个单词用 1 次且仅用 1 次。

解：这题是混合路的欧拉路径问题。

1. 首先判断图的连通性，若不连通，无解。

2. 然后任意定向无向边，计算每个点 i 的入度和出度之差 $\deg[i]$ 。若 $\deg[i]$ 为奇数，无解。

3. 设立源点 s 和汇点 t ，若某点 i 入度 < 出度，连边($s, i, -\deg[i]/2$)，若入度 > 出度，连边($i, t, \deg[i]/2$)；对于任意定向的无向边($i, j, 1$)。

4. 若有两个度数为奇数的点，假设存在欧拉路径，添加一条容量为 1 的边，构成欧拉回路，不影响结果。若全为偶数，直接最大流。

5. 若从 S 发出的边全部满流，证明存在欧拉回路(路径)，否则不存在。

ps：若要求输出路径，将网络中有(无)流量的边反向，加上原图的有向边，用套圈算法即可。

5. 全局最小割

```
1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4 #include <algorithm>
5 using namespace std;
6 const int INF = 0x3f3f3f3f;
7 const int maxn = 510;
```

```

8 struct UMinCut
9 {
10     int c[maxn][maxn], n;
11     int cut;
12     int id[maxn], b[maxn];
13     void init(int _n)
14     {
15         memset(c, 0, sizeof c);
16         n = _n;
17     }
18     int minCut()
19     {
20         for (int i = 0; i < n; i++) id[i] = i;
21         cut = INF;
22         for (; n > 1; n--)
23         {
24             memset(b, 0, sizeof b);
25             for (int i = 0; i + 1 < n; i++)
26             {
27                 int p = i + 1;
28                 for (int j = i + 1; j < n; j++)
29                 {
30                     b[id[j]] += c[id[i]][id[j]];
31                     if (b[id[p]] < b[id[j]]) p = j;
32                 }
33                 swap(id[i+1], id[p]);
34             }
35             cut = min(cut, b[id[n-1]]);
36             for (int i = 0; i < n-2; i++)
37             {
38                 c[id[i]][id[n-2]] += c[id[i]][id[n-1]];
39                 c[id[n-2]][id[i]] += c[id[n-1]][id[i]];
40             }
41         }
42         return cut;
43     }
44 } ccc;
45
46 int main()
47 {
48     int n, m;
49
50     while (~scanf("%d%d", &n, &m))
51     {
52         ccc.init(n);
53         for (int i = 0; i < m; i++)
54         {
55             int u, v, w;
56             scanf("%d%d%d", &u, &v, &w);

```

```

57         ccc.c[u][v] += w;
58         ccc.c[v][u] += w;
59     }
60     printf("%d\n", ccc.minCut());
61 }
62
63 return 0;
64 }

```

三 最短路

1. Dijkstra 复杂度上限 $O(M \log N)$

```

1 memset(use, 0x3f, sizeof(use));
2 priority_queue < pair < int, int > > q;
3 use[s] = 0;
4 q.push(mp(0, s));
5 while (!q.empty())
6 {
7     int cur = q.top().B;
8     q.pop();
9     if (use[cur] != -q.top().A)
10         continue;
11     for (unsigned i = 0; i < g[cur].size(); i++)
12         if (use[g[cur][i].B] > use[cur] + g[cur][i].A)
13         {
14             use[g[cur][i].B] = use[cur] + g[cur][i].A;
15             q.push(mp(-use[g[cur][i].B], g[cur][i].B));
16         }
17 }

```

2. SPFA 估计复杂度 $O(2M)$ 带 SLF 优化

```

1 deque < int > q;
2 use[s] = 0;
3 q.push_back(s);
4 memset(inq, 0, sizeof(inq));
5 inq[s] = true;
6 while (!q.empty())
7 {
8     int cur = q.front();
9     inq[cur] = false;

```

```

10     q.pop_front();
11     for (unsigned i = 0; i < g[cur].size(); i++)
12         if (use[g[cur][i].B] > use[cur] + g[cur][i].A)
13             {
14                 use[g[cur][i].B] = use[cur] + g[cur][i].A;
15                 if (!inq[g[cur][i].B])
16                     {
17                         if (!q.empty() && use[g[cur][i].B] <=
18 use[q.front()])
19                             q.push_front(g[cur][i].B);
20                         else q.push_back(g[cur][i].B);
21                         inq[g[cur][i].B] = true;
22                     }
23             }
24 }

```

3. 差分约束系统

根据二元约束关系建图，设源点 $D[s] = 0$ 。

约束关系表示	问题	算法
$D[j] - D[i] \leq C$	约束关系下的最大值	最短路
$D[j] - D[i] \geq C$	约束关系下的最小值	最长路

4. 次短路径

次短路径可以看作是 k 短路径问题的一种特殊情况，求 k 短路径有 Yen 算法等较为复杂的方法，对于次短路径，可以有更为简易的方法。下面介绍一种求两个顶点之间次短路径的解法。

我们要对一个有向赋权图(无向图每条边可以看作两条相反的有向边)的顶点 s 到 t 之间求次短路径，首先应求出 s 的单源最短路径。遍历有向图，标记出可以在最短路径上的边，加入集合 K 。然后枚举删除集合 K 中每条边，求从 s 到 t 的最短路径，记录每次求出的路径长度值，其最小值就是次短路径的长度。

在这里我们以为次短路径长度可以等于最短路径长度，如果想等，也可以

看作是从 s 到 t 有不只一条最短路径。如果我们规定求从 s 到 t 大于最短路径长度的次短路径，则答案就是每次删边后大于原最短路径的 s 到 t 的最短路径长度的最小值。

用 Dijkstra+堆求单源最短路径，则每次求最短路径时间复杂度为 $O(N \log(N+M) + M)$ ，所以总的时间复杂度为 $O(NM \log(N+M) + M^2)$ 。该估计是较为悲观的，因为一般来说，在最短路径上的边的条数要远远小于 M ，所以实际效果要比预想的好。

