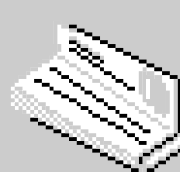
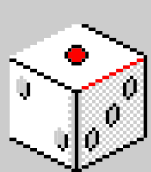
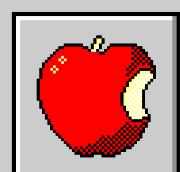


# ROUTE\_x86

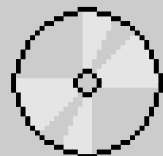
Routing Algorithm for Ocean Shipping and Urban Deliveries



Desenho de Algoritmos



04:20PM

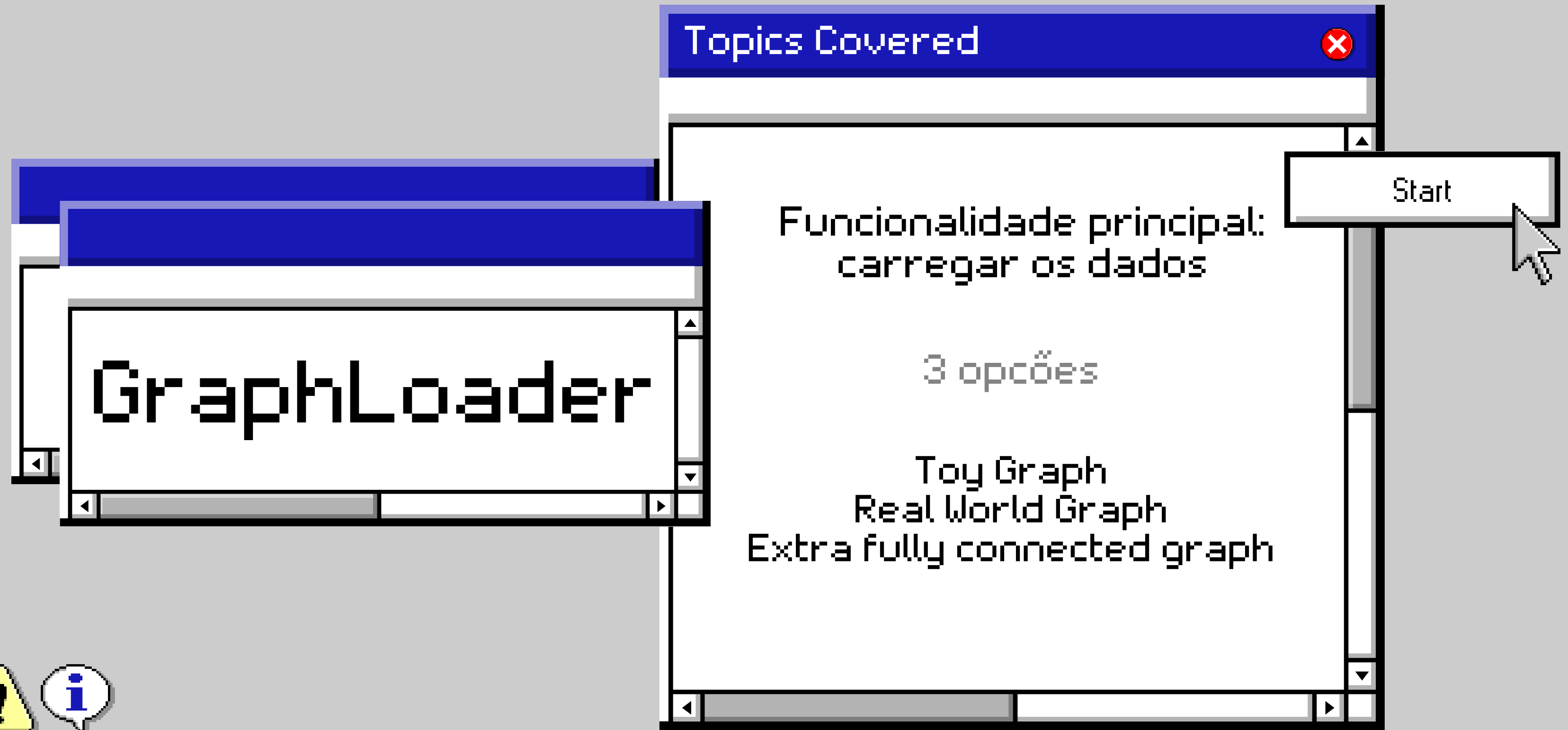


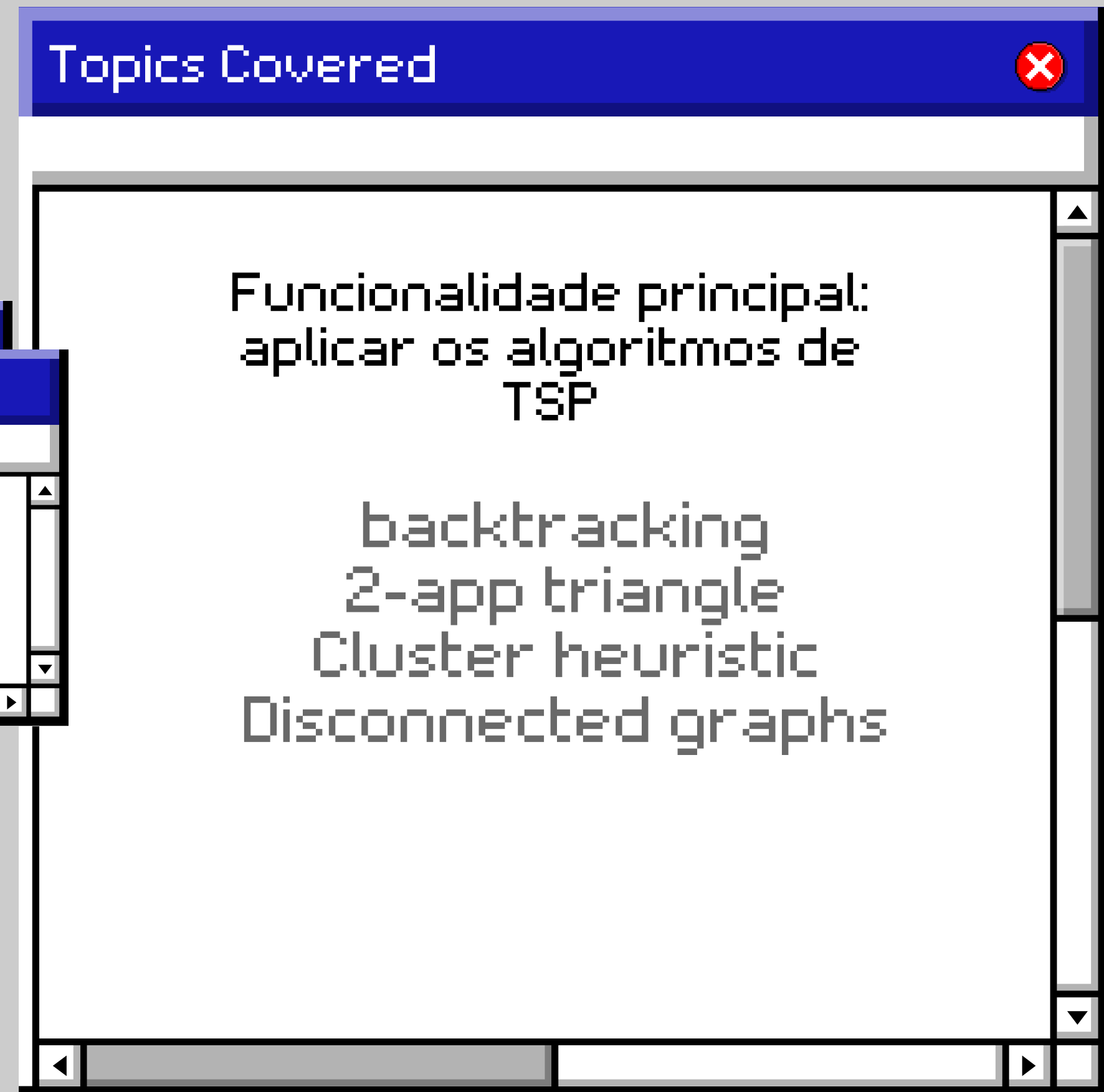
# Enumeração de Classes



[Back](#)

