/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Hungarian Algorithm(求最大匹配)

O(NM)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

伪代码：

Function FindPath(u)  
 For v∈u的相邻节点  
 标记v已经查询过  
 If v未匹配 or FindPath(v的匹配的点) Then  
 更改u的匹配为v  
 Return Ture  
 End If  
 End For  
Return False  
  
For i ∈ V  
 清空标记  
 FindPath(i)  
End If

C++实现：

#include <string.h>

#include <algorithm>

#include <vector>

using namespace std;

const int MAX = 1000; // 总点数，总！

int n, m;

vector<int> e[MAX];

int link[MAX];

bool vis[MAX];

bool check(int u)

{

for (auto v : e[u])

if (!vis[v])

{

vis[v] = 1;

if (link[v] == 0 || check(link[v]))

{

link[v] = u;

link[u] = v;

return 1;

}

}

return 0;

}

int solve()

{

int ans = 0;

memset(link, 0, sizeof link);

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

{

memset(vis, 0, sizeof vis);

if (check(i))

ans++;

}

return ans;

}