四 生成树

次小生成树

类比上述次短路径求法，很容易想到一个“枚举删除最小生成树上的每条边，再求最小生成树”的直观解法。如果用 Prim+堆，每次最小生成树时间复杂度为 $O(N \log(N+M) + M)$ ，枚举删除有 $O(N)$ 条边，时间复杂度就是 $O(N^2 \log(N+M) + N \cdot M)$ ，当图很稠密时，接近 $O(N^3)$ 。这种方法简易直观，但我们有一个更简单，而且效率更高的 $O(N^2 + M)$ 的解法，下面介绍这种方法。

首先求出原图最小生成树，记录权值之和为 MinST 。枚举添加每条不在最小生成树上的边 (u, v) ，加上以后一定会形成一个环。找到环上权值第二大的边(即除了 (u, v) 以外的权值最大的边)，把它删掉，计算当前生成树的权值之和。取所有枚举修改的生成树权值之和的最小值，就是次小生成树。

具体实现时，更简单的方法是从每个节点 i 遍历整个最小生成树，定义 $F[j]$ 为从 i 到 j 的路径上最大边的权值。遍历图求出 $F[j]$ 的值，然后对于添加每条不在最小生成树中的边 (i, j) ，新的生成树权值之和就是 $\text{MinST} + w(i, j) - F[j]$ ，记录其最小值，则为次小生成树。

该算法的时间复杂度为 $O(N^2 + M)$ 。由于只用求一次最小生成树，可以用

最简单的 Prim，时间复杂度为 $O(N^2)$ 。算法的瓶颈不在求最小生成树，而在 $O(N^2+M)$ 的枚举加边修改，所以用更好的最小生成树算法是没有必要的。

计算几何

五 树的直径

(调用时语句: `dis[bfs(bfs(1))]`; 注意初始化等问题)

```

1  const int maxn = 100010;
2  vector<int> g[maxn];
3  int dis[maxn], vis[maxn];
4
5  int bfs(int start)
6  {
7      queue<int> q;
8      q.push(start);
9      vis[start] = 1;
10     memset(dis, 0x3f, sizeof dis);
11     memset(vis, 0, sizeof vis);
12     dis[start] = 0;
13     int p = 0;
14     while (!q.empty())
15     {
16         int cur = q.front();
17         p = cur;
18         q.pop();
19         for (int i = 0; i < (int)g[cur].size(); i++)
20         {
21             if (!vis[g[cur][i]])
22             {
23                 q.push(g[cur][i]);
24                 vis[g[cur][i]] = 1;
25                 dis[g[cur][i]] = dis[cur] + 1;
26             }
27         }
28     }
29     return p;
30 }
```

一 二维几何

1. 基本操作 【部分未验证】

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <vector>
5  #include <algorithm>
6  #include <cmath>
7  using namespace std;
8
9  const double eps = 1e-11;
10
11 struct point
12 {
13     double x, y;
14     point () {}
15     point (double _x, double _y) : x(_x), y(_y) {}
16
17     point operator+ (point b)
18     {
19         return point(x + b.x, y + b.y);
20     }
21     point operator- (point b)
22     {
23         return point(x - b.x, y - b.y);
24     }
25     point operator* (double k)
26     {
27         return point(k*x, k*y);
28     }
29     point operator/ (double k)
30     {
31         return point(x/k, y/k);
32     }
33
34     double dot(point b)
35     {
36         return x * b.x + y * b.y;
37     }
38     double cross(point b)
```

```

2      {
3          return x * b.y - y * b.x;
4      }
5
6      double dist()
7      {
8          return sqrt(x*x + y*y);
9      }
10     double dist2()
11     {
12         return x*x+y*y;
13     }
14
15     inline int getQuad()
16     {
17         if (x > eps && y > -eps) return 1;
18         else if (x < eps && y > eps) return 2;
19         else if (x < -eps && y < eps) return 3;
20         else return 4;
21     }
22
23     void rotate90()
24     {
25         double nx = x, ny = y;
26         x = -ny, y = nx;
27     }
28 };
29
30 struct line
31 {
32     point a, b;
33     line () {}
34     line (point _a, point _b): a(_a), b(_b) {}
35 };
36 typedef line segment;
37
38 double distSegmentPoint(point, segment);
39 double distLinePoint(point, line);
40 bool judSegmentInt(segment, segment);
41 point getLineInt(line, line);
42
43 struct circle
44 {
45     point centre;
46     double r;
47
48     circle (point _c, double _r = 0)
49     {
50         centre = _c, r = _r;

```

```

51     }
52     circle (point a, point b)
53     {
54         centre.x = (a.x + b.x) / 2;
55         centre.y = (a.y + b.y) / 2;
56         r = (a-b).dist() / 2;
57     }
58
59     circle (point a, point b, point c)
60     {
61         point u1((a.x + b.x) / 2, (a.y + b.y) / 2), v1;
62         v1.x = u1.x - (b.y - a.y), v1.y = u1.y + (b.x - a.x);
63
64         point u2((a.x + c.x) / 2, (a.y + c.y) / 2), v2;
65         v2.x = u2.x - (c.y - a.y), v2.y = u2.y + (c.x - a.x);
66
67         centre = getLineInt(line(u1, v1), line(u2, v2));
68         r = (centre - a).dist();
69     }
70
71     bool inCircle(point px)
72     {
73         if ((px - centre).dist() < r + eps)
74             return true;
75         else
76             return false;
77     }
78
79     bool judIntSegment(segment s)
80     {
81         return distSegmentPoint(centre, s) < r + eps &&
82             (r < (centre - s.a).dist() + eps || r < (centre -
83 s.b).dist() + eps);
84     }
85
86     bool judIntCircle(circle b)
87     {
88         double dis = (centre - b.centre).dist();
89         return dis < r + b.r + eps && fabs(r - b.r) < dis + eps;
90     }
91
92     int getIntLine(line l, vector<point> &ret)
93     {
94         double x = (l.a - centre).dot(l.b - l.a);
95         double y = (l.b - l.a).dist2();
96         double d = x*x - y*((l.a - centre).dist2() - r*r);
97         if (d < -eps) return 0;
98         if (d < 0) d = 0;
99         point q1 = l.a - ((l.b - l.a)*(x/y));

```



