1 dinic 最大流 组装电脑 POJ 3436 ACM Computer Factory

1. **const** **int** inf = 0x3f3f3f3f;
2. **const** **int** maxn = 1e3+100;
3. **const** **int** maxm = 1e6+100;
5. **int** n, m;//点数、边数
6. **int** sp, tp;//原点、汇点
8. **struct** node {
9. **int** u;
10. **int** v, next;
11. **int** cap;
12. }mp[maxm];
14. **int** pre[maxn], dis[maxn], cur[maxn];//cur为当前弧优化，dis存储分层图中每个点的层数（即到原点的最短距离），pre建邻接表
15. **int** cnt = 0;
17. **void** init() {  //不要忘记初始化
18. cnt = 0;
19. memset(pre, -1, **sizeof**(pre));
20. }
22. **void** add(**int** u, **int** v, **int** w) { //加边
23. mp[cnt].u = u;
24. mp[cnt].v = v;
25. mp[cnt].cap = w;
26. mp[cnt].next = pre[u];
27. pre[u] = cnt++;
29. mp[cnt].u = v;
30. mp[cnt].v = u;
31. mp[cnt].cap = 0;
32. mp[cnt].next = pre[v];
33. pre[v] = cnt++;
34. }
36. **bool** bfs() {  //建分层图
37. memset(dis, -1, **sizeof**(dis));
38. queue<**int**>q;
39. **while**(!q.empty())
40. q.pop();
41. q.push(sp);
42. dis[sp] = 0;
43. **int** u, v;
44. **while**(!q.empty()) {
45. u = q.front();
46. q.pop();
47. **for**(**int** i = pre[u]; i != -1; i = mp[i].next) {
48. v = mp[i].v;
49. **if**(dis[v] == -1 && mp[i].cap > 0) {
50. dis[v] = dis[u] + 1;
51. q.push(v);
52. **if**(v == tp)
53. **break**;
54. }
55. }
56. }
57. **return** dis[tp] != -1;
58. }
60. **int** dfs(**int** u, **int** cap) {//寻找增广路
61. **if**(u == tp || cap == 0)
62. **return** cap;
63. **int** res = 0, f;
64. **for**(**int** &i = cur[u]; i != -1; i = mp[i].next) {//
65. **int** v = mp[i].v;
66. **if**(dis[v] == dis[u] + 1 && (f = dfs(v, min(cap - res, mp[i].cap))) > 0) {
67. mp[i].cap -= f;
68. mp[i ^ 1].cap += f;
69. res += f;
70. **if**(res == cap)
71. **return** cap;
72. }
73. }
74. **if**(!res)
75. dis[u] = -1;
76. **return** res;
77. }
79. **int** dinic() {
80. **int** ans = 0;
81. **while**(bfs()) {
82. **for**(**int** i = sp; i <= tp; i++)
83. cur[i] = pre[i];
84. ans += dfs(sp, inf);
85. }
86. **return** ans;
87. }
89. **int** infw[55][15];
90. **int** ofw[55][15];
92. **int** main(){
93. **while**(~scanf("%d %d", &n, &m)) {
94. **int** c;
95. init();
96. sp = 0, tp = m \* 2 + 1;
97. **for**(**int** i = 1; i <= m; i++) {
98. scanf("%d", &c);
99. add(i, i + m, c); // i -> i+m
100. **int** flag = 1;
101. **for**(**int** j = 1; j <= n; j++) {
102. scanf("%d", &infw[i][j]);
103. **if**(infw[i][j] == 1) {
104. flag = 0;
105. }
106. }
107. **if**(flag) { // sp 连边到所有接收 "0000..." 或 "若干个0及若干个2" 的机器，容量为无穷大
108. add(sp, i, inf); // 0 -> i
109. }
110. flag = 1;
111. **for**(**int** j = 1; j <= n; j++) {
112. scanf("%d", &ofw[i][j]);
113. **if**(ofw[i][j] == 0) {
114. flag = 0;
115. }
116. }
117. **if**(flag) { // 能产出成品的节点，连边到T，容量无穷大。
118. add(i + m, tp, inf); // i+m -> m\*2+1
119. }
120. **for**(**int** j = 1; j < i; j++) { // 产出节点连边到能接受其产品的接收节点，容量无穷大
121. flag = 1;
122. **for**(**int** k = 1; k <= n; k++) {
123. **if**(infw[j][k] == 1 && ofw[i][k] == 0 || infw[j][k] == 0 && ofw[i][k] == 1) {
124. flag = 0;
125. **break**;
126. }
127. }
128. **if**(flag) {
129. add(i + m, j, inf); // i+m -> j
130. }
131. flag = 1;
132. **for**(**int** k = 1; k <= n; k++) {
133. **if**(infw[i][k] == 1 && ofw[j][k] == 0 || infw[i][k] == 0 && ofw[j][k] == 1) {
134. flag = 0;
135. **break**;
136. }
137. }
138. **if**(flag) {
139. add(j + m, i, inf); // j+m -> i
140. }
141. }
142. }
143. **int** ans = dinic();
144. **int** sum = 0;
146. **for**(**int** i = 1; i <= m; i++) {
147. **for**(**int** p = pre[i + m]; p; p = mp[p].next) {
148. **if**(mp[p].v != i && mp[p].v != sp && mp[p].v != tp && mp[p].cap != inf) {
149. sum++;
150. }
151. }
152. }
153. printf("%d %d\n", ans, sum);
154. **for**(**int** i = 1; i <= m; i++) {
155. **for**(**int** p = pre[i + m]; p; p = mp[p].next) {
156. **if**(mp[p].v != i && mp[p].v != sp && mp[p].v != tp && mp[p].cap != inf) {
157. printf("%d %d %d\n", i, mp[p].v, inf - mp[p].cap);
158. }
159. }
160. }
161. }
162. **return** 0;
163. }

