МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

DEVOPS: ЕГО РОЛЬ В СОВРЕМЕННОЙ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ОТЧЁТ

студента 3 курса 351 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Устюшина Богдана Антоновича

Проверено:
доцент, к. п. н. _____ М. С. Портенко

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Отчёт по теории графов. **Используемый язык** – C++.

1 Задание 1: создание класса графов

В первом задании предлагалось создать свой собственный класс графов, в который входят:

1. Структуру хранения графа (список смежности или список рёбер)
2. Конструкторы (пустой граф, граф из файла, конструктор-копию)
3. Методы (добавления/удаления вершин, добавления/удаления рёбер или дуг, вывод списка смежности в файл, т.е. сохранения графа)
4. Должны поддерживаться как ориентированные/неориентированные графы
5. Должен быть создан минималистичный консольный интерфейс

Весь код представлен в приложении А.

Структура хранения выбрана как словарь словарей (`map<map<string, int>>`): то есть у каждой вершины есть `map`, который соответствует вершинам, с которыми она смежна (в случае неориентированного графа) или вершинам, в которые идут дуги (в случае ориентированного графа).

Представлены конструкторы на основе:

1. Bool-переменной `isOriented`, которая определяет, является ли граф ориентированным или нет (поддержка различия двух графов);
2. Файла, в котором граф представляется в json-формате с помощью библиотеки nlohmann-json;
3. Ссылки на константное значение класса графа – конструктор копирования.

Написать примеры создания графа

Также реализован независимый от класса графа консольный интерфейс (файлы `App.h` и `App.cpp`), функции которого потом используются в функции `main` в главном файле `main.cpp`.

В нём созданы:

1. `enum string_code` – введённая пользователем строка на основе функции Hashing будет преобразовываться в номер для выполнения определённой команды.
2. `CommandMessage()` – выводит справку о поддерживаемых командах, появляется при старте программы и вызове функции `help` (h)
3. `CreateGraph` – интерфейс создания графа (вызывается при пустом файле, указанном в переменной `DATA_FILE_1` имени графа, используемой в главном файле `main.cpp`).

ЗАКЛЮЧЕНИЕ

В этом реферате были рассмотрены основные инструменты DevOps инженера.

ПРИЛОЖЕНИЕ А

Файлы класса графов

Graph.h файл:

```
1  #pragma once
2
3  #include <map>
4  #include <iostream>
5  #include <fstream>
6  #include <string>
7  #include <nlohmann/json.hpp>
8
9  using json = nlohmann::json;
10
11 using std::string;
12 using std::map;
13 using std::vector;
14 using std::pair;
15 using std::ofstream;
16 using std::ifstream;
17
18 class Graph
19 {
20         friend void to_json(json& j, const Graph& graph);
21 public:
22
23         enum graph_orientation
24         {
25                 undirected = 0,
26                 directed = 1
27         };
28
29         enum code_error
30         {
31                 no_error = 0,
```

```
32
33                  vertice_exists,
34                  no_vertice1,
35                  no_vertice2,
36                  edge_exists,
37                  no_edge
38          };
39
40          Graph(bool orient = true);
41          Graph(ifstream& file);
42          Graph(const Graph& copiedValue);
43          Graph(map<string, map<string, int32_t>>, bool isOriented);
44
45          const map<string, map<string, int32_t>> GetAdjacencyList()
    ↪ const;
46          bool GetOrientation();
47          void ChangeOrientation();
48
49          bool isVertice(string s);
50          bool isEdge(string s1, string s2);
51
52          static graph_orientation Hashing(string const& inString);
53
54          uint8_t AddVertice(const string& value);
55          uint8_t AddEdge(const string& startVertice, const string&
    ↪ endVertice, const int32_t& weight);
56
57          uint8_t RemoveVertice(const string& vertice);
58          uint8_t RemoveEdge(const string& startVertice, const
    ↪ string& endVertice);
59
60          uint8_t ChangeWeight(const string& startVertice, const
    ↪ string& endVertice, const int32_t& weight);
61          void Unweight();
```

```
62
63          void Save(string fileName);
64  private:
65          map<string, map<string, int32_t>> adjacencyList;
66          bool isOriented;
67  };
```