```

100     point q2 = l.b - (l.a*(sqrt(d)/y));
101     ret.push_back(q1);
102     ret.push_back(q2);
103     return 2;
104 }
105
106 int getIntCircle(circle &c2, vector <point> &ret)
107 {
108     double x = (centre - c2.centre).dist2();
109     double y = ((r*r - c2.r*c2.r) / x + 1.0) / 2.0;
110     double d = r*r/x - y*y;
111     if (d < -eps) return 0;
112
113     point q1 = centre + (c2.centre - centre) * y;
114     if (d < eps)
115     {
116         ret.push_back(q1);
117         return 1;
118     }
119     point q2 = (c2.centre - centre) * sqrt(d);
120     ret.push_back(q1 - q2);
121     ret.push_back(q1 + q2);
122     return 2;
123 }
124
125 int getTangentFromP(Point p, vector <point> &ret)
126 {
127     double x = (p - centre).dist2();
128     double d = x - r*r;
129     if (d < -eps) return 0;
130
131     if (d < 0) d = 0;
132 }
133 };
134
135 double distSegmentPoint(point p, segment s)
136 {
137     if ((s.b - s.a).dot(p - s.a) < eps) return (p - s.a).dist();
138     if ((s.a - s.b).dot(p - s.b) < eps) return (p - s.b).dist();
139     return distLinePoint(p, s);
140 }
141
142 double distLinePoint(point p, line l)
143 {
144     return abs((l.b - l.a).cross(p - l.a)) / (l.b - l.a).dist();
145 }
146
147 bool onSegment(segment a, point b)

```

```

149 {
150     return (min(a.a.x, a.b.x) < b.x + eps) && (max(a.a.x, a.b.x) >
151 b.x - eps)
152         && (min(a.a.y, a.b.y) < b.y + eps) && (max(a.a.y, a.b.y)
153 > b.y - eps);
154 }
155 bool judSegmentInt(segment a, segment b)
156 {
157     double d1, d2, d3, d4;
158     d1 = (b.a - a.a).cross(a.b - a.a);
159     d2 = (b.b - a.a).cross(a.b - a.a);
160     d3 = (a.a - b.a).cross(b.b - b.a);
161     d4 = (a.b - b.a).cross(b.b - b.a);
162
163     if (((d1 > eps && d2 < -eps) || (d1 < -eps && d2 > eps))
164         && ((d3 > eps && d4 < -eps) || (d3 < -eps && d4 > eps)))
165         return true;
166
167     if (fabs(d1) < eps) return onSegment(a, b.a);
168     if (fabs(d2) < eps) return onSegment(a, b.b);
169     if (fabs(d3) < eps) return onSegment(b, a.a);
170     if (fabs(d4) < eps) return onSegment(b, a.b);
171     return false;
172 }
173 point getLineInt(line a, line b)
174 {
175     point ret = a.a;
176
177     double t = ((a.a.x - b.a.x) * (b.a.y - b.b.y) - (a.a.y - b.a.y)
178 * (b.a.x - b.b.x))
179 / ((a.a.x - a.b.x) * (b.a.y - b.b.y) - (a.a.y - a.b.y)
180 * (b.a.x - b.b.x));
181
182     ret.x += (a.b.x - a.a.x) * t;
183     ret.y += (a.b.y - a.a.y) * t;
184     return ret;
185 }
186
187 point getProj(line a, point p)
188 {
189     return a.a + ((a.b - a.a)*((a.b - a.a).dot(p - a.a) / (a.b -
190 a.a).dist()));
191 }

```


2. 凸包 $O(N \log N)$

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cmath>
4  #include <cstring>
5  #include <algorithm>
6  #define MAXN 10001
7  #define eps 1e-9
8  #define SQ(x) ((x)*(x))
9  using namespace std;
10
11 struct point
12 {
13     double x, y;
14     point (double _x = 0, double _y = 0)
15     {
16         x = _x, y = _y;
17     }
18     double operator * (point b)
19     {
20         return x * b.x + y * b.y;
21     }
22     double cross (point b)
23     {
24         return x * b.y - y * b.x;
25     }
26 }
27
28 point operator - (point b)
29 {
30     return point(x - b.x, y - b.y);
31 }
32 };
33 point pts[MAXN];
34 point _Basis;
35 double direction(point a, point b, point c)
36 {
37     return (c - a).cross(b - a);
38 }
39
40 bool cmp(point a, point b)
41 {
42     if (fabs(direction(_Basis, a, b)) < eps)
43         return SQ(a - _Basis) < SQ(b - _Basis) - eps;
44     return direction(_Basis, a, b) < -eps;
45 }
46

```

```

47 void getConvex(point p[], int n, point *convex, int &hv)
48 {
49
50     int Basis = 0, Top;
51     for (int i = 0; i < n; i++)
52     {
53         if (fabs(p[i].y - p[Basis].y) < eps)
54             if (p[i].x < p[Basis].x - eps)
55                 Basis = i;
56         Basis = p[i].y < p[Basis].y - eps ? i : Basis;
57     }
58     _Basis = p[Basis];
59     swap(p[0], p[Basis]);
60     sort(p + 1, p + n, cmp);
61     for (int i = 0; i < 3; i++)
62         convex[i] = p[i];
63     Top = 2;
64     for (int i = 3; i < n; i++)
65     {
66         while (direction(convex[Top - 1], convex[Top], p[i]) > -eps
67             && Top > 0)
68             Top--;
69         convex[++Top] = p[i];
70     }
71     hv = Top + 1;
72 }
73
74 double getConvexArea(point conv[], int num) // 计算凸包面积
75 {
76     double ans = 0;
77     for (int i = 0; i < num; i++)
78         ans += direction(point(0, 0), conv[i], conv[(i + 1) % num]);
79     return fabs(ans / 2.0); // 计算结果必须 / 2.0
80 }