2 dinic 最大流 源点食物牛牛饮料汇点 POJ 3281 Dining

1. **const** **int** inf = 0x3f3f3f3f;
2. **const** **int** maxn = 510;
3. **const** **int** maxm = 1e5+100;
5. **int** n, F, D;//点数、边数
6. **int** sp, tp;//原点、汇点
8. **struct** node {
9. **int** u;
10. **int** v, next;
11. **int** cap;
12. }mp[maxm];
14. **int** pre[maxn], dis[maxn], cur[maxn];//cur为当前弧优化，dis存储分层图中每个点的层数（即到原点的最短距离），pre建邻接表
15. **int** cnt = 0;
17. **void** init() {  //不要忘记初始化
18. cnt = 0;
19. memset(pre, -1, **sizeof**(pre));
20. }
22. **void** add(**int** u, **int** v, **int** w) { //加边
23. mp[cnt].u = u;
24. mp[cnt].v = v;
25. mp[cnt].cap = w;
26. mp[cnt].next = pre[u];
27. pre[u] = cnt++;
29. mp[cnt].u = v;
30. mp[cnt].v = u;
31. mp[cnt].cap = 0;
32. mp[cnt].next = pre[v];
33. pre[v] = cnt++;
34. }
36. **bool** bfs() {  //建分层图
37. memset(dis, -1, **sizeof**(dis));
38. queue<**int**>q;
39. **while**(!q.empty())
40. q.pop();
41. q.push(sp);
42. dis[sp] = 0;
43. **int** u, v;
44. **while**(!q.empty()) {
45. u = q.front();
46. q.pop();
47. **for**(**int** i = pre[u]; i != -1; i = mp[i].next) {
48. v = mp[i].v;
49. **if**(dis[v] == -1 && mp[i].cap > 0) {
50. dis[v] = dis[u] + 1;
51. q.push(v);
52. **if**(v == tp)
53. **break**;
54. }
55. }
56. }
57. **return** dis[tp] != -1;
58. }
60. **int** dfs(**int** u, **int** cap) {//寻找增广路
61. **if**(u == tp || cap == 0)
62. **return** cap;
63. **int** res = 0, f;
64. **for**(**int** &i = cur[u]; i != -1; i = mp[i].next) {//
65. **int** v = mp[i].v;
66. **if**(dis[v] == dis[u] + 1 && (f = dfs(v, min(cap - res, mp[i].cap))) > 0) {
67. mp[i].cap -= f;
68. mp[i ^ 1].cap += f;
69. res += f;
70. **if**(res == cap)
71. **return** cap;
72. }
73. }
74. **if**(!res)
75. dis[u] = -1;
76. **return** res;
77. }
79. **int** dinic() {
80. **int** ans = 0;
81. **while**(bfs()) {
82. **for**(**int** i = sp; i <= tp; i++)
83. cur[i] = pre[i];
84. ans += dfs(sp, inf);
85. }
86. **return** ans;
87. }
89. **int** infw[55][15];
90. **int** ofw[55][15];
92. **int** main(){
93. scanf("%d %d %d", &n, &F, &D);
94. **int** c;
95. init();
96. sp = 0, tp = F + n + n + D + 1;
97. **for**(**int** i = 1; i <= F; ++i){
98. add(sp, i, 1);
99. }
100. **for**(**int** i = 1; i <= D; ++i){
101. add(F+n+n+i, tp, 1);
102. }
103. **int** k, kk, v;
104. **for**(**int** i = 1; i <= n; ++i){
105. add(F+i, F+n+i,1);
106. scanf("%d%d", &k, &kk);
107. **while**(k--){
108. scanf("%d", &v);
109. add(v, F+i, 1);
110. }
111. **while**(kk--){
112. scanf("%d", &v);
113. add(F+n+i, F+n+n+v, 1);
114. }
115. }
117. **int** ans = dinic();
118. **int** sum = 0;
119. printf("%d\n", ans);
121. **return** 0;
122. }

3 dinic 最大流 插座设备转换器 POJ 1087 A Plug for UNIX

1. **const** **int** inf = 0x3f3f3f3f;
2. **const** **int** maxn = 1e3+100;
3. **const** **int** maxm = 1e6+100;
5. **int** n, m, t;
6. **int** sp, tp;//原点、汇点
8. **struct** node {
9. **int** u;
10. **int** v, next;
11. **int** cap;
12. }mp[maxm];
14. **int** pre[maxn], dis[maxn], cur[maxn];//cur为当前弧优化，dis存储分层图中每个点的层数（即到原点的最短距离），pre建邻接表
15. **int** cnt = 0;
17. **void** init() {  //不要忘记初始化
18. cnt = 0;
19. memset(pre, -1, **sizeof**(pre));
20. }
22. **void** add(**int** u, **int** v, **int** w) { //加边
23. mp[cnt].u = u;
24. mp[cnt].v = v;
25. mp[cnt].cap = w;
26. mp[cnt].next = pre[u];
27. pre[u] = cnt++;
29. mp[cnt].u = v;
30. mp[cnt].v = u;
31. mp[cnt].cap = 0;
32. mp[cnt].next = pre[v];
33. pre[v] = cnt++;
34. }
36. **bool** bfs() {  //建分层图
37. memset(dis, -1, **sizeof**(dis));
38. queue<**int**>q;
39. **while**(!q.empty())
40. q.pop();
41. q.push(sp);
42. dis[sp] = 0;
43. **int** u, v;
44. **while**(!q.empty()) {
45. u = q.front();
46. q.pop();
47. **for**(**int** i = pre[u]; i != -1; i = mp[i].next) {
48. v = mp[i].v;
49. **if**(dis[v] == -1 && mp[i].cap > 0) {
50. dis[v] = dis[u] + 1;
51. q.push(v);
52. **if**(v == tp)
53. **break**;
54. }
55. }
56. }
57. **return** dis[tp] != -1;
58. }
60. **int** dfs(**int** u, **int** cap) {//寻找增广路
61. **if**(u == tp || cap == 0)
62. **return** cap;
63. **int** res = 0, f;
64. **for**(**int** &i = cur[u]; i != -1; i = mp[i].next) {//
65. **int** v = mp[i].v;
66. **if**(dis[v] == dis[u] + 1 && (f = dfs(v, min(cap - res, mp[i].cap))) > 0) {
67. mp[i].cap -= f;
68. mp[i ^ 1].cap += f;
69. res += f;
70. **if**(res == cap)
71. **return** cap;
72. }
73. }
74. **if**(!res)
75. dis[u] = -1;
76. **return** res;
77. }
79. **int** dinic() {
80. **int** ans = 0;
81. **while**(bfs()) {
82. **for**(**int** i = sp; i <= tp; i++)
83. cur[i] = pre[i];
84. ans += dfs(sp, inf);
85. }
86. **return** ans;
87. }
89. map<string, **int**> Hash;
90. **char** str1[30],str2[30];
92. **int** main(){
93. ios::sync\_with\_stdio(**false**);
94. cin.tie(0);
95. **int** cot = 0;
96. string ch1, ch2;
97. cin >> n;
98. sp = 0, tp = 500;
99. init();
100. **for**(**int** i = 1; i <= n; ++i){// 插座
101. cin >> ch1;
102. **if**(Hash[ch1] == 0){
103. Hash[ch1] = ++cot;
104. add(Hash[ch1], tp, 1);
105. }
106. }
107. cin >> m;
108. **for**(**int** i = 1; i <= m; ++i){
109. cin >> ch2 >> ch1;
110. Hash[ch2] = ++cot;// 设备
111. add(sp, Hash[ch2], 1);
112. **if**(!Hash[ch1])// 插座
113. Hash[ch1] = ++cot;
114. add(Hash[ch2], Hash[ch1], 1);
115. }
116. cin>>t;
117. **for**(**int** i = 1; i <= t; ++i){
118. cin >> ch1 >> ch2;
119. **if**(!Hash[ch1])// 插座
120. Hash[ch1] = ++cot;
121. **if**(!Hash[ch2])// 插座
122. Hash[ch2] = ++cot;
123. add(Hash[ch1], Hash[ch2], inf);
124. }
126. **int** ans = dinic();
127. cout<<m - ans<<endl;
129. **return** 0;
130. }