Graph.cpp файл:

```
1   #include "Graph.h"
2
3   Graph::Graph(bool isOriented)
4   {
5           this->isOriented = isOriented;
6   }
7
8   Graph::Graph(ifstream& file)
9   {
10          json j;
11          file >> j;
12          this->isOriented = j["orient"];
13          for (auto& adjList : j["vertices"].items())
14          {
15                  this->adjacencyList[adjList.key()] =
    ↪  j["vertices"][adjList.key()].get<map<string, int32_t>>();
16          }
17  }
18
19  Graph::Graph(const Graph& copiedValue)
20  {
21          this->isOriented = copiedValue.isOriented;
22          this->adjacencyList = copiedValue.adjacencyList;
23  }
24
25  Graph::Graph(map <string, map<string, int32_t>> list, bool
    ↪  isOriented)
```

```
26 {
27         this->adjacencyList = list;
28         this->isOriented = isOriented;
29 }
30
31 //getters
32 const map<string, map<string, int32_t>> Graph::GetAdjacencyList()
    ↪ const
33 {
34         return adjacencyList;
35 }
36
37 bool Graph::GetOrientation()
38 {
39         return isOriented;
40 }
41
42 void Graph::ChangeOrientation()
43 {
44         isOriented = !isOriented;
45 }
46
47 bool Graph::isVertice(string s)
48 {
49         return !(adjacencyList.find(s) == adjacencyList.end());
50 }
51
52 bool Graph::isEdge(string s1, string s2)
53 {
54         return adjacencyList[s1].find(s2) !=
    ↪ adjacencyList[s1].end();
55 }
56
57
```

```
58 Graph::graph_orientation Graph::Hashing(std::string const&
    ↪ inString)
59 {
60         if (inString == "0")
61                 return graph_orientation::undirected;
62         else
63                 return graph_orientation::directed;
64 }
65
66 //methods
67 uint8_t Graph::AddVertice(const string& vertice)
68 {
69         if (!isVertice(vertice))
70         {
71                 adjacencyList[vertice];
72                 return code_error::no_error;
73         }
74         else
75                 return code_error::vertice_exists;
76 }
77
78 uint8_t Graph::ChangeWeight(const string& startVertice, const
    ↪ string& endVertice, const int32_t& weight)
79 {
80         //fail: no first vertice
81         if (!isVertice(startVertice))
82                 return code_error::no_vertice1;
83
84         //fail: no second vertice
85         if (!isVertice(endVertice))
86                 return code_error::no_vertice2;
87
88         //fail: no such edge
89         if (!isEdge(startVertice, endVertice))
```

```
90                  return code_error::no_edge;
91
92          //success
93          adjacencyList[startVertice][endVertice] = weight;
94          if (!isOriented)
95                  adjacencyList[endVertice][startVertice] = weight;
96          return code_error::no_error;
97  }
98
99  uint8_t Graph::AddEdge(const string& startVertice, const string&
    ↪  endVertice, const int32_t& weight)
100 {
101         //fail: no first vertice
102         if (!isVertice(startVertice))
103                 return code_error::no_vertice1;
104
105         //fail: no second vertice
106         if (!isVertice(endVertice))
107                 return code_error::no_vertice2;
108
109         //fail: edge already exists
110         if (isEdge(startVertice, endVertice))
111                 return code_error::edge_exists;
112
113         //success
114         adjacencyList[startVertice][endVertice] = weight;
115         if (!isOriented)
116                 adjacencyList[endVertice][startVertice] = weight;
117         return code_error::no_error;
118 }
119
120 uint8_t Graph::RemoveVertice(const string& removedVertice)
121 {
122         //fail: no such vertice
```

```
123         if (!isVertice(removedVertice))
124                 return code_error::no_vertice1;
125
126         //success, cleaning other vertices and their edges
127         for (auto vert : adjacencyList)
128         {
129                 this->RemoveEdge(vert.first, removedVertice);
130                 this->RemoveEdge(removedVertice, vert.first);
131         }
132         adjacencyList.erase(removedVertice);
133         return code_error::no_error;
134 }
135
136 uint8_t Graph::RemoveEdge(const string& startVertice, const
    ↪ string& endVertice)
137 {
138         if (adjacencyList[startVertice].find(endVertice) !=
    ↪ adjacencyList[startVertice].end())
139         {
140                 adjacencyList[startVertice].erase(endVertice);
141                 if (!isOriented)
142                 {
143                         adjacencyList[endVertice].erase(startVertice);
144                 }
145                 //success
146                 return no_error;
147         }
148         else
149                 return code_error::no_edge;
150 }
151
152 void Graph::Unweight()
153 {
```

```
154         for (auto vert1 = adjacencyList.begin(); vert1 !=
    ↪  adjacencyList.end(); vert1++)
155         {
156                 for (auto vert2 = adjacencyList.begin(); vert2 !=
    ↪  adjacencyList.end(); vert2++)
157                 {
158                         if (isEdge(vert1->first, vert2->first))
159                                 adjacencyList
    ↪  [vert1->first][vert2->first] = 1;
160                 }
161         }
162 }
163
164 void Graph::Save(string fileName)
165 {
166         json j = *this;
167         std::ofstream data(fileName);
168         data << std::setw(4) << j;
169 }
170
171 void to_json(json& j, const Graph& graph)
172 {
173         j["orient"] = graph.isOriented;
174         for (auto const& mapEl : graph.adjacencyList)
175         {
176                 j["vertices"][mapEl.first] = mapEl.second;
177         }
178 }
```

