Нужно решить 3 задания на неориентированном невзвешенном графе: найти количество компонент связности графа, найти мосты в графе, определить подлежит ли граф сильной ориентации.

Третья задача решается сходу: граф подлежит сильной ориентации, если содержит ровно одну компоненту связности и не содержит мостов.

Рассмотрим решение первой задачи. Компонента связности графа -- это множество взаимно достижимых вершин. Изолированная вершина -- это тоже компонента связности, которая состоит из одной вершины, например, на рисунке 1 граф имеет 3 компоненты связности.

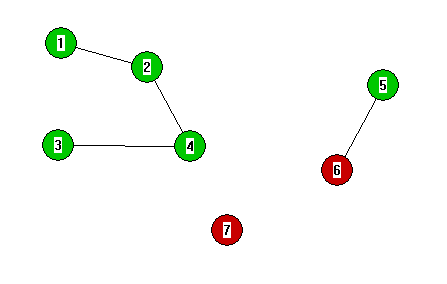


Рисунок 1

Первая задача решается с помощью псевдокода:

function comps:uint

for vertex in graph visited[vertex]=false;

count=0;

for vertex in graph begin

if (visited[vertex]) continue;

deep(vertex);

count++;

end;

return count;

end;

function deep(vertex):

visited[vertex]=1;

for child in graph[vertex] begin

if visited[child] continue;

deep(child);

end;

end;

Теперь рассмотрим вторую задачу. Мост -- это ребро, удаление которого приведёт к увеличению количества компонент связности. Таким образом, надо пройти по всем рёбрам и выяснить удаление каких рёбер ведет к отделению компонент связности. Псевдокод второй задачи:

function bridges:uint

count=comps(graph);

for link in links begin

delete(graph, link);

count2=comps(graph);

add(graph,link);

if (count2>count) output(link);

end;

end;

На рисунке 2 мы имеем 4 моста и программа их определяет.

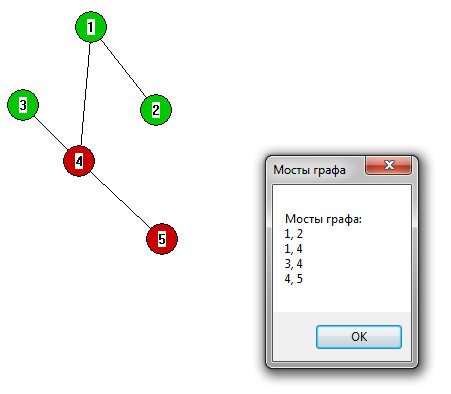


Рисунок 2

Граф хранится в массиве списков вершин incMatrix и в массиве длин списков вершин lens.

То есть, для каждой вершины известно количество смежных вершин и сами смежные вершины, которые хранятся в порядке возрастания номера.

Если нам надо вставить новое ребро (a,b), то надо вставить вершину b в массив incMatrix[a], lens[a]++ и надо вставить вершину a в массив incMatrix[b], lens[b]++. Если нам надо удалить ребро (a,b), то надо удалить вершину b из массива incMatrix[a], lens[a]-- и надо удалить вершину a из массива incMatrix[b], lens[b]--.

Если надо добавить вершину, то надо скопировать массив incMatrix в буфер, затем скопировать вершины из буфера в incMatrix по новым адресам.

Псевдокод добавления вершины:

function addNode:

copy(incMatrix, buffer, nodes\*nodes)

nodes1=nodes+1

for i=0 to nodes-1 begin

for j=0 to lens[i] begin

incMatrix[i\*nodes1+j]=buffer[i\*nodes+j];

end;

end;

lens[nodes]=0; nodes++;

end;

Если надо удалить вершину, то надо удалить длину её списка смежности из массива lens, надо удалить её список смежности из массива incMatrix, а также уменьшить номера следующих за ней вершин.

Сначала копируем в буфер список смежности вершины и удаляем из графа связующие рёбра. Затем в списках смежности проводим декремент всех вершин, которые старше удаляемой.

