Exercise 5

Introduction

This exercise requires to implement a Greedy Algorithm that solves the same problem of the previous exercise, that is the software assignment problem. Greedy programming is an algorithmic paradigm that builds up a solution step by step, always choosing the solution that offers the most obvious and immediate benefit. A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment.

This means that the algorithm picks the best solution in that moment without regard for consequences. So, the problems where choosing locally optimal also leads to a near globally optimal solution are best fit for Greedy.

The choices in a Greedy Algorithm are made according to a principle of local optimality and in each step, a subproblem of smaller and smaller dimension is solved. The solution depends on the previous choices, but not on future ones.

Greedy Algorithm Properties

Advantages:

- They are easier to implement.
- They are easier to understand.
- They require much fewer computing resources.
- They are much faster to execute.
- Greedy algorithms are used to solve optimization problems.

Disadvantages:

- Their only disadvantages are that they do not always reach the global optimum solution.
- On the other hand, even when the global optimum solution is not reached, most of the times the reached result is a very good solution.

Solution chosen

In our solution, we implemented three function: “call_func(g)”, “_func(g,degrees,edges)” and “_delete_incident_edges(g,edge,vertex)”. In the first function we defined a set “edges” in which all the edges of the graph are saved with “edges = g.edges()” assignment, and we initialized a map called “degrees” in which the keys are the vertices of the graph and the values are the vertices’ degree. After this, the function returns the solution of “_func(g,degrees,edges)” method that is called passing three parameters:

- “g” is the graph;
- “degrees” is the map;
- “edges” is the set of the edges.

In this function, we declared a list called “solution” in which will be saved all the vertices where the software will be installed (i.e. the solution of our problem). Then, we used a loop that will be run

until the length of the "edges" set becomes zero. As first thing, we select the vertex with higher degree among those in the map "degrees" using the "selected_vertex = max(degrees.items(), key=operator.itemgetter(1))[0]" statement. After choosing the vertex, we call the "_delete_incident_edges(g,degrees,edges,selected_vertex)" method. In this function, we implemented a foreach to visit all the incident edges of the selected vertex using "for e in g.incident_edges(vertex)" statement and deleting all the incident edges that are already included in the edges set with "edges.remove(e)" statement. At this point, we take the endpoints of the edge with the "u,v = e.endpoints()" statement and we decrease by one the vertex's degree for each incident edges. The "endpoints()" function, a method in the nested Edge class in the graph library, returns a tuple for vertices "u,v". So, the first is the origin of the edge and the second is its destination.

When the "_delete_incident_edges(g,degrees,edges,vertex)" method is finished, the selected vertex is added in the solution list with the "solution.append(selected_vertex)".

The loop ends when the "edges" length becomes equal to zero. At this point, the function returns the list of selected vertices.

We used a map because it is very convenient for the operations that have been carried out, for example the search of the vertex with higher degree takes constant time O(1). Our greedy choice is that to take the vertex with higher degree and deleting all the incident edges of the selected vertex. Then, we decrease, for each edge, the degree of its endpoints.

Computational complexity

In this case, the data structure for representing the graph is an adjacency map in which for each vertex v its adjacency list is implemented with a Map. Adjacency map essentially achieves optimal running times for all methods, making an excellent all-purpose choice as a graph representation. In our algorithm, we have an initialization phase in which we call the "edges()" method only one time, that requires O(m) to be performed and we initialize the degrees' map that takes O(n). Then, in each iteration, we select from the map the vertices with maximum degree. This operation takes O(n) to be performed. After this, for each selected vertex, we have to remove all its incident edges and update the degrees' map (that takes O(1)). The function "delete_incident_edges(g,edges,vertex)" has a time complexity of $O(d_{vertex})$. In conclusion the algorithm is performed in $O(m(n+ d_{vertex}))$.

Space complexity

Let n the number of the vertices and m the number of the edges, in our algorithm we save the vertices in a map and the edges in a set. So, the space complexity of the algorithm is O(n + m).

Solution Bound

As said before, the algorithm does not provide a global optimum solution. In order to calculate the solution bound of our algorithm, we tested it on two kinds of graphs, with 450 and 1150 nodes respectively. The optimal solutions of the graphs are known, so we calculated the approximation ratio and the solution bound of the algorithm.

Benchmark	Number of the vertices	Optimal Solution	Greedy algorithm	Approximation ratio
frb30-15-1	450	420	429	1.021
frb30-15-2	450	420	431	1.026
frb30-15-3	450	420	429	1.021
frb30-15-4	450	420	429	1.021
frb30-15-5	450	420	430	1.023
frb50-23-1	1150	1100	1130	1.027
frb50-23-2	1150	1100	1129	1.026
frb50-23-3	1150	1100	1130	1.027
frb50-23-4	1150	1100	1127	1.024
frb50-23-5	1150	1100	1127	1.024

According to these results, we calculated the average approximation of the graphs:

- First five graphs with 450 nodes: (1.021 + 1.026 + 1.021 + 1.021 + 1.023)/5 = 1.022
- Last five graphs with 1150 nodes: (1.027 + 1.026 + 1.027 +1.024 + 1.024)/5 = 1.025

The solution bound is the average of the two average approximation times the optimal solution:

- **Solution Bound:** ((1.022 + 1.025)/2) * (optimal solution) = 1.023*(optimal solution)