1 Tarjan 强连通图 入度出度 POJ 1236 Network of Schools

1. **const** **int** maxn = 110;
3. **int** n;
4. **int** dfn[maxn], low[maxn], vst[maxn];
5. **int** belong[maxn];//belong[i]表示i属于缩点后的哪个节点
6. **int** cnt;
7. **int** out[maxn], in[maxn];
8. **int** tot;
10. **struct** Edge {
11. **int** v;
12. **int** next;
13. } edge[maxn\*maxn];
15. **int** edgecount;
16. **int** head[maxn];
18. **void** init() {
19. edgecount = 0;
20. memset(head, -1, **sizeof**(head));
21. }
23. **void** add(**int** u, **int** v) {
24. edge[++edgecount].v = v;
25. edge[edgecount].next = head[u];
26. head[u] = edgecount;
27. }
29. stack<**int**> s;
30. **void** Tarjan(**int** u) { //从节点x开始搜索
31. dfn[u] = low[u] = ++tot;
32. vst[u] = 1;//为1表示在队列里面
33. s.push(u);
34. **for**(**int** k = head[u]; k != -1; k = edge[k].next) {
35. **int** v = edge[k].v;
36. **if**(!dfn[v]) { //还未访问过
37. Tarjan(v);
38. low[u] = min(low[u], low[v]);
39. } **else** **if**(vst[v]) { //被访问过，还在队列里
40. low[u] = min(low[u], dfn[v]);
41. }
42. }
43. **if**(low[u] == dfn[u]) {
44. **int** x;
45. ++cnt;
46. **while**(1) {
47. x = s.top();
48. s.pop();
49. vst[x] = 0;
50. belong[x] = cnt;
51. **if**(x == u)
52. **break**;
53. }
54. }
55. }
57. **int** main() {
58. **int** v;
59. **while**(~scanf("%d", &n)) {
60. init();
61. **for**(**int** u=1; u<=n; u++) {
62. **while**(scanf("%d",&v) && v!=0)
63. add(u,v);
64. }
65. tot = 0;
66. cnt = 0;//缩点后的点数
67. **for**(**int** i = 0; i <= n; ++i) {
68. dfn[i] = low[i] = 0;
69. in[i] = out[i] = 0;
70. vst[i] = 0;
71. }
72. **for**(**int** i = 1; i <= n; ++i) {
73. **if**(dfn[i] == 0)
74. Tarjan(i);
75. }
76. **for**(**int** u = 1; u <= n; ++u) {
77. **for**(**int** k = head[u]; k != -1; k = edge[k].next) {
78. **int** v = edge[k].v;
79. **if**(belong[u] != belong[v]) {
80. out[belong[u]]++;   // 缩点belong[u]的出度
81. in[belong[v]]++;    // 缩点belong[v]的入度
82. }
84. }
85. }
87. **int** ans1 = 0;
88. **int** ans2 = 0;
89. **for**(**int** i = 1; i <= cnt; ++i) {
90. **if**(in[i] == 0)ans1++;
91. **if**(out[i] == 0)ans2++;
92. }
93. ans2 = max(ans1, ans2);
94. **if**(cnt == 1)printf("1\n0\n");
95. **else** printf("%d\n%d\n", ans1, ans2);
96. }
97. **return** 0;
98. }

2 Tarjan 割点模板 重边 UVA 315 Network

1. #define ms(x, n) memset(x,n,sizeof(x));
2. **typedef**  **long** **long** LL;
3. **const** **int** inf = 1<<30;
4. **const** LL maxn = 110;
6. **int** N;
7. vector<**int**> G[maxn];
8. **int** low[maxn], dfn[maxn];
9. **int** par[maxn], indx;
10. **bool** isCut[maxn];
11. **void** Tarjan(**int** u, **int** pa) {
12. par[u] = pa;
13. dfn[u] = low[u] = indx++;
14. **for**(**int** i = 0; i < G[u].size(); ++i) {
15. **int** v = G[u][i];
16. **if**(dfn[v]==-1) {
17. Tarjan(v, u);
18. low[u] = min(low[u], low[v]);
19. } **else** **if**(pa != v) //特判重边, 即v也是u的父亲
20. low[u] = min(low[u], dfn[v]);
21. }
22. }
23. **void** solve() {
24. **int** rootCnt = 0;
25. Tarjan(1, 0);
26. **for**(**int** v = 2; v <= N; ++v) {
27. **int** u = par[v];
28. **if**(u==1) ++rootCnt; //统计根节点子树的数目
29. **else** **if**(low[v]>=dfn[u])
30. isCut[u] = **true**;
31. }
32. **if**(rootCnt>1)
33. isCut[1] = **true**;
34. }
35. **void** init() {
36. ms(G, 0);
37. ms(dfn, -1);
38. ms(par, 0);
39. ms(isCut, 0);
40. ms(low, -1);
41. indx = 0;
42. }
43. **int** main() {
44. **int** a, b;
45. **while**(cin >> N, N) {
46. init();
47. **while**(cin >> a, a) {
48. **while**(getchar()!='\n') {
49. cin >> b;
50. G[a].push\_back(b);
51. G[b].push\_back(a);
52. }
53. }
54. solve();
55. **int** ans = 0;
56. **for**(**int** i = 1; i <= N; ++i)
57. **if**(isCut[i]) ++ans;
58. cout << ans << endl;
59. }
61. **return** 0;
62. }

