Analiza algorytmów

Dokumentacja wstępna projektu

Zagadnienie:

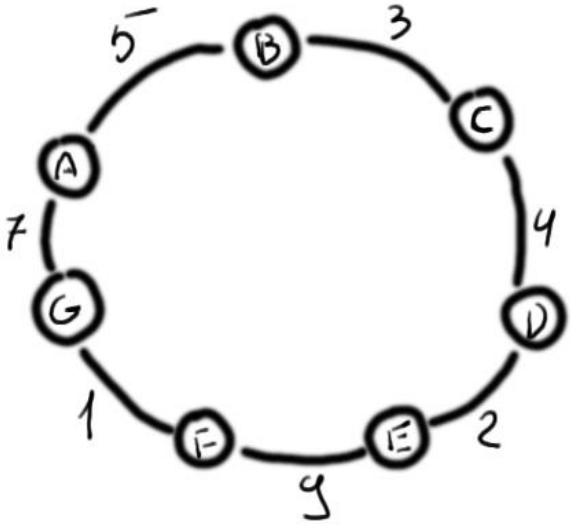
Wzdłuż obwodnicy miejskiej ustawiono maszty, na których znajdują się kamery monitoringu miejskiego. Następnie spostrzeżono, że znaczna część kierowców przekracza prędkość. Postanowiono zainstalować radary mierzące prędkość a do ich instalacji wykorzystać istniejące maszty. Postanowiono, że radary będą zainstalowane na najbardziej oddalonych od siebie masztach.

Mając N masztów i informacje o odległości pomiędzy sąsiadującymi masztami zaproponuj algorytm, który wybierze dwa maszty, na których powinny zostać zainstalowane radary. Oceń jego złożoność czasową oraz pamięciową.

Założenia:

• Maszty nie są ustawione w równych odległościach

• Obwodnica jest drogą, którą można przedstawić jako graf cykliczny



Algorytmy:

1. Algorytm naiwny/brutalny, czyli sprawdzanie każdej pary masztów.

```
list<IDa, IDb, cost> max;
int actualCost = 0;

list<ID, x, y, cost> vert = readVertices(FILE* file);
int bestCost = sumDistance(vert) >> 1;
pair<ID, cost> first = pair<1, vert.at[0].cost>
pair<ID, cost> last = pair<vert.length, vert.at[vert.length-1].cost>

for(pair<ID, cost> chosen = first; chosen != last; chosen++){
    for(pair<ID, cost> target = chosen++; target != chosen; target++){
        for(pair<ID, cost> temp = chosen; temp != target; temp++)
            actualCost += temp.cost;

        if(actualCost < bestCost)
            if(actualCost > max.cost){
                max.cost = actualCost;
                max.IDa = chosen.ID;
                max.IDb = target.ID;
            }
        actualCost = 0;
    }
}
for(pair<ID, cost> target = first; target != chosen; target++){
    for(pair<ID, cost> temp = chosen; temp != target; temp++){
        actualCost += temp.cost;
    }
    if(actualCost < bestCost)
        if(actualCost > max.cost){
            max.cost = actualCost;
            max.IDa = chosen.ID;
            max.IDb = target.ID;
        }
}
printf(max);
```

2. Algorytm autorski.

```
list<IDa, IDb, cost> max;
int actualCost = 0;

list<ID, x, y, cost> vert = readVertices(FILE* file);
int bestCost = sumDistance(vert) >> 1;
pair<ID, cost> first = pair<1, vert.at[0].cost>;
pair<ID, cost> last = pair<vert.length, vert.at[vert.length-1].cost>;
pair<ID, cost> prev = pair<0,0>;

for(pair<ID, cost> chosen = first; chosen != last; chosen++){
    for(pair<ID, cost> temp = chosen;; temp++){
        actualCost += temp.cost;
        if(actualCost >= bestCost){
            if(prev.cost == bestCost || actualCost == bestCost){
                checkModule(temp.ID, actualCost, prev);
                return max;
            }
            if(max.cost > bestCost && max.cost < actualCost)
                checkModule(prev, max); //If prev better, than it's new max
            else if(max.cost < bestCost && max.cost > prev.cost)
                checkModule(temp.ID, actualCost, max); //The same as upper
            else if((max.cost < bestCost && max.cost < prev.cost) ||
                    (max.cost > bestCost && max.cost > actualCost)){
                checkModule(temp.ID, actualCost, prev); /The same as upper
            }
            break;
        }
        prev.ID = temp.ID;
        prev.cost = actualCost;
    }
    actualCost = 0;
}
for(pair<ID, cost> temp = chosen;; temp++){
    actualCost += temp.cost;
    if(actualCost >= bestCost){
        if(prev.cost == bestCost || actualCost == bestCost){
            checkModule(temp.ID, actualCost, prev);
            return max;
        }
        if(max.cost > bestCost && max.cost < actualCost)
            checkModule(prev, max); //If prev better, than it's new max
        else if(max.cost < bestCost && max.cost > prev.cost)
            checkModule(temp.ID, actualCost, max); //The same as upper
        else if((max.cost < bestCost && max.cost < prev.cost) ||
                (max.cost > bestCost && max.cost > actualCost)){
            checkModule(temp.ID, actualCost, prev); /The same as upper
        }
        break;
    }
    prev.ID = temp.ID;
    prev.cost = actualCost;
}
return max;
```