

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

УТВЕРЖДАЮ

Зав.кафедрой,

к. ф.-м. н., доцент

_____ С. В. Миронов

ОТЧЕТ О ПРАКТИКЕ

студента 3 курса 311 группы факультета КНиИТ
Козырева Юрия Дмитриевича

вид практики: учебная

кафедра: математической кибернетики и компьютерных наук

курс: 3

семестр: 2

продолжительность: 2 нед., с _____ г. по _____ г.

Руководитель практики от университета,

старший преподаватель _____

М. С. Портенко

Руководитель практики от организации (учреждения, предприятия),

старший преподаватель _____

М. С. Портенко

Тема практики: «Теория графов»

СОДЕРЖАНИЕ

1	Создание класса Граф	5
1.1	Задание	5
1.2	Решение	6
2	Список смежности Ia(1)	18
2.1	Задание	18
2.2	Решение	18
2.3	Пример	18
3	Список смежности Ia(2)	20
3.1	Задание	20
3.2	Решение	20
3.3	Пример	20
4	Список смежности Ib: несколько графов	22
4.1	Задание	22
4.2	Решение	22
4.3	Пример	23
5	Обходы графа II(1)	25
5.1	Задание	25
5.2	Решение	25
5.3	Пример	26
6	Обходы графа II(2)	27
6.1	Задание	27
6.2	Решение	27
6.3	Пример	28
7	Каркас III	30
7.1	Задание	30
7.2	Решение	30
7.3	Пример	32
8	Весы IV a	33
8.1	Задание	33
8.2	Решение	33
8.3	Пример	35
9	Весы IV b	36
9.1	Задание	36

9.2	Решение	36
9.3	Пример	37
10	Весы IV с	39
10.1	Задание	39
10.2	Решение	39
10.3	Пример	41
11	V Максимальный поток	42
11.1	Задание	42
11.2	Решение	42
11.3	Пример	43

1 Создание класса Граф

1.1 Задание

Необходимо создать класс (или иерархию классов - на усмотрение разработчика), содержащий:

- Структуру для хранения списка смежности графа (не работать с графом через матрицы смежности, если в некоторых алгоритмах удобнее использовать список ребер - реализовать метод, преобразующий список смежности в список ребер);
- Конструкторы (не менее 3-х):
 - конструктор по умолчанию, создающий пустой граф;
 - конструктор, заполняющий данные графа из файла;
 - конструктор-копию (аккуратно, не все сразу делают именно копию);
 - специфические конструкторы для удобства тестирования.
- ;
- Методы:
 - добавляющие вершину;
 - добавляющие ребро (дугу);
 - удаляющие вершину;
 - удаляющие ребро (дугу);
 - выводящие список смежности в файл (в том числе в пригодном для чтения конструктором формате).

Не выполняйте некорректные операции, сообщайте об ошибках;

- Должны поддерживаться как ориентированные, так и неориентированные графы. Заранее предусмотрите возможность добавления меток и или весов для дуг. Поддержка мультиграфа не требуется;
- Добавьте минималистичный консольный интерфейс пользователя (не смешивая его с реализацией!), позволяющий добавлять и удалять вершины и рёбра (дуги) и просматривать текущий список смежности графа;
- Сгенерируйте не менее 4 входных файлов с разными типами графов (балансируйте на комбинации ориентированность-взвешенность) для тестирования класса в этом и последующих заданиях. Графы должны содержать не менее 7-10 вершин, в том числе, петли и изолированные вершины.

