Problem C5. Graphs, Shortest Path

What is the Shortest Path through the graph shown below using Dijkstra’s algorithm?

Show all the steps in the computation of the shortest path.

Initialize

\[ V = \{a, b, c, d, e\} \]
\[ E = \{(a,b), (a,c), (b,e), (b,c), (c,b), (c,d), (c,e), (d,a), (d,e), (e,d)\} \]
\[ S = \{\emptyset\} \]
\[ Q = \{a, b, c, d, e\} \]

\[ D = [0, \infty, \infty, \infty, \infty] \]
Previous = [0, 0, 0, 0, 0]

Start at A

Relax (a,b,10)
Relax (a,c,4)

\[ S = \{a\} \]
\[ Q = \{b, c, d, e\} \]

\[ D = [0, 10, 4, \infty, \infty] \]
Previous = [0, a, 0, 0, 0]

Move to C

Relax(c,b,1)
Relax (c,d,2)
Relax(c,e,9)

\[ S = \{a,c\} \]
\[ Q = \{b, d, e\} \]
D = [0, 5, 4, 6, 13]
Previous = [0, c, a, c, c]

Move to B
Relax(b,e,1)
Relax(b,c,3)

S = {a, c, b}
Q = {d, e}

D = [0, 5, 4, 6, 6]
Previous = [0, c, a, c, b]

Move to D (You can also move to E)
Relax(d,a,6)
Relax(d,e,4)

S = {a, c, b, d}
Q = {e}

D = [0, 5, 4, 6, 6]
Previous = [0, c, a, c, b]

Move to E
Relax(e,d,4)

S = {a, c, b, d, e}
Q = {}

D = [0, 5, 4, 6, 6]
Previous = [0, c, a, c, b]
C6 Tree Traversal

a. 
   i. Inorder Traversal: a/b-c*d*e/f+g
   ii. Preorder Traversal: * -/ a b c / * d e + f g
   iii. Postorder Traversal: ab/c-de*fg+/*

b. Algorithm for generating an expression tree from a prefix expression

1. Read the prefix expression from right to left
2. Examine the next element in the input.
3. If it is operand then
   a. create a leaf node i.e. node having no child (node->left_child=node->right_child=NULL)
   b. copy the operand in data part
   c. PUSH node's address on stack
4. If it is an operator, then
   a. create a node
   b. copy the operator in data part
   c. POP address of node from stack and assign it to node->left_child
   d. POP address of node from stack and assign it to node->right_child
   e. PUSH node's address on stack
5. If there is more input go to step 2
6. If there is no more input, POP the address from stack, which is the address of the ROOT node of Expression Tree.

c. Procedure for creating an expression tree from a prefix expression

```text
procedure Create_Expression_Tree_From_Prefix (Root : out Nodeptr) is
Temp,
Leftchild,
Rightchild : Nodeptr;
Nodeptr_Stack : My_Stack;
Length        : Integer;
Expression    : String (1 .. 80);
begin
   -- create a temporary stack for
   My_Pointer_Stack.Create(Nodeptr_Stack);
   Get_Line(Expression,Length);
   for I in reverse 1 .. Length loop
      if Isdigit(Expression(I)) then
         Temp := new Node;
         Temp.Element := Expression(I);
         Temp.Left_Child := null;
```

Temp.Right_Child := null
Push(Nodeptr_Stack,Temp);
end if;

if Isoperator(Expression(I)) then
    Temp := new Node;
    Temp.Element := Expression(I);
    Pop(Nodeptr_Stack, Leftchild);
    Pop(Nodeptr_Stack, Rightchild);
    Temp.Left_Child := Leftchild;
    Temp.Right_Child := Rightchild;
    Push(Nodeptr_Stack, Temp);
end if;
end loop;
Pop(Nodeptr_Stack, Temp);
if Isempty(Nodeptr_Stack) then
    Root := Temp;
else
    Put_Line("Cannot create the expression tree");
    Root := null;
end if;
end Create_Expression_Tree_From_Prefix;
C7 Spanning Trees/ Depth First and Breadth First Traversal

a. The modification to `expression_tree.ads` is shown below:

```pascal
procedure Traverse_Dfs( Root : in Nodeptr);
procedure Traverse_Bfs( Root : in Nodeptr);
```

