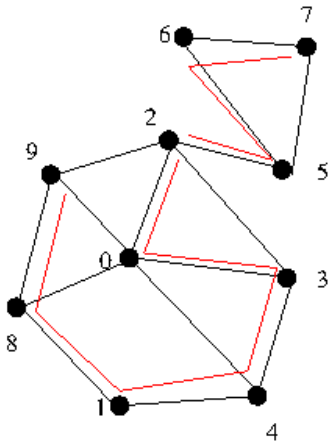
# Quiz 2
# (Solutions)

1. [3 marks] For any dfs tree T, let S={v1,...,vk} be the set of leaves. We would like to compute the sum = d(v1)+...+d(vk), the sum of the distances of the leaves from the root, along the dfs tree. For the example below, S={7,9} and sum=3+6=9. Modify the dfs code below to compute the sum.




dfs=[2 0 3 4 1 8 9 5 6 7]

```
void Graph::DFS(int v)
{
    visited[v] = true;

    for (int i = 0; i<deg[v]; i=i+1)
        if (visited[elist[v][i]]== false)
            DFS(elist[v][i]);
}
```

```
global sum=0;
function DFS(v,d) // d is the
{
    visited[v]=true;
    found=0;
    for (i=0; i<deg[v]; i=i+1)
      If (visited[elist[v][i]]==false)
      {    DFS(elist[v][i],d+1);
           found=1;
      }; // o if
    If found==0
      sum=sum+d;
}

main();
{
  sum=0;
  DFS(s,0);
};
```

2. [6 marks] Game of Sticks. There are n sticks on a table. Two players Red and Blue play the game. In each play, one of the players picks up either 1,2 or 3 sticks from

the table and removes them from the game. Blue begins first and the players alternate. The player who picks up the last stick(s) from the table, loses. (i) Model this game as a directed bipartite graph. List the vertex sets and the edge sets clearly. Give examples. (2 marks).

VB=Vertices from which Blue is to play. ={0B,1B,...,nB}
VR=vertices from which red is to play = {0R, 1R,..., nR}
Edges E ={(iB,(i-1)R), (iB,(i-2)R),(iB(i-3)R)} union {(iR,(i-1)B,(iR,(i-2)B),(iR,(i-3)B)}
Example (5R,4B), (5R,3B) and (5R,2B)

How would you describe a winning strategy for the Blue Player in terms of the above graph, in two sentences? (2 marks).

A winning strategy would be given by a preferred edge or edges from every vertex labeled iB which if taken will ensure a win for the Blue player. Thus for example, (4B->1R) is the winning strategy.

Suppose for all m<n you have computed the winning strategy for the Blue Player, if it exists, when there were m sticks on the table. How would you check if you have a winning strategy for n sticks? How would you then construct the strategy for n sticks? Answer in exactly 2 sentences. (2 marks)

Given the vertex/number nB, i will check (n-1)R, (n-2)R and (n-3)R. If for some (n-i)R all the outgoing edges only go to winning vertices/numbers, i.e., (n-i-1)B and (n-i-2)B and (n-i-3)B are winning vertices, then (nB->(n-i)R) is the winning strategy. Example: 4B,3B,2B are winning vertices. So 7B->5R is a winning strategy for 7B and 7B is a winning vertex.

3. [3 marks] We say that an edge in an undirected graph is a cycle-edge if it is contained in some cycle in the graph. We want to use the bfs to detect and classify cycle edges. Complete the following fill-in-the-blanks:
   (A) A non-tree edge in a bfs is a cycle edge if and only if ____Always_____.
   (B) A tree-edge in a bfs is a cycle edge if and only if _(assuming (u->v) in the tree) there is a non-tree edge (v,x) or (w,x), where w is a descendent of v in the tree. _____.

4. [5 marks] Now modify the bfs code below to label all tree edges as cycle-edges/non-cycle-edges.
   (i) List the new variable(s) introduced and their meaning (1 mark)

   The new global variable is incycle[w]: for any tree edge, we look at (n[w],w), where n[w] is the parent of w in T. incycle[w] is made 1 if that is in a cycle.

   (ii) Make the changes in the code (2 marks)
   See below.

(iii) Analyse the time complexity of your code. More efficient code carries more marks. (2 marks)

The total additional work which is done is O(E). Note that for any vertex u such that (u,v), the incycle of all edges on the path from v to the root s and w to s are to be made 1. However, if on this path, an edge is already seen to be 1, then all the edges above are guaranteed to be 1. Hence only edges to the first incycle[x]=1 is to be freshly made 1. Thus the bound on the total time spent in the inner while loops is O(E).

```
BFS(G,s)
01 for each vertex u ∈ V[G]-{s}
02    color[u] ← white
03    d[u] ← ∞
04    π[u] ← NIL
05 color[s] ← gray
06 d[s] ← 0
07 π[u] ← NIL
08 Q ← {s}
09 while Q ≠ ∅ do
10    u ← head[Q]
11    for each v ∈ Adj[u] do
12       if color[v] = white then
13          color[v] ← gray
14          d[v] ← d[u] + 1
15          π[v] ← u
16          Enqueue(Q,v)
17    Dequeue(Q)
18    color[u] ← black
```

```
global incycle;
for all w  incycle[w]=0;//incycle[w]=1 pimples [n[w],w] is in a cycle

…

While Q!= empty do
    u= head[Q]
    For each v adjacent to u do
```

```
          If color[v]==white then
             color[v]=gray
             d[v]=d[u]+1;
             n[v]=u;
             Enqueue(Q,v);
        Else // backedge
             incycle[u]=1;incycle[v]=1;
             x=u; // the current path up
             While (incycle[n[x]]!=1 OR n[x]!=nil)
                   incycle[x]=1;
                   x=n[x];
              End;
             x=v;  // the other path up
             While (incycle[n[x]]!=1 OR n[x]!=nil)
                   incycle[x]=1;
                   x=n[x];
              End;
       End; // of If
      Dequeue(Q);
      color[u]=black;
end;// whileQ!=empty
```