```

3. 旋转卡壳

```

1 // 用以求凸包上距离最远两点的距离
2 // 如果有三点共线的情况，取消代码注释部分
3 // 返回 ans 最大两点距离平方
4 double getMaxSQDistance(/*point p[], int n,*/ point conv[], int num)
5 {
6     /*
7     if (num <= 1)
8     {
9         double ret = 0;
10        for (int i = 0; i < n; i++)

```

```

15         ret = max(ret, SQ(p[i] - p[0]));
16     return ret;
17 }*/
18 int q = 1;
19 double ans = 0;
20 for (int i = 0; i < num; i++)
21 {
22     while (abs(direction(conv[i], conv[i + 1], conv[q]))
23         < abs(direction(conv[i], conv[i + 1], conv[q +
24 1]))-eps)
25         q = (q + 1) % num;
26     ans = max(ans, max(SQ(conv[i] - conv[q]), SQ(conv[i + 1] -
27 conv[q + 1])));
28 }
29 return ans;
30 }

```

4. 半平面交 $O(N \log N)$

```

1 // POJ 2451 Uyuw's Concert
2 // POJ 3335
3 // POJ 1279
4 #include <iostream>
5 #include <vector>
6 #include <cstdio>
7 #include <cmath>
8 #include <cstring>
9 #include <algorithm>
10 #define SQ(x) ((x)*(x))
11 #define A first
12 #define B second
13 using namespace std;
14
15 const double eps = 1e-9;
16 const double pi = acos(-1.0);
17 const int maxn = 1010;
18
19 struct point
20 {
21     double x, y;
22     point (double _x = 0, double _y = 0)
23     {
24         x = _x, y = _y;
25     }
26     point operator * (double b)
27     {

```

```

28         return point(x*b, y*b);
29     }
30
31     inline double dot(point b)
32     {
33         return x * b.x + y * b.y;
34     }
35
36     inline double cross (point b)
37     {
38         return x * b.y - y * b.x;
39     }
40
41     point operator - (point b)
42     {
43         return point(x - b.x, y - b.y);
44     }
45
46     inline int getQuad()
47     {
48         if (x > eps && y > -eps) return 1;
49         else if (x < eps && y > eps) return 2;
50         else if (x < -eps && y < eps) return 3;
51         else return 4;
52     }
53     inline double dist()
54     {
55         return x*x + y*y;
56     }
57 };
58 typedef pair <point, point> Line;
59
60 point pts[maxn];
61 Line line[maxn];
62
63 double direction(point a, point b, point c)
64 {
65     return (c - a).cross(b - a);
66 }
67
68 bool cmpPointbyAngle(point a, point b)
69 {
70     int q1 = a.getQuad(), q2 = b.getQuad();
71     if (q1 == q2)
72     {
73         if (fabs(a.cross(b)) < eps)
74             return a.dist() < b.dist();
75
76         return a.cross(b) > eps;

```

```

77     }
78     return q1 < q2;
79 }
80
81 bool cmpLinebyAngle(Line a, Line b)
82 {
83     point v1 = a.B - a.A, v2 = b.B - b.A;
84
85     return cmpPointbyAngle(v1, v2);
86 }
87
88 bool onLeft(Line L, point p)
89 {
90     return (L.B - L.A).cross(p - L.A) > eps; // 注意: 如果相交的点
91     也算作半平面交的解 将此句改为 (L.B - L.A).cross(p - L.A) > -eps
92 }
93
94 bool isParallel(Line a, Line b) /// 平行且同向
95 {
96     return fabs((a.B - a.A).cross(b.B - b.A)) < eps &&
97     (a.B - a.A).dot(b.B - b.A) > eps;
98 }
99
100 point lineInt(Line a, Line b)
101 { // 注意 事先必须判断两直线不平行
102     point ret = a.A;
103
104     double t = ((a.A.x - b.A.x) * (b.A.y - b.B.y) - (a.A.y - b.A.y)
105     * (b.A.x - b.B.x))
106     / ((a.A.x - a.B.x) * (b.A.y - b.B.y) - (a.A.y - a.B.y)
107     * (b.A.x - b.B.x));
108
109     ret.x += (a.B.x - a.A.x) * t;
110     ret.y += (a.B.y - a.A.y) * t;
111     return ret;
112 }
113
114 point intersect[maxn];
115 Line my_deque[maxn];
116
117 int halfplaneIntersection(Line *L, int n, vector<point> &poly)
118 {
119     sort(L, L + n, cmpLinebyAngle);
120
121     int first, last;
122
123     point *p = intersect;
124     Line *q = my_deque;
125     q[first = last = 0] = L[0];

```

```

126
127     /** p 为双端队列对应的交点 p[i-1] = q[i] 交 q[i-1] */
128
129     for (int i = 1; i < n; i++)
130     {
131         /** 消除无用直线, 加入新直线 */
132         while (first < last && !onLeft(L[i], p[last-1])) last--;
133         while (first < last && !onLeft(L[i], p[first])) first++;
134         q[++last] = L[i];
135
136         /** 判断平行, 平行时保留内侧直线 */
137         if (first < last && isParallel(q[last], q[last-1]))
138         {
139             last--;
140             if (onLeft(q[last], L[i].A)) q[last] = L[i];
141         }
142         if (first < last)
143             p[last-1] = lineInt(q[last-1], q[last]);
144     }
145     while (first < last && !onLeft(q[first], p[last-1])) last--;
146     if (last - first <= 1) return 0;
147     p[last] = lineInt(q[last], q[first]);
148     for (int i = first; i <= last; i++) poly.push_back(p[i]);
149     return (int)poly.size();
150 }

