```
1.  get_score( e_1: tuple, e_2: tuple):
2.    --------------------------------
3.    props_1 = get_edge_props( e_1 )  // List[str]
4.    props_2 = get_edge_props( e_2 )  // List[str]
5.
6.    // this will create a full bipartite graph between props_1 and props_2
7.    similarity_edges = get_edges_weights( props_1, props_2 ) // List[Tuple[str, str, float]]
8.
9.    // clustering is using AgglomerativeClustering of sklearn.cluster
10.   // https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html
11.   // distance_threshold  → how close the props in the cluster
12.   clusters_props_1 = clustering( props_1, distance_threshold ) // Dict[int, List[str]]
13.   clusters_props_2 = clustering( props_2, distance_threshold ) // Dict[int, List[str]]
14.
15.   // between every two clusters (from the opposite side of the bipartite) we will take
16.   // only one edge, which will be the one with the maximum weight.
17.   clusters_edges = get_clusters_edges( similarity_edges, clusters_props_1, clusters_props_1 )
18.
19.   // we want the maximum-weight of full bipartite matching
20.   // we will use networkx algorithm of minimum_weight_full_matching
21.   // https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.bipartite.matching.minimum_weight_full_matching.html
22.   best_matching = maximum_weight_full_matching( clusters_edges )
23.   --------------------------------
24.   return TODO
```

```
1.  get_edge_props( subject: string, object: string ):
2.    --------------------------------
3.    props_1 = get_props_from_quasimodo( subject, object, n_largest = 10 )  // sorted by plausibility
4.    props_2 = get_props_from_google( subject, object )  // why do, why does, how do, how does
5.    props_3 = get_props_from_conceptnet( subject, object )  // sorted by concept-net weights
6.    --------------------------------
7.    return props_1 + props_2 + props_3
```

```
1.  get_edges_weights( props_edge_1: List[string], props_edge_2: List[string] ):
2.    --------------------------------
3.    edges = []
4.    for p_1 in props_edge_1:
5.      for p_2 in props_edge_2:
6.        // similarity is calculated by cosine-similarity.
7.        // https://pytorch.org/docs/stable/generated/torch.nn.CosineSimilarity.html
8.        edges.append( (p_1, p_2, similarity( p_1,p_2 )) )
9.    --------------------------------
10.   return edges
```