4 dinic spfa 最小费用最大流 人找屋 POJ 2195 Going Home

1. //最小费用最大流模版.求最大费用最大流建图时把费用取负即可。
2. //无向边转换成有向边时需要拆分成两条有向边。即两次加边。
3. **const** **int** maxn = 1010;
4. **const** **int** maxm = 1000200;
5. **const** **int** inf = 0x3f3f3f3f;
7. **struct** Edge {
8. **int** v, cap, cost, next;
9. } p[maxm << 1];
11. **int** e, sumFlow, n, m, st, en;
12. **int** head[maxn], dis[maxn], pre[maxn];
13. **bool** vis[maxn];
14. **void** init() {
15. e = 0;
16. memset(head, -1, **sizeof**(head));
17. }
19. **void** addEdge(**int** u, **int** v, **int** cap, **int** cost) {
20. p[e].v = v;
21. p[e].cap = cap;
22. p[e].cost = cost;
23. p[e].next = head[u];
24. head[u] = e++;
25. p[e].v = u;
26. p[e].cap = 0;
27. p[e].cost = -cost;
28. p[e].next = head[v];
29. head[v] = e++;
30. }
32. **bool** spfa(**int** s,**int** t, **int** n) {
33. **int** u, v;
34. queue<**int**>q;
35. memset(vis, **false**, **sizeof**(vis));
36. memset(pre, -1, **sizeof**(pre));
37. **for**(**int** i = 0; i <= n; i++)
38. dis[i] = inf;
39. vis[s] = **true**;
40. dis[s] = 0;
41. q.push(s);
42. **while**(!q.empty()) {
43. u = q.front();
44. q.pop();
45. vis[u] = **false**;
46. **for**(**int** i = head[u]; i != -1; i = p[i].next) {
47. v = p[i].v;
48. **if**(p[i].cap && dis[v] > dis[u] + p[i].cost) {
49. dis[v] = dis[u] + p[i].cost;
50. pre[v] = i;
51. **if**(!vis[v]) {
52. q.push(v);
53. vis[v] = **true**;
54. }
55. }
56. }
57. }
58. **if**(dis[t] == inf)
59. **return** **false**;
60. **return** **true**;
61. }
63. **int** MCMF(**int** s, **int** t, **int** n) {
64. **int** flow = 0; // 总流量
65. **int** minflow, mincost;
66. mincost = 0;
67. **while**(spfa(s, t, n)) {
68. minflow = inf + 1;
69. **for**(**int** i = pre[t]; i != -1; i = pre[p[i^1].v]) {
70. **if**(p[i].cap < minflow) {
71. minflow = p[i].cap;
72. }
73. }
74. flow += minflow;
75. **for**(**int** i = pre[t]; i != -1; i = pre[p[i^1].v]) {
76. p[i].cap -= minflow;
77. p[i^1].cap += minflow;
78. }
79. mincost += dis[t] \* minflow;
80. }
81. sumFlow = flow; // 最大流
82. **return** mincost;
83. }
85. **const** **int** N = 110;
86. **int** x1[N], x2[N], y1[N], y2[N];
88. **int** main(){
89. **int** t, Case = 0;
90. **int** n, m;
91. **while**(~scanf("%d %d", &n, &m) && n && m){
92. **int** k1 = 0, k2 = 0;
93. **char** str[N];
94. **for**(**int** i = 0; i < n; ++i){
95. scanf("%s", str);
96. **for**(**int** j = 0; j < m; ++j){
97. **if**(str[j] == 'm'){
98. x1[++k1] = i;
99. y1[k1] = j;
100. }
101. **if**(str[j] == 'H'){
102. x2[++k2] = i;
103. y2[k2] = j;
104. }
105. }
106. }
107. init();
108. n = k1;
109. **for**(**int** i = 1; i <= n; ++i){
110. addEdge(0, i, 1, 0);
111. addEdge(n+i, n\*2+1, 1, 0);
112. **for**(**int** j = 1; j <= n; ++j){
113. **int** k = (abs(x1[i] - x2[j]) + abs(y1[i] - y2[j]));
114. addEdge(i, j+n, 1, k);
115. }
116. }
117. **int** ans = MCMF(0, 2\*n+1, 2\*n+1);// 0 到 2\*n+1 共2\*n+1个点
118. printf("%d\n", ans);
119. }
120. **return** 0;
121. }

5 dinic spfa K次供应地 店家 商品 POJ 2516 Minimum Cost

1. //最小费用最大流模版.求最大费用最大流建图时把费用取负即可。
2. //无向边转换成有向边时需要拆分成两条有向边。即两次加边。
3. **const** **int** maxn = 1010;
4. **const** **int** maxm = 1000200;
5. **const** **int** inf = 0x3f3f3f3f;
7. **struct** Edge {
8. **int** v, cap, cost, next;
9. } p[maxm << 1];
11. **int** e, sumFlow, st, en;
12. **int** head[maxn], dis[maxn], pre[maxn];
13. **bool** vis[maxn];
14. **void** init() {
15. e = 0;
16. memset(head, -1, **sizeof**(head));
17. }
19. **void** addEdge(**int** u, **int** v, **int** cap, **int** cost) {
20. p[e].v = v;
21. p[e].cap = cap;
22. p[e].cost = cost;
23. p[e].next = head[u];
24. head[u] = e++;
25. p[e].v = u;
26. p[e].cap = 0;
27. p[e].cost = -cost;
28. p[e].next = head[v];
29. head[v] = e++;
30. }
32. **bool** spfa(**int** s,**int** t, **int** n) {
33. **int** u, v;
34. queue<**int**>q;
35. memset(vis, **false**, **sizeof**(vis));
36. memset(pre, -1, **sizeof**(pre));
37. **for**(**int** i = 0; i <= n; i++)
38. dis[i] = inf;
39. vis[s] = **true**;
40. dis[s] = 0;
41. q.push(s);
42. **while**(!q.empty()) {
43. u = q.front();
44. q.pop();
45. vis[u] = **false**;
46. **for**(**int** i = head[u]; i != -1; i = p[i].next) {
47. v = p[i].v;
48. **if**(p[i].cap && dis[v] > dis[u] + p[i].cost) {
49. dis[v] = dis[u] + p[i].cost;
50. pre[v] = i;
51. **if**(!vis[v]) {
52. q.push(v);
53. vis[v] = **true**;
54. }
55. }
56. }
57. }
58. **if**(dis[t] == inf)
59. **return** **false**;
60. **return** **true**;
61. }
63. **int** MCMF(**int** s, **int** t, **int** n) {
64. **int** flow = 0; // 总流量
65. **int** minflow, mincost;
66. mincost = 0;
67. **while**(spfa(s, t, n)) {
68. minflow = inf + 1;
69. **for**(**int** i = pre[t]; i != -1; i = pre[p[i^1].v]) {
70. **if**(p[i].cap < minflow) {
71. minflow = p[i].cap;
72. }
73. }
74. flow += minflow;
75. **for**(**int** i = pre[t]; i != -1; i = pre[p[i^1].v]) {
76. p[i].cap -= minflow;
77. p[i^1].cap += minflow;
78. }
79. mincost += dis[t] \* minflow;
80. }
81. sumFlow = flow; // 最大流
82. **return** mincost;
83. }
85. **int** n, m, k;
86. **int** a[60][60], b[60][60], c[60][60][60];
87. **int** sup[60], need[60];
89. **int** main() {
90. **while**(~scanf("%d %d %d", &n, &m, &k) && n && m && k) {
91. memset(need, 0, **sizeof**(need));
92. memset(sup, 0, **sizeof**(sup));
93. **int** v;
94. **int** sp = 0, tp = n+m+1;
96. **for**(**int** i = 1; i <= n; ++i) {
97. **for**(**int** j = 1; j <= k; ++j) {
98. scanf("%d", &a[i][j]); // 表示i店需要的商品j数量
99. need[j] += a[i][j];// 表示总需的商品j数量
100. }
101. }
102. **for**(**int** i = 1; i <= m; ++i) {
103. **for**(**int** j = 1; j <= k; ++j) {
104. scanf("%d", &b[i][j]);// 表示存储在该供应地点i的商品j数量
105. sup[j] += b[i][j];// 表示商品j总存货数量
106. }
107. }
108. **for**(**int** p = 1; p <= k; ++p) {
109. **for**(**int** i = 1; i <= n; ++i) {
110. **for**(**int** j = 1; j <= m; ++j) {
111. scanf("%d", &c[p][i][j]);// 第p个货物的一个单位从第j个供应地运送到第i个店主
112. }
113. }
114. }
115. **int** flag = 0;
116. **for**(**int** i = 1; i <= k; ++i) {
117. **if**(sup[i] < need[i]) {
118. flag = 1;
119. **break**;
120. }
121. }
122. **if**(flag) {
123. printf("-1\n");
124. **continue**;
125. }
126. // 0 -> 供应地 -> 店家 -> n+m+1
127. **int** ans=0;
128. **for**(**int** i=1; i<=k; i++) {
129. init();
130. **for**(**int** j=1; j<=m; j++)
131. addEdge(sp,j,b[j][i],0);
132. **for**(**int** j=1; j<=n; j++)
133. addEdge(j+m,tp,a[j][i],0);
135. **for**(**int** j=1; j<=n; j++)
136. **for**(**int** l=1; l<=m; l++) {
137. addEdge(l,j+m,inf,c[i][j][l]);
138. }
139. ans += MCMF(0, n+m+1, n+m+1);
140. //            printf("%4d\n", ans);
141. }
143. printf("%d\n", ans);
144. }
145. **return** 0;
146. }

