Dokumentation Algorithmus

Datenstruktur:

```
// List of visited nodes
std::vector<std::string> visited;
// Heap
std::priority_queue<HeapNode, std::vector<HeapNode>, CompareHeapNode> heap;
// Map for storing last connection for every node
std::map<std::string,Connection*> lastcons;
```

Wir benutzen für unseren Algorithmus 3 verschiedenen Datenstrukturen.

Vektor: Ein Vektor von Strings wird zur Speicherung von bereits besuchten Knotenpunkte verwendet.

Priority_Queue: Der Heap speichert alle verfügbaren Verbindung von den bereits besuchten Stationen geordnet nach dem Gewicht der Verbindungsstrecke.

Map: Zur Speicherung der letzten Verbindungen zu jedem Knotenpunkt um nachvollziehen zu können, woher man kam.

Der Dijkstra Algorithmus:

Wir verwenden den Dijkstra und gehen Knoten für Knoten in sämtliche Richtungen durch unsere vorgegebenen Verbindungen. Bei jedem Schritt wird der Knoten mit dem kürzesten Gewicht zu unserem Startpunkt gewählt von welchen aus wieder weitergegangen wird. Dieser Vorgang wird solange wiederholt, bis das Ziel erreicht wurde beziehungsweise bis festgestellt wurde, dass keine Route zwischen den Stationen verfügbar ist.

Wenn das gewünschte Ziel als Knotenpunkt gefunden wird, wird die kürzeste Route ausgegeben, welche im Heap gespeichert wird. Falls keine Route zwischen den Punkten verfügbar ist, wird eine Meldung ausgegeben und der Programmverlauf wird beendet.

Gesucht wird vom eingegebenen Zielknoten zum Startknoten.

```cpp
const void ConnectionPlan::getShortestPath(std::string start_station,
std::string end_station) {
    // List of visited nodes
    std::vector<std::string> visited;
    // Heap
    std::priority_queue<HeapNode, std::vector<HeapNode>, CompareHeapNode>
heap;
    // Map for storing last connection for every node
    std::map<std::string,Connection*> lastcons;

    // Init variables
    int mintime=0;
    // Start search from destination
    std::string currentnode = end_station;
    Connection* lastconnection = nullptr;

    do {
        // Set current node visited
        visited.push_back(currentnode);
        // Iterate through each neighbour of current node
        for(unsigned int i = 0; i < stations[currentnode].size(); i++){
            // Push neighbour to heap if not already visited
            if (std::find(visited.begin(), visited.end(),
stations[currentnode].at(i)->getDestination()) == visited.end()) {
                // Check if line changes
                if(lastconnection == nullptr || lastconnection->getLine()
== stations[currentnode].at(i)->getLine()) {
                    heap.push(HeapNode(stations[currentnode].at(i),
                                       mintime +
stations[currentnode].at(i)->getTraveltime(),
                                       lastconnection));
                }else{
                    // If line changes, add time for changing
                    heap.push(HeapNode(stations[currentnode].at(i),
                                       mintime +
stations[currentnode].at(i)->getTraveltime() + changingtime,
                                       lastconnection));
                }
            }
        }
        do {
            if(heap.empty()){
                // Heap empty, problem occured
                std::cout << "No path could be found! :(" << std::endl;
                return;
            }
            // Pop node with least weight from heap and update currentnode
and mintime
            HeapNode node = heap.top();
            currentnode = node.connection->getDestination();
            mintime = node.weight;
            lastconnection = node.connection;
            heap.pop();
            lastcons.insert({currentnode, node.lastconnection});
            // Ignore nodes which are already visited, go to next one
        }while (std::find(visited.begin(), visited.end(), currentnode) !=
visited.end());
        // Do as long as current node is not start station
    } while (currentnode!=start_station);

    // Print found path
    printPath(lastcons, start_station, end_station);
}
```

Kommentiert [AD1]: Es wird der Vektor, Heap und die Map initialisiert die ich oben in der Dokumentation erläutert habe.

Kommentiert [AD2]: Der Integer Minimalzeit speichert stets die kürzeste Strecke zu unseren Startknoten.

Kommentiert [AD3]: Ich gehe durch alle Nachbarknoten des aktuellen Knotens und wenn sich Gewichte in den Verbindungsstrecken ändern, speichere ich diese ab.

