

3. DIRECTED GRAPHS

Topics:

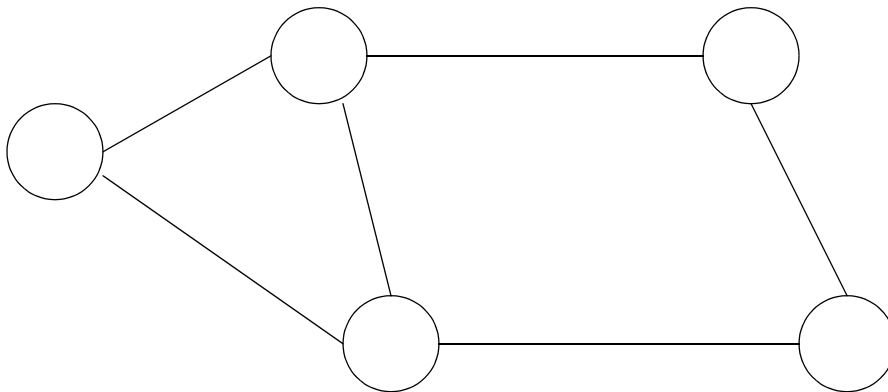
-

Objectives:

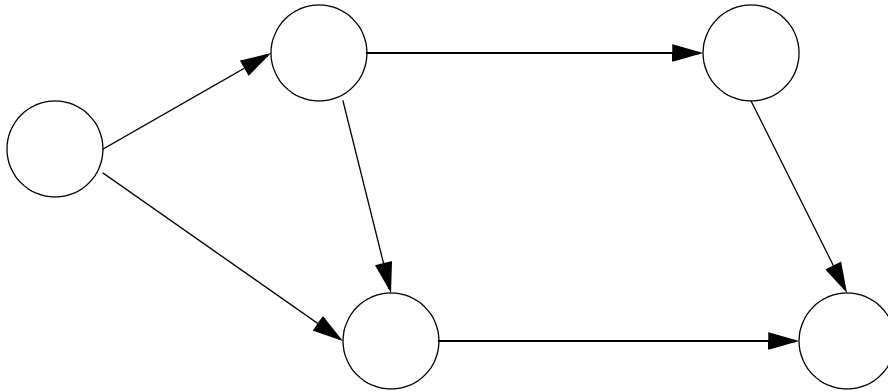
-

3.1 Introduction

- Graphs are normally data structures that have vertices (circles) and edges (lines) as shown below.



- In a directed graph the edges will have a direction assigned, as shown below.



3.2 Data Structures

- A simple data structure for representing a graph is shown below.

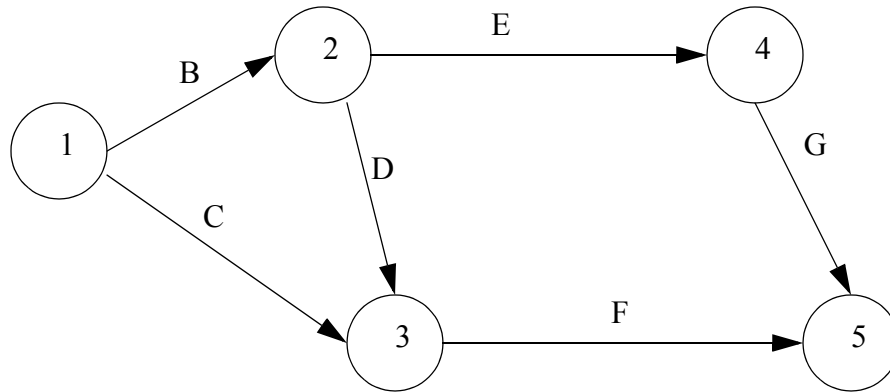
Vertex list:

Vertex

Edge list:

Edge	From	To
------	------	----

- Consider the graph below.



Vertex list:

Vertex
1
2
3
4
5

Edge list:

Edge	From vert.	To vert.
B	1	2
C	1	3
D	2	3
E	2	4
F	3	5
G	4	5

- Representing a graph in Scilab

```

function foo = find_index(_list, name) // a function to find list numbers given names
    foo = -1; // use -1 to indicate no match found yet
    A = size(_list); // find the rows and columns in the list
    cnt = A(1, 1); // get the rows in the list
    for i = 1 : cnt, // loop through the rows
        if name == _list(i) then // look for a name match
            foo = i; // record the matching row number
            break; // no point continuing the for loop
        end,
    end
    if foo == -1 then mprintf("ERROR: list name %s not found\n", name); end
endfunction

vertex_names = ["1" ; "2" ; "3" ; "4" ; "5"]; // define the vertices

function foo = vert_number(name) // a function to find vertex numbers given names
    foo = find_index(vertex_names, name);
endfunction

edge_names = ["B" ; "C" ; "D" ; "E" ; "F" ; "G"]; // define the edges.

function foo = edge_number(name) // a function to find edge numbers given names
    foo = find_index(edge_names, name);
endfunction

edge = [edge_number("B"), vert_number("1"), vert_number("2")];
edge = [edge ; [edge_number("C"), vert_number("1"), vert_number("3")]];
edge = [edge ; [edge_number("D"), vert_number("2"), vert_number("3")]];
edge = [edge ; [edge_number("E"), vert_number("2"), vert_number("4")]];
edge = [edge ; [edge_number("F"), vert_number("3"), vert_number("5")]];
edge = [edge ; [edge_number("G"), vert_number("4"), vert_number("5")]];