6 dinic 最大流 发电站用电站 POJ 1459 Power Network

1. **const** **int** inf = 0x3f3f3f3f;
2. **const** **int** maxn = 1e3+100;
3. **const** **int** maxm = 1e6+100;
5. **int** n, m;//点数、边数
6. **int** sp, tp;//原点、汇点
8. **struct** node {
9. **int** u;
10. **int** v, next;
11. **int** cap;
12. }mp[maxm];
14. **int** pre[maxn], dis[maxn], cur[maxn];//cur为当前弧优化，dis存储分层图中每个点的层数（即到原点的最短距离），pre建邻接表
15. **int** cnt = 0;
17. **void** init() {  //不要忘记初始化
18. cnt = 0;
19. memset(pre, -1, **sizeof**(pre));
20. }
22. **void** add(**int** u, **int** v, **int** w) { //加边
23. mp[cnt].u = u;
24. mp[cnt].v = v;
25. mp[cnt].cap = w;
26. mp[cnt].next = pre[u];
27. pre[u] = cnt++;
29. mp[cnt].u = v;
30. mp[cnt].v = u;
31. mp[cnt].cap = 0;
32. mp[cnt].next = pre[v];
33. pre[v] = cnt++;
34. }
36. **bool** bfs() {  //建分层图
37. memset(dis, -1, **sizeof**(dis));
38. queue<**int**>q;
39. **while**(!q.empty())
40. q.pop();
41. q.push(sp);
42. dis[sp] = 0;
43. **int** u, v;
44. **while**(!q.empty()) {
45. u = q.front();
46. q.pop();
47. **for**(**int** i = pre[u]; i != -1; i = mp[i].next) {
48. v = mp[i].v;
49. **if**(dis[v] == -1 && mp[i].cap > 0) {
50. dis[v] = dis[u] + 1;
51. q.push(v);
52. **if**(v == tp)
53. **break**;
54. }
55. }
56. }
57. **return** dis[tp] != -1;
58. }
60. **int** dfs(**int** u, **int** cap) {//寻找增广路
61. **if**(u == tp || cap == 0)
62. **return** cap;
63. **int** res = 0, f;
64. **for**(**int** &i = cur[u]; i != -1; i = mp[i].next) {//
65. **int** v = mp[i].v;
66. **if**(dis[v] == dis[u] + 1 && (f = dfs(v, min(cap - res, mp[i].cap))) > 0) {
67. mp[i].cap -= f;
68. mp[i ^ 1].cap += f;
69. res += f;
70. **if**(res == cap)
71. **return** cap;
72. }
73. }
74. **if**(!res)
75. dis[u] = -1;
76. **return** res;
77. }
79. **int** dinic() {
80. **int** ans = 0;
81. **while**(bfs()) {
82. **for**(**int** i = sp; i <= tp; i++)
83. cur[i] = pre[i];
84. ans += dfs(sp, inf);
85. }
86. **return** ans;
87. }
89. **int** infw[55][15];
90. **int** ofw[55][15];
91. **int** np, nc;
93. **int** main(){
94. ios::sync\_with\_stdio(**false**);
95. cin.tie();
96. **while**(cin >> n >> np >> nc >> m) {
97. init();
98. **char** ch;
99. sp = 0, tp = n + 1;
100. **int** x, y, z;
101. **for**(**int** i = 1; i <= m; ++i){\
102. cin >> ch >> x >> ch >> y >> ch >> z;
103. add(x+1, y+1, z);
104. }
105. **for**(**int** i = 1; i <= np; ++i){
106. cin >> ch >> x >> ch >> y;
107. add(sp, x+1, y);
108. }
109. **for**(**int** i = 1; i <= nc; ++i){
110. cin >> ch >> x >> ch >> y;
111. add(x+1, tp, y);
112. }
113. **int** ans = dinic();
114. cout << ans << endl;
115. }
116. **return** 0;
117. }

7 dinic 双向最大流 任意至任意 岛船 HDU 4280 Island Transport

1. #define inf 0x3f3f3f3f
2. **const** **int** maxn = 100100;
3. **const** **int** maxm = 300000;
4. **struct** node {
5. **int** v, w, ne;
6. } ed[maxm];
7. **int** n, m, cnt;
8. **int** head[maxn], dis[maxn], cur[maxn];
9. **void** init() {
10. cnt = 0;
11. memset(head, -1, **sizeof**(head));
12. }
13. **void** add(**int** u, **int** v, **int** w) { //加边
14. ed[cnt].v = v;
15. ed[cnt].w = w;
16. ed[cnt].ne = head[u];
17. head[u] = cnt++;
18. ed[cnt].v = u;
19. ed[cnt].w = w;
20. ed[cnt].ne = head[v];
21. head[v] = cnt++;
22. }
23. **int** bfs(**int** sp, **int** tp) { //建分层图
24. queue<**int**>q;
25. memset(dis, 0, **sizeof**(dis));
26. dis[sp] = 1;
27. q.push(sp);
28. **while** (!q.empty()) {
29. **int** u = q.front();
30. q.pop();
31. **if** (u == tp)**return** 1;
32. **for** (**int** s = head[u]; ~s; s = ed[s].ne) {
33. **int** v = ed[s].v;
34. **if** (dis[v] == 0 && ed[s].w > 0) {
35. **if** (dis[v] == 0 && ed[s].w > 0) {
36. dis[v] = dis[u] + 1;
37. q.push(v);
38. }
39. }
40. }
41. }
42. **return** dis[tp] != 0;
43. }
44. **int** dfs(**int** sp, **int** tp, **int** flow) { //寻找增广路
45. **int** ret = flow, a;
46. **if** (sp == tp || flow == 0)**return** flow;
47. **for** (**int** &s = cur[sp]; ~s; s = ed[s].ne) {
48. **int** v = ed[s].v;
49. **if** (dis[v] == dis[sp] + 1 && (a = dfs(v, tp, min(ret, ed[s].w)))) {
50. ed[s].w -= a;
51. ed[s ^ 1].w += a;
52. ret -= a;
53. **if** (!ret)**break**;
54. }
55. }
56. **if** (ret == flow)dis[sp] = 0;
57. **return** flow - ret;
58. }
59. **int** dinic(**int** sp, **int** tp) {
60. **int** ans = 0;
61. **while** (bfs(sp, tp)) {
62. **for** (**int** s = 0; s <= n; s++)// 遍历  任意点至任到任意点优化
63. cur[s] = head[s];
64. ans += dfs(sp, tp, inf);
65. }
66. **return** ans;
67. }
68. **int** main() {
69. **int** t;
70. **int** sp = 0, tp = 0;
71. scanf("%d", &t);
72. **while**(t--) {
73. init();
74. **int** x, y, z;
75. **int** Max = 0;
76. **int** Min = inf;
77. scanf("%d%d", &n, &m);
78. **for**(**int** i = 1; i <= n; ++i){
79. scanf("%d %d", &x, &y);
80. **if**(Max < x){
81. Max = x;
82. tp = i;
83. }
84. **if**(Min > x){
85. Min = x;
86. sp = i;
87. }
88. }
89. **while**(m--) {
90. scanf("%d%d%d", &x, &y, &z);
91. add(x, y, z);
92. }
93. **int** ans = dinic(sp, tp);
94. printf("%d\n", ans);
95. }
96. }