```

5. 最小包围圆 O(N)

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cmath>
4  #include <cstring>
5  #include <algorithm>
6  #define MAXN 50001
7  #define eps 1e-9
8  #define SQ(x) ((x)*(x))
9  using namespace std;
10
11  int n;
12
13  struct point
14  {
15      double x, y;
16      point (double _x = 0, double _y = 0)
17      {
18          x = _x, y = _y;
19      }

```

```

24 double operator * (point b)
25 {
26     return x * b.x + y * b.y;
27 }
28 double cross (point b)
29 {
30     return x * b.y - y * b.x;
31 }
32
33 point operator - (point b)
34 {
35     return point(x - b.x, y - b.y);
36 }
37 };
38
39 point lineInt(point a1, point a2, point b1, point b2)
40 { // 注意 事先必须判断两直线不平行
41     point ret = a1;
42
43     double t = ((a1.x - b1.x) * (b1.y - b2.y) - (a1.y - b1.y) *
44 (b1.x - b2.x))
45 /((a1.x - a2.x) * (b1.y - b2.y) - (a1.y - a2.y) *
46 (b1.x - b2.x));
47
48     ret.x += (a2.x - a1.x) * t;
49     ret.y += (a2.y - a1.y) * t;
50     return ret;
51 }
52
53 double dist(point a, point b)
54 {
55     return sqrt(SQ(a - b));
56 }
57
58 struct circle
59 {
60     point centre;
61     double r;
62
63     circle (point _c, double _r = 0)
64     {
65         centre = _c, r = _r;
66     }
67     circle (point a, point b)
68     {
69         centre.x = (a.x + b.x) / 2;
70         centre.y = (a.y + b.y) / 2;
71         r = dist(a, b) / 2;
72     }

```

```

73     circle (point a, point b, point c)
74     {
75         point u1((a.x + b.x) / 2, (a.y + b.y) / 2), v1;
76         v1.x = u1.x - (b.y - a.y), v1.y = u1.y + (b.x - a.x);
77
78         point u2((a.x + c.x) / 2, (a.y + c.y) / 2), v2;
79         v2.x = u2.x - (c.y - a.y), v2.y = u2.y + (c.x - a.x);
80
81         centre = lineInt(u1, v1, u2, v2);
82         r = dist(centre, a);
83     }
84
85     bool inCircle(point px)
86     {
87         if (sqrt(SQ(px - centre)) < r + eps)
88             return true;
89         else
90             return false;
91     }
92 };
93
94 double direction(point a, point b, point c)
95 {
96     return (c - a).cross(b - a);
97 }
98
99 point pt[MAXN];
100 int Basis, Top;
101
102 circle minCoverDisc(point p[], int n)
103 {
104     random_shuffle(p, p + n);
105
106     if (n == 1)
107         return circle(p[0], 0);
108     circle ret(p[0], p[1]);
109
110     for (int i = 2; i < n; i++)
111     {
112         if (ret.inCircle(p[i]))
113             continue;
114         ret = circle(p[0], p[i]);
115         for (int j = 1; j < i; j++)
116         {
117             if (ret.inCircle(p[j]) || i == j)
118                 continue;
119             ret = circle(p[i], p[j]);
120
121             for (int k = 0; k < j; k++)

```

```

122     {
123         if (ret.inCircle(p[k]) || k == j || k == i)
124             continue;
125         ret = circle(p[i], p[j], p[k]);
126     }
127 }
128 }
129 return ret;
130 }
131
132 int main()
133 {
134     int n;
135     //freopen("10005.in", "r", stdin);
136     while (1)
137     {
138         scanf("%d", &n);
139         if (!n) break;
140         for (int i = 0; i < n; i++)
141         {
142             scanf("%lf%lf", &pt[i].x, &pt[i].y);
143         }
144
145         double rr = 0;
146         circle c = minCoverDisc(pt, n);
147         scanf("%lf", &rr);
148
149         if (rr < c.r - eps)
150         {
151             printf("There is no way of packing that polygon.\n");
152         }
153         else
154         {
155             printf("The polygon can be packed in the circle.\n");
156         }
157         printf("(%.3f, %.3f)   r = %.11f\n", c.centre.x, c.centre.y,
158 c.r);
159     }
160     return 0;
161 }

```

二 三维几何

1. 基本操作（点和线等）

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <cmath>
5  #include <vector>
6  using namespace std;
7  const double eps = 0;
8  struct point3D
9  {
10     double x, y, z;
11     point3D (double _x = 0, double _y = 0, double _z = 0)
12     {
13         x = _x, y = _y, z = _z;
14     }
15     point3D operator + (point3D b)
16     {
17         return point3D(x + b.x, y + b.y, z + b.z);
18     }
19
20     point3D operator - (point3D b)
21     {
22         return point3D(x - b.x, y - b.y, z - b.z);
23     }
24     point3D operator * (double k)
25     {
26         return point3D(k*x, k*y, k*z);
27     }
28     point3D operator / (double k)
29     {
30         return point3D(x/k, y/k, z/k);
31     }
32
33     point3D cross(point3D b)
34     {
35         return point3D(y * b.z - z * b.y,
36                         z * b.x - x * b.z,
37                         x * b.y - y * b.x);
38     }
39     double dot(point3D b)
40     {
41         return x * b.x + y * b.y + z * b.z;