1.2 Решение

```
1      #pragma once
2      #include <iostream>
3      #include <vector>
4      #include <map>
5      #include <string>
6      #include <sstream>
7      #include <set>
8      #include <utility>
9      #include <algorithm>
10     using namespace std;
11     class edge;
12     class vertex {
13     private:
14         long long id;
15         map<long long, edge*> outEdges;
16         map<long long, edge*> inEdges;
17         static long long lastId;
18     public:
19         long long getId() {
20             return id;
21         }
22
23         void setId(long long id) {
24             this->id = id;
25         }
26
27         map<long long, edge*> getInEdges() {
28             return inEdges;
29         }
30
31         void setInEdges(map<long long, edge*> inEdges) {
32             this->inEdges = inEdges;
33         }
34
35         map<long long, edge*> getOutEdges() {
36             return outEdges;
37         }
38
39         void setOutEdges(map<long long, edge*> outEdges) {
```

```

40         this->outEdges = outEdges;
41     }
42
43     vertex() {
44         inEdges = map<long long, edge*>();
45         outEdges = map<long long, edge*>();
46         id = 0;
47     }
48
49     vertex(long long n) {
50         inEdges = map<long long, edge*>();
51         outEdges = map<long long, edge*>();
52         id = n;
53     }
54
55     vertex(const vertex &v) {
56         id = v.id;
57         inEdges = v.inEdges;
58         outEdges = v.outEdges;
59     }
60
61     bool operator==(const vertex& v) {
62         return this->getId() == v.id;
63     }
64 };
65 bool operator==(const pair<const long long, vertex*>& v1, const pair<const
↪ long long, vertex*>& v2) {
66     return v1.second == v2.second;
67 }
68
69 class edge {
70 private:
71     vertex* start;
72     vertex* end;
73     long long id;
74     double weight;
75     bool weighted;
76 public:
77     static long long lastId;
78     vertex* getStart() {
79         return start;

```

```

80     }
81
82     void setStart(vertex* start) {
83         this->start = start;
84     }
85
86     vertex* getEnd() {
87         return end;
88     }
89
90     void setEnd(vertex* end) {
91         this->end = end;
92     }
93
94     long long getId() {
95         return id;
96     }
97
98     double getWeight() {
99         if (weighted) {
100             return weight;
101         }
102         else {
103             throw "Error: graph is not weighted";
104             return 0;
105         }
106     }
107
108     void setWeight(double weight) {
109         if (weighted) {
110             this->weight = weight;
111         }
112         else {
113             throw "Error: graph is not weighted\n";
114             return;
115         }
116     }
117
118     bool isWeighted() {
119         return weighted;
120     }

```



```

121
122     edge() {
123         start = new vertex();
124         end = new vertex();
125         weighted = 0;
126         weight = 0;
127         lastId++;
128         id = lastId;
129     }
130
131     edge(vertex* start, vertex* end, bool weighted, double weight = 0) {
132         this->start = start;
133         this->end = end;
134         this->weighted = weighted;
135         this->weight = weight;
136         lastId++;
137         id = lastId;
138     }
139
140     edge(const edge &e) {
141         start = e.start;
142         end = e.end;
143         weighted = e.weighted;
144         weight = e.weight;
145         id = e.id;
146     }
147
148     bool operator ==(edge e) {
149         return this->id == e.id;
150     }
151 };
152 long long edge::lastId = -1;
153
154 class graph {
155     map<long long, vertex*> V;
156     map<long long, edge*> E;
157     bool weighted;
158     bool oriented;
159     bool multy;
160 public:
161     graph() {

```

```

162         V = map<long long, vertex*>();
163         E = map<long long, edge*>();
164         weighted = 0;
165         oriented = 0;
166         multy = 0;
167     }
168
169     graph(const graph &g) {
170         V = g.V;
171         E = g.E;
172         weighted = g.weighted;
173         oriented = g.oriented;
174         multy = g.multy;
175     }
176
177     void setType(bool w, bool o, bool m) {
178         weighted = w;
179         oriented = o;
180         multy = m;
181     }
182
183     bool isWeighted() {
184         return weighted;
185     }
186
187     bool isOriented() {
188         return oriented;
189     }
190
191     bool isMulty() {
192         return multy;
193     }
194
195     void add(long long p1) {
196         if (find_if(V.begin(), V.end(),
197             [p1](pair <long long, vertex*> v) {return v.second->getId() ==
198                 ↪ p1; }) == V.end()) {
199             V.insert(pair<long long, vertex*>(p1, new vertex(p1)));
200         }
201         else {
202             throw "Error: repeating vertex:" + to_string(p1) + "\n";