2 Tarjan 割点模板 注释 UVA 315 Network

1. /\*\*
2. 求割点
3. 一个顶点u是割点，当且仅当满足(1)或(2)
4. (1) u为树根，且u有多于一个子树。
5. (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 dfn(u)<=low(v)。
6. （也就是说 V 没办法绕过 u 点到达比 u dfn要小的点）
7. 注：这里所说的树是指，DFS下的搜索树\*/
9. #define N 10010
10. **int** dfn[N];  ///代表最先遍历到这个点的时间
11. **int** low[N];///这个点所能到达之前最早的时间点
12. **int** Father[N];///保存这个节点的父亲节点
13. **int** n,Time;
14. vector<vector<**int**> >G;
16. **void** init() {
17. G.clear();
18. G.resize(n+1);
19. memset(low,0,**sizeof**(low));
20. memset(dfn,0,**sizeof**(dfn));
21. memset(Father,0,**sizeof**(Father));
22. Time=0;
23. }
25. **void** Tarjan(**int** u,**int** fa) {
26. low[u]=dfn[u]=++Time;
27. Father[u]=fa;
28. **int** len=G[u].size(), v;
29. **for**(**int** i=0; i<len; i++) {
30. v=G[u][i];
31. **if**(!dfn[v]) {
32. Tarjan(v,u);
33. low[u]=min(low[u],low[v]);
34. } **else** **if**(fa!=v)
35. low[u]=min(dfn[v],low[u]);
36. }
37. }
39. **void** solve() {
40. **int** Rootson=0,ans=0;  ///根节点儿子的数量
41. **bool** Cut[N]= {**false**}; ///标记数组，判断这个点是否是割点

44. Tarjan(1,0);ge'dian
46. **for**(**int** i=2; i<=n; i++) {
47. **int** v=Father[i];
48. **if**(v==1)  ///也是就说 i的父亲是根节点
49. Rootson++;
50. **else** **if**(dfn[v]<=low[i])  ///判断是否满足条件（2）
51. Cut[v]=**true**;
52. }
53. **for**(**int** i=2; i<=n; i++) {
54. **if**(Cut[i])
55. ans++;
56. }
57. **if**(Rootson>1)  ///满足条件（1）
58. ans++;
59. printf("%d\n",ans);
60. }
62. **int** main() {
63. **while**(scanf("%d",&n),n) {
64. **int** a,b;
65. **char** ch;
66. init();
67. **while**(scanf("%d",&a),a) {
68. **while**(scanf("%d%c",&b,&ch)) {
69. G[a].push\_back(b);
70. G[b].push\_back(a);
71. **if**(ch=='\n')
72. **break**;
73. }
74. }
75. solve();
76. }
77. **return** 0;
78. }

2 Tarjan 模板 割点 UVA 315 Network

1. **const** **int** maxn = 100010;
2. **const** **int** maxm = 200010;
4. **struct** edge {
5. **int** u;
6. **int** v;
7. **int** next;
8. } e[2 \* maxm];
10. **int** edge\_num, head[maxn];
12. **void** add(**int** u, **int** v) {
13. e[edge\_num].u = u;
14. e[edge\_num].v = v;
15. e[edge\_num].next = head[u];
16. head[u] = edge\_num++;
17. }
19. **int** par[maxn];
20. **int** Find(**int** x) {
21. **return** par[x] == x ? x : par[x] = Find(par[x]);
22. }
23. **void** Union(**int** x, **int** y) {
24. **int** xx = Find(x);
25. **int** yy = Find(y);
26. **if** (xx != yy)
27. par[xx] = yy;
28. }
30. **int** n, m;
31. **int** dfn[maxn], low[maxn], vst[maxn];
33. **int** fa[maxn];
34. **int** ans;
35. **int** tot;
37. **void** Tarjan(**int** u, **int** pre) {
38. dfn[u] = low[u] = ++tot;
39. vst[u] = 1;
40. **int** son = 0;
41. fa[u] = pre; //
42. **for** (**int** i = head[u]; ~i; i = e[i].next) {
43. **int** v = e[i].v;
44. **if**(v == pre)// 无向图割点，防止回溯。
45. **continue**;
46. **if**(!dfn[v]){
47. son++;
48. Tarjan(v, u);
49. low[u] = min(low[u], low[v]);
50. **if**(u != pre && dfn[u] < low[v]){
51. ans++;
52. } **else** {
53. Union(u, v);
54. }
56. }
57. **else** **if**(vst[u])
58. low[u] = min(low[u], dfn[v]);
59. }
60. vst[u] = 2;
61. }
63. **void** func(**int** u) {
64. **int** xx = Find(u);
65. **int** yy = Find(fa[u]);
66. **if** (xx != yy) {
67. ans--;
68. par[xx] = yy;
69. }
70. }
72. **void** LCA(**int** u, **int** v) {
73. **while** (dfn[u] > dfn[v]) {
74. func(u);
75. u = fa[u];
76. }
78. **while** (dfn[v] > dfn[u]) {
79. func(v);
80. v = fa[v];
81. }
83. **while** (u != v) {
84. func(u);
85. func(v);
86. u = fa[u];
87. v = fa[v];
88. }
89. }
91. **int** main() {
92. **int** t = 1;
93. **while** (~scanf("%d%d", &n, &m)) {
94. **if** (n == 0 && m == 0)
95. **break**;
96. memset(head, -1, **sizeof**(head));
97. edge\_num = 0;
98. **int** u, v;
99. **for** (**int** i = 0; i < m; ++i) {
100. scanf("%d%d", &u, &v);
101. add(u, v);
102. add(v, u);
103. }
105. ans = tot = 0;
106. memset(dfn, 0, **sizeof**(dfn));
107. memset(low, 0, **sizeof**(low));
108. memset(vst, 0, **sizeof**(vst));
109. **for** (**int** i = 0; i <= n; ++i)
110. par[i] = i;
112. **for**(**int** i = 1; i <= n; ++i)
113. **if**(!dfn[i])
114. Tarjan(i, i);
116. printf("Case %d:\n", t++);
118. **int** q;
119. scanf("%d", &q);
120. **for** (**int** i = 0; i < q; ++i) {
121. scanf("%d%d", &u, &v);
122. **if** ( Find(u) != Find(v) )
123. LCA(u, v);
124. printf("%d\n", ans);
125. }
126. printf("\n");
127. }
128. }