b. Implementation of depth-first and depth first search

```pascal
procedure Traverse_Dfs (  
    Root : in    Nodeptr) is  
begin  
    Traverse_Preorder(Root);  
end Traverse_Dfs;

procedure Traverse_Bfs (  
    Root : in    Nodeptr) is  
    -- create a queue to hold node pointers  
    My_Bfs_Queue : My_Queue;  
    Temp         : Nodeptr;  
begin  
    -- initialize the queue  
    Init_Queue(My_Bfs_Queue);  
    --queue in the root node  
    Enqueue(My_Bfs_Queue, Root);  
    -- loop until there are no nodes in the queue  
    loop  
       exit when Empty_Queue(My_Bfs_Queue);  
       --get the first node from the queue  
       Dequeue(My_Bfs_Queue, Temp);  
       -- display the element  
       Ada.Text_Io.Put(Temp.Element);  
       Ada.Text_Io.New_Line;  
       --if the left child is not null, enqueue it  
       if Temp.Left_Child /= null then  
           Enqueue(My_Bfs_Queue, Temp.Left_Child);  
       end if;  
       --if the right child is not null, enqueue it  
       if Temp.Right_Child /= null then  
           Enqueue(My_Bfs_Queue, Temp.Right_Child);  
       end if;  
    end loop;  
null;  
end Traverse_Bfs;
```
c. Minimum Weight Spanning Tree Using Prim’s Algorithm

Step 1.

Step 2
Step 3.

MST

E \rightarrow F \quad 1 \rightarrow 3

Parent

A \quad F \quad E

B \quad G \quad D \quad C \quad H

Fringe Set

Step 4

MST

E \rightarrow F \quad 1 \rightarrow 3

D

Parent

E \quad D

H \quad B \quad C \quad G

Fringe Set
1.\[ g(t) \]
\[ e^{-2t} \]

\[ -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3 \quad t \]

\[ u(t) \]
\[ e^{-t} \]

\[ -2 \quad -1 \quad 1 \quad 2 \quad 3 \quad t \]

2. To convolve, flip \( g(t) \), slide left and right, multiply by \( u(t) \), and find the area.

There will be 3 important ranges:

\[ t < -1 \quad \text{no overlap} \]

\[ -1 \leq t < 1 \quad \text{overlap from } -1 \text{ to } t \]

\[ t > 1 \quad \text{overlap from } -1 \text{ to } 1 \]

These are illustrated below:
So the convolution will look like:

\[ y(t) \]
3. The convolution is:

\[ t < -1 \quad y(t) = 0 \]

\[ -1 < t < 1 \quad y(t) = \int_{-1}^{t} e^{-2(t-\tau)} e^{-\tau} d\tau \]

\[ = e^{-2t} \int_{-1}^{t} e^{\tau} d\tau \]

\[ = e^{-2t} \left. e^{\tau} \right|_{\tau=-1}^{t} \]

\[ = e^{-2t} (e^{t} - e^{-1}) \]

\[ = e^{-t} - e^{-2t-1} \]

\[ t > 1 \quad y(t) = \int_{-1}^{1} e^{-2(t+\tau)} e^{-\tau} d\tau \]

\[ = e^{-2t} \left. e^{\tau} \right|_{\tau=-1}^{1} \]

\[ = e^{-2t} (e^{1} - e^{-1}) \]

So,

\[ y(t) = \begin{cases} 
0 & t < -1 \\
\frac{e^{-t} - e^{-2t-1}}{2} & -1 < t < 1 \\
e^{-2t} (e^{1} - e^{-1}) & t > 1
\end{cases} \]

4. This part is very much like S6. See S6 solution for general approach. The result is
1. Because \( g(t) \) & \( u(t) \) are piecewise constant, \( y(t) \) will be continuous and piecewise linear. We can find \( y(t) \) at the "corners" by evaluating at \( t = \text{integer} \), since the "corners" of \( g(t) \) & \( u(t) \) occur at the integers.

So do flip & slide:

There is no overlap for \( t < -5 \) or \( t > 5 \). So do \( t = -4, -3, -2, \ldots, 4 \):

\[ y(-4) = \int g(t-4)u(t-4) \, dt = 1 \]

\( t = -3; \)
So \( y(-3) = \int g(t-\tau)u(\tau) \, d\tau = 0 \)

Continuing in this fashion, we have

<table>
<thead>
<tr>
<th>( t )</th>
<th>( y(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>1</td>
</tr>
<tr>
<td>-3</td>
<td>0</td>
</tr>
<tr>
<td>-2</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

So \( y(t) \) is:

2. By linearity and time invariance, delaying \( u(t) \)
    by \( T \) will simply delay \( y(t) \), so the convolution
    \( g(t) \times u(t-T) \) is as above, shifted right by \( T \).
3. T is easily identified as the time at which the max of y(t) occurs.

4. Since we are modeling a notch filter we want one frequency and one time to "show up". So we can't use a "flipped" u(t) for y(t) because they will match at only one point and "conflict" at other points producing small values for y(t).