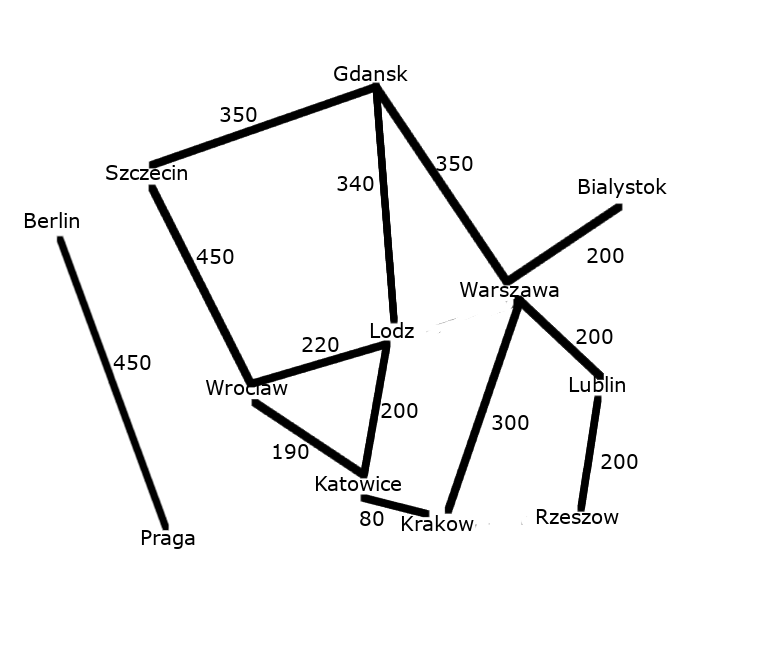
The Map, “Story”

The Map is library that allows you to create a map of cities, for example map of cities in Poland. The user will be able to add new Cities to the map and link some cities, specifying the distance between them. Map will allow you to check how to get from one city to another. For example, if you check how to get from Krakow to Warsaw, you may see that the best route is go to Katowice, then to Łodź and then to Gdansk.

The Map library is designed to use when map is modified rarely, but you very often check how to get from one city to another. After modification of Map (adding a city or changing a distance between two cities) user, to get the correct best path from one city to another, will have to explicitly call function “recalculate”. Function recalculate finds the best path from every city in the map to every other city in the map (if any connection between two cities is possible). After recalculation each city remembers how to get to every other city (if any path is possible, cities can be in two disconnected graphs. On the beginning function “recalculate” force each city in the map to forget all it's connections.Let’s consider such map (map1).

If there are at least two convenient paths connecting two cities, after recalculation each of this two cities will know the best and second the best path to other city. So in the case of map1 after recalculation city “Gdansk” will know that the best city to Krakow is through Lodz and Katowice, and second the best path to Krakow is through Warszawa. City “Krakow” will also know two routes to Gdansk (same as from Gdansk to Krakow reversed). If there is only one path connecting two cities then each of those two cities will know this path and will know that this is the only convenient path. So in case of map1 city “Rzeszow”, after recalculation, will know that the only convenient path to Bialystok is through Lublin and Warszawa. City “Bialystok will know know that the only convenient path to Rzeszow is through Warszawa and Lublin.

If there is no convenient way from two cities each of those two cities will know that there in so way to get to other city.

After recalculation user may call function “how\_to\_get”, passing names of two cities as arguments, to check how to get from one city to another.

bool how\_to\_get(std::string from, std::string destination, std::ostream& stream)

If any of two passed cities is not on the map fuction ”how\_to\_get” will print this information.

If there is no way to get from one city to another fuction ”how\_to\_get” will print this information.

If there is only one convenient path from one city to another, fuction ”how\_to\_get” will print this path and print that this is the only possible path.

If there are at least two possible paths connecting those two cities fuction ”how\_to\_get” will print those path as best\_path and second\_best\_path.

User is building a map by calling a function “link” ( bool link(std::string city1\_name, std::string city2\_name, unsigned int distance) ). If any of passed cites is not yet on the map, function first creates new city on new map with given name. If cities were unlink before (one wasn’t a neighbour of other) function links them (adds second city to list of neighbours of city1 and vice versa). If cities were linked before function reset distance between them, so user can use this function to change distance between cities. Function returns true if new link was created, false otherwise.

User can unlink two cities using function “unlink” ( bool unlink(std::string city1\_name,  city2\_name); ). Function returns false and do nothing if one of the cities wasn’t on the map, or if cities wasn’t linked. If given cities were linked, function unlinks them and returns true. Unlinking means removing first city of neighbours list of second city and removing second city from neighbours list of first city.

The Map “Case study”

How recalculation is done:

Calculation is done by “sending paths” from on city to another. Path is represented by object of class Path. Path contains array of id’s of cities which are in the path and distances between them. Let’s consider map1. Let’s give some cities an id: Bialsytok is nr 1, Warszawa, Lublin, Krakow, and Gdansk are in order 2,3,4,5. Firstly city nr1 (Bialystok) send path which contains id 1 and distance 200 to it’s only neighbour, city nr 2 (Warszawa). City nr 2 then checks on the connection table. If it is necessary it adds path to it’s routing table. Then city nr 2 sends to all it's neighbour the copy of the path with addition of neighbour id and distance between them. So it’s send path with id 1, distance 200, id 4, distance 300 to Kraków and so on… Each city that receives the path check it’s own connection table and if necessary adds path to the connection table. Then it send the copy of the path with addition of neighbour id and distance to **each neighbour which isn’t already in the path. This way no loops are created.**

After this each city know the best path (or two path) to Bialystok. Then operation is repeated, but now starting from city nr2, then from city nr3… Because of many operations recalculation of maps is time-consuming.

Important note: Paths are send as pointers to Path objects. The city that receive the path is responsible to delete this path, if necessary.

Program structure:

Obraz zawierający tekst

Opis wygenerowany automatycznie

A member of class Map object is single linked list of keys type std::string (city names) and value types City.