Analiza algorytmów

Dokumentacja wstępna projektu

Zagadnienie:

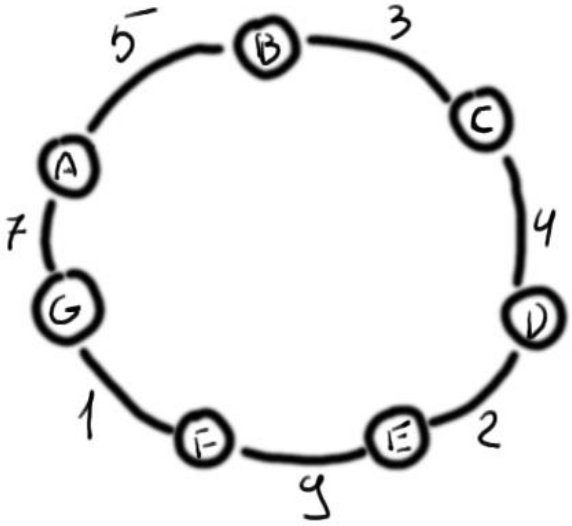
Wzdłuż obwodnicy miejskiej ustawiono maszty, na których znajdują się kamery monitoringu miejskiego. Następnie spostrzeżono, że znaczna część kierowców przekracza prędkość. Postanowiono zainstalować radary mierzące prędkość a do ich instalacji wykorzystać istniejące maszty. Postanowiono, że radary będą 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 o złożoności czasowej: $O(n^3)$ i złożoności pamięciowej: $O(n)$

```
vector<Vertex> vert = readVertices(FILE* file);
unsigned averageDistance = sumDistance(vert) >> 1;

pair<Vertex, Vertex> bestVertices;
unsigned bestDistance = 0;
unsigned actualDistance = 0;

pair<unsigned, pair<Vertex, Vertex>> result;

unsigned len = vert.length();

for(unsigned chosen = 0; chosen < len; chosen++){
    for(unsigned foo = (chosen+1)%len; foo != chosen; foo = (foo+1)%len){
        for(unsigned bar = chosen; bar != foo; bar = (bar+1)%len)
            actualDistance += vert.at(bar).getDistance();
        if(actualDistance <= averageDistance &&
           actualDistance > bestDistance){
                bestVertices.first = vert.at(chosen);
                bestVertices.second = vert.at(target);
                bestDistance = actualDistance;
            }
        actualDistance = 0;
    }
}
result.first = bestDistance;
result.second = bestVertices;
return result;
```

2. Algorytm autorski mniej naiwny o złożoności czasowej: $O(n^2)$ i złożoności pamięciowej: $O(n)$

```
#define D Distance
#define V Vertices

void writeNew(unsigned first, unsigned second, unsigned distance){
   bestV.first = vertices.at(first);
   bestV.second = vertices.at(second);
   bestD = distance;
}

vector<Vertex> vert = readV(FILE* file);
unsigned averageD = sumD(vert) >> 1;

pair<Vertex, Vertex> bestV;
unsigned bestD = 0;
unsigned actualD = 0;
unsigned prevD = 0;

pair<unsigned, pair<Vertex, Vertex>> result;

unsigned len = vert.length();

for(unsigned chosen = 0; chosen < len; chosen++){
    for(unsigned temp = chosen; temp != len; temp = (temp+1)%len){
        actualD += vert.at(temp).getD();
        if(actualD >= averageD){
            if(prevD == averageD || actualD == averageD){
                if((prevD - averageD) == 0)
                    writeNew(chosen, temp, prevD);
                else
                    writeNew(chosen, (temp+1)%len, actualD);
                result.first = bestD;
                result.second = bestV;
                return result;
            }
            else if(bestD > averageD && bestD < actualD)
                if((bestD - averageD) > (averageD - prevD))
                    writeNew(chosen, temp, prevD);
            else if(bestD < averageD && bestD > prevD)
                if((averageD - bestD) > (actualD - averageD))
                    writeNew(chosen, (temp+1)%len, actualD);
            else if((bestD < averageD && bestD < prevD) ||
                    (bestD > averageD && bestD > actualD)){
                if((actualD - averageD) >= (averageD - prevD))
                    writeNew(chosen, temp, prevD);
                else
                    writeNew(chosen, (temp+1)%len, actualD);
            }
            break;
        }
        prevD = actualD;
    }
    prevD = 0;
    actualD = 0;
}
result.first = bestD;
result.second = bestV;
return result;
```

3. Algorytm autorski o złożoności czasowej: $O(n)$ i złożoności pamięciowej: $O(n)$

```
#define D Distance
#define V Vertices

void writeNew(unsigned first, unsigned second, unsigned distance){
   bestV.first = vertices.at(first);
   bestV.second = vertices.at(second);
   bestD = distance;
}

vector<Vertex> vert = readV(FILE* file);
unsigned averageD = sumD(vert) >> 1;
pair<Vertex, Vertex> bestV;
unsigned bestD = 0;
unsigned actualD = 0;
unsigned prevD = 0;

pair<unsigned, pair<Vertex, Vertex>> result;
unsigned len = vert.length();
unsigned back = len;

for(unsigned front = 0; back != 0; front = (front+1)%len){
    if(actualD >= averageD){
        if(prevD == averageD || actualD == averageD){
            if((prevD - averageD) == 0)
                writeNew(back, (front-1)%len, prevD);
            else
                writeNew(back, front, actualD);
            result.first = bestD;
            result.second = bestV;
            return result;
        }
        else if(bestD > averageD && bestD < actualD)
            if((bestD - averageD) > (averageD - prevD))
                writeNew(back, (front-1)%len, prevD);
        else if(bestD < averageD && bestD > prevD)
            if((averageD - bestD) > (actualD - averageD))
                writeNew(back, front, actualD);
        else if((bestD < averageD && bestD < prevD) ||
                (bestD > averageD && bestD > actualD)){
            if((actualD - averageD) >= (averageD - prevD))
                writeNew(back, (front-1)%len, prevD);
            else
                writeNew(back, front, actualD);
        }
        actualD -= vert.at(back%len).getD();
        prevD -= vert.at(back%len).getD();
        back = (back+1)%len;
        front = (front-1)%len;
        continue;
    }
    prevD = actualD;
    actualD += vert.at(temp).getD();
}
result.first = bestD;
result.second = bestV;
return result;
```