Задание.

Найти все пути между двумя вершинами в графе.

Решение.

```
:-use_rendering(graphviz).
/* member( element, list ): если элемент element есть в списке list
*/
member( X, [X|_] ).
member( X, [_|L] ):-
  member( X, L ).

/* notmember( element, list ): если элемент element не
 *  содержится в списке lis
 */
notmember(_,[]).
notmember(Z,[Z|_]):-!, fail.
notmember(Z,[_|ZT]):-
  notmember(Z,ZT).

/* edge_exists( (Node1, Node2), Edges ): существует ли ребро
 * (Node1,Node2) или (Node2,Node1) в списке ребер Edges
 */
edge_exists( (Node1, Node2), Edges ):-
  member((Node1, Node2), Edges ).
edge_exists( (Node1, Node2), Edges ):-
  member((Node2, Node1), Edges ).

/* edges_equal( (Node1, Node2), (Node3, Node4) ):
 * если (Node1,Node2) и (Node3,Node4) - записи одного и того же ребра графа
 */
edges_equal( (Node1, Node2), (Node3, Node4) ):-
  Node1=Node3,
  Node2=Node4.
edges_equal( (Node1, Node2), (Node3, Node4) ):-
  Node1=Node4,
  Node2=Node3.

/* delete_all(element, lis1, lis2). lis2 - это список ребер lis1, из которого
удалены
   все вхождения ребра element
*/
delete_all(_,[],[]).
delete_all(X1,[X2|L],L1):-
  edges_equal(X1,X2),
  delete_all(X1,L,L1).
delete_all(X,[Y|L],[Y|L1]):-
  not(edges_equal(X,Y)),
  delete_all(X,L,L1).


/* list_set( lis1, lis2 ). lis2 - это список ребер lis1, из которого удалены
повторные
   вхождения ребер
```

```
*/
list_set([],[]).
list_set([H|T],[H|T2]):-
  delete_all(H,T,T1),
  list_set(T1,T2),!.

/* set_minus( set1, set2, set3 ). Множество ребер set3 есть разность множеств
set1, set2
*/
set_minus([], _, []).
set_minus([H|L1], L2, L3):-
  edge_exists(H,L2),
  !,
  set_minus(L1, L2, L3).
set_minus([H|L1], L2, [H|L3]):-
  set_minus(L1, L2, L3).

/* path(From, To, VisitedNodes, Path, Edges ) - построение
 * пути Path от узла From до узла To по ребрам из
 * списка Edges с занесением списка посещенных узлов в VisitedNodes
 */
/* Конечный узел пути не должен содержаться в посещенных узлах
 */
path(To,To,NodesBefore,[],_):-
  notmember(To,NodesBefore).
/* Если в поисках пути From->To существует промежуточный
 * узел Middle, ранее не посещенный, что ребро (From,Middle)
 * есть в графе. Тогда пытаемся отыскать путь Middle->To.
 */
path(From,To,NodesBefore,[(From,Middle)|RemPath],Edges):-
  edge_exists((From,Middle),Edges),
  not( From=Middle ),
  notmember(Middle,[From|NodesBefore]),
  path(Middle,To,[From|NodesBefore],RemPath, Edges ).


/* list_declarations( list1, list2 ): по списку list1 термов "(a,b)"
 * составляет список list2 деклараций для библиотеки dot
 */
list_declarations( [], [] ).
list_declarations( [ X1 | L1 ], [ X2 | L2 ] ):-
  edge_declaration( X1, X2 ),
  list_declarations( L1, L2 ).


/* edge_declaration( X, Color, X ):
 * по терму X="(a,b)" и имени цвета (Color="Red") формирует
   dot-декларацию ребра графа вида X="a--b[color=\"Red\"];"
 */
edge_declaration( (A,B), Color, X ):-
    string_concat( A, "--", X1 ),
    string_concat( X1, B, X2 ),
    string_concat( X2, "[color=\"", X3 ),
    string_concat( X3, Color, X4 ),
    string_concat( X4, "\"];", X ).

/* edges_declarations( List, Color, Decl1, Decl2 )
```

```
 * добавляет к строке Decl1 описания всех ребер из списка
 * List с цветом Color. Получающаяся строка - Decl2
 */
edges_declarations( [], _, Decl, Decl ).
edges_declarations( [H|L], Color, DeclBefore, DeclAfter ):-
    edge_declaration( H, Color, DeclEdge ),
    string_concat( DeclBefore, DeclEdge, DeclAfter1 ),
    edges_declarations( L, Color, DeclAfter1, DeclAfter ).

/* По списку красных ребер RedEdges, чёрных ребер BlackEdges
 * формирует описание графа Decl в dot-формате
 */
graph_declaration( RedEdges, BlackEdges, Decl ):-
    edges_declarations( RedEdges, "Red", "graph{ ", Decl1 ),
    edges_declarations( BlackEdges, "Black", Decl1, Decl2 ),
    string_concat( Decl2, "}", Decl ).


/* find_path( Edges, From, To, Pic ): нахождение пути от From до To по
 * графу с ребрами Edges без повторных посещений одной и той же вершины
 * с отрисовкой картинки Pic
 */
find_path( Edges, From, To, Pic ):-
  /* Создаем по списку ребер Edges
   * множество ребер */
  list_set(Edges,GraphEdges),
  /* Ищем путь From->To в графе с
   * множеством ребер GraphEdges
   * с изначально пустым множеством
   * посещённых ребер */
  path( From,To,[],Path,GraphEdges ),
  /* Множество ребер, не вошедших в путь Path */
  set_minus( GraphEdges, Path, NotPath ),
  /* Построение dot-декларации графа */
  graph_declaration( Path, NotPath, GraphDecl ),
  /* Отрисовка графа */
  Pic = dot(GraphDecl),
  /* Распечатываем путь */
  write( Path ).

/*
 * Пример запроса поиска всех путей от a до k, в неориентированном графе с
указанным списком ребер, с построением картинки Pic
 *
find_path([(a,b),(a,c),(a,g),(b,c),(b,p),(c,d),(c,e),(c,g),(d,f),(d,m),(d,p),
(e,f),(e,g),(e,h),(f,k)],a,k, Pic ).
*/
```