EXPERIMENT 11

Objective: Implementation of the Hungarian algorithm for maximum weight matching.

Brief Theory:

The Hungarian algorithm, also known as the Kuhn-Munkres algorithm, is used to find the maximum weight matching in a weighted bipartite graph. This matching maximizes the total weight of the selected edges while ensuring no two edges share a vertex.

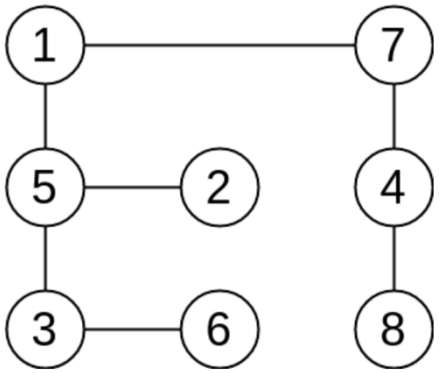
Key Concepts:

1. Weighted Bipartite Graph: A graph with two disjoint sets of vertices where edges between sets have weights.
2. Matching: A subset of edges where no two edges share a vertex.
3. Maximum Weight Matching: A matching where the total weight of the edges is maximized.

```
FUNCTION KuhnMunkres(weight_matrix):
  N ← number of vertices on each side (square matrix assumed)
  label_U ← array of size N, initialized as max of each row in weight_matrix
  label_V ← array of size N, initialized to 0
  matching_V ← array of size N, initialized to -1

  FOR u FROM 0 TO N - 1:
    slack ← array of size N, initialized to ∞
    slack_from ← array of size N
    prev ← array of size N, initialized to -1
    visited_U ← array of size N, initialized to FALSE
    visited_V ← array of size N, initialized to FALSE

    queue ← empty queue
    queue.push(u)

    WHILE TRUE:
      WHILE queue IS NOT empty:
        current_U ← queue.pop()
        visited_U[current_U] ← TRUE

        FOR v FROM 0 TO N - 1:
          IF NOT visited_V[v]:
            delta ← label_U[current_U] + label_V[v] - weight_matrix[current_U][v]
            IF delta < slack[v]:
              slack[v] ← delta
              slack_from[v] ← current_U
          IF slack[v] == 0:
            visited_V[v] ← TRUE
            IF matching_V[v] == -1:
              // Augmenting path found
              UPDATE matching using prev and slack_from
              GOTO next_u
            ELSE:
              queue.push(matching_V[v])
              prev[matching_V[v]] ← current_U

      // Update labels
      delta ← min(slack[v] for v WHERE NOT visited_V[v])
      FOR i FROM 0 TO N - 1:
        IF visited_U[i]: label_U[i] ← label_U[i] - delta
        IF visited_V[i]: label_V[i] ← label_V[i] + delta
        ELSE: slack[i] ← slack[i] - delta

      FOR v FROM 0 TO N - 1:
        IF NOT visited_V[v] AND slack[v] == 0:
          visited_V[v] ← TRUE
          IF matching_V[v] == -1:
            // Augmenting path found
            UPDATE matching using prev and slack_from
            GOTO next_u
          ELSE:
            queue.push(matching_V[v])
            prev[matching_V[v]] ← slack_from[v]

    LABEL next_u

  RETURN matching_V
```

Task:

1) Implement the Hungarian algorithm and compute the maximum weight matching for the given graph. Display the matching and the total weight.



Apparatus and components required: Computer with C or C++ Compiler and Linux/Windows platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.