HydroHarmony

A water management system

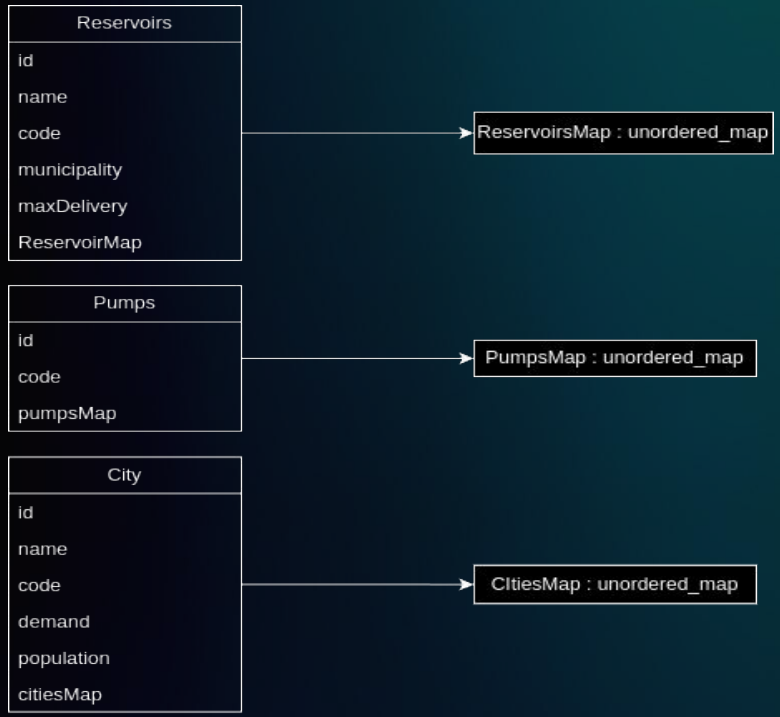
DA Project 1

- Up202007189 André David Aires de Freitas
- Up202206351 Bruno Coutinho Pereira
- Up202206349 Ricardo Alexandre Alves Ramos

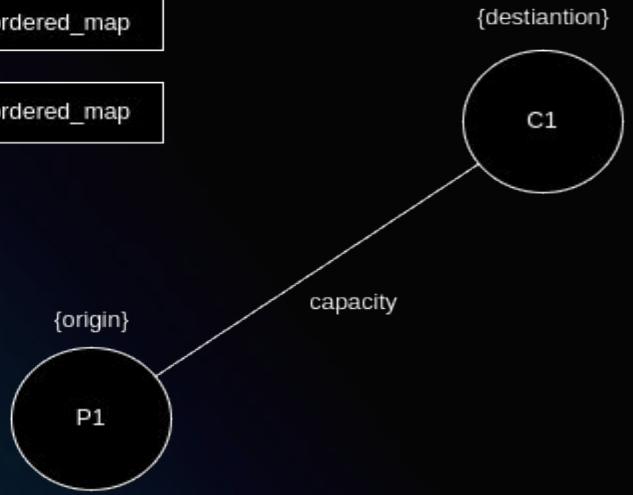
Classes


Reservoirs
id
name
code
municipality
maxDelivery
ReservoirMap
Pumps
id
code
pumpsMap
City
id
name
code
demand
population
citiesMap
Pipes
origin
destination
capacity
direction
pipesMultiMap
DataReader
readCities
readPipes
readReservoirs
readStations
Graph
vertexSet
sourceSet
sinkSet
superSet
superSink
DataManager
buildGraph
resetUsing
edmondsKarp
Vertex
code
adj
path
visited
dist
incoming
Edge
dest
orig
reverse
weight
flow

LEITURA DE DADOS



Grafo



Pipe to citie example

Criar o grafo

```
Graph DataManager::buildGraph(City::CitiesMap citiesMap, Pipe::PipesMultiMap pipesMultiMap, Pump::PumpsMap pumpsMap,
                              Reservoir::ReservoirsMap reservoirsMap) {

    Graph graph = Graph();

    for (const auto& reservoir : const pair<...> & : reservoirsMap)
        if (!graph.addSource( code: reservoir.second->getCode(), weight: reservoir.second->getMaxDelivery()))
            throw std::logic_error("error adding source vertex");

    for (const auto& pump : const pair<...> & : pumpsMap)
        if (!graph.addVertex( code: pump.second->getCode())) throw std::logic_error("error adding vertex");
    for (const auto& city : const pair<...> & : citiesMap) {...}
    for (const auto& pipe : const pair<...> & : pipesMultiMap.getRtPMAP()) {...}
    for (const auto& pipe : const pair<...> & : pipesMultiMap.getPtPMAP()) {

        if (pipe.second->getDirection()) {
            if (!graph.addEdge( src: pipe.first.first, dest: pipe.first.second, w: pipe.second->getCapacity()))
                throw std::logic_error("error adding RtP edge");
        } else {
            if (!graph.addBidirectionalEdge( src: pipe.first.first, dest: pipe.first.second, w: pipe.second->getCapacity()))
                throw std::logic_error("error adding RtP edge");
        }
    }
    for (const auto& pipe : const pair<...> & : pipesMultiMap.getPtCMAP()) {

        if (pipe.second->getDirection()) {
            if (!graph.addEdge( src: pipe.first.first, dest: pipe.first.second, w: pipe.second->getCapacity()))
                throw std::logic_error("error adding RtP edge");
        } else {
            if (!graph.addBidirectionalEdge( src: pipe.first.first, dest: pipe.first.second, w: pipe.second->getCapacity()))
                throw std::logic_error("error adding RtP edge");
        }
    }

    return graph;
}
```