8 dinic 最大流 源点食物人人饮料汇点 HDU 4292 Food

1. **const** **int** inf = 0x3f3f3f3f;
2. **const** **int** maxn = 100105;
3. **const** **int** maxm = 3000005;
5. **int** n, F, D;//点数、边数
6. **int** sp, tp;//原点、汇点
8. **struct** node {
9. **int** u;
10. **int** v, next;
11. **int** cap;
12. }mp[maxm];
14. **int** pre[maxn], dis[maxn], cur[maxn];//cur为当前弧优化，dis存储分层图中每个点的层数（即到原点的最短距离），pre建邻接表
15. **int** cnt = 0;
17. **void** init() {  //不要忘记初始化
18. cnt = 0;
19. memset(pre, -1, **sizeof**(pre));
20. }
22. **void** add(**int** u, **int** v, **int** w) { //加边
23. mp[cnt].u = u;
24. mp[cnt].v = v;
25. mp[cnt].cap = w;
26. mp[cnt].next = pre[u];
27. pre[u] = cnt++;
29. mp[cnt].u = v;
30. mp[cnt].v = u;
31. mp[cnt].cap = 0;
32. mp[cnt].next = pre[v];
33. pre[v] = cnt++;
34. }
36. **bool** bfs() {  //建分层图
37. memset(dis, -1, **sizeof**(dis));
38. queue<**int**>q;
39. **while**(!q.empty())
40. q.pop();
41. q.push(sp);
42. dis[sp] = 0;
43. **int** u, v;
44. **while**(!q.empty()) {
45. u = q.front();
46. q.pop();
47. **for**(**int** i = pre[u]; i != -1; i = mp[i].next) {
48. v = mp[i].v;
49. **if**(dis[v] == -1 && mp[i].cap > 0) {
50. dis[v] = dis[u] + 1;
51. q.push(v);
52. **if**(v == tp)
53. **break**;
54. }
55. }
56. }
57. **return** dis[tp] != -1;
58. }
60. **int** dfs(**int** u, **int** cap) {//寻找增广路
61. **if**(u == tp || cap == 0)
62. **return** cap;
63. **int** res = 0, f;
64. **for**(**int** &i = cur[u]; i != -1; i = mp[i].next) {//
65. **int** v = mp[i].v;
66. **if**(dis[v] == dis[u] + 1 && (f = dfs(v, min(cap - res, mp[i].cap))) > 0) {
67. mp[i].cap -= f;
68. mp[i ^ 1].cap += f;
69. res += f;
70. **if**(res == cap)
71. **return** cap;
72. }
73. }
74. **if**(!res)
75. dis[u] = -1;
76. **return** res;
77. }
79. **int** dinic() {
80. **int** ans = 0;
81. **while**(bfs()) {
82. **for**(**int** i = sp; i <= tp; i++)
83. cur[i] = pre[i];
84. ans += dfs(sp, inf);
85. }
86. **return** ans;
87. }
89. **int** infw[55][15];
90. **int** ofw[55][15];
92. **int** main(){
93. **while**(~scanf("%d %d %d", &n, &F, &D)){
94. sp = 0;
95. tp = n+n+F+D+1;
96. init();
97. **int** v;
98. **char** ch[210];
99. // 0 -> 食物 -> 人 -> 人 -> 饮料
100. **for**(**int** i = 1; i <= F; ++i){
101. scanf("%d", &v);
102. add(0, i, v);
103. }
104. **for**(**int** i = 1; i <= D; ++i){
105. scanf("%d", &v);
106. add(n+n+F+i, tp, v);
107. }
108. **for**(**int** i = 1; i <= n; ++i){
109. scanf("%s", ch);
110. **for**(**int** j = 0; j < F; ++j){
111. **if**(ch[j] == 'Y'){
112. add(j+1, i+F, inf);
113. }
114. }
115. }
116. **for**(**int** i = 1; i <= n; ++i){
117. scanf("%s", ch);
118. **for**(**int** j = 0; j < D; ++j){
119. **if**(ch[j] == 'Y'){
120. add(i+n+F, j+1+n+n+F, inf);
121. }
122. }
123. }
124. **for**(**int** i = 1; i <= n; ++i)
125. add(i+F, i+F+n, 1);
126. **int** ans = dinic();
127. printf("%d\n", ans);
128. }
130. **return** 0;
131. }

9 dinic 最小割 罪犯炸城市 HDU 4289 Control

1. **const** **int** inf = 0x3f3f3f3f;
2. **const** **int** maxn = 510;
3. **const** **int** maxm = 1e5+100;
5. **int** n, m;
6. **int** sp, tp;//原点、汇点
8. **struct** node {
9. **int** u;
10. **int** v, next;
11. **int** cap;
12. }mp[maxm];
14. **int** pre[maxn], dis[maxn], cur[maxn];//cur为当前弧优化，dis存储分层图中每个点的层数（即到原点的最短距离），pre建邻接表
15. **int** cnt = 0;
17. **void** init() {  //不要忘记初始化
18. cnt = 0;
19. memset(pre, -1, **sizeof**(pre));
20. }
22. **void** add(**int** u, **int** v, **int** w) { //加边
23. mp[cnt].u = u;
24. mp[cnt].v = v;
25. mp[cnt].cap = w;
26. mp[cnt].next = pre[u];
27. pre[u] = cnt++;
29. mp[cnt].u = v;
30. mp[cnt].v = u;
31. mp[cnt].cap = 0;
32. mp[cnt].next = pre[v];
33. pre[v] = cnt++;
34. }
36. **bool** bfs() {  //建分层图
37. memset(dis, -1, **sizeof**(dis));
38. queue<**int**>q;
39. **while**(!q.empty())
40. q.pop();
41. q.push(sp);
42. dis[sp] = 0;
43. **int** u, v;
44. **while**(!q.empty()) {
45. u = q.front();
46. q.pop();
47. **for**(**int** i = pre[u]; i != -1; i = mp[i].next) {
48. v = mp[i].v;
49. **if**(dis[v] == -1 && mp[i].cap > 0) {
50. dis[v] = dis[u] + 1;
51. q.push(v);
52. **if**(v == tp)
53. **break**;
54. }
55. }
56. }
57. **return** dis[tp] != -1;
58. }
60. **int** dfs(**int** u, **int** cap) {//寻找增广路
61. **if**(u == tp || cap == 0)
62. **return** cap;
63. **int** res = 0, f;
64. **for**(**int** &i = cur[u]; i != -1; i = mp[i].next) {//
65. **int** v = mp[i].v;
66. **if**(dis[v] == dis[u] + 1 && (f = dfs(v, min(cap - res, mp[i].cap))) > 0) {
67. mp[i].cap -= f;
68. mp[i ^ 1].cap += f;
69. res += f;
70. **if**(res == cap)
71. **return** cap;
72. }
73. }
74. **if**(!res)
75. dis[u] = -1;
76. **return** res;
77. }
79. **int** dinic() {
80. **int** ans = 0;
81. **while**(bfs()) {
82. **for**(**int** i = sp; i <= tp; i++)
83. cur[i] = pre[i];
84. ans += dfs(sp, inf);
85. }
86. **return** ans;
87. }
89. **int** main(){
90. **while**(~scanf("%d%d", &n, &m)) {
91. init();
92. scanf("%d%d", &sp, &tp);
93. **int** x, y, v;
94. **for**(**int** i = 1; i <= n; ++i){
95. scanf("%d", &v);
96. add(i, i+n, v);
97. }
98. **for**(**int** i = 1; i <= m; ++i){
99. scanf("%d%d", &x, &y);
100. add(x+n, y, inf);
101. add(y+n, x, inf);
102. }
103. add(0, sp, inf);
104. add(tp+n, 2\*n+1, inf);
105. sp = 0;
106. tp = n+n+1;
107. **int** ans = dinic();
108. printf("%d\n", ans);
109. }
110. **return** 0;
111. }