```

• Representing a graph in C

```

int find_number(char list[][], char *name){
    int i;
    if(name == NULL) return -1;
    for(i = 0; list[i][0] != NULL; i++){
        if(strcmp(name, list[i]) == 0){
            return i;
        }
    }
    printf("ERROR: search name %s not found \n", name);
    return -1;
}

char vert_names[][] = {"1"}, {"2"}, {"3"}, {"4"}, {"5"}, NULL;
int vert_number(char *name){ return find_number(vert_names, name);}

char edge_names[][] = {"B"}, {"C"}, {"D"}, {"E"}, {"F"}, {"G"}, NULL;
int edge_number(char *name){ return find_number(edge_names, name);}

#define MAX_EDGES 10
int edge_cnt = 0;
int edges[MAX_EDGES][3];

void add_edge(char *edge, char *from_vert, char *to_vert){
    edges[edge_cnt][0] = edge_number(edge);
    edges[edge_cnt][1] = vert_number(from_vert);
    edges[edge_cnt][2] = vert_number(to_vert);
    edge_cnt++;
}

int main(){
    add_edge("B", "1", "2");
    add_edge("C", "1", "3");
    add_edge("D", "2", "3");
    add_edge("E", "2", "4");
    add_edge("F", "3", "5");
    add_edge("G", "4", "5");

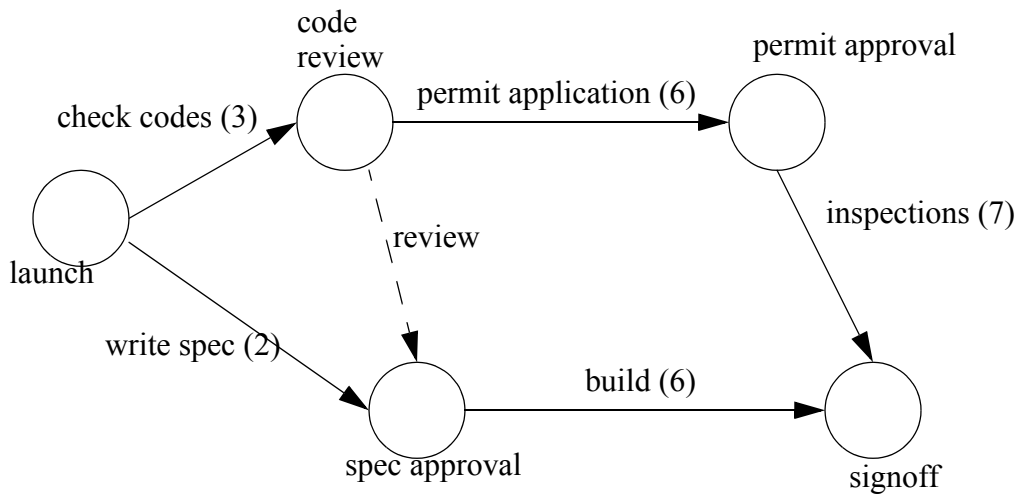
    return 0;
}

```

3.3 Applications

3.3.1 Precedence Identification

- To determine a precedence we add a precedence count for each vertex. The basic check is,
 1. Set all the precedence values to 1 (lowest precedence)
 2. Cycle through and check each edge. For each edge ensure that the 'to' vertex has a precedence that is at least one greater than the 'from' vertex.
- Consider a Network Diagram for a CPM problem.