Kommentiert [AD4]: Wenn der Heap leer ist, gebe aus, dass keine Route gefunden wurde, ansonsten aktualisiere meinen aktuellen Knoten und speichere meine Minimalzeit entsprechend ab. Gehe weiter durch die Knoten bis unser Ziel erreicht wird und gib dann den kürzesten Weg aus.

Testfälle:

Funktionierende Testbeispiele:

```
Where do you want to start your journey?Leopoldau
Leopoldau
What is your destination?Huetteldorf
Huetteldorf

Using U1

Starting from Leopoldau
in 2 Minutes to Grossfeldsiedlung
in 1 Minutes to Aderklaaer Strasse
in 1 Minutes to Rennbahnweg
in 2 Minutes to Kagraner Platz
in 2 Minutes to Kagran
in 1 Minutes to Alte Donau
in 2 Minutes to Kaisermuehlen-VIC
in 1 Minutes to Donauinsel
in 2 Minutes to Vorgartenstrasse
in 1 Minutes to Praterstern
in 1 Minutes to Nestroyplatz
in 1 Minutes to Schwedenplatz

In Schwedenplatz change to U4 (5 Minutes)

in 2 Minutes to Landstrasse
in 1 Minutes to Stadtpark
in 2 Minutes to Karlsplatz
in 2 Minutes to Kettenbrueckengasse
in 1 Minutes to Pilgramgasse
in 2 Minutes to Margaretenguertel
in 2 Minutes to Laengenfeldgasse
in 1 Minutes to Meidling Hauptstrasse
in 1 Minutes to Schoenbrunn
in 2 Minutes to Hietzing
in 1 Minutes to Braunschweiggasse
in 2 Minutes to Unter Sank Veit
in 1 Minutes to Ober Sankt Veit
in 2 Minutes to Huetteldorf

Total time needed: 44 Minutes
```

```
Where do you want to start your journey?Heiligenstadt
Heiligenstadt
What is your destination?Altes Landgut
Altes Landgut

Using U4

Starting from Heiligenstadt
in 2 Minutes to Spittelau
in 1 Minutes to Friedensbruecke
in 1 Minutes to Rossauerlaende
in 2 Minutes to Schottenring
in 1 Minutes to Schwedenplatz

In Schwedenplatz change to U1 (5 Minutes)

in 1 Minutes to Stephansplatz
in 2 Minutes to Karlsplatz
in 2 Minutes to Taubstummengasse
in 1 Minutes to Suedtirolerplatz
in 2 Minutes to Keplerplatz
in 1 Minutes to Reumannplatz

In Reumannplatz change to 67 (5 Minutes)

in 2 Minutes to Troststrasse
in 1 Minutes to Schleiergasse
in 2 Minutes to Altes Landgut

Total time needed: 31 Minutes
```

Wenn keine Route aufgrund Fehlern gefunden werden konnte:

Wenn nicht auf Case-Sensitive geachtet wird:

```
Press s for searching a path, c for setting the time for changing lines or any other character to exit the program!
s
s
Where do you want to start your journey?handelskai
handelskai
What is your destination?heiligenstadt
heiligenstadt

No path could be found! :(
```

Wenn eine Station als Start- und Zielhaltestelle angegeben wird:

```
Press s for searching a path, c for setting the time for changing lines or any other character to exit the program!
s
s
Where do you want to start your journey?Handelskai
Handelskai
What is your destination?Handelskai
Handelskai

No path could be found! :(
```

Testfall mit zirkularer U-Bahnverbindung:

```
Press s for searching a path, c for setting the time for changing lines or any other character to exit the program!
s
s
Where do you want to start your journey?Jaegerstrasse
Jaegerstrasse
What is your destination?Tscherttegasse
Tscherttegasse

Using U6

Starting from Jaegerstrasse
in 2 Minutes to Dresdner Strasse
in 1 Minutes to Handelskai
in 2 Minutes to Neue Donau
in 2 Minutes to Floridsdorf
in 1 Minutes to Siebenhirten
in 2 Minutes to Perfektastrasse
in 1 Minutes to Erlaaer Strasse
in 2 Minutes to Alterlaa
in 2 Minutes to Am Schoepfwerk
in 1 Minutes to Tscherttegasse

Total time needed: 16 Minutes
```