ПРИЛОЖЕНИЕ Б

Файлы класса приложения

App.h

```
1  #pragma once
2
3  #include <string>
4  #include <iostream>
5
6  #include "Graph.h"
7
8  enum string_code
9  {
10         printVertices = 1,
11         addVertice,
12         removeVertice,
13         addEdge,
14         removeEdge,
15         changeWeight,
16         saveGraph,
17         unweightGraph,
18
19         task2 = 10,
20         task3,
21         task4,
22         task5,
23         task6,
24         task7,
25         task8,
26         task9,
27         task10,
28         task11,
29
30         help = 30,
31         quit
```

```
32 };
33
34 string_code Hashing(std::string const& inString);
35 void CommandMessage();
36 Graph* CreateGraph(string& command);
37
38 bool is_number(const string& s);
39 void PrintVertices(Graph* graph);
40 void AddVertice(Graph* graph);
41 void RemoveVertice(Graph* graph);
42 void AddEdge(Graph* graph);
43 void RemoveEdge(Graph* graph);
44 void ChangeWeight(Graph* graph);
45 void Unweight(Graph* graph);
```

App.cpp

```
1  #include "App.h"
2
3  using std::string;
4  using std::getline;
5  using std::cin;
6  using std::cout;
7
8  string_code Hashing(std::string const& inString) {
9          if (inString == "1") return printVertices;
10         if (inString == "2") return addVertice;
11         if (inString == "3") return removeVertice;
12         if (inString == "4") return addEdge;
13         if (inString == "5") return removeEdge;
14         if (inString == "6") return changeWeight;
15         if (inString == "7") return saveGraph;
16         if (inString == "8") return unweightGraph;
17
18         if (inString == "T2") return task2;
```

```
19          if (inString == "T3") return task3;
20          if (inString == "T4") return task4;
21          if (inString == "T5") return task5;
22          if (inString == "T6") return task6;
23          if (inString == "T7") return task7;
24          if (inString == "T8") return task8;
25          if (inString == "T9") return task9;
26          if (inString == "T10") return task10;
27          if (inString == "T11") return task11;
28
29          if (inString == "h") return help;
30          if (inString == "q") return quit;
31  }
32
33  void CommandMessage()
34  {
35          std::cout << "Select command: \n"
36                  << "1 - Print Vertices \n"
37                  << "2 - Add vertice \n"
38                  << "3 - Remove vertice \n"
39                  << "4 - Add edge \n"
40                  << "5 - Remove edge \n"
41                  << "6 - Change edge's weight \n"
42                  << "7 - Save graph \n"
43                  << "8 - Unweight graph \n"
44                  << '\n'
45                  << "T2 - task 2 \n"
46                  << "T3 - task 3 \n"
47                  << "T4 - task 4 \n"
48                  << "T5 - task 5 \n"
49                  << "T6 - task 6 \n"
50                  << "T7 - task 7 \n"
51                  << "T8 - task 8 \n"
52                  << "T9 - task 9 \n"
```

```
53                 << "T10 - task 10\n"
54                 << "T11 - task 11\n"
55                 << '\n'
56                 << "'h' to print this message\n"
57                 << "'q' to exit program\n";
58 }
59
60 Graph* CreateGraph(string& command)
61 {
62         while (true)
63         {
64                 std::cout << "No graph found. Choose directed or
    ↪  undirected graph\n"
65                         << "1 - Directed graph\n"
66                         << "2 - Undirected graph\n";
67                 getline(std::cin, command);
68                 switch (Graph::Hashing(command))
69                 {
70                 case Graph::undirected:
71                         std::cout << "Created a new undirected
    ↪  graph\n";
72                         return new Graph(false);
73                 case Graph::directed:
74                         std::cout << "Created a new directed
    ↪  graph\n";
75                         return new Graph(true);
76                 default:
77                         break;
78                 }
79         }
80 }
81
82 //Hidden
83 bool is_number(const std::string& s)
```

```
84 {
85         std::string::const_iterator it = s.begin();
86         while (it != s.end() && (std::isdigit(*it) || (*it) ==
   ↪  '-')) ++it;
87         return !s.empty() && it == s.end();
88 }
89
90 void PrintVertices(Graph* graph)
91 {
92         auto adjacencyList = graph->GetAdjacencyList();
93
94         for (auto& list : adjacencyList)
95         {
96                 std::cout << list.first << ": ";
97                 for (auto& el : list.second)
98                 {
99                         std::cout << "(" << el.first << ", " <<
   ↪  el.second << ") ";
100                 }