```

```

202         return;
203     }
204 }
205
206 void add(long long p1, long long p2, double w = 0) {
207     map<long long, vertex*>::iterator startI =
208         find_if(V.begin(), V.end(),
209             [p1](pair <long long, vertex*> v) {return v.second->getId()
210                 ↪ == p1; });
211     if (startI == V.end()) {
212         throw "Error: the graph doesn't contain starting vertex of this
213             ↪ edge\n ";
214         return;
215     }
216     map<long long, vertex*>::iterator endI =
217         find_if(V.begin(), V.end(),
218             [p2](pair <long long, vertex*> v) {return v.second->getId()
219                 ↪ == p2; });
220     if (endI == V.end()) {
221         throw "Error: the graph doesn't contain ending vertex of this
222             ↪ edge\n ";
223         return;
224     }
225     vertex* newStart = (*startI).second;
226     vertex* newEnd = (*endI).second;
227     edge* newEdge = new edge(newStart, newEnd, weighted, w);
228     E.insert(pair<long long, edge*>(newEdge->getId(), newEdge));
229     map<long long, edge*> newOutEdges = newStart->getOutEdges();
230     newOutEdges.insert(pair<long long, edge*>(newEdge->getId(),
231         ↪ newEdge));
232     newStart->setOutEdges(newOutEdges);
233     map<long long, edge*> newInEdges = newEnd->getInEdges();
234     newInEdges.insert(pair<long long, edge*>(newEdge->getId(),
235         ↪ newEdge));
236     newEnd->setInEdges(newInEdges);
237     if (!oriented) {
238         edge* newREdge = new edge(newEnd, newStart, weighted, w);
239         E.insert(pair<long long, edge*>(newREdge->getId(), newREdge));
240         map<long long, edge*> newROutEdges = newEnd->getOutEdges();
241         newOutEdges.insert(pair<long long, edge*>(newREdge->getId(),
242             ↪ newREdge));

```

```

236         newEnd->setOutEdges(newROutEdges);
237         map<long long, edge*> newRInEdges = newStart->getInEdges();
238         newRInEdges.insert(pair<long long, edge*>(newREdge->getId(),
239             ↪ newREdge));
240         newEnd->setInEdges(newRInEdges);
241     }
242 }
243
244 void addVertex(vertex* v) {
245     V.insert(pair<long long, vertex*>(V.size(), v));
246 }
247
248 void addEdge(edge* e) {
249     map<long long, vertex*>::iterator startI =
250         find_if(V.begin(), V.end(), [e](pair <long long, vertex*> v)
251             ↪ {return v.second->getId() == e->getStart()->getId(); });
252     if (startI == V.end()) {
253         throw "Error: the graph doesn't contain starting vertex of this
254             ↪ edge\n ";
255         return;
256     }
257     map<long long, vertex*>::iterator endI =
258         find_if(V.begin(), V.end(), [e](pair <long long, vertex*> v)
259             ↪ {return v.second->getId() == e->getEnd()->getId(); });
260     if (endI == V.end()) {
261         throw "Error: the graph doesn't contain ending vertex of this
262             ↪ edge\n ";
263         return;
264     }
265     vertex* newStart = (*startI).second;
266     vertex* newEnd = (*endI).second;
267     edge* newEdge = new edge(newStart, newEnd, weighted,
268         ↪ e->getWeight());
269     E.insert(pair<long long, edge*>(E.size(), newEdge));
270     map<long long, edge*> newOutEdges = newStart->getOutEdges();
271     newOutEdges.insert(pair<long long, edge*>(newOutEdges.size(),
272         ↪ newEdge));
273     newStart->setOutEdges(newOutEdges);
274     map<long long, edge*> newInEdges = newEnd->getInEdges();
275     newInEdges.insert(pair<long long, edge*>(newInEdges.size(),
276         ↪ newEdge));