Classes	Atributos
UserInterface	GraphController, GraphLoader
GraphController	graph, graphAdj
GraphLoader	nenhum
Vertex	vertexId, adj, coordinates
Edge	weight, origin, destination
MutablePriorityQueue	nenhum





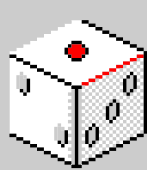
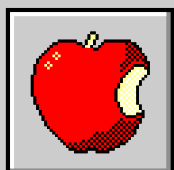
# Backtracking

```
Enter choice: 1
Enter filename: tourism
Graph loaded successfully.
1. Load Graph
2. Execute TSP Backtracking
3. Execute Triangular Approximation
4. Execute Cluster Approximation
5. Exit
Enter choice: 2
For the given graph the minimum cost for a hamiltonian cycle is 2600
With the following path: 0, 3, 2, 1, 4,
```

```
void GraphController::recursiveBacktracking(Vertex* curr_vertex, double distance,
const std::set<uint16_t>& visited, std::vector<uint16_t> path){
    std::set<uint16_t> curr_visited = visited;
    curr_visited.insert( x: curr_vertex->getId());

    path.push_back(curr_vertex->getId());

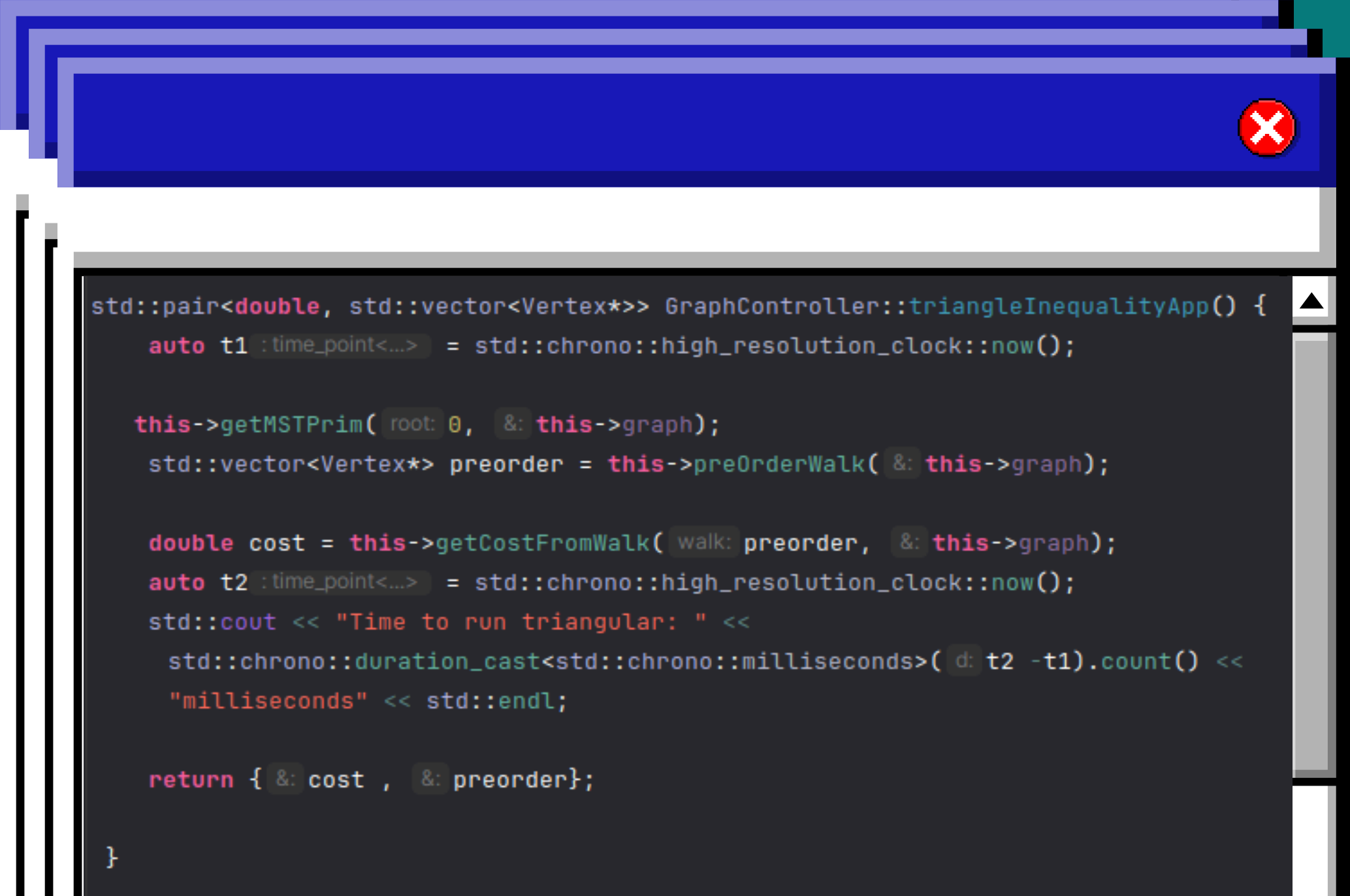
    for(Edge* adj: curr_vertex->getAdj()){
        double updatedDistance = distance + adj->getWeight();
        if(adj->getDestination()->getId() == 0 ){
            if(this->allVerticesVisited( visited: curr_visited) && this->min_distance > updatedDistance){
                this->min_distance = updatedDistance;
                this->minPath = path;
            }
            continue;
        }
        // this vertex is not endpoint
        if(!this->Bound( curr_idx: adj->getDestination()->getId(),
            pathSum: updatedDistance, visited)) continue;
        recursiveBacktracking( curr_vertex: adj->getDestination(),
            distance: updatedDistance, visited: curr_visited, path);
    }
}
```