2 Tarjan 无向图求割点 UVA 315 Network

1. /\*
2. \*  求 无向图的割点和桥
3. \*  可以找出割点和桥，求删掉每个点后增加的连通块。
4. \*  需要注意重边的处理，可以先用矩阵存，再转邻接表，或者进行判重
5. \*/
6. **const** **int** maxn = 10010;
7. **const** **int** maxm = 100010;
8. **struct** Edge {
9. **int** to,next;
10. **bool** cut;//是否为桥的标记
11. } edge[maxm];
12. **int** head[maxn],tot;
13. **int** Low[maxn],DFN[maxn],Stack[maxn];
14. **int** Index,top;
15. **bool** Instack[maxn];
16. **bool** cut[maxn];
17. **int** add\_block[maxn];//删除一个点后增加的连通块
18. **int** bridge;
20. **void** addedge(**int** u,**int** v) {
21. edge[tot].to = v;
22. edge[tot].next = head[u];
23. edge[tot].cut = **false**;
24. head[u] = tot++;
25. }
27. **void** Tarjan(**int** u,**int** pre) {
28. **int** v;
29. Low[u] = DFN[u] = ++Index;
30. Stack[top++] = u;
31. Instack[u] = **true**;
32. **int** son = 0;
33. **for**(**int** i = head[u]; i != -1; i = edge[i].next) {
34. v = edge[i].to;
35. **if**(v == pre)**continue**;
36. **if**( !DFN[v] ) {
37. son++;
38. Tarjan(v,u);
39. **if**(Low[u] > Low[v])Low[u] = Low[v];
40. //桥
41. //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)。
42. **if**(Low[v] > DFN[u]) {
43. bridge++;
44. edge[i].cut = **true**;
45. edge[i^1].cut = **true**;
46. }
47. //割点
48. //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。
49. //(2) u不为树根，且满足存在(u,v)为树枝边(或称父子边，
50. //即u为v在搜索树中的父亲)，使得DFS(u)<=Low(v)
51. **if**(u != pre && Low[v] >= DFN[u]) { //不是树根
52. cut[u] = **true**;
53. add\_block[u]++;
54. }
55. } **else** **if**( Low[u] > DFN[v])
56. Low[u] = DFN[v];
57. }
58. //树根，分支数大于1
59. **if**(u == pre && son > 1)cut[u] = **true**;
60. **if**(u == pre)add\_block[u] = son - 1;
61. Instack[u] = **false**;
62. top--;
63. }
65. **void** solve(**int** N) {
66. memset(DFN,0,**sizeof**(DFN));
67. memset(Instack,**false**,**sizeof**(Instack));
68. memset(add\_block,0,**sizeof**(add\_block));
69. memset(cut,**false**,**sizeof**(cut));
70. Index = top = 0;
71. bridge = 0;
72. **for**(**int** i = 1; i <= N; i++)
73. **if**(!DFN[i])
74. Tarjan(i,i);
75. **int** ans = 0;
76. **for**(**int** i = 1; i <= N; i++)
77. **if**(cut[i])
78. ans++;
79. printf("%d\n",ans);
80. }
81. **void** init() {
82. tot = 0;
83. memset(head,-1,**sizeof**(head));
84. }
85. **int** g[110][110];
86. **char** buf[1010];
87. **int** main() {
88. **int** n;
89. **while**(scanf("%d",&n),n){
90. **int** a, b;
91. **char** ch;
92. memset(g,0,**sizeof**(g));
93. **while**(scanf("%d",&a),a){
94. **while**(scanf("%d%c",&b,&ch)){
95. g[a][b] = g[b][a] = 1;
96. **if**(ch=='\n')
97. **break**;
98. }
99. }
100. init();
101. **for**(**int** i = 1; i <= n; i++)
102. **for**(**int** j = i+1; j <= n; j++)
103. **if**(g[i][j]) {
104. addedge(i,j);
105. addedge(j,i);
106. }
107. solve(n);
108. }
110. **return** 0;
111. }