```

```

271     newEnd->setInEdges(newInEdges);
272     if (!oriented) {
273         edge* newREdge = new edge(newEnd, newStart, weighted,
274             ↪ e->getWeight());
275         E.insert(pair<long long, edge*>(E.size(), newREdge));
276         map<long long, edge*> newROutEdges = newEnd->getOutEdges();
277         newOutEdges.insert(pair<long long, edge*>(newROutEdges.size(),
278             ↪ newREdge));
279         newEnd->setOutEdges(newROutEdges);
280         map<long long, edge*> newRInEdges = newStart->getInEdges();
281         newRInEdges.insert(pair<long long, edge*>(newRInEdges.size(),
282             ↪ newREdge));
283         newStart->setInEdges(newRInEdges);
284     }
285 }
286
287 graph(string s) {
288     V = map<long long, vertex*>();
289     E = map<long long, edge*>();
290     stringstream ss = stringstream(s);
291     ss >> weighted >> oriented >> multy;
292     string regime = "none";
293     while(!ss.eof()){
294         string line;
295         getline(ss, line, '\n');
296         if (line == "vertexes:"){
297             regime = "vertexes";
298             continue;
299         }
300         if (line == "edges:"){
301             regime = "edges";
302             continue;
303         }
304         if (regime == "vertexes"){
305             char delim = line[line.size() - 1];
306             line = line.substr(0, line.size() - 1);
307             stringstream linereader = stringstream(line);
308             long long vernum;
309             if (!linereader.eof()) {
310                 linereader >> vernum;
311                 add(vernum);

```

```

309         }
310         if (delim == '.'){
311             regime = "none";
312         }
313         continue;
314     }
315     if (regime == "edges"){
316         char delim = line[line.size() - 1];
317         line = line.substr(0, line.size() - 1);
318         stringstream linereader = stringstream(line);
319         if (weighted) {
320             long long ver1num;
321             long long ver2num;
322             double w;
323             if (!linereader.eof()) {
324                 linereader >> ver1num >> ver2num >> w;
325                 add(ver1num, ver2num, w);
326             }
327         }
328         else{
329             long long ver1num;
330             long long ver2num;
331             linereader >> ver1num >> ver2num;
332             add(ver1num, ver2num);
333         }
334         if (delim == '.'){
335             regime = "none";
336         }
337         continue;
338     }
339     if (regime == "none"){
340         continue;
341     }
342 }
343 /*example:
344     0 0 0
345     vertexes:
346         1,
347         2,
348         3.
349     edges:

```

```

350         1 2.
351     vertices:
352         4.
353     edges:
354         3 4,
355         1 3.
356     */
357 }
358
359 void eraseEdge(long long id) {
360     map<long long, edge*>::iterator ei = find_if(E.begin(), E.end(),
361         [id](pair<long long, edge*> e) {return e.second->getId() == id;
362             ↪ });
363     if (ei == E.end()) {
364         throw "Error: this edge doesn't exist\n";
365         return;
366     }
367     map<long long, edge*> startsOut =
368         ↪ ei->second->getStart()->getOutEdges();
369     startsOut.erase(ei->second->getId());
370     ei->second->getStart()->setOutEdges(startsOut);
371     map<long long, edge*> endsIn = ei->second->getEnd()->getInEdges();
372     endsIn.erase(ei->second->getId());
373     ei->second->getEnd()->setInEdges(endsIn);
374     if (!oriented) {
375         map<long long, edge*> startsIn =
376             ↪ ei->second->getStart()->getInEdges();
377         startsIn.erase(ei->second->getId() + 1);
378         ei->second->getStart()->setInEdges(startsIn);
379         map<long long, edge*> endsOut =
380             ↪ ei->second->getEnd()->getOutEdges();
381         endsOut.erase(ei->second->getId() + 1);
382         ei->second->getEnd()->setInEdges(endsOut);
383     }
384     E.erase(id);
385     E.erase(id + 1);
386 }
387
388 void eraseEdge(map<long long, edge*>::iterator ei) {
389     map<long long, edge*> startsOut =
390         ↪ ei->second->getStart()->getOutEdges();