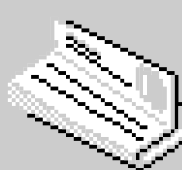
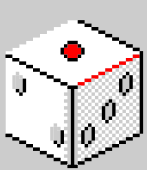
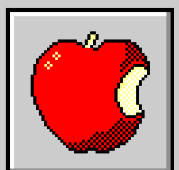
[Back to Agenda Page](#)

# TriangleInequality

## 2-approximation



```
std::pair<double, std::vector<Vertex*>> GraphController::triangleInequalityApp() {  
    auto t1 : time_point<...> = std::chrono::high_resolution_clock::now();  
  
    this->getMSTPrim( root: 0,  &: this->graph);  
    std::vector<Vertex*> preorder = this->preOrderWalk( &: this->graph);  
  
    double cost = this->getCostFromWalk( walk: preorder,  &: this->graph);  
    auto t2 : time_point<...> = std::chrono::high_resolution_clock::now();  
    std::cout << "Time to run triangular: " <<  
        std::chrono::duration_cast<std::chrono::milliseconds>( d: t2 -t1).count() <<  
        "milliseconds" << std::endl;  
  
    return { &: cost ,  &: preorder};  
}
```



[Back to Agenda Page](#)

# ClusterHeuristic

Este algoritmo consiste em dividir o grafo em clusters usando kmeans, e aplicar multi threading para resolver o tsp (nearest neighbour) em cada um dos cluster.

```
std::pair<double, std::vector<uint16_t>> GraphController::clusterHeuristic() {
    auto t1 :time_point<...> = std::chrono::high_resolution_clock::now();
    if(this->graph.begin()->second->getCoordinates().latitude == std::numeric_limits<double>::infinity()) return this->fullNN();

    // K-means clustering for geographic based clustering
    std::vector<Cluster> clusters;
    clusters = this->kmeansClustering();

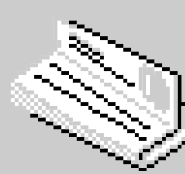
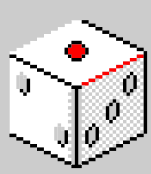
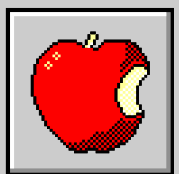
    Cluster rootCluster;

    // for each cluster get a tour and a walk cost
    double totalCost = 0;

    std::vector<std::thread> threads;
    for (Cluster& cluster : clusters) {
        threads.emplace_back([this, &cluster]() {
            // ...
        });
    }
    // Join threads to ensure all complete before moving on
    for (std::thread& t : threads) {
        if (t.joinable()) {
            t.join();
        }
    }
    // generate cost from intra-cluster and find cluster with 0
    for(Cluster& cluster: clusters){
        totalCost += cluster.tourCost;
        if(cluster.rootCluster) rootCluster = cluster;
    }

    std::pair<double, std::vector<uint16_t>> finalTour = this->tourNNInterClusters(&rootCluster, &clusters);
    finalTour.first += totalCost;

