3. The bottleneck TSP problem is the problem of finding a Hamiltonian cycle in a given weighted undirected graph such that the length of the longest edge in the cycle is minimized. Assuming that the input graph satisfies the triangle inequality, show that this problem has a polynomial-time approximation algorithm with ratio 3.

Important Notes

(1) Hints for the problem are provided in CLRS (page 1033).
(2) The solution given below is not mine - it is an expansion of a solution provided by a document found at the URL below:

http://www.mpi-sb.mpg.de/~rbeier/opt2001/loesung10.pdf

Let G = (V, E) be the graph given by the problem, and let T be a MST for G. As mentioned in the discussion session, it suffices to show that it is possible to traverse T using shortcuts without shortcutting more than two consecutive vertices. Moreover, it is sufficient to show how, beginning at some vertex v with parent p, there is such a traversal that visits every descendant of p and ends with p itself. If we can accomplish this, then a Hamiltonian cycle for T is constructed by

(1) Choosing an arbitrary non-leaf root r.
(2) Following the edge from r to the first child of r, c.
(3) Applying the procedure described below to obtain a traversal beginning with c and ending with r.

The pseudocode given below is an attempt to precisely describe such a traversal (I am not confident that the code is without error). The code is an interpretation of the algorithm specified (somewhat less precisely) by the document referenced above. The input to the recursive procedure Walk() is a vertex v and its parent p.

We use the terms "first", "second", …, "last" to refer to the positions of a vertex v's children. So, "first child" refers to v's leftmost child, "second child" refers to the first child's sibling to the right, and so on. We assume for simplicity that, given a non-leaf node v, all of v's leaf children are to the left of all of v's non-leaf children.

(Incidentally, the following is slightly different than what I put on the whiteboard this evening)

Walk(v,p)

```
If v is a leaf,
    If v has no sibling to its right,
        Move to p and return.
    If v has leaf siblings,
        Traverse all of v's leaf siblings.
    If v has non-leaf-siblings s_1, s_2, …, s_k,
        Let s_{i-first-child} be the first child of s_i for 1 ≤ i ≤ k.
        Walk(s_{1-first-child}, s_1).
        Walk(s_{2-first-child}, s_2).
        …
        Walk(s_{k-first-child}, s_k).
        Move to p and return.
```

Else (v is not a leaf),
    If v has leaf children,
        Traverse all of v's leaf children.
    If v has non-leaf-children $c_1,c_2,\ldots,c_k$,
        Let $c_{i\text{-first-child}}$ be the first child of c_i for $1 \leq i \leq k$.
        Walk($c_{1\text{-first-child}},c_1$).
        Walk($c_{2\text{f-irst-child}},c_2$).
        …
        Walk($c_{k\text{-first-child}},c_k$).
    If v has no sibling to its right,
        Move to p and return.
    Else (v has a sibling s to its right),
        Walk(s,p).