101                 std::cout << '\n';
102         }
103 }
104
105 void AddVertice(Graph* graph)
106 {
107         string vertice;
108         std::cout << "Enter vertice to add: ";
109         getline(cin, vertice);
110         switch (graph->AddVertice(vertice))
111         {
112         case Graph::code_error::no_error:
113                 std::cout << "Vertice added succesfully\n";
114                 break;
115         case Graph::code_error::vertice_exists:
```

```
116                 std::cout << "Vertice already exists\n";
117                 break;
118         }
119 }
120
121 void RemoveVertice(Graph* graph)
122 {
123         string vertice;
124         std::cout << "Enter vertice to remove: ";
125         getline(cin, vertice);
126         switch (graph->RemoveVertice(vertice))
127         {
128         case Graph::code_error::no_error:
129                 std::cout << "Vertice removed successfully\n";
130                 break;
131         case Graph::code_error::no_vertice1:
132                 std::cout << "No such vertice in graph\n";
133                 break;
134         }
135 }
136
137 void AddEdge(Graph* graph)
138 {
139         string vertice1;
140         string vertice2;
141         string weightMsg;
142         std::cout << "Enter start vertice: ";
143         getline(cin, vertice1);
144         std::cout << "Enter end vertice: ";
145         getline(cin, vertice2);
146         std::cout << "Enter edge weight (default = 1): ";
147         getline(cin, weightMsg);
148         if (weightMsg.empty())
149                 weightMsg = "1";
```

```
150         while (!is_number(weightMsg))
151         {
152                 std::cout << "Wrong weight value! Enter integer:
    ↪ ";
153                 getline(cin, weightMsg);
154                 if (weightMsg.empty())
155                         weightMsg = "1";
156         }
157
158         switch (graph->AddEdge(vertice1, vertice2,
    ↪ std::stoi(weightMsg)))
159         {
160         case Graph::no_vertice1:
161                 std::cout << "No vertice 1 represented in
    ↪ graph\n";
162                 break;
163         case Graph::no_vertice2:
164                 std::cout << "No vertice 2 represented in
    ↪ graph\n";
165                 break;
166         case Graph::edge_exists:
167                 std::cout << "Edge already exists between these 2
    ↪ vertices\n";
168                 break;
169         case Graph::no_error:
170                 std::cout << "Edge added successfully\n";
171                 break;
172         }
173 }
174
175 void RemoveEdge(Graph* graph)
176 {
177         string vertice1;
178         string vertice2;
```

```
179         std::cout << "Enter start vertice: ";
180         getline(cin, vertice1);
181         std::cout << "Enter end vertice: ";
182         getline(cin, vertice2);
183 
184         switch (graph->RemoveEdge(vertice1, vertice2))
185         {
186         case Graph::code_error::no_edge:
187                 std::cout << "No such edge in graph\n";
188                 break;
189         case Graph::code_error::no_error:
190                 std::cout << "Edge removed successfully\n";
191                 break;
192         }
193 }
194 
195 void ChangeWeight(Graph* graph)
196 {
197         string vertice1;
198         string vertice2;
199         string weightMsg;
200         std::cout << "Enter start vertice: ";
201         getline(cin, vertice1);
202         std::cout << "Enter end vertice: ";
203         getline(cin, vertice2);
204         std::cout << "Enter edge weight (default = 1): ";
205         getline(cin, weightMsg);
206         if (weightMsg.empty())
207                 weightMsg = "1";
208         while (!is_number(weightMsg))
209         {
210                 std::cout << "Wrong weight value! Enter integer:
    ↪   ";
211                 getline(cin, weightMsg);
```

```
212         }
213
214         switch (graph->ChangeWeight(vertice1, vertice2,
    ↪ std::stoi(weightMsg)))
215         {
216         case Graph::no_vertice1:
217                 std::cout << "No vertice 1 represented in
    ↪ graph\n";
218                 break;
219         case Graph::no_vertice2:
220                 std::cout << "No vertice 2 represented in
    ↪ graph\n";
221                 break;
222         case Graph::no_edge:
223                 std::cout << "No edge exists between these 2
    ↪ vertices\n";
224                 break;
225         case Graph::no_error:
226                 std::cout << "Weight changed successfully\n";
227                 break;
228         }
229 }
230
231 void Unweight(Graph* graph)
232 {
233         graph->Unweight();
234         std::cout << "Weight of all edges changed to 1\n";
235 }
```