10 dinic 最小割 无向图输出割边 城市1与2分离 UVA 10480 Sabotage

1. #define inf 0x3f3f3f3f
2. **const** **int** maxn = 100100;
3. **const** **int** maxm = 300000;
5. **struct** node {
6. **int** v, w, ne;
7. } ed[maxm];
8. **int** n, m, cnt;
9. **int** head[maxn], dis[maxn], cur[maxn];
10. **void** init() {
11. cnt = 0;
12. memset(head, -1, **sizeof**(head));
13. }
14. **void** add(**int** u, **int** v, **int** w) { //加边
15. ed[cnt].v = v;
16. ed[cnt].w = w;
17. ed[cnt].ne = head[u];
18. head[u] = cnt++;
19. ed[cnt].v = u;
20. ed[cnt].w = w;
21. ed[cnt].ne = head[v];
22. head[v] = cnt++;
23. }
24. **int** bfs(**int** sp, **int** tp) { //建分层图
25. queue<**int**>q;
26. memset(dis, 0, **sizeof**(dis));
27. dis[sp] = 1;
28. q.push(sp);
29. **while** (!q.empty()) {
30. **int** u = q.front();
31. q.pop();
32. **if** (u == tp)**return** 1;
33. **for** (**int** s = head[u]; ~s; s = ed[s].ne) {
34. **int** v = ed[s].v;
35. **if** (dis[v] == 0 && ed[s].w > 0) {
36. **if** (dis[v] == 0 && ed[s].w > 0) {
37. dis[v] = dis[u] + 1;
38. q.push(v);
39. }
40. }
41. }
42. }
43. **return** dis[tp] != 0;
44. }
45. **int** dfs(**int** sp, **int** tp, **int** flow) { //寻找增广路
46. **int** ret = flow, a;
47. **if** (sp == tp || flow == 0)**return** flow;
48. **for** (**int** &s = cur[sp]; ~s; s = ed[s].ne) {
49. **int** v = ed[s].v;
50. **if** (dis[v] == dis[sp] + 1 && (a = dfs(v, tp, min(ret, ed[s].w)))) {
51. ed[s].w -= a;
52. ed[s ^ 1].w += a;
53. ret -= a;
54. **if** (!ret)**break**;
55. }
56. }
57. **if** (ret == flow)dis[sp] = 0;
58. **return** flow - ret;
59. }
60. **int** dinic(**int** sp, **int** tp) {
61. **int** ans = 0;
62. **while** (bfs(sp, tp)) {
63. **for** (**int** s = 0; s <= n; s++)// 遍历  任意点至任到任意点优化
64. cur[s] = head[s];
65. ans += dfs(sp, tp, inf);
66. }
67. **return** ans;
68. }
70. **int** a[maxn], b[maxn];
72. **int** main() {
73. **int** t;
74. **int** sp = 0, tp = 0;
75. **while**(~scanf("%d%d", &n, &m)) {
76. **if**(!n && !m)
77. **break**;
78. init();
79. **int** x, y, z;
80. **for**(**int** i = 1; i <= m; ++i){
81. scanf("%d%d%d", &x, &y, &z);
82. add(x, y, z);
83. a[i] = x;
84. b[i] = y;
85. }
86. **int** ans = dinic(1, 2);
87. **for**(**int** i = 1; i <= m; ++i){
88. **if**( (!dis[ a[i] ] && dis[ b[i] ] ) || (dis[ a[i] ] && !dis[ b[i] ] )  ){
89. printf("%d %d\n", a[i], b[i]);
90. }
91. }
92. printf("\n")
93. }
94. }

11 dinic 最大流 蜥蜴跳柱子逃离网格 HDU 2732 Leapin' Lizards

1. **const** **int** inf = 0x3f3f3f3f;
2. **const** **int** maxn = 1010;
3. **const** **int** maxm = 1e5+100;
5. **int** n, m;
6. **int** sp, tp;//原点、汇点
8. **struct** node {
9. **int** u;
10. **int** v, next;
11. **int** cap;
12. }mp[maxm];
14. **int** pre[maxn], dis[maxn], cur[maxn];//cur为当前弧优化，dis存储分层图中每个点的层数（即到原点的最短距离），pre建邻接表
15. **int** cnt = 0;
17. **void** init() {  //不要忘记初始化
18. cnt = 0;
19. memset(pre, -1, **sizeof**(pre));
20. }
22. **void** add(**int** u, **int** v, **int** w) { //加边
23. mp[cnt].u = u;
24. mp[cnt].v = v;
25. mp[cnt].cap = w;
26. mp[cnt].next = pre[u];
27. pre[u] = cnt++;
29. mp[cnt].u = v;
30. mp[cnt].v = u;
31. mp[cnt].cap = 0;
32. mp[cnt].next = pre[v];
33. pre[v] = cnt++;
34. }
36. **bool** bfs() {  //建分层图
37. memset(dis, -1, **sizeof**(dis));
38. queue<**int**>q;
39. **while**(!q.empty())
40. q.pop();
41. q.push(sp);
42. dis[sp] = 0;
43. **int** u, v;
44. **while**(!q.empty()) {
45. u = q.front();
46. q.pop();
47. **for**(**int** i = pre[u]; i != -1; i = mp[i].next) {
48. v = mp[i].v;
49. **if**(dis[v] == -1 && mp[i].cap > 0) {
50. dis[v] = dis[u] + 1;
51. q.push(v);
52. **if**(v == tp)
53. **break**;
54. }
55. }
56. }
57. **return** dis[tp] != -1;
58. }
60. **int** dfs(**int** u, **int** cap) {//寻找增广路
61. **if**(u == tp || cap == 0)
62. **return** cap;
63. **int** res = 0, f;
64. **for**(**int** &i = cur[u]; i != -1; i = mp[i].next) {//
65. **int** v = mp[i].v;
66. **if**(dis[v] == dis[u] + 1 && (f = dfs(v, min(cap - res, mp[i].cap))) > 0) {
67. mp[i].cap -= f;
68. mp[i ^ 1].cap += f;
69. res += f;
70. **if**(res == cap)
71. **return** cap;
72. }
73. }
74. **if**(!res)
75. dis[u] = -1;
76. **return** res;
77. }
79. **int** dinic() {
80. **int** ans = 0;
81. **while**(bfs()) {
82. **for**(**int** i = sp; i <= tp; i++)
83. cur[i] = pre[i];
84. ans += dfs(sp, inf);
85. }
86. **return** ans;
87. }
89. **char** str[30][30];// 柱子可用次数
90. vector<pair<**int**, **int**> > point1, point2;// 柱子
92. **int** main(){
93. **int** t, d;
94. scanf("%d", &t);
95. **for**(**int** Case = 1; Case <= t; ++Case){
96. init();
97. scanf("%d%d", &n, &d);
98. point1.clear();
99. point2.clear();
100. memset(str, 0, **sizeof**(str));
101. // 0 -> 柱子入 -> 柱子出 -> tp
102. **int** cot = 0;
103. **for**(**int** i = 0; i < n; ++i){
104. scanf("%s", str[i]);
105. **if**(!i)  m = strlen(str[i]);
106. **for**(**int** j = 0; j < m; ++j){
107. **if**(str[i][j] == '0')
108. **continue**;
109. point1.push\_back({i, j});
110. }
111. }
112. **char** ch[30];
113. **for**(**int** i = 0; i < n; ++i){
114. scanf("%s", ch);
115. **for**(**int** j = 0; j < m; ++j){
116. **if**(ch[j] == 'L'){
117. point2.push\_back({i, j});
118. cot ++;
119. }
120. }
121. }
122. sp = 0;
123. tp = 2 \* n \* m + 1;
124. **for**(**int** i = 0; i < point1.size(); ++i){
125. **int** x = point1[i].first, y = point1[i].second;
126. add(x\*m+y+1, n\*m + x\*m+y+1, str[x][y]-'0'); // 柱子入 - > 柱子出
127. **if**( x-d < 0 || x+d >= n || y-d < 0 || y+d >= m ){
128. add(n\*m + x\*m+y+1, tp, inf);// 柱子出 - > tp
129. }
130. **for**(**int** j = 0; j < point1.size(); ++j){
131. **if**(i == j)
132. **continue**;
133. **int** xx = point1[j].first, yy = point1[j].second;
134. **if**(abs(x-xx) + abs(y-yy) <= d){
135. add(n\*m + x\*m+y+1, xx\*m+yy+1, inf);// 柱子出 -> 柱子入
136. }
137. }
138. }
140. **for**(**int** i = 0; i < point2.size(); ++i){
141. add(sp, point2[i].first \* m + point2[i].second + 1, 1);// 0 -> 柱子入
142. }
143. **int** ans = cot - dinic();
144. **if**(ans == 0)
145. printf("Case #%d: no lizard was left behind.\n", Case);
146. **else** **if**(ans == 1)
147. printf("Case #%d: 1 lizard was left behind.\n", Case);
148. **else**
149. printf("Case #%d: %d lizards were left behind.\n", Case, ans);
151. }
153. **return** 0;
154. }