```

```

4     }
5     double dist2()
6     {
7         return x * x + y * y + z * z;
8     }
9     double dist()
10    {
11        return sqrt(dist2());
12    }
13    void read()
14    {
15        scanf("%lf%lf%lf", &x, &y, &z);
16    }
17    void write()
18    {
19        printf("%.6lf %.6lf %.6lf", x, y, z);
20    }
21 };
22
23 double disLP(point3D p1, point3D p2, point3D q)
24 {
25     return ((p2 - p1).cross(q - p1)).dist() / (p2 - p1).dist();
26 }
27
28 double disLL(point3D p1, point3D p2, point3D q1, point3D q2,
29              point3D &is1, point3D &is2)
30 {
31     point3D p = q1 - p1, u = p2 - p1, v = q2 - q1;
32     double d = u.dist2() * v.dist2() - u.dot(v) * u.dot(v);
33     if (fabs(d) < eps) return disLP(q1, q2, p1);
34     double s = (p.dot(u) * v.dist2() - p.dot(v) * u.dot(v)) / d;
35     double t = -(u.dist2() * p.dot(v) - p.dot(u) * v.dot(u)) / d;
36     is1 = p1 + (u * s);
37     is2 = q1 + (v * t);
38     return disLP(q1, q2, p1+(u * s));
39 }
40
41 vector<point3D> isPL(point3D p, point3D o, point3D q1, point3D q2)
42 {
43     double a = o.dot(q2-p);
44     double b = o.dot(q1-p);
45     double d = a - b;
46     vector<point3D> retv;
47     if (fabs(d) < eps) return retv;
48     retv.push_back((q1 * a - q2 * b)/d);
49     return retv;
50 }
51
52 bool isPP(point3D p1, point3D o1, point3D p2, point3D o2,

```

```

53 pair<point3D, point3D> L)
54 {
55     point3D e = o1.cross(o2);
56     point3D v = o1.cross(e);
57     double d = o2.dot(v);
58     if (fabs(d) < eps) return false;
59     point3D q = p1 + (v * (o2.dot(p2 - p1)) / d);
60     L.first = q, L.second = q + e;
61     return true;
62 }

```

2. 凸包 $O(N^2)$

来源于网络 验过

```

1  /*
2  模板来源: 互联网
3  功能:
4      1. 求三维凸包
5      2. 三维凸包求表面积
6      3. 三维凸包求面数
7      4. 求质心位置
8
9  */
10 #include <stdio.h>
11 #include <string.h>
12 #include <math.h>
13 #include <algorithm>
14 using namespace std;
15 const int PR = 1e-8;
16 const int eps = 1e-8;
17 #define N 510
18 struct TPoint
19 {
20     double x,y,z;
21     TPoint(){ }
22     TPoint(double _x,double _y,double _z):x(_x),y(_y),z(_z){ }
23     TPoint operator-(const TPoint p) {return
24     TPoint(x-p.x,y-p.y,z-p.z);}
25     TPoint operator*(const TPoint p) {return
26     TPoint(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y*p.x);} //叉积
27     double operator^(const TPoint p) {return x*p.x+y*p.y+z*p.z;} //点
28     积
29 };
30 struct fac
31 {
32     int a,b,c; //凸包一个面上的三个点的编号

```

```

12     bool ok;//该面是否是最终凸包中的面
13 };
14 struct T3dhull
15 {
16     int n;//初始点数
17     TPoint ply[N];//初始点
18     int trianglecnt;//凸包上三角形数
19     fac tri[N];//凸包三角形
20     int vis[N][N];//点 i 到点 j 是属于哪个面
21     double dist(TPoint a){return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);}//两
22 点长度
23     double area(TPoint a,TPoint b,TPoint c){return
24 dist((b-a)*(c-a));}//三角形面积*2
25     double volume(TPoint a,TPoint b,TPoint c,TPoint d){return
26 (b-a)*(c-a)^(d-a);}//四面体有向体积*6
27     double ptoplane(TPoint &p,fac &f)//正：点在面同向
28     {
29         TPoint m=ply[f.b]-ply[f.a],n=ply[f.c]-ply[f.a],t=p-ply[f.a];
30         return (m*n)^t;
31     }
32     void deal(int p,int a,int b)
33     {
34         int f=vis[a][b];
35         fac add;
36         if(tri[f].ok)
37         {
38             if((ptoplane(ply[p],tri[f]))>PR) dfs(p,f);
39             else
40             {
41                 add.a=b,add.b=a,add.c=p,add.ok=1;
42                 vis[p][b]=vis[a][p]=vis[b][a]=trianglecnt;
43                 tri[trianglecnt++]=add;
44             }
45         }
46     }
47     void dfs(int p,int cnt)//维护凸包，如果点 p 在凸包外更新凸包
48     {
49         tri[cnt].ok=0;
50         deal(p,tri[cnt].b,tri[cnt].a);
51         deal(p,tri[cnt].c,tri[cnt].b);
52         deal(p,tri[cnt].a,tri[cnt].c);
53     }
54     bool same(int s,int e)//判断两个面是否为同一面
55     {
56         TPoint a=ply[tri[s].a],b=ply[tri[s].b],c=ply[tri[s].c];
57         return fabs(volume(a,b,c,ply[tri[e].a]))<PR
58             &&fabs(volume(a,b,c,ply[tri[e].b]))<PR
59             &&fabs(volume(a,b,c,ply[tri[e].c]))<PR;
60     }