Затем перемещаем элементы списков смежности на новые места, пропуская список удалённой вершины. Псевдокод удаления вершины:

function deleteNode(node):

copy(incMatrix+ nodes\*node, buffer, lens[node]);

len=lens[node];

for i=0 to len-1 delete(buffer[i], node);

nodes1=nodes-1;

for i=0 to nodes-1 begin

for j=0 to lens[i]-1if incMatrix[i\*nodes+j]>node incMatrix[i\*node+j]--;

end;

for i=0 to node-1 begin

for j=0 to lens[i]-1 incMatrix[i\*nodes1+j]=incMatrix[i\*nodes+j];

end;

for i=node+1 to nodes-1 begin

for j=0 to lens[i]-1 incMatrix[(i-1)\*nodes1+j]=incMatrix[i\*nodes+j];

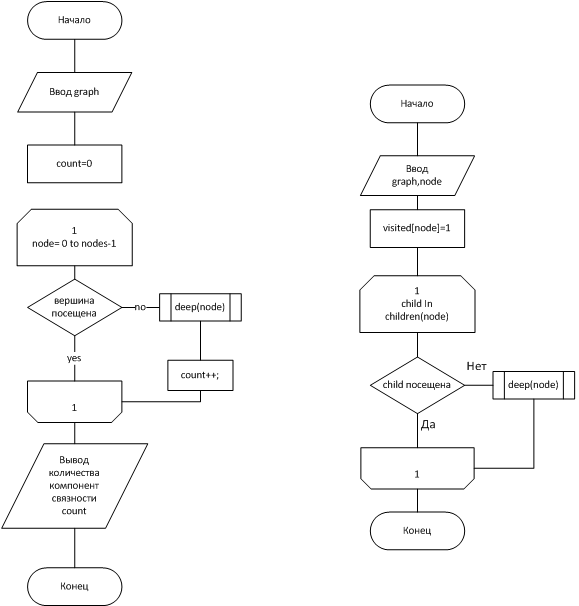
end;

for i=node to nodes-2 lens[i]=lens[i+1];

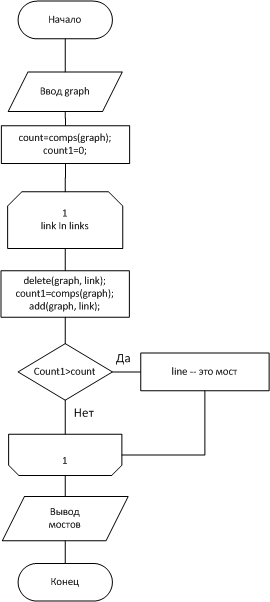
lens[nodes1]=0; nodes=nodes1;

end;

Блок-схема первой задачи:



Блок-схема второй задачи:



Код первой задачи:

int traverse\_graph\_nodes(Graph \*graph, unsigned node){

int child = -1;

unsigned i = 0, line=node\*graph[0].nodeCount;

if (node>graph[0].nodeCount) return 0;

if (graph[0].visited[node]) return 1;

graph[0].visited[node] = 1;

for (i=0;i<graph[0].lens[node];i++){

child = graph[0].incMatrix[line + i];

if (graph[0].visited[child]) continue;

traverse\_graph\_nodes(graph, child);

}

return 1;

}

int graph\_getConnectedCount(Graph \*graph, int fill){

unsigned i = 0, count = 0;

if (fill) graph\_fillIncMatrix(graph);

graph[0].nconn = 0;

for (i = 0; i < graph[0].nodeCount; i++){graph[0].visited[i] = 0;}

for (i = 0; i < graph[0].nodeCount; i++){

if (graph[0].visited[i]) continue;

traverse\_graph\_nodes(graph, i);

count++;

}

graph[0].nconn = count;

return count;

}

Код второй задачи:

int graph\_getBridges(Graph \*graph, int fill){

unsigned i = 0, j = 0, line = 0, child = 0;

unsigned conn = 0, conn1=0, count=0;

unsigned \*visited2 = graph[0].visited + graph[0].nodeCount;

graph[0].timer = 0; graph[0].nbridge = 0;

graph[0].cut[0] = graph[0].cut[1] = -1;

conn = graph\_getConnectedCount(graph, 1);

for (i = 0; i < graph[0].nodeCount;i++){

graph[0].visited[graph[0].nodeCount + i] = 0;}

// Цикл по ребрам графа

for (i = 0, line=0; i < graph[0].nodeCount; i++,

line += graph[0].nodeCount){

for (j = 0; j < graph[0].lens[i]; j++){

child = graph[0].incMatrix[line+j];

if (child == i || child<i) continue;

deleteLinkFromGraph(i, child);

graph\_fillIncMatrix(graph);

/\* Если удаление ребра не вызвало

увеличение количества компонент

связности, то это не мост\*/

conn1=graph\_getConnectedCount(graph, 0);

addLinkToGraph(i, child, 1);

graph\_fillIncMatrix(graph);

if (conn1 > conn); else continue;

graph[0].bridges[count \* 2] = i;

graph[0].bridges[count \* 2 + 1] = child;

count++; graph[0].nbridge++;

}

visited2[i] = 1;

}

graph[0].cut[0] = graph[0].cut[1] = -1;

return 1;

}