13 dinic 最大流 人移民星球 缩点 HDU 3605 Escape

1. **const** **int** inf = 0x3f3f3f3f;
2. **const** **int** maxn = 1e5+100;
3. **const** **int** maxm = 1e5+100;
5. **int** n, m;
6. **int** sp, tp;//原点、汇点
8. **struct** node {
9. **int** u;
10. **int** v, next;
11. **int** cap;
12. }mp[maxm];
14. **int** pre[maxn], dis[maxn], cur[maxn];//cur为当前弧优化，dis存储分层图中每个点的层数（即到原点的最短距离），pre建邻接表
15. **int** cnt = 0;
17. **void** init() {  //不要忘记初始化
18. cnt = 0;
19. memset(pre, -1, **sizeof**(pre));
20. }
22. **void** add(**int** u, **int** v, **int** w) { //加边
23. mp[cnt].u = u;
24. mp[cnt].v = v;
25. mp[cnt].cap = w;
26. mp[cnt].next = pre[u];
27. pre[u] = cnt++;
29. mp[cnt].u = v;
30. mp[cnt].v = u;
31. mp[cnt].cap = 0;
32. mp[cnt].next = pre[v];
33. pre[v] = cnt++;
34. }
36. **bool** bfs() {  //建分层图
37. memset(dis, -1, **sizeof**(dis));
38. queue<**int**>q;
39. **while**(!q.empty())
40. q.pop();
41. q.push(sp);
42. dis[sp] = 0;
43. **int** u, v;
44. **while**(!q.empty()) {
45. u = q.front();
46. q.pop();
47. **for**(**int** i = pre[u]; i != -1; i = mp[i].next) {
48. v = mp[i].v;
49. **if**(dis[v] == -1 && mp[i].cap > 0) {
50. dis[v] = dis[u] + 1;
51. q.push(v);
52. **if**(v == tp)
53. **break**;
54. }
55. }
56. }
57. **return** dis[tp] != -1;
58. }
60. **int** dfs(**int** u, **int** cap) {//寻找增广路
61. **if**(u == tp || cap == 0)
62. **return** cap;
63. **int** res = 0, f;
64. **for**(**int** &i = cur[u]; i != -1; i = mp[i].next) {//
65. **int** v = mp[i].v;
66. **if**(dis[v] == dis[u] + 1 && (f = dfs(v, min(cap - res, mp[i].cap))) > 0) {
67. mp[i].cap -= f;
68. mp[i ^ 1].cap += f;
69. res += f;
70. **if**(res == cap)
71. **return** cap;
72. }
73. }
74. **if**(!res)
75. dis[u] = -1;
76. **return** res;
77. }
79. **int** dinic() {
80. **int** ans = 0;
81. **while**(bfs()) {
82. **for**(**int** i = sp; i <= tp; i++)
83. cur[i] = pre[i];
84. ans += dfs(sp, inf);
85. }
86. **return** ans;
87. }
89. **const** **int** cot = 1024+10;
90. **int** type[cot];
92. **int** main(){
93. **while**(~scanf("%d%d", &n, &m)){
94. init();
95. memset(type, 0, **sizeof**(type));
96. // 0 - > 人 - > 星球 - > tp
97. sp = 0;
98. tp = cot + m + 1;
99. **int** v;
100. **for**(**int** i = 1; i <= n; ++i){
101. **int** res = 0;
102. **for**(**int** j = 0; j < m; ++j){
103. scanf("%d", &v);
104. res += (v<<j);
105. }
106. type[res]++;
107. }
108. **for**(**int** i = 1; i < cot; ++i){
109. **if**(type[i] > 0){
110. add(sp, i, type[i]);// 0 -> 人
111. **for**(**int** j = 0; j < m; ++j){
112. **if**( (i >> j)&1 ){
113. add(i, cot+j+1, inf);// 人 -> 星球
114. }
115. }
116. }
117. }
118. **for**(**int** i = 1; i <= m; ++i){
119. scanf("%d", &v);
120. add(cot+i, tp, v);// 星球 - > tp
121. }
122. **int** ans = dinic();
123. **if**( n == ans)
124. printf("YES\n");
125. **else**
126. printf("NO\n");
127. }
128. **return** 0;
129. }