3 Tarjan 模板 桥 UVA 796 Critical Links

1. // Tarjan算法   复杂度O(n+m)
2. **const** **int** maxn = 10010;//点数
3. **const** **int** maxm = 100010;//边数
4. **struct** Edge {
5. **int** to, next;
6. **bool** cut;
7. } edge[maxm];
8. **int** head[maxn], cnt;
9. **int** low[maxn], dfn[maxn];
10. **int** tot, bridge;
11. **bool** vst[maxn], cut[maxn];
13. **void** add(**int** u, **int** v) {
14. edge[cnt].to = v;
15. edge[cnt].next = head[u];
16. edge[cnt].cut = 0;
17. head[u] = cnt++;
18. }
20. stack<**int**> s;
21. **void** Tarjan(**int** u, **int** pre) {
22. low[u] = dfn[u] = ++tot;
23. s.push(u);
24. vst[u] = 1;
25. **for**(**int** k = head[u]; ~k; k = edge[k].next) {
26. **int** v = edge[k].to;
27. **if**(v == pre)
28. **continue**; //忽略直接相连的边
29. **if**(!dfn[v]) {
30. Tarjan(v, u);
31. low[u] = min(low[u], low[v]);
32. //桥
33. //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)。
34. **if**(dfn[u] < low[v]) {
35. bridge++;
36. edge[k].cut = edge[k^1].cut = 1;
37. }
38. } **else** **if**(vst[u])
39. low[u] = min(low[u], dfn[v]);
40. }
41. vst[u] = 0;
42. s.pop();
43. }
45. **void** solve(**int** n) {
46. **for**(**int** i = 1; i <= n; i++)
47. **if**(!dfn[i])
48. Tarjan(i, i);
50. printf("%d critical links\n", bridge);
51. vector<pair<**int**,**int**> > ans;
53. **for**(**int** u = 1; u <= n; ++u)
54. **for**(**int** v = head[u]; ~v; v = edge[v].next)
55. **if**(edge[v].cut && edge[v].to > u)
56. ans.push\_back( make\_pair(u, edge[v].to) );
57. sort(ans.begin(), ans.end());
58. **for**(**int** i = 0; i <ans.size(); ++i)
59. printf("%d - %d\n", ans[i].first-1, ans[i].second-1);
60. printf("\n");
61. }
63. **void** init() {
64. cnt = 0;
65. tot = bridge= 0;
66. memset(head, -1, **sizeof**(head));
67. memset(dfn, 0, **sizeof**(dfn));
68. memset(vst, 0, **sizeof**(vst));
69. memset(low, 0, **sizeof**(low));
70. }
72. **int** main() {
73. **int** n, a, b, m;
74. **while**(~scanf("%d",&n)) {
75. init();
76. **for**(**int** j = 0; j < n; ++j) {
77. scanf("%d (%d)", &a, &m);
78. a++;
79. **for**(**int** i = 0; i < m; ++i) {
80. scanf("%d", &b);
81. b++;
82. **if**(b <= a)
83. **continue**;
84. add(a,b);
85. add(b,a);
86. }
87. }
88. solve(n);
89. }
90. **return** 0;
91. }

3 Tarjan 无向图求桥 UVA 796 Critical Links

1. #define ms(x, n) memset(x,n,sizeof(x));
2. **typedef**  **long** **long** LL;
3. **const** **int** inf = 1<<30;
4. **const** LL maxn = 11000;
6. **int** N, head[maxn], ecnt;
7. **struct** node {
8. **int** u, v;
9. **int** next;
10. } es[maxn\*10];
11. **void** addEdge(**int** u, **int** v) {
12. es[ecnt].v = v;
13. es[ecnt].next = head[u];
14. head[u] = ecnt++;
15. }
16. **int** low[maxn], dfn[maxn];
17. **int** par[maxn], indx;
19. **typedef** pair<**int**, **int**> P;
20. vector<P> ans;
21. **bool** cmp(node a, node b) {
22. **return** a.u < b.u;
23. }
25. **void** Tarjan(**int** u, **int** pa) {
26. par[u] = pa;
27. dfn[u] = low[u] = ++indx;
28. **bool** flag = **false**;
29. **for**(**int** i = head[u]; i != -1; i = es[i].next) {
30. **int** v = es[i].v;
31. **if**(v==pa && !flag) { //判重边
32. flag = **true**;
33. **continue**;
34. }
35. **if**(!dfn[v]) {
36. Tarjan(v, u);
37. low[u] = min(low[u], low[v]);
38. } **else** **if**(pa != v)
39. low[u] = min(low[u], dfn[v]);
40. }
41. }
42. **void** init() {
43. ms(es, 0);
44. ms(low, 0);
45. ms(dfn, 0);
46. ms(par, 0);
47. ms(head, -1);
48. ans.clear();
49. indx = 0, ecnt = 0;
50. }
52. **int** main() {
53. **int** u, tn, v;
54. **while**(scanf("%d",&N)!=EOF) {
55. init();
56. **for**(**int** i = 1; i <= N; ++i) {
57. scanf("%d (%d)",&u,&tn);
58. **while**(tn--) {
59. scanf(" %d",&v);
60. **if**(v <= u) **continue**; //减少重边
61. addEdge(u, v);
62. addEdge(v, u);
63. }
64. }
66. **for**(**int** i = 0; i < N; ++i)
67. **if**(!dfn[i])
68. Tarjan(i, -1);

