

PRÁCTICA III

Algoritmos Greedy

Por: Antonio Gámiz

Problema:

Un electricista necesita hacer n reparaciones urgentes, y sabe de antemano el tiempo que le va a llevar cada una de ellas: en la tarea i -ésima tardará t_i minutos. Como en su empresa le pagan dependiendo de la satisfacción del cliente, necesita decidir el orden en el que atenderá los avisos para minimizar el tiempo medio de atención de los clientes (desde el inicio hasta que su reparación sea efectuada).

Problema (segunda versión):

Adaptar el problema a N electricistas.

[Aclaración]: en ninguno de los algoritmos cierro el camino, ya que el problema busca que los problemas sean resueltos en el menor tiempo posible, por lo que el tiempo que tardaría en volver el electricista después de haber terminado el último trabajo, no influye en el tiempo que ha tardado en realizar los anteriores.

Función de selección y “aprobación”

```
bool cycle(vector<node>& v, struct node new_node){ return is_in(new_node.label, v); }

int nodeHeuristic(graph &g, struct node n, float min_bound, vector<node> taken){
    float min=LONG_MAX;
    int index_min=-1;
    float current_weight;

    for(int i=0; i<g.size(); i++)
    {
        current_weight=g.get_weight(i, n.label);
        if( current_weight > 0 && current_weight < min && current_weight > min_bound && !is_in(i, taken) )
        {
            min=current_weight;
            index_min=i;
        }
    }

    return index_min;
}
```

Algoritmo greedy

```
vector<node> greedy(graph &g, int begin)
{
    vector<node> nodes=g.get_nodes();
    vector<node> solution;

    int index_nextNode;

    struct node current_node=nodes[begin];
    bool first_time=true;
    solution.push_back(current_node);

    struct node nextNode;
    first_time=true;

    while(solution.size() < nodes.size())
    {
        first_time=true;
        do
        {
            index_nextNode=nodeHeuristic(g, current_node, (first_time)? 0:g.get_weight(current_node.label, nextNode.label), solution);
            if( index_nextNode==-1 ) return vector<node>();
            nextNode=nodes[index_nextNode];

            first_time=false;
        } while( cycle( solution, nextNode ) );

        solution.push_back(nextNode);
        current_node=nextNode;
    }

    return solution;
}
```

Adaptación a N-electricistas

```
vector<vector<node> > greedy(graph &g, int begin, int n_electricians)
{
    vector<node> nodes=g.get_nodes();
    vector<vector<node> > solution(n_electricians);
    vector<node> taken;

    int index_nextNode;


    struct node current_node=nodes[begin];
    taken.push_back(current_node);
    bool first_time=true;

    for(int i=0; i<n_electricians; i++)
        solution[i].push_back(current_node);

    struct node nextNode;
    first_time=true;
    int j=0;
    while( taken.size() < nodes.size() )
    {
        if(j==n_electricians) j=0;
        current_node=solution[j].back();
        /*-----searching next node-----*/
        first_time=true;
        do
        {
            index_nextNode=nodeHeuristic(g, current_node, (first_time)? 0:g.get_weight(current_node.label, nextNode.label), taken);
            if( index_nextNode==-1 ) return vector<vector<node> >();
            nextNode=nodes[index_nextNode];

            first_time=false;
        } while( cycle( solution[j], nextNode ) );
        /*-----searching next node-----*/
        solution[j].push_back(nextNode);
        taken.push_back(nextNode);
        j++;
    }

    return solution;
}
```



FIN