struct secondValueComp {

constexpr bool operator()(

std::pair<int, int> const& a,

std::pair<int, int> const& b)

const noexcept

{

return a.second > b.second;

}

};

std::pair<std::map<int, int>, int> Graph::djikstra(int s, int t)

{

if (this->nout.find(s) != this->nout.end() && this->nout.find(t) != this->nout.end()) // check if the two given vertices exist

{

std::priority\_queue<std::pair<int, int>, std::vector<std::pair<int, int>>, secondValueComp> q; // create a min heap priority queue,

// sorted by the second element of the pair, which is the priority

std::map<int, int> prev; // a map that associates, to each accessible vertex, the cost of the minimum cost walk from s to it

std::map<int, int> dist; // a map that maps each accessible vertex to its predecessor on a path from s to it

int x;

q.push(std::make\_pair(s, 0));

dist[s] = 0;

bool found = false;

while (!q.empty() && !found)

{

x = q.top().first; // gets the element with minimum value of priority

q.pop(); // then dequeues it

for (int y : this->parse\_nout(x))

{

if (dist.find(y) == dist.end() || dist[x] + this->get\_cost(x, y) < dist[y])

{

dist[y] = dist[x] + this->get\_cost(x, y);

q.push(std::make\_pair(y, dist[y]));

prev[y] = x;

}

}

if (x == t)

{

found = true;

}

}

if (found == true) // check if there is a path between the two vertices

return std::make\_pair(prev, dist[t]); // returns prev and the the cost of the minimum cost walk from s to t

else

throw "No path found";

}

else

{

throw "Invalid vertices";

}

}