71. //一条边(u,v)是桥当且仅当满足(u,v)为树枝边，且low[v]>dfn[u]
72. **for**(**int** v = 0; v < N; ++v) {
73. **int** u = par[v];
74. **if**(u!=-1 && dfn[u] < low[v]) {// u!=-1不是根即树枝边。
75. **if**(v < u) ans.push\_back(P(v, u)); //小的在前
76. **else** ans.push\_back(P(u, v));
77. }
78. }
80. sort(ans.begin(), ans.end());
81. printf("%d critical links\n",ans.size());
82. **for**(**int** i = 0; i < (**int**)ans.size(); ++i)
83. printf("%d - %d\n",ans[i].first,ans[i].second);
84. printf("\n");
85. }
87. **return** 0;
88. }

4 Tarjan 模板 LCA 在线割边 POJ 3694 Network

1. #define ms(a,b) memset((a),(b),sizeof((a)))
2. **const** **int** maxn = 1e6+10;
4. **struct** Edge {
5. **int** to, next;
6. } edge[maxn], edge0[maxn];  //edge为初始图， edge0为重建图
7. **int** tot, head[maxn], tot0, head0[maxn];
9. **int** index, dfn[maxn], low[maxn];
10. **int** top, Stack[maxn], instack[maxn];
11. **int** belong[maxn];
12. **int** fa[maxn], depth[maxn];  //fa用于重建图时记录当前节点的父亲节点，depth记录当前节点的深度
13. **int** sum\_bridge;
15. //找到x最终所属的结合
16. **int** Find(**int** x) {
17. **return** belong[x]==x?x:belong[x]=Find(belong[x]);
18. }
20. **void** addedge(**int** u, **int** v, Edge edge[], **int** head[], **int** &tot) {
21. edge[tot].to = v;
22. edge[tot].next = head[u];
23. head[u] = tot++;
24. }
26. **void** Tarjan(**int** u, **int** pre) {
27. dfn[u] = low[u] = ++index;
28. Stack[top++] = u;
29. instack[u] = **true**;
30. **for**(**int** i = head[u]; i!=-1; i = edge[i].next) {
31. **int** v = edge[i].to;
32. **if**(v==pre) **continue**;
33. **if**(!dfn[v]) {
34. Tarjan(v, u);
35. low[u] = min(low[u], low[v]);
36. **if**(low[v]>dfn[u]) sum\_bridge++;
37. } **else** **if**(instack[v])
38. low[u] = min(low[u], dfn[v]);
39. }
41. **if**(dfn[u]==low[u]) {
42. **int** v;
43. **do** {
44. v = Stack[--top];
45. instack[v] = **false**;
46. belong[v] = u;  //把集合的编号设为联通分量的第一个点
47. } **while**(v!=u);
48. }
49. }
51. **void** build(**int** u, **int** pre) {
52. fa[u] = pre;    //记录父亲节点
53. depth[u] = depth[pre] + 1;  //记录深度
54. **for**(**int** i  = head0[u]; i!=-1; i=edge0[i].next)
55. **if**(edge0[i].to!=pre)    //防止往回走
56. build(edge0[i].to, u);
57. }
59. **int** LCA(**int** u, **int** v) { //左一步右一步地找LCA
60. **if**(u==v) **return** u;  //因为两个结点一定有LCA， 所以一定有u==v的时候
62. //可能爬一步就爬了几个深度，因为中间的结点已经往上缩点了
63. **if**(depth[u]<depth[v]) swap(u, v);   //深度大的往上爬
64. sum\_bridge--;
65. **int** lca = LCA(Find(fa[u]), v);
66. **return** belong[u] = lca;     //找到了LCA，在沿路返回的时候把当前节点的所属集合置为LCA的所属集合
67. }
69. **void** init() {
70. tot = tot0 = 0;
71. memset(head, -1, **sizeof**(head));
72. memset(head0, -1, **sizeof**(head0));
74. index = top = 0;
75. memset(dfn, 0, **sizeof**(dfn));
76. memset(low, 0, **sizeof**(low));
77. memset(instack, 0, **sizeof**(instack));
79. sum\_bridge = 0;
80. }
82. **int** main() {
83. **int** n, m, kase = 0;
84. **while**(scanf("%d%d", &n, &m) && (n||m) ) {
85. init();
86. **for**(**int** i = 1; i<=m; i++) {
87. **int** u, v;
88. scanf("%d%d", &u, &v);
89. addedge(u, v, edge, head, tot);
90. addedge(v, u, edge, head, tot);
91. }
93. Tarjan(1, 1);
94. **for**(**int** u = 1; u<=n; u++)   //重建建图
95. **for**(**int** i = head[u]; i!=-1; i = edge[i].next) {
96. **int** tmpu = Find(u);
97. **int** tmpv = Find(edge[i].to);
98. **if**(tmpu!=tmpv)
99. addedge(tmpu, tmpv, edge0, head0, tot0);
100. }
102. depth[Find(1)] = 0;
103. build(Find(1), Find(1));    //把无根树转为有根树
105. **int** q, a, b;
106. scanf("%d", &q);
107. printf("Case %d:\n", ++kase);
108. **while**(q--) {
109. scanf("%d%d", &a, &b);
110. LCA(Find(a), Find(b));
111. printf("%d\n", sum\_bridge);
112. }
113. printf("\n");
114. }
115. }

