

cs280s21-b.sg

[Dashboard](#) / [My courses](#) / [cs280s21-b.sg](#) / [General](#) / [Assignment 4: Graphs](#)

- [Description](#)
- [Submission](#)
- [Edit](#)
- Submission view

Grade

Reviewed on Thursday, April 1, 2021, 6:40 PM by Automatic grade
grade: 91.67 / 100.00

- Assessment report**[\[-\]](#)
- [\[+\]](#) **Test 12: TestBig(99, 2)**
- [\[+\]](#) **Summary of tests**

Submitted on Thursday, April 1, 2021, 6:39 PM ([Download](#))
ALGraph.h

```

1  ▾ /*****
2  ▾  */
3  \file:      ALGraph.h
4  \author:    Goh Wei Zhe, weizhe.goh, 44000119
5  \par email: weizhe.goh\@digipen.edu
6  \date:      March 28, 2021
7  \brief      This file contains the declarations needed to construct a graph and
8              implementing Dijkstra's algorithm.
9
10 Copyright (C) 2021 DigiPen Institute of Technology.
11 Reproduction or disclosure of this file or its contents without the
12 prior written consent of DigiPen Institute of Technology is prohibited.
13 */
14 ▾ /*****
15
16 //-----
17 #ifndef ALGRAPH_H
18 #define ALGRAPH_H
19 //-----
20 #include <vector>
21 #include <algorithm>
22 #include <queue>
23 #include <list>
24 #include <iostream>
25
26 struct DijkstraInfo
27 ▾ {
28     unsigned cost;
29     std::vector<unsigned> path;
30 };
31
32 struct AdjacencyInfo
33 ▾ {
34     unsigned id;
35     unsigned weight;
36 };
37
38 typedef std::vector<std::vector<AdjacencyInfo> > ALIST;
39
40 class ALGraph
41 ▾ {
42     public:
43         ALGraph(unsigned size);
44         ~ALGraph(void);
45         void AddDEdge(unsigned source, unsigned destination, unsigned weight);
46         void AddUEdge(unsigned node1, unsigned node2, unsigned weight);
47
48         std::vector<DijkstraInfo> Dijkstra(unsigned start_node) const;
49         ALIST GetAList(void) const;
50
51     private:
52
53         // An EXAMPLE of some other classes you may want to create and
54         // implement in ALGraph.cpp
55         class GNode;
56         class GEdge;
57
58         struct AdjInfo
59 ▾ {
60             // GNode *node;
61             unsigned id;
62             unsigned weight;
63
64             //unsigned cost;
65             //AdjInfo();
66             bool operator<(const AdjInfo& rhs) const;
67             bool operator>(const AdjInfo& rhs) const;
68         };
69
70         // Other private fields and methods
71         struct NodeInfo
72 ▾ {
73             unsigned id;
74             unsigned cost;
75             unsigned prev;
76
77             bool operator < (const NodeInfo& rhs) const;
78             bool operator == (const unsigned& value) const;
79         };
80
81         unsigned size_;
82         std::vector<std::vector<AdjInfo>> graph;
83
84         unsigned extract_min(std::vector<AdjacencyInfo>& pq) const;
85     };
86
87 #endif
88

```

ALGraph.cpp

```

1  /*****
2  /*!
3  \file:      ALGraph.cpp
4  \author:    Goh Wei Zhe, weizhe.goh, 44000119
5  \par email: weizhe.goh@digipen.edu
6  \date:      March 28, 2021
7  \brief      This file contains the definitions needed to construct a graph and
8               implementing Dijkstra's algorithm.
9
10 Copyright (C) 2021 DigiPen Institute of Technology.
11 Reproduction or disclosure of this file or its contents without the
12 prior written consent of DigiPen Institute of Technology is prohibited.
13 */
14 /*****/
15 #include "ALGraph.h"
16
17 const unsigned INFINITY_ = static_cast<unsigned>(-1);
18
19 /*****/
20 /*!
21 \fn      ALGraph::ALGraph(unsigned size)
22
23 \brief   Constructor for ALGraph
24 */
25 /*****/
26 ALGraph::ALGraph(unsigned size):size_{size}
27 {
28     //fill the vector of graph with vector of AdjInfo
29     for(unsigned i = 0; i < size; ++i)
30     {
31         graph.push_back(std::vector<AdjInfo>());
32     }
33 }
34
35 /*****/
36 /*!
37 \fn      ALGraph::~ALGraph(void)
38
39 \brief   Destructor for ALGraph
40 */
41 /*****/
42 ALGraph::~ALGraph(void){}
43
44 /*****/
45 /*!
46 \fn      void ALGraph::AddDEdge(unsigned source, unsigned destination,
47                                   unsigned weight)
48
49 \brief   Adds directed edge between two nodes.
50
51 \param   source - The starting node
52
53 \param   destination - The destination node
54
55 \param   weight - Cost to get from source to destination
56
57 */
58 /*****/
59 void ALGraph::AddDEdge(unsigned source, unsigned destination, unsigned weight)
60 {
61     auto i = source - 1;
62     graph[i].push_back(AdjInfo{destination, weight});
63
64     auto sortLambda = [](AdjInfo& LHS, AdjInfo& RHS)
65     {
66         return (LHS.weight == RHS.weight)
67             ? (LHS.id < RHS.id) : (LHS.weight < RHS.weight);
68     };
69
70     std::sort(graph[i].begin(), graph[i].end(), sortLambda);
71 }
72
73 /*****/
74 /*!
75 \fn      void ALGraph::AddUEdge(unsigned node1, unsigned node2, unsigned weight)
76
77 \brief   Adds undirected nodes between two nodes
78
79 \param   node1 - first node
80
81 \param   node2 - second node
82
83 \param   weight - cost to get from node1 to node 2 and node2 to node 1
84
85 */
86 /*****/
87 void ALGraph::AddUEdge(unsigned node1, unsigned node2, unsigned weight)
88 {
89     AddDEdge(node1, node2, weight);
90     AddDEdge(node2, node1, weight);
91 }
92
93 /*****/
94 /*!
95 \fn      std::vector<DijkstraInfo> ALGraph::Dijkstra(unsigned start_node) const
96
97 \brief   Performs Dijkstra's algorithm on the graph to find shortest path from a
98           starting node to every possible node.
99
100 \param   start_node - the node that we are starting at.
101
102 \return  Returns the result of Dijkstra algorithm that constaints a vector of
103           DijkstraInfo
104 */
105 /*****/
106 std::vector<DijkstraInfo> ALGraph::Dijkstra(unsigned start_node) const
107 {