Funcionalidade e algoritmos

01

Utility Menu

Menu built with Ncurses

02

Max Flow

Max flow built with Edmonds Karp algorithm

03

Maintenance and Reliability

You can describe the topic of the section here

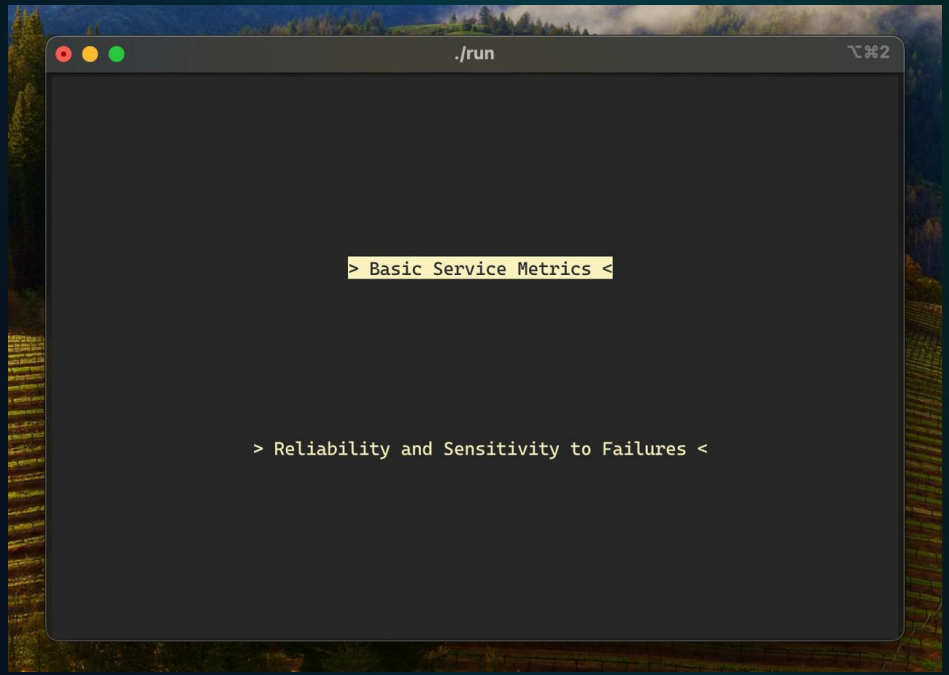
04

Balance of the flow

Tentative of implementation of flow balancing

Funcionalidades e algoritmos

01



Funcionalidades e algoritmos

Max Flow in the of the network

02

```
double DataManager::findMinResidualAlongPath(Vertex *src, Vertex *dest) {

    double f = INF;

    for (Vertex* vertex = dest; vertex != src;) {

        Edge* edge = vertex->getPath();

        if (edge->getDest() == vertex) {
            f = std::min(f, edge->getWeight() - edge->getFlow());
            vertex = edge->getOrig();
        } else {
            f = std::min(f, edge->getFlow());
            vertex = edge->getDest();
        }
    }

    return f;
}


void DataManager::augmentFlowAlongPath(Vertex *src, Vertex *dest, double f) {

    for (Vertex* vertex = dest; vertex != src; ) {
        Edge* edge = vertex->getPath();
        double flow = edge->getFlow();
        if (edge->getDest() == vertex) {
            edge->setFlow(flow + f);
            vertex = edge->getOrig();
        } else {
            edge->setFlow(flow - f);
            vertex = edge->getDest();
        }
    }
}
```

```
void DataManager::testAndVisit(std::queue<Vertex*> &q, Edge *edge, Vertex *w, double residual) {

    if (!w->isVisited() && residual > 0 && w->isUsing()) {
        w->setVisited(true);
        w->setPath(edge);
        q.push( x: w);
    }
}


bool DataManager::findAugmentingPath(Graph *graph, Vertex *src, Vertex *dest) {

    for(Vertex* vertex : graph->getVertexSet()) {
        vertex->setVisited(false);
    }

    src->setVisited(true);
    std::queue<Vertex *> q;
    q.push( x: src);

    while(! q.empty() && ! dest->isVisited()) {
        Vertex* vertex = q.front();
        q.pop();
        for (Edge* edge: vertex->getAdj()) {
            if (edge->isUsing())
                testAndVisit( &: q, edge, w: edge->getDest(), residual: edge->getWeight() - edge->getFlow());
        }
        for (Edge* edge: vertex->getIncoming()) {
            if (edge->isUsing())
                testAndVisit( &: q, edge, w: edge->getOrig(), residual: edge->getFlow());
        }
    }
    return dest->isVisited();
}
```