4 Tarjan 无向图 并查集缩点+割边 在线求割边数量 POJ 3694 Network

1. **const** **int** maxn = 100010;
2. **const** **int** maxm = 200010;
4. **struct** edge {
5. **int** u;
6. **int** v;
7. **int** next;
8. } e[2 \* maxm];
9. **int** edge\_num, head[maxn];
11. **void** add(**int** u, **int** v) {
12. e[edge\_num].u = u;
13. e[edge\_num].v = v;
14. e[edge\_num].next = head[u];
15. head[u] = edge\_num++;
16. }
18. **int** par[maxn];
19. **int** Find(**int** x) {
20. **return** par[x] == x ? x : par[x] = Find(par[x]);
21. }
23. **void** Union(**int** x, **int** y) {
24. **int** xx = Find(x);
25. **int** yy = Find(y);
26. **if** (xx != yy)
27. par[xx] = yy;
28. }
30. **int** n, m;
31. **int** dfn[maxn], low[maxn], vst[maxn];
33. **int** fa[maxn];
34. **int** ans;
36. **void** tarjan(**int** u, **int** last, **int** vsttime) {
37. dfn[u] = low[u] = vsttime;
38. vst[u] = 1;
39. fa[u] = last;
40. **for** (**int** i = head[u]; ~i; i = e[i].next) {
41. **int** v = e[i].v;
43. **if** (vst[v] == 1 && v != last)
44. low[u] = min(low[u], dfn[v]);
46. **if** (!vst[v]) {
47. tarjan(v, u, vsttime + 1);
48. low[u] = min(low[u], low[v]);
49. **if** (dfn[u] < low[v]) {
50. ans++;
51. } **else** {
52. Union(u, v);
53. }
54. }
55. }
56. vst[u] = 2;
57. }
59. **void** func(**int** u) {
60. **int** xx = Find(u);
61. **int** yy = Find(fa[u]);
62. **if** (xx != yy) {
63. ans--;
64. par[xx] = yy;
65. }
66. }
68. **void** LCA(**int** u, **int** v) {
69. **while** (dfn[u] > dfn[v]) {
70. func(u);
71. u = fa[u];
72. }
73. **while** (dfn[v] > dfn[u]) {
74. func(v);
75. v = fa[v];
76. }
78. **while** (u != v) {
79. func(u);
80. func(v);
81. u = fa[u];
82. v = fa[v];
83. }
84. }
86. **int** main() {
87. **int** t = 1;
88. **while** (~scanf("%d%d", &n, &m)) {
89. **if** (n == 0 && m == 0)
90. **break**;
91. memset(head, -1, **sizeof**(head));
92. edge\_num = 0;
93. **int** u, v;
94. **for** (**int** i = 0; i < m; i++) {
95. scanf("%d%d", &u, &v);
96. add(u, v);
97. add(v, u);
98. }
100. ans = 0;
101. memset(dfn, 0, **sizeof**(dfn));
102. memset(low, 0, **sizeof**(low));
103. memset(vst, 0, **sizeof**(vst));
104. **for** (**int** i = 0; i <= n; i++)
105. par[i] = i;
107. tarjan(1, 0, 1);
109. printf("Case %d:\n", t++);
111. **int** q;
112. scanf("%d", &q);
113. **for** (**int** i = 0; i < q; i++) {
114. scanf("%d%d", &u, &v);
115. **if** (Find(u) != Find(v))
116. LCA(u, v);
117. printf("%d\n", ans);
118. }
119. printf("\n");
120. }
121. }