14 dinic 最大流 结婚配对 二分并查集最大流 HDU 3081 Marriage Match II

1. **const** **int** maxn= 2020+5;
2. **const** **int** inf = 0x3f3f3f3f;
4. **struct** Edge {
5. **int** from,to,cap,flow;
6. Edge() {}
7. Edge(**int** f,**int** t,**int** c,**int** flow):from(f),to(t),cap(c),flow(flow) {}
8. };
10. **struct** Dinic {
11. **int** n,m,s,t;
12. vector<Edge> edges;
13. vector<**int**> G[maxn];
14. **int** d[maxn];
15. **bool** vis[maxn];
16. **int** cur[maxn];
18. **void** init(**int** n,**int** s,**int** t) {
19. **this**->n=n, **this**->s=s, **this**->t=t;
20. edges.clear();
21. **for**(**int** i=0; i<n; i++) G[i].clear();
22. }
24. **void** AddEdge(**int** from,**int** to,**int** cap) {
25. edges.push\_back( Edge(from,to,cap,0) );
26. edges.push\_back( Edge(to,from,0,0) );
27. m=edges.size();
28. G[from].push\_back(m-2);
29. G[to].push\_back(m-1);
30. }
32. **bool** BFS() {
33. queue<**int**> Q;
34. memset(vis,0,**sizeof**(vis));
35. vis[s]=**true**;
36. d[s]=0;
37. Q.push(s);
38. **while**(!Q.empty()) {
39. **int** x= Q.front();
40. Q.pop();
41. **for**(**int** i=0; i<G[x].size(); ++i) {
42. Edge& e=edges[G[x][i]];
43. **if**(!vis[e.to] && e.cap>e.flow) {
44. vis[e.to]=**true**;
45. d[e.to]=d[x]+1;
46. Q.push(e.to);
47. }
48. }
49. }
50. **return** vis[t];
51. }
53. **int** DFS(**int** x,**int** a) {
54. **if**(x==t || a==0) **return** a;
55. **int** flow=0,f;
56. **for**(**int**& i=cur[x]; i<G[x].size(); ++i) {
57. Edge& e=edges[G[x][i]];
58. **if**(d[e.to]==d[x]+1 && (f=DFS(e.to,min(a,e.cap-e.flow) ) )>0) {
59. e.flow +=f;
60. edges[G[x][i]^1].flow -=f;
61. flow +=f;
62. a-=f;
63. **if**(a==0) **break**;
64. }
65. }
66. **return** flow;
67. }
69. **int** max\_flow() {
70. **int** ans=0;
71. **while**(BFS()) {
72. memset(cur,0,**sizeof**(cur));
73. ans +=DFS(s,inf);
74. }
75. **return** ans;
76. }
77. } DC;
79. **int** par[maxn];
80. **int** Find(**int** x) {
81. **return** par[x]==-1?x : par[x]=Find(par[x]);
82. }
83. **void** Union(**int** x,**int** y) {
84. **int** xx = Find(x);
85. **int** yy = Find(y);
86. **if**(xx != yy) par[xx] = yy;
87. }
89. **bool** G[maxn][maxn];
91. **bool** solve(**int** n,**int** limit) {
92. **int** src=0, dst=2\*n+1;
93. DC.init(n\*2+2,src,dst);
94. **for**(**int** i=1; i<=n; ++i)
95. **for**(**int** j=n+1; j<=2\*n; ++j)**if**(G[i][j])
96. DC.AddEdge(i,j,1);
97. **for**(**int** i=1; i<=n; i++) DC.AddEdge(src,i,limit);
98. **for**(**int** j=n+1; j<=n\*2; ++j) DC.AddEdge(j,dst,limit);
100. **return** DC.max\_flow() == n\*limit;
101. }
103. **int** main() {
104. **int** T;
105. scanf("%d",&T);
106. **while**(T--) {
107. **int** n,m,f;
108. scanf("%d%d%d",&n,&m,&f);
109. memset(par,-1,**sizeof**(par));
110. memset(G,0,**sizeof**(G));
111. **while**(m--) {
112. **int** u,v;
113. scanf("%d%d",&u,&v);
114. G[u][v+n]=**true**;
115. }
116. **while**(f--) {
117. **int** u,v;
118. scanf("%d%d",&u,&v);
119. Union(u,v);
120. }
121. **for**(**int** i=1; i<=n; i++)
122. **for**(**int** j=i+1; j<=n; j++)
123. **if**(Find(i) == Find(j))//属于同一个并查集的 就合并他们的所有与男生的关系
124. **for**(**int** k=n+1; k<=n\*2; ++k) {
125. G[i][k] = G[j][k] = (G[i][k] || G[j][k]);
126. }
128. **int** l=0, r=n, ans = 0;
129. **while**(l <= r) {
130. **int** mid = (l+r)/2;
131. **if**(solve(n,mid)){
132. ans=mid;
133. l=mid+1;
134. }
135. **else**
136. r=mid-1;
137. }
138. printf("%d\n",ans);
139. }
140. **return** 0;
141. }

15 dinic 最大流 结婚配对 最短路最大流 HDU 3416 Marriage Match IV

1. #define ll long long
2. #define lson l,m,rt<<1
3. #define rson m+1,r,rt<<1|1
5. **const** **int** inf = 0x3f3f3f3f;
6. **const** **int** maxn = 1010;
8. **struct** node {
9. **int** to;
10. **int** w;
11. **int** next;
12. } edge[maxn\*100\*2],e[maxn\*200\*5];
14. **int** ind,pre[maxn],vst[maxn],d[maxn],head[maxn],ind1;
15. **int** now[maxn],S,T;
16. **int** n,m;
18. **void** add1(**int** x,**int** y,**int** z) {
19. e[ind1].to = y;
20. e[ind1].w = z;
21. e[ind1].next = head[x];
22. head[x] = ind1 ++;
23. }
24. **void** spfa(**int** s) {
25. **for**(**int** i = 1; i <= n; i++) {
26. d[i] = inf;
27. vst[i] = 0;
28. }
29. vst[s] = 1;
30. d[s] = 0;
31. queue<**int**> q;
32. q.push(s);
33. **while**(!q.empty()) {
34. **int** k = q.front();
35. q.pop();
36. vst[k] = 0;
37. **for**(**int** i = head[k]; i != -1; i = e[i].next) {
38. **int** t = e[i].to;
39. **if**(d[t] > d[k] + e[i].w) {
40. d[t] = d[k] + e[i].w;
41. **if**(!vst[t]) {
42. vst[t] = 1;
43. q.push(t);
44. }
45. }
46. }
47. }
48. }
50. **void** add(**int** x,**int** y,**int** z) {
51. edge[ind].to = y;
52. edge[ind].w = z;
53. edge[ind].next = pre[x];
54. pre[x] = ind ++;
55. }
57. **void** dfs1(**int** rt) {
58. vst[rt] = 1;
59. **if**(rt == T)**return** ;
60. **for**(**int** i = head[rt]; i != -1; i = e[i].next) {
61. **int** t = e[i].to;
62. **if**(now[rt] + d[t] + e[i].w == d[S]) {
63. now[t] = now[rt] + e[i].w;
64. add(rt,t,1);
65. add(t,rt,0);
66. **if**(!vst[t]) {
67. dfs1(t);
68. }
69. }
70. }
71. }
73. **int** bfs() {
74. memset(vst,-1,**sizeof**(vst));
75. queue<**int**>q;
76. vst[S] = 0;
77. q.push(S);
78. **while**(!q.empty()) {
79. **int** k = q.front();
80. q.pop();
81. **for**(**int** i = pre[k]; i != -1; i = edge[i].next) {
82. **int** t = edge[i].to;
83. **if**(vst[t] == -1 && edge[i].w) {
84. vst[t] = vst[k] + 1;
85. q.push(t);
86. }
87. }
88. }
89. **if**(vst[T] == -1)**return** 0;
90. **return** 1;
91. }
93. **int** dfs(**int** rt,**int** low) {
94. **int** used = 0;
95. **if**(rt == T)**return** low;
96. **for**(**int** i = pre[rt]; i != -1 && used < low; i = edge[i].next) {
97. **int** t = edge[i].to;
98. **if**(vst[t] == vst[rt] + 1 && edge[i].w) {
99. **int** a = dfs(t,min(low-used,edge[i].w));
100. used += a;
101. edge[i].w -= a;
102. edge[i^1].w += a;
103. }
104. }
105. **if**(used == 0)vst[rt] = -1;
106. **return** used;
107. }
108. **int** x[maxn\*100],y[maxn\*100],z[maxn\*100];
110. **void** init(**int** flag) {
111. ind1 = 0;
112. memset(head,-1,**sizeof**(head));
113. **for**(**int** i = 1; i <= m; i++) {
114. **if**(!flag)
115. add1(y[i],x[i],z[i]);
116. **else**
117. add1(x[i],y[i],z[i]);
118. }
119. }
121. **int** main() {
122. **int** t;
123. scanf("%d",&t);
124. **while**(t--) {
125. scanf("%d%d", &n, &m);
126. **for**(**int** i = 1; i <= m; ++i) {
127. scanf("%d%d%d", &x[i], &y[i], &z[i]);
128. }
129. init(0);
130. scanf("%d%d",&S,&T);
131. spfa(T);
132. init(1);
133. ind = 0;
134. memset(now,0,**sizeof**(now));
135. memset(pre,-1,**sizeof**(pre));
136. dfs1(S);
137. **int** ans = 0;
138. **while**(bfs()) {
139. **while**(1) {
140. **int** a = dfs(S,inf);
141. **if**(!a)**break**;
142. ans += a;
143. }
144. }
145. printf("%d\n",ans);
146. }
147. **return** 0;
148. }