```

```

61 void construct();//构建凸包
62 {
63     int i,j;
64     trianglecnt=0;
65     if(n<4) return ;
66     bool tmp=true;
67     for(i=1;i<n;i++)//前两点不共点
68     {
69         if((dist(ply[0]-ply[i]))>PR)
70         {
71             swap(ply[1],ply[i]); tmp=false; break;
72         }
73     }
74     if(tmp) return;
75     tmp=true;
76     for(i=2;i<n;i++)//前三点不共线
77     {
78         if((dist((ply[0]-ply[1])*(ply[1]-ply[i]))>PR)
79         {
80             swap(ply[2],ply[i]); tmp=false; break;
81         }
82     }
83     if(tmp) return ;
84     tmp=true;
85     for(i=3;i<n;i++)//前四点不共面
86     {
87         if(fabs((ply[0]-ply[1])*(ply[1]-ply[2])^(ply[0]-ply[i]))>PR)
88         {
89             swap(ply[3],ply[i]); tmp=false; break;
90         }
91     }
92     if(tmp) return ;
93     fac add;
94     for(i=0;i<4;i++)//构建初始四面体
95     {
96         add.a=(i+1)%4,add.b=(i+2)%4,add.c=(i+3)%4,add.ok=1;
97         if((ptoplane(ply[i],add))>0) swap(add.b,add.c);
98         vis[add.a][add.b]=vis[add.b][add.c]=vis[add.c]
99 [add.a]=trianglecnt;
100         tri[trianglecnt++]=add;
101     }
102     for(i=4;i<n;i++)//构建更新凸包
103     {
104         for(j=0;j<trianglecnt;j++)
105         {
106             if(tri[j].ok&&(ptoplane(ply[i],tri[j]))>PR)
107             {
108                 dfs(i,j); break;
109             }

```

```

110     }
111 }
112 }
113 int cnt=trianglecnt;
114 trianglecnt=0;
115 for(i=0;i<cnt;i++)
116 {
117     if(tri[i].ok)
118         tri[trianglecnt++]=tri[i];
119 }
120 }
121 double area()//表面积
122 {
123     double ret=0;
124     for(int i=0;i<trianglecnt;i++)
125         ret+=area(ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);
126     return ret/2.0;
127 }
128 double volume()//体积
129 {
130     TPoint p(0,0,0);
131     double ret=0;
132     for(int i=0;i<trianglecnt;i++)
133     {
134         ret+=volume(p,ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);
135         return fabs(ret/6);
136     }
137 int facetri() {return trianglecnt;}//表面三角形数
138 int facepolygon()//表面多边形数
139 {
140     int ans=0,i,j,k;
141     for(i=0;i<trianglecnt;i++)
142     {
143         for(j=0,k=1;j<i;j++)
144         {
145             if(same(i,j)) {k=0;break;}
146         }
147         ans+=k;
148     }
149     return ans;
150 }
151
152 TPoint massPoint() // 凸包重心
153 {
154     TPoint ret(0, 0, 0);
155     double sumv = 0.0;
156     for (int i = 0; i < trianglecnt; i++)
157     {
158         TPoint cur;

```

```

159         cur = ply[tri[i].a] + ply[tri[i].b] + ply[tri[i].c];
160         cur.x /= 4.0, cur.y /= 4.0, cur.z /= 4.0;
161         double v = volume(TPoint(0,0,0), ply[tri[i].a],
162 ply[tri[i].b], ply[tri[i].c]);
163         cur.x *= v, cur.y *= v, cur.z *= v;
164         sumv += v;
165         ret = ret + cur;
166     }
167
168     ret.x /= sumv, ret.y /= sumv, ret.z /= sumv;
169     return ret;
170 }
171
172 }hull;

```

3. 凸包切割 $O(N^2)$

参考东大模板

其他算法专题

Dancing Links X 精确覆盖

```

1 #include <iostream>
2 #include <cstring>
3 #include <cstdio>
4 #include <vector>
5 #include <algorithm>
6 using namespace std;
7
8 const int maxr = 1010, maxc = 1010, inf = 0x3f3f3f3f;
9
10 int findcnt = 0;
11 struct DancingLinks
12 {
13     struct Node
14     {
15         int L, R, U, D, Col, Row;
16         void setLink(int _l, int _r, int _u, int _d, int _col, int
17 _row)
18         {
19             L = _l, R = _r, U = _u, D = _d, Col = _col, Row = _row;

```



```

31     }
32 } mat[maxr*maxc];
33
34 int head, nodecnt, S[maxc], c, r;
35 int ans[maxr], ansdep;
36
37 void init(int _r, int _c) // Set all nums in matrix 1
38 {
39     c = _c, r = _r;
40     head = 0;
41     nodecnt = c+r+1;
42     memset(S, 0, sizeof S);
43
44     for (int i = 1; i <= c; i++)
45     {
46         mat[i].setLink(i-1, i+1, i, i, i, 0);
47     }
48     mat[c].R = head;
49     mat[1].L = 0;
50     for (int i = 1; i <= r; i++)
51     {
52         mat[c+i].setLink(i+c, i+c, i-1+c, i+1+c, 0, i+c);
53     }
54     mat[c+1].U = 0;
55     mat[c+r].D = 0;
56
57     mat[head].setLink(c, 1, c+r, c+1, 0, 0);
58 }
59
60 void setRow(int row, const vector<int> cols) // Set 1 from the
61 top to the bottom, cols should be sorted.
62 {
63     for (int i = 0; i < (int)cols.size(); i++)
64     {
65         int rtail = mat[row+c].L, ctail = mat[cols[i]].U, rhead
66 = row+c, chead = cols[i];
67         mat[nodecnt].setLink(rtail, rhead, ctail, chead, chead,
68 rhead);
69         mat[rtail].R = nodecnt, mat[ctail].D = nodecnt;
70         mat[rhead].L = nodecnt, mat[chead].U = nodecnt;
71         S[cols[i]]++;
72         nodecnt++;
73     }
74 }
75
76 void removeColumn(int col)
77 {
78     mat[mat[col].L].R = mat[col].R;
79     mat[mat[col].R].L = mat[col].L;

