**Q2:** Acyclic Pathfinder

The graph is represented by the setf function on lines 1 – 6. Each node along with its successors are stored as a tuple with the nodes successors being stored as ‘successors.

The path finding algorithm find-path is defined from lines 9-22. Recursively loops over all possible paths until a path from the start node to the end node is found, which is then printed. It also stops the recursion if an already visited node is trying to be added to the path. Lines 23-24 are just an abstraction function to hide the passing in of a nil argument to find-path.

Function Call: (pathfinder 'A 'C)

Output: (A B C)

(A B E C)

(A E B C)

(A E C)

Function Call: (pathfinder 'C 'D)

Output: (C D)

(C E D)

Function Call: (pathfinder 'A 'D)

Output: (A B C D)

(A B C E D)

(A B E D)

(A B E C D)

(A E B C D)

(A E D)

(A E C D)

**Q3.** Shortest Path

The graph designated for Q3 is represented in the same way as Q2 but with an extra data abstraction called weights, which stores the weights of the associated vertex in the same index as the corresponding child node.

The path finding algorithm is identical to the one in Q2 except it adds the total weight of the path found to the end of the final path. This is then passed in to the shortest path function on lines 31-40. This function simply iterates over the list of paths found from the previous function call, stored in \*paths\* and prints the one with the lowest total weight (i.e the shortest path).

Function Call: (shortest-path 'A 'C)

Output: (A B C 4)

Function Call: (shortest-path 'A 'D)

Output: (A B C D 6)

Function Call: (shortest-path 'D 'B)

Output: (D B 2)

Q4: Syllables

Iterates over the hard coded list \*word\* as I had a difficult time figuring out how to represent the problem in a recursive way. This implementation ended up being quite simple, as it requires checking membership of the already processed letters in the vowel or consonant list and looking ahead 2 letters in the case for the second syllable type. If the first case or second case is matched in retval, a ‘-’ is pushed on the list to separate the syllables.

\*word\* = ‘(a n a l o g)

Function Call: (syllables \*word\*)

Output: (A N A - L O G)

\*word\* = ‘(b u s t e r)

Function Call: (syllables \*word\*)

Output: (B U S - T E R)

\*word\* = ‘(a n a c o n d a)

Function Call: (syllables \*word\*)

Output: (A N A - C O N - D A)