```

```

386     startsOut.erase(ei->second->getId());
387     ei->second->getStart()->setOutEdges(startsOut);
388     map<long long, edge*> endsIn = ei->second->getEnd()->getInEdges();
389     endsIn.erase(ei->second->getId());
390     ei->second->getEnd()->setInEdges(endsIn);
391     E.erase(ei->second->getId());
392     if (!oriented) {
393         map<long long, edge*> startsIn =
394             ↪ ei->second->getStart()->getInEdges();
395         startsIn.erase(ei->second->getId() + 1);
396         ei->second->getStart()->setInEdges(startsIn);
397         map<long long, edge*> endsOut =
398             ↪ ei->second->getEnd()->getOutEdges();
399         endsOut.erase(ei->second->getId() + 1);
400         ei->second->getEnd()->setInEdges(endsOut);
401         E.erase(ei->second->getId() + 1);
402     }
403 }
404
405 void eraseVertex(long long id) {
406     map<long long, vertex*>::iterator vi = find_if(V.begin(), V.end(),
407         [id](pair<long long, vertex*> v) {return v.second->getId() ==
408             ↪ id; });
409     while (!vi->second->getInEdges().empty()) {
410         eraseEdge(vi->second->getInEdges().begin());
411     }
412     while (!vi->second->getOutEdges().empty()) {
413         eraseEdge(vi->second->getOutEdges().begin());
414     }
415     V.erase(vi);
416 }
417
418 string printForm() {
419     string ans = "";
420     stringstream ss;
421     ss << noboolalpha << weighted << " " << oriented << " " << multy
422         ↪ << " \n ";
423     ans += ss.str();
424     if (!V.empty()) {
425         ans += "vertexes: \n ";
426         for (map<long long, vertex*>::iterator i = V.begin(); i !=
427             ↪ V.end(); i++) {

```



```

423         ans += to_string(i->first) + ", \n ";
424     }
425     ans[ans.size() - 2] = '.';
426 }
427 if (!V.empty()) {
428     ans += "edges: \n ";
429     for (map<long long, edge*>::iterator i = E.begin(); i !=
430         ↪ E.end(); i++) {
431         if (!weighted) {
432             ans += to_string(i->second->getStart()->getId()) + " "
433                 ↪ + to_string(i->second->getEnd()->getId()) + ", \n ";
434         }
435         else {
436             ans += to_string(i->second->getStart()->getId()) + " "
437                 ↪ +
438                 to_string(i->second->getEnd()->getId()) + " " +
439                 ↪ to_string(i->second->getWeight()) + ", \n ";
440         }
441     }
442     ans[ans.size() - 2] = '.';
443 }
444 return ans;
445 }
446
447 void setVertexes(map<long long, vertex*> V) {
448     this->V = V;
449 }
450
451 map<long long, vertex*> getVertexes() {
452     return V;
453 }
454
455 void setEdges(map<long long, edge*> E) {
456     this->E = E;
457 }
458
459 map<long long, edge*> getEdges() {
460     return E;
461 }
462 };

```

2 Список смежности Ia(1)

2.1 Задание

Для каждой вершины орграфа вывести её степень.

2.2 Решение

```
1 #include <iostream>
2 #include "F:\5th semester\graphs\graph\graph\graph.h"
3 #include <fstream>
4 #include "F:\5th semester\graphs\graph\graph\graphInterface.h"
5 using namespace std;
6 int main() {
7     graph g;
8     g = graphInterface();
9     map<long long, vertex*> V = g.getVertexes();
10    for (map<long long, vertex*>::iterator i = V.begin(); i != V.end(); i++) {
11        if (g.isOriented()) {
12            cout << i->first << ": " << i->second->getInEdges().size() << ", " <<
13                << i->second->getOutEdges().size() << "\n ";
14        }
15        else {
16            cout << i->first << ": " << i->second->getInEdges().size() << "\n ";
17        }
18    }
```

2.3 Пример

```
1 open example1.txt
2 Result:
3 0 1 0
4 vertexes:
5 1,
6 2,
7 3,
8 4,
9 5,
10 6,
11 7.
12 edges:
```

```
13 1 2,  
14 2 3,  
15 1 4,  
16 7 2,  
17 5 7,  
18 6 4.  
  
19  
20 -----  
21 exit  
22 1: 0, 2  
23 2: 2, 1  
24 3: 1, 0  
25 4: 2, 0  
26 5: 0, 1  
27 6: 0, 1  
28 7: 1, 1
```

3 Список смежности Ia(2)

3.1 Задание

Определить, можно ли попасть из вершины u в вершину v через одну какую-либо вершину орграфа. Вывести такую вершину.