5 Tarjan 补边 边双连通图 POJ 3177 Redundant Paths

1. **const** **int** maxn = 1e5+10;
3. **int** n, m;
4. **int** dfn[maxn], low[maxn], vst[maxn];
5. **int** belong[maxn];//belong[i]表示i属于缩点后的哪个节点
6. **int** cnt;
7. **int** tot;
8. **int** degree[maxn];
10. **struct** Edge {
11. **int** v;
12. **int** next;
13. } edge[maxn \* 2];
15. **int** edgecount;
16. **int** head[maxn];
18. **void** init() {
19. edgecount = 0;
20. memset(head, -1, **sizeof**(head));
21. memset(degree, 0, **sizeof**(degree));
22. }
24. **void** add(**int** u, **int** v) {
25. edge[++edgecount].v = v;
26. edge[edgecount].next = head[u];
27. head[u] = edgecount;
28. }
30. stack<**int**> s;
31. **void** Tarjan(**int** u, **int** pre) { //从节点x开始搜索
32. dfn[u] = low[u] = ++tot;
33. vst[u] = 1;//为1表示在队列里面
34. s.push(u);
35. **for**(**int** k = head[u]; k != -1; k = edge[k].next) {
36. **int** v = edge[k].v;
37. **if**(v == pre) // 双连通分量，不能回走
38. **continue**;
39. **if**(!dfn[v]) { //还未访问过
40. Tarjan(v, u);
41. low[u] = min(low[u], low[v]);
42. } **else** **if**(vst[v]) { //被访问过，还在队列里
43. low[u] = min(low[u], dfn[v]);
44. }
45. }
46. **if**(low[u] == dfn[u]) {
47. **int** x;
48. ++cnt;
49. **while**(1) {
50. x = s.top();
51. s.pop();
52. vst[x] = 0;
53. belong[x] = cnt;
54. **if**(x == u)
55. **break**;
56. }
57. }
58. }
60. **const** **int** Hash = 5010;
61. **bool** mp[Hash\*Hash];
63. **int** main() {
64. scanf("%d%d", &n, &m);
65. init();
66. **int** x, y;
67. **for**(**int** i = 0; i < m; ++i){
68. scanf("%d%d", &x, &y);
69. **int** v = x \* Hash + y; // 去重边
70. **if**( mp[v] )
71. **continue**;
72. mp[x\*Hash+y] = 1;
73. add(x, y);
74. add(y, x);
75. }
76. tot = 0;
77. cnt = 0;// 缩点后的点数
78. **for**(**int** i = 0; i <= n; ++i) {
79. dfn[i] = low[i] = 0;
80. vst[i] = 0;
81. }
82. **for**(**int** i = 1; i <= n; ++i) {
83. **if**(dfn[i] == 0)
84. Tarjan(i, i);
85. }
86. **for**(**int** u = 1; u <= n; ++u) {
87. **for**(**int** k = head[u]; k != -1; k = edge[k].next) {
88. **int** v = edge[k].v;
89. **if**(belong[u] != belong[v]) {
90. degree[belong[u]] ++;   // u v | v u 重复出现
91. }
92. }
93. }
95. **int** ans = 0;
96. **for**(**int** i = 1; i <= cnt; ++i)
97. **if**(degree[i] == 1)
98. ans++;
100. printf("%d\n", (ans+1)/2 );
102. **return** 0;
103. }

7 Tarjan 添边图依然不强联通 HDU 4635 Strongly connected

1. **typedef** **long** **long** ll;
3. **const** **int** maxn = 1e5+7;
4. **const** **int** inf = 0x3f3f3f3f;
6. vector<**int**> G[maxn];
7. stack<**int**> s;
8. **int** dfn[maxn], low[maxn], vst[maxn], belong[maxn], num[maxn];
9. **int** in[maxn],out[maxn];
10. **int** n, m, tot, cnt, msum;
11. **void** Tarjan(**int** x){
12. dfn[x] = low[x] = ++tot;
13. s.push(x);
14. vst[x] = 1;
15. **for** (**int** i = 0; i < G[x].size();i++){
16. **int** v = G[x][i];
17. **if**(!dfn[v]){
18. Tarjan(v);
19. low[x] = min(low[x], low[v]);
20. }
21. **else** **if**(vst[v]){
22. low[x] = min(low[x], dfn[v]);
23. }
24. }
25. **if**(dfn[x] == low[x]){
26. cnt++;
27. **while** (1){
28. **int** now = s.top();
29. s.pop();
30. vst[now] = 0;
31. num[cnt]++;
32. belong[now] = cnt;
33. **if**(now==x) **break**;
34. }
35. }
36. }
38. **void** init(){
39. memset(dfn, 0, **sizeof**(dfn));
40. memset(low, 0, **sizeof**(low));
41. memset(vst, 0, **sizeof**(vst));
42. memset(belong, 0, **sizeof**(belong));
43. memset(num, 0, **sizeof**(num));
44. memset(in, 0, **sizeof**(in));
45. memset(out, 0, **sizeof**(out));
46. **for**(**int** i = 1;i <= n; ++i)
47. G[i].clear();
48. }
50. **int** main(){
51. **int** t;
52. **int** k = 1;
53. scanf("%d",&t);
54. **while** (t--){
55. init();
56. scanf("%d %d", &n, &m);
57. **for** (**int** i = 0;i < m; ++i){
58. **int** x, y;
59. scanf("%d %d", &x, &y);
60. G[x].push\_back(y);
61. }
62. tot = 0, cnt = 0;
63. **for** (**int** i = 1;i <= n; ++i){
64. **if**(!dfn[i])
65. Tarjan(i);
66. }
68. **if**(cnt == 1){
69. printf("Case %d: -1\n", k++);
70. **continue**;
71. }
73. **for**(**int** i = 1; i <= n; ++i) {
74. **for**(**int** j = 0; j < G[i].size(); ++j) {
75. **int** u = belong[i];
76. **int** v = belong[ G[i][j] ];
77. **if**(u != v){
78. out[u] ++;
79. in[v] ++;
80. }
81. }
82. }
83. msum = inf;
84. **for** (**int** i = 1;i <= cnt; ++i){
85. **if**(!out[i])
86. msum = min(msum, num[i]);
87. **if**(!in[i])
88. msum = min(msum, num[i]);
89. }
90. ll ans = (**long** **long**) n \* (n - 1);// 完全图的边数(n\*(n-1))
91. // 这个特殊的DAG与外界不能相连的边数,即该DAG里的顶点个数乘以除该DAG的顶点外的所有顶点数
92. ans -= (**long** **long**) msum \* (n - msum);
93. printf("Case %d: %d\n", k++, ans - m);// 原来图本就存在的边数m
94. }
95. **return** 0;
96. }