```

```

108     std::vector<AdjacencyInfo> pq;
109     std::vector<DijkstraInfo> dijkstra;
110
111     for(unsigned i = 0; i < size_ ; ++i)
112     {
113         DijkstraInfo di;
114
115         if(i + 1 == start_node)
116         {
117             di.cost = 0;
118             di.path.push_back(start_node);
119         }
120         else
121             di.cost = INFINITY_;
122
123         dijkstra.push_back(di);
124         pq.push_back(AdjacencyInfo{i+1, di.cost});
125     }
126
127     while(!pq.empty())
128     {
129         //Returns index of min cost node
130         unsigned u = extract_min(pq);
131
132         //if extracted cost is infinite, it is not updated and is not connected
133         //to any nodes
134
135         if(dijkstra[u-1].cost == INFINITY_)
136             break;
137
138         //for each neighbour v of u
139         for(unsigned v = 0; v < graph[u - 1].size(); ++v)
140         {
141             //if dist[u] + cost(u,v) < dist[v]
142             if(dijkstra[u - 1].cost + graph[u-1][v].weight <
143                 dijkstra[graph[u-1][v].id-1].cost)
144             {
145                 //update cost
146                 dijkstra[graph[u-1][v].id-1].cost =
147                     dijkstra[u-1].cost + graph[u-1][v].weight;
148
149                 //update previous path
150                 dijkstra[graph[u-1][v].id-1].path = dijkstra[u-1].path;
151                 dijkstra[graph[u-1][v].id-1].path.push_back(graph[u-1][v].id);
152
153                 //Update info in priority_queue
154                 for(unsigned i = 0; i < pq.size(); ++i)
155                 {
156                     if(pq[i].id == graph[u-1][v].id)
157                     {
158                         //update cost
159                         pq[i].weight = dijkstra[graph[u-1][v].id-1].cost;
160                     }
161                 }
162             }
163         }
164
165         //Remove minimum cost node from priority queue
166         for(unsigned i = 0; i < pq.size(); ++i)
167         {
168             if(pq[i].id == u)
169                 pq.erase(pq.begin() + i);
170         }
171     }
172
173     return dijkstra;
174 }
175
176 /*****
177  *!
178  \fn      ALIST ALGraph::GetAList(void) const
179
180  \brief   Gets adjacency matrix for the graph
181
182  \return  Returns adjacency matrix of the graph
183  */
184 /*****
185  ALIST ALGraph::GetAList(void) const
186  {
187      ALIST adjList;
188
189      //copy graph to adjList matrix
190      for(auto i : graph)
191      {
192          std::vector<AdjacencyInfo> v;
193
194          for(auto j : i)
195          {
196              v.push_back(AdjacencyInfo{j.id, j.weight});
197          }
198
199          adjList.push_back(v);
200      }
201      return adjList;
202  }
203
204 /*****
205  *!
206  \fn      bool ALGraph::AdjInfo::operator < (const AdjInfo& rhs) const
207
208  \brief   Compares to AdjacencyInfo node for sorting
209
210  \param   rhs - right hand side node
211
212  \return  Returns true if weight is less than rhs.weight, else, return false.
213  */
214 /*****

```

```
215 bool ALGraph::AdjInfo::operator < (const AdjInfo& rhs) const
216 {
217     return weight < rhs.weight;
218 }
219
220 /*****
221  *!
222  \fn      bool ALGraph::AdjInfo::operator > (const AdjInfo& rhs) const
223
224  \brief   Compares to AdjacencyInfo node for sorting
225
226  \param   rhs - right hand side node
227
228  \return  Returns true if weight is more than rhs.wight, else, return false.
229  */
230 /*****
231  bool ALGraph::AdjInfo::operator > (const AdjInfo& rhs) const
232  {
233      return weight > rhs.weight;
234  }
235
236 /*****
237  *!
238  \fn      unsigned ALGraph::extract_min(std::vector<AdjacencyInfo>& pq) const
239
240  \brief   Extract the node with the minimum cost
241
242  \param   pq - priority queue that contains a vector of nodes
243
244  \return  Returns the node index with the minimum cost
245  */
246 /*****
247  unsigned ALGraph::extract_min(std::vector<AdjacencyInfo>& pq) const
248  {
249      if(pq.empty())
250          return 0;
251
252      unsigned id = pq[0].id;
253      unsigned min_weight = INFINITY;
254
255      for(unsigned i = 0; i < pq.size(); ++i)
256      {
257          if(pq[i].weight < min_weight)
258          {
259              id = pq[i].id;
260              min_weight = pq[i].weight;
261          }
262      }
263
264      return id;
```

[VPL](#)

[◀ Assignment 3: AVL Trees](#)

Jump to...

[Assignment 5: Hashing ▶](#)

You are logged in as [Wei Zhe GOH](#) ([Log out](#))
[cs280s21-b.sg](#)
[Data retention summary](#)
[Get the mobile app](#)