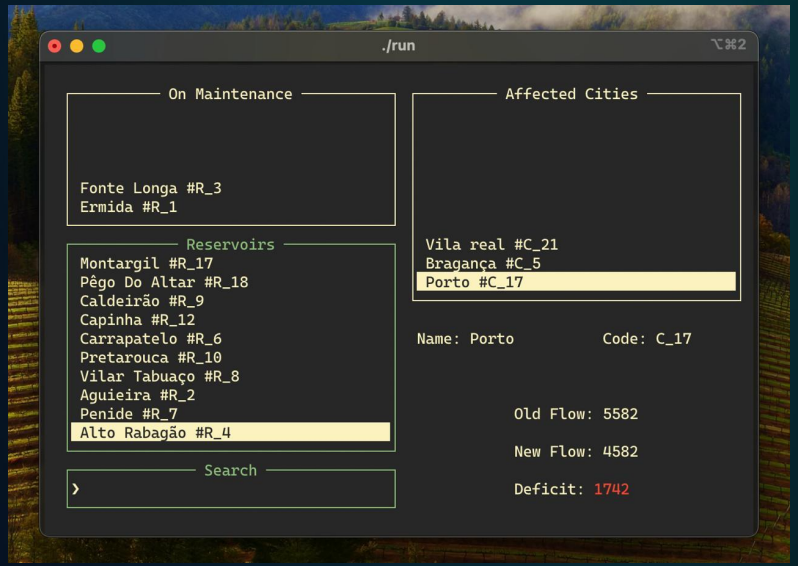
03

FUNDRAISING CHANNELS

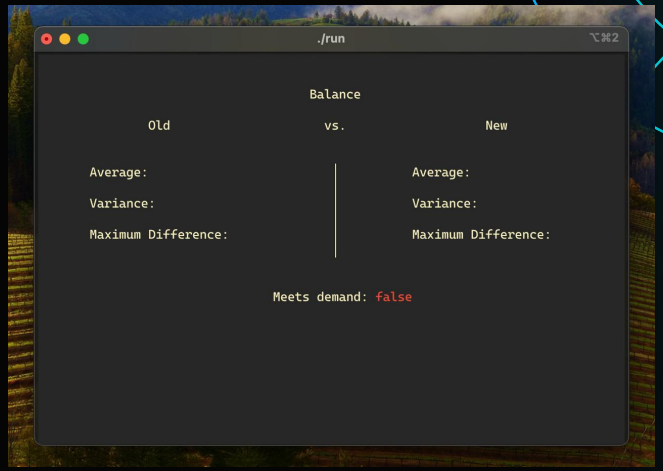
04

```
void DataManager::edmondsKarp(Graph *graph, const std::string& source, const std::string& target) {

    Vertex* src= nullptr;
    Vertex* dest= nullptr;

    if (source.empty()) src = graph->getSuperSource();
    else src = graph->findVertex( code: source);

    if (target.empty()) dest= graph->getSuperSink();
    else dest= graph->findVertex( code: target);


    if (src == nullptr || dest== nullptr || src == dest)
        throw std::logic_error("Invalid source and/or target vertex");

    for (Vertex* vertex : graph->getVertexSet()) {
        for (Edge* edge: vertex->getAdj()) {
            edge->setFlow(0);
        }
    }

    while( findAugmentingPath(graph, src, dest) ) {
        double f = findMinResidualAlongPath(src, dest);
        augmentFlowAlongPath(src, dest, f);
    }
}
```

Maintenance and Reliability



Balance of the network




```
void DataManager::balanceFlow(Graph* graph, std::vector<Edge*>& edgeSet) {
    std::set<Vertex*> processing;

    for (Vertex* vertex : graph->getVertexSet()) processing.insert( x: vertex);

    while (!processing.empty()) {
        Vertex* vertex = *processing.begin();
        processing.erase( position: processing.begin());
        processVertex(vertex, & processing, & edgeSet, graph);
        for (Edge* edge : edgeSet) edge->setUsing(false);
    }
}

void DataManager::removeFullEdges(std::vector<Edge*>& edgeSet) {

    auto it : iterator<...> = edgeSet.begin();
    while (it != edgeSet.end()) {
        if ((*it)->getFlow() == (*it)->getWeight()) it = edgeSet.erase( position: it);
        else it++;
    }
}

void DataManager::balanceGraph(Graph* graph, int maxFlow) {

    DataManager::resetUsing(graph);

    double usedFlow = 0;
    std::vector<Edge*> edgeSet = graph->getEdgeSet();

    //while (usedFlow < maxFlow - 10) {
    for (int i = 0; i < 1; i++)
    {
        addAvgToEdges(maxFlow, & usedFlow, & edgeSet);
        balanceFlow(graph, & edgeSet);
        removeFullEdges( & edgeSet);
    }
}
```