3.2 Решение

```
1 #include <iostream>
2 #include "F:\5th semester\graphs\graph\graph\graph.h"
3 #include <fstream>
4 #include "F:\5th semester\graphs\graph\graph\graphInterface.h"
5 using namespace std;
6 int main() {
7     graph g;
8     g = graphInterface();
9     long long un;
10    long long vn;
11    cin >> un >> vn;
12    vertex* u = g.getVertexes()[un];
13    map<long long, edge*> ti = u->getOutEdges();
14    for (map<long long, edge*>::iterator i = ti.begin(); i != ti.end(); i++) {
15        map<long long, edge*> tj = i->second->getEnd()->getOutEdges();
16        for (map<long long, edge*>::iterator j = tj.begin(); j != tj.end(); j++)
17            ↪ {
18                if (j->second->getEnd()->getId() == vn) {
19                    cout << i->second->getEnd()->getId();
20                    return 0;
21                }
22            }
23    cout << "There is no such way \n ";
24 }
```

3.3 Пример

```
1 open example1.txt
2 Result:
3 0 1 0
4 vertexes:
```

```
5 1,
6 2,
7 3,
8 4,
9 5,
10 6,
11 7.
12 edges:
13 1 2,
14 2 3,
15 1 4,
16 7 2,
17 5 7,
18 6 4.
19
20 -----
21 exit
22 1 3
23 2
```

4 Список смежности Iб: несколько графов

4.1 Задание

Построить граф, являющийся объединением двух заданных.

4.2 Решение

```
1 #include <iostream>
2 #include "F:\5th semester\graphs\graph\graph\graph.h"
3 #include <fstream>
4 #include "F:\5th semester\graphs\graph\graph\graphInterface.h"
5 using namespace std;
6 int main() {
7     graph g1;
8     graph g2;
9     g1 = graphInterface();
10    g2 = graphInterface();
11    if ((g1.isOriented() == g2.isOriented()) && (g1.isMulty() == g2.isMulty())
        ↪ && (g1.isWeighted() == g2.isWeighted())) {
12        graph g = graph();
13        map<long long, vertex*> tv = g1.getVertexes();
14        map<long long, vertex*> v = g2.getVertexes();
15        for (map<long long, vertex*>::iterator i = v.begin(); i != v.end(); i++)
            ↪ {
16            tv.insert(*i);
17        }
18        map<long long, edge*> te = g1.getEdges();
19        map<long long, edge*> e = g2.getEdges();
20        for (map<long long, edge*>::iterator i = e.begin(); i != e.end(); i++) {
21            te.insert(*i);
22        }
23        g.setType(g1.isWeighted(), g1.isOriented(), g1.isMulty());
24        g.setVertexes(tv);
25        g.setEdges(te);
26        cout << g.printForm();
27    }
28    else {
29        cout << "The graphs are of different types\n";
30        return 0;
31    }
32 }
```

4.3 Пример

```
1 open example1.txt
2 Result:
3 0 1 0
4 vertexes:
5 1,
6 2,
7 3,
8 4,
9 5,
10 6,
11 7.
12 edges:
13 1 2,
14 2 3,
15 1 4,
16 7 2,
17 5 7,
18 6 4.
19
20 -----
21 exit
22 open example1.txt
23 Result:
24 0 1 0
25 vertexes:
26 1,
27 2,
28 3,
29 4,
30 5,
31 6,
32 7.
33 edges:
34 1 2,
35 2 3,
36 1 4,
37 7 2,
38 5 7,
39 6 4.
```

```
40
41 -----
42 addedge 3 4
43 exit
44 0 1 0
45 vertexes:
46 1,
47 2,
48 3,
49 4,
50 5,
51 6,
52 7.
53 edges:
54 1 2,
55 2 3,
56 1 4,
57 7 2,
58 5 7,
59 6 4,
60 1 2,
61 2 3,
62 1 4,
63 7 2,
64 5 7,
65 6 4,
66 3 4.
```


5 Обходы графа II(1)

5.1 Задание

Проверить граф на ацикличность.

5.2 Решение

```
1 #include <iostream>
2 #include "F:\5th semester\graphs\graph\graph\graph.h"
3 #include <fstream>
4 #include <stack>
5 #include "F:\5th semester\graphs\graph\graph\graphInterface.h"
6 using namespace std;
7 bool acycle(graph g) {
8     map<long long, vertex*> V = g.getVertexes();
9     stack<long long> s = stack<long long>();
10    map<long long, long long> color = map<long long, long long>();
11    for (