ПРИЛОЖЕНИЕ В
Файл Main.cpp

main.cpp

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <map>
5
6  #include "Graph.h"
7  #include "App.h"
8  #include "Tasks.h"
9
10 const string DATA_FILE1 = "task11_2.json";
11 const string DATA_FILE2 = "data4.json";
12
13 int main()
14 {
15         string command;
16
17         Graph* graph1;
18         std::ifstream file(DATA_FILE1);
19         if (file.is_open())
20                 graph1 = new Graph(file);
21         else
22                 graph1 = CreateGraph(command);
23         file.close();
24
25         file.open(DATA_FILE2);
26         Graph* graph2 = new Graph(file);
27         file.close();
28
29         CommandMessage();
30         while (true)
31         {
```

```
32                  std::cout << "Command: ";
33                  std::getline(std::cin, command);
34                  switch (Hashing(command))
35                  {
36                  case string_code::printVertices:
37                          PrintVertices(graph1);
38                          break;
39                  case string_code::addVertice:
40                          AddVertice(graph1);
41                          break;
42                  case string_code::removeVertice:
43                          RemoveVertice(graph1);
44                          break;
45                  case string_code::addEdge:
46                          AddEdge(graph1);
47                          break;
48                  case string_code::removeEdge:
49                          RemoveEdge(graph1);
50                          break;
51                  case string_code::changeWeight:
52                          ChangeWeight(graph1);
53                          break;
54                  case string_code::unweightGraph:
55                          Unweight(graph1);
56                          break;
57
58                  case string_code::saveGraph:
59                          graph1->Save(DATA_FILE1);
60                          std::cout << "Graph saved succesfully \n";
61                          break;
62
63                  case string_code::task2:
64                          task2_14(graph1);
65                          break;
```

```
66                  case string_code::task3:
67                          task3_9(graph1);
68                          break;
69                  case string_code::task4:
70                          task4_10(graph1, graph2);
71                          break;
72                  case string_code::task5:
73                          task5_2(graph1);
74                          break;
75                  case string_code::task6:
76                          task6_20(graph1);
77                          break;
78                  case string_code::task7:
79                          task7_prim(graph1);
80                          break;
81                  case string_code::task8:
82                          task8_11(graph1);
83                          break;
84                  case string_code::task9:
85                          task9_17(graph1);
86                          break;
87                  case string_code::task10:
88                          task10_1(graph1);
89                          break;
90                  case string_code::task11:
91                          task11_net(graph1);
92                          break;
93
94                  case string_code::help:
95                          CommandMessage();
96                          break;
97                  case string_code::quit:
98                          graph1->Save(DATA_FILE1);
99                          return 0;
```

```
100
101                     default:
102                             std::cout << "Wrong Command Number\n";
103                     }
104             }
105     }
```