    auto t2 :time_point<...> = std::chrono::high_resolution_clock::now();
    std::cout << "Time to run triangular: " << std::chrono::duration_cast<std::chrono::milliseconds>(t2 - t1).count() << "milliseconds" <<
        std::endl;
    return finalTour;
}
```

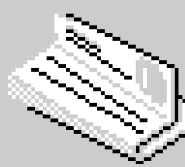
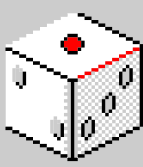
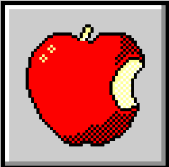


[Back to Agenda Page](#)

# Benchmarks



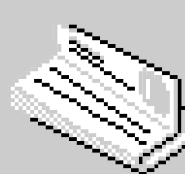
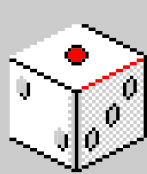
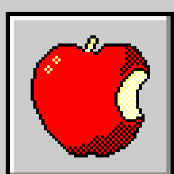
	Backtracking	Triangle Inequality	Cluster Heuristic (NN)	Factor of speed
Tourism	0ms [2600]	0ms [2600]	0ms [2600]	0x faster
Stadiums	3500ms[341]	0ms[391.4]	0ms[407.4]	0x faster
Graph1	INF	222ms[1.14e6]	29ms[1.25e6]	7.7x faster
Graph2	INF	2200ms[2.10e6]	332ms[2.50e6]	6.6x faster
Graph3	INF	6200ms[3.23e6]	675ms[3.45e06]	9.18x faster





# TSP disconnected

```
std::pair<double, std::vector<uint16_t>> GraphController::findTSPForDisconnectedGraph(Vertex* startVertex) {  
    std::set<uint16_t> nodes;  
    std::vector<uint16_t> tour;  
    double tourCost = 0;  
  
    for (const auto& v : const pair<...> & : graph) {  
        nodes.insert( x v.first);  
    }  
  
    uint16_t currNode = startVertex->getId();  
    while (!nodes.empty()) {  
        double minDistance = std::numeric_limits<double>::infinity();  
        uint16_t minVertex;  
  
        for (uint16_t adjNode : nodes) {  
            if (adjNode == currNode) continue;  
  
            double distance = this->graphAdj[currNode][adjNode];  
            if (distance < minDistance) {  
                minVertex = adjNode;  
                minDistance = distance;  
            }  
        }  
  
        nodes.erase( x currNode);  
        tour.push_back(currNode);  
        currNode = minVertex;  
        if (minDistance == std::numeric_limits<double>::infinity()) break;  
        tourCost += minDistance;  
    }  
  
    if (this->graphAdj[tour.front()][tour.back()] != std::numeric_limits<double>::infinity()) {  
        tourCost += this->graphAdj[tour.front()][tour.back()];  
    }  
  
    return { & tourCost, & tour};  
}
```

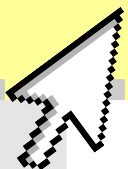


[VOLTAR À PÁGINA INICIAL](#)



LIVE DEMO

[Back to Agenda Page](#)





# FIM

- Rodrigo Ferreira Alves - up202207478
- Gonçalo de Abreu Matias - up202108703
- David Tavares Simões - up202210329

