cs380su21-meta.sg

Grade

Reviewed on Tuesday, 15 June 2021, 12:25 AM by Automatic grade
grade: 100.00 / 100.00

Assessment report [-]

[-]**Summary of tests**

```
+-----------------------------+
| 10 tests run/10 tests passed |
+-----------------------------+
```

Submitted on Tuesday, 15 June 2021, 12:25 AM (Download)

functions.cpp

```cpp
/*!*****************************************************************************
\file functions.cpp
\author Vadim Surov, Goh Wei Zhe
\par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
\par Course: CS380
\par Section: B
\par Programming Assignment 5
\date 06-13-2021
\brief
This file has declarations and definitions that are required for submission
*******************************************************************************/

#include "functions.h"

namespace AI
{


} // end namespace
```

functions.h

```cpp
/*!*****************************************************************************
\file functions.h
\author Vadim Surov, Goh Wei Zhe
\par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
\par Course: CS380
\par Section: B
\par Programming Assignment 5
\date 06-13-2021
\brief
This file has declarations and definitions that are required for submission
*******************************************************************************/

#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <iostream>
#include <vector>
#include <array>
#include <climits>
#include <algorithm>

#include "data.h"

#define UNUSED(x) (void)x;

namespace AI
{
    const int null = -1;
    const int inf = INT_MAX;

    // An implementation of the Bellman-Ford algorithm.
    // The algorithm finds the shortest path between a
    // starting node and all other nodes in the graph.
    // The algorithm also detects negative cycles.
    template<int SIZE = 0>
     class BellmanFord
    {
        int* matrix; // the cost adjacency matrix
        int* distance;
        int* predecessor;

    public:

        /*!*************************************************************************
        \brief
        Constructor for class BellmanFord

        \param matrix
        A matrix array of integers

        \return
        None.
        ***************************************************************************/
        BellmanFord(int* matrix = nullptr)
            : matrix{ matrix }, distance{ nullptr }, predecessor{ nullptr }
        {
           this->distance = new int[SIZE];
           this->predecessor = new int[SIZE];
         }

        /*!*************************************************************************
        \brief
        Destructor for class BellmanFord

        \param
        None.

        \return
        None.
        ***************************************************************************/
        ~BellmanFord()
        {
           delete[] distance;
           delete[] predecessor;
        }

        /*!*************************************************************************
        \brief
        Function to run Bellman-Ford's Algorithm

        \param starting
        The starting position of the array

        \return
        Returns tue if cycles are found, else return false if negative cycles
        ***************************************************************************/
        bool run(int starting = 0)
        {
            for (int i = 0; i < SIZE; ++i)
            {
                //set distance all to infinite
                //set predecessor all to null
                this->distance[i] = inf;
                this->predecessor[i] = null;

                //set starting node distance to 0
                if(starting == i)
                    this->distance[starting] = 0;

            }

            // For each node, apply relaxation for all the edges
            for (int k = 0; k < SIZE-1; k++)
            {
                //number of relaxation
                int counter = 0;

                for (int i = 0; i < SIZE; i++)
```

```
109             {
110                 for (int j = 0; j < SIZE; j++)
111                 {
112                     if ((i != j) && (this->distance[i] != inf) &&
113                         (this->matrix[(i*SIZE) + j] != inf))
114                     {
115                         int new_distance =
116                             this->distance[i] + this->matrix[(i*SIZE) + j];
117 
118                         if (new_distance < this->distance[j])
119                         {
120                             //Relaxation
121                             this->distance[j] = new_distance;
122                             this->predecessor[j] = i;
123                             counter++;
124                         }
125                     }
126                 }
127             }
128 
129             // Stop when no more relaxation
130             // There is no negative cycles
131             if (counter == 0)
132                 return true;
133         }
134 
135         // Run algorithm a second time to detect which nodes are part
136         // of a negative cycle. A negative cycle has occurred if we
137         // can find a better path beyond the optimal solution.
138         for (int i = 0; i < SIZE; i++)
139         {
140             for (int j = 0; j < SIZE; j++)
141             {
142                 if ((i != j) && (this->distance[i] != inf) &&
143                     (this->matrix[(i*SIZE)+ j] != inf)
144                     && ((this->distance[i] + this->matrix[(i*SIZE) + j])
145                         < this->distance[j]))
146                     return false;
147             }
148         }
149 
150         // There is no negative cycles
151         return true;
152     }
153 
154     /*!*************************************************************************
155     \brief
156     Function to reconstruct the shortest path from starting point to target
157 
158     \param target
159     Target to get the path from
160 
161     \return
162     Returns an array of int values from starting point to target.
163     *************************************************************************/
164     std::vector<int> getPath(int target)
165     {
166         std::vector<int> path{};
167 
168         for (int i = target; predecessor[i] != null; i = predecessor[i])
169             path.push_back(i);
170 
171         std::reverse(path.begin(), path.end());
172 
173         return path;
174     }
175 
176     /*!*************************************************************************
177     \brief
178     Function to create a route (step-by-step description) of the shortest
179     path from start to end with cost
180 
181     \param target
182     Target to get the path from
183 
184     \return
185     Returns a vector of int array values from starting point to target.
186     *************************************************************************/
187     std::vector<std::array<int, 3>> getRoute(int target)
188     {
189         std::vector<std::array<int, 3>> route{};
190 
191         for (int i = target; predecessor[i] != null; i = predecessor[i])
192             route.push_back({ this->predecessor[i], i, this->distance[i] });
193 
194         for (unsigned j = 0; j < route.size() - 1; ++j)
195             route[j][2] -= route[j + 1][2];
196 
197         std::reverse(route.begin(), route.end());
198 
199         return route;
200     }
201 
202     /*!*************************************************************************
203     \brief
204     Serialization. An overloading insertion operator function that takes
205     and return a stream object.
206 
207     \param os
208     Output stream to perform output.
209 
210     \param rhs
211     Right hand side object.
212 
213     \return
214     Returns the output through ostream.
215     *************************************************************************/
216     friend std::ostream& operator<<(std::ostream& os,const BellmanFord& rhs)
```

```
217     {
218         Print(os, &rhs);
219         return os;
220     }
221
222     /*!*************************************************************************
223     \brief
224     Function to print output.
225
226     \param os
227     Output stream to perform output.
228
229     \param rhs
230     Right hand side object.
231
232     \return
233     None.
234     *************************************************************************/
235     static void Print(std::ostream& os, const BellmanFord* rhs)
236     {
237         os << "[";
238
239         for (int i = 0; i < SIZE; i++)
240         {
241             if ((rhs->distance[i] == inf) && (i != SIZE - 1))
242                 os << "inf,";
243             else if ((rhs->distance[i] == inf) && (i == SIZE - 1))
244                 os << "inf";
245             else if (i != SIZE-1)
246                 os << rhs->distance[i] << ",";
247             else
248                 os << rhs->distance[i];
249         }
250
251         os << "] [";
252
253         for (int i = 0; i < SIZE; i++)
254         {
255             if ((rhs->predecessor[i] == null) && (i != SIZE - 1))
256                 os << "null,";
257             else if ((rhs->predecessor[i] == null) && (i == SIZE - 1))
258                 os << "null";
259             else if (i != SIZE-1)
260                 os << rhs->predecessor[i] << ",";
261             else
262                 os << rhs->predecessor[i];
263         }
264
265         os << "]";
266     }
267 };
268
269 } // end namespace
270
271 #endif
```

VPL

◄ Showcase: Pickup and delivery simulation

Jump to...

Showcase: Traveling Salesman Problem ►