9 Tarjan 重边可走桥 HDU 4738 Caocao's Bridges

1. **const** **int** maxn = 1e3 + 100;
2. **const** **int** maxm = 1e6 + 100;
3. **const** **int** inf = 0x3f3f3f3f;
5. **int** n, m;
6. **int** par[maxn];
8. **struct** Edge {
9. **int** to, w, next;
10. **bool** cut;
11. } edge[maxm];
12. **int** head[maxm], cnt;
13. **int** low[maxm], dfn[maxm];
14. **int** tot, bridge;
15. **bool** vst[maxm], cut[maxm];
17. **void** add(**int** u, **int** v, **int** w) {
18. edge[cnt].to = v;
19. edge[cnt].w = w;
20. edge[cnt].next = head[u];
21. edge[cnt].cut = 0;
22. head[u] = cnt++;
23. }
25. stack<**int**> s;
26. **void** Tarjan(**int** u, **int** pre) {
27. low[u] = dfn[u] = ++tot;
28. s.push(u);
29. vst[u] = 1;
30. **int** num = 0;
31. **for**(**int** k = head[u]; ~k; k = edge[k].next) {
32. **int** v = edge[k].to;
33. **if**(v == pre && !num){// 忽略直接相连的边 可以访问重边
34. num ++;
35. **continue**;
36. }
37. **if**(!dfn[v]) {
38. Tarjan(v, u);
39. low[u] = min(low[u], low[v]);
40. //桥
41. //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)。
42. **if**(dfn[u] < low[v]) {
43. bridge++;
44. edge[k].cut = edge[k^1].cut = 1;
45. }
46. } **else** **if**(vst[u])
47. low[u] = min(low[u], dfn[v]);
48. }
49. vst[u] = 0;
50. s.pop();
51. }
53. **int** Find(**int** x){
54. **if**(par[x] == x)
55. **return** x;
56. **return** par[x] = Find(par[x]);
57. }
59. **void** Union(**int** x, **int** y){
60. **int** xx = Find(x);
61. **int** yy = Find(y);
62. **if**(xx != yy)
63. par[xx] = yy;
64. }
66. **void** init(){
67. **for**(**int** i = 0; i <= n; ++i)
68. par[i] = i;
69. **while**(!s.empty())
70. s.pop();
71. cnt = 0;
72. tot = bridge = 0;
73. memset(head, -1, **sizeof**(head));
74. memset(dfn, 0, **sizeof**(dfn));
75. memset(vst, 0, **sizeof**(vst));
76. memset(low, 0, **sizeof**(low));
77. }
79. **int** main(){
80. **while**(~scanf("%d%d", &n, &m)){
81. **if**(!n && !m)
82. **break**;
83. **int** x, y, z;
84. init();
85. **for**(**int** i = 1; i <= m; ++i){
86. scanf("%d%d%d", &x, &y, &z);
87. add(x, y, z);
88. add(y, x, z);
89. Union(x, y);
90. }
91. **int** flag = 0;
92. **for**(**int** i = 1; i <= n; ++i){
93. **if**(Find(i) != Find(y)){
94. flag = 1;
95. **continue**;
96. }
97. }
98. **if**(flag){
99. printf("0\n");
100. **continue**;
101. }
102. **for**(**int** i = 1; i <= n; ++i)
103. **if**(!dfn[i])
104. Tarjan(i, i);
105. **if**(bridge == 0){
106. printf("-1\n");
107. **continue**;
108. }
110. **int** ans = inf;
111. **for**(**int** u = 1; u <= n; ++u)
112. **for**(**int** v = head[u]; ~v; v = edge[v].next)
113. **if**(edge[v].cut)
114. ans = min(ans, edge[v].w);
115. printf("%d\n", ans == 0 ? 1 : ans);
117. }
118. **return** 0;
119. }