- The algorithm implemented in Scilab (assuming the 'find_index' function is the same as before)

```

vert_names = ["launch" ; "code review" ; "spec approval" ; "permit approval" ; "signoff"]; // vertices
A = size(vert_names); // find the rows and columns in the list
vert_cnt = A(1, 1); // get the rows in the list

edge_names = ["check codes" ; "write spec" ; "permit application" ; "review" ; "build" ; "inspections"]; /
/ define the edges.
A = size(edge_names); // find the rows and columns in the list
edge_cnt = A(1, 1); // get the rows in the list

//----- Previous Code -----

function foo = find_index(_list, name) // a function to find list numbers given names
    foo = -1; // use -1 to indicate no match found yet
    A = size(_list); // find the rows and columns in the list
    cnt = A(1, 1); // get the rows in the list
    for i = 1 : cnt, // loop through the rows
        if name == _list(i) then // look for a name match
            foo = i; // record the matching row number
            break; // no point continuing the for loop
        end,
    end
    if foo == -1 then mprintf("ERROR: list name %s not found\n", name); end
endfunction

function foo = vert_number(name) // a function to find vertex numbers given names
    foo = find_index(vert_names, name);
endfunction
function foo = edge_number(name) // a function to find edge numbers given names
    foo = find_index(edge_names, name);
endfunction

//-----

// Note: an extra column is added to include the time for each activity
edge = [edge_number("check codes"), vert_number("launch"), vert_number("code review"), 3];
edge = [edge ; [edge_number("write spec"), vert_number("launch"), vert_number("spec approval"), 2]];
edge = [edge ; [edge_number("permit application"), vert_number("code review"), vert_number("permit
approval"), 6]];
edge = [edge ; [edge_number("review"), vert_number("code review"), vert_number("spec approval"), 0]];
edge = [edge ; [edge_number("build"), vert_number("spec approval"), vert_number("signoff"), 6]];
edge = [edge ; [edge_number("inspections"), vert_number("permit approval"), vert_number("signoff"), 7]];

// initialize all precedence values to 1
precedence = [1];
for i = 2 : vert_cnt,
    precedence = [precedence ; 1];
end

// Loop through and update precedences
for i = 1 : edge_cnt - 1,
    for j = 1 : edge_cnt,
        prev_vert = edge(j, 2); // find the previous and current vertices
        next_vert = edge(j, 3);
        prev_precedence = precedence(prev_vert); // find the precedence of
previous/next vertices
        next_precedence = precedence(next_vert);
precedence needs to be updated
        if (prev_precedence + 1) > next_precedence then // check to see if the
            precedence(next_vert) = prev_precedence + 1;
        end,
    end,
end

```

3.3.2 Graph Searching

- Identifying the vertex costs.

```

// Assumes the code for vertices has already been defined and run.

// initialize all vertex cost values to 0
cost = [0];
for i = 2 : vert_cnt,
    cost = [cost ; 0];
end

maximum_cost = 0;
last_vert = 0;

// Loop through and update precedences
for i = 1 : edge_cnt - 1,
    for j = 1 : edge_cnt,
        prev_vert = edge(j, 2); // find the previous and current vertices
        next_vert = edge(j, 3);
        edge_cost = edge(j, 4);
        prev_cost = cost(prev_vert); // find the cost of previous/next vertices
        next_cost = cost(next_vert);
        if (prev_cost + edge_cost) > next_cost then // check to see if the cost
needs to be updated
            cost(next_vert) = prev_cost + edge_cost;
        end,
        if cost(next_vert) > maximum_cost then
            maximum_cost = cost(next_vert);
            last_vert = next_vert;
        end
    end,
end

mprintf("The critical path cost is %f \n", maximum_cost);

```

3.4 Other Topics

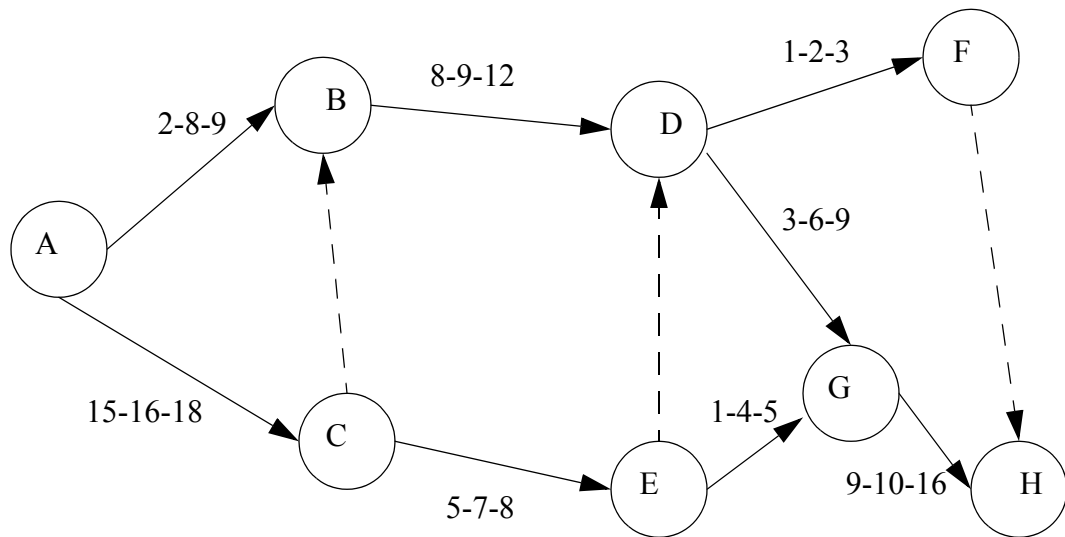
- ADD: resource limited plans

1.5 Summary

•

1.6 Problems

1. Write a general program that will accept a network diagram entered as vertices and arcs. The program will then do a PERT analysis to determine the T_e value and the standard deviation. Test the program with the following network diagram.



(ans. $T_e=42.333$, S.D.=1.826)

1.7 Challenge Problems