```

```

80
81     for (int i = mat[col].D; i != col; i = mat[i].D)
82     {
83         for (int j = mat[i].R; j != i; j = mat[j].R)
84         {
85             if (mat[j].Col == 0) continue;
86             mat[mat[j].U].D = mat[j].D;
87             mat[mat[j].D].U = mat[j].U;
88             S[mat[j].Col]--;
89         }
90     }
91 }
92
93 void resumeColumn(int col)
94 {
95     for (int i = mat[col].U; i != col; i = mat[i].U)
96     {
97         for (int j = mat[i].L; j != i; j = mat[j].L)
98         {
99             if (mat[j].Col == 0) continue;
100             mat[mat[j].U].D = j;
101             mat[mat[j].D].U = j;
102             S[mat[j].Col]++;
103         }
104     }
105     mat[mat[col].L].R = col;
106     mat[mat[col].R].L = col;
107 }
108
109 bool dfs(int dep)
110 {
111     findcnt++;
112
113     if (mat[head].R == head)
114     {
115         // one answer has been found
116         ansdep = dep;
117         return true;
118     }
119     int mins = inf, rc = -1;
120     for (int i = mat[head].R; i != head; i = mat[i].R)
121         if (mins > S[mat[i].Col]) mins = S[mat[i].Col], rc =
122 mat[i].Col;
123
124     removeColumn(rc);
125
126     for (int i = mat[rc].D; i != rc; i = mat[i].D)
127     {

```

```

129     for (int j = mat[i].R; j != i; j = mat[j].R)
130     {
131         if (mat[j].Col != 0)
132             removeColumn(mat[j].Col);
133     }
134
135     ans[dep] = mat[i].Row - c;
136     if (dfs(dep+1)) return true;
137
138     for (int j = mat[i].L; j != i; j = mat[j].L)
139         if (mat[j].Col != 0)
140             resumeColumn(mat[j].Col);
141     }
142     resumeColumn(rc);
143     return false;
144 }
145 };
146
147 DancingLinks dl;
148
149 int main()
150 {
151     int n, m;
152
153     while (~scanf("%d%d", &n, &m))
154     {
155         dl.init(n, m);
156
157         for (int i = 1; i <= n; i++)
158         {
159             int c;
160             scanf("%d", &c);
161             vector<int> v;
162             for (int j = 0; j < c; j++)
163             {
164                 int p;
165                 scanf("%d", &p);
166                 v.push_back(p);
167             }
168             sort(v.begin(), v.end());
169             dl.setRow(i, v);
170         }
171
172         if (dl.dfs(0))
173         {
174             printf("%d ", dl.ansdep);
175             for (int i = 0; i < dl.ansdep; i++)
176             {
177                 if (i) printf(" ");

```

```

178             printf("%d", dl.ans[i]);
179         }
180         printf("\n");
181     }
182     else
183         printf("NO\n");
184 }
185
186 return 0;
187 }

```

语言和环境

一 python 可视化小工具

```

1 """
2     This file is a part of TkDraw
3
4     Copyright (C) 2011  Rujia Liu
5
6     This program is free software; you can redistribute it and/or
7     modify it under the terms of the GNU Lesser General Public
8     License as published by the Free Software Foundation; either
9     version 2.1 of the License, or (at your option) any later version.
10
11     This program is distributed in the hope that it will be useful,
12     but WITHOUT ANY WARRANTY; without even the implied warranty of
13     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
14     Lesser General Public License for more details.
15
16     You should have received a copy of the GNU Lesser General Public
17     License along with this program; if not, write to the Free
18     Software
19     Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
20     02110-1301
21     USA
22 """
23
24 import sys
25 from Tkinter import *
26
27 def onMouseDown(event):
28     statusText.set("%d,%d" % (event.x, event.y))
29

```

```

35 def onRightMouseDown(event):
36     global scale, dx, dy
37     scale = 1
38     dx = dy = 0
39     f = open(sys.argv[1], 'r')
40     solo = False
41     inblock = True
42     draw("CLEAR")
43     for line in f.readlines():
44         line = line.strip()
45         if line == '-s':
46             if not solo:
47                 solo = True # start to draw a solo block
48                 draw("CLEAR")
49             else:
50                 break # end of solo
51         elif line == '-[':
52             if inblock:
53                 draw("CLEAR") # clear previously drawn content
54                 inblock = True
55         elif line == '-]':
56             inblock = False
57         else:
58             if inblock:
59                 draw(line) # draw if we're inside a block
60     f.close()
61
62 def buildGUI():
63     global canvas, statusText
64
65     canvas = Canvas(width=800, height=600, background='#ffffff')
66     canvas.bind("<Button-1>", onMouseDown)
67     canvas.bind("<Button-3>", onRightMouseDown)
68     canvas.pack()
69
70     statusText = StringVar()
71     status = Label(root, textvariable=statusText, bd=1, relief=SUNKEN,
72 anchor=W)
73     status.pack(side=BOTTOM, fill=X)
74
75 def draw(command):
76     global scale, dx, dy
77     cmd = command.split()
78     if cmd[0] == "TRANSLATE":
79         dx = float(cmd[1])
80         dy = float(cmd[2])
81     if cmd[0] == "SCALE":
82         scale = float(cmd[1])
83     if cmd[0] == "LINE":

```

```

84         x1 = (float(cmd[1]) + dx) * scale
85         y1 = (float(cmd[2]) + dy) * scale
86         x2 = (float(cmd[3]) + dx) * scale
87         y2 = (float(cmd[4]) + dy) * scale
88         canvas.create_line(x1, y1, x2, y2)
89     if cmd[0] == 'CIRCLE':
90         x = (float(cmd[1]) + dx) * scale
91         y = (float(cmd[2]) + dy) * scale
92         r = float(cmd[3]) * scale
93         canvas.create_oval(x-r, y-r, x+r, y+r)
94     if cmd[0] == 'CLEAR':
95         all = canvas.find_all()
96         for item in all:
97             canvas.delete(item)
98
99     root = Tk()
100     buildGUI()
101     root.mainloop()

```

二 准备工作

配终端

gnome-terminal [--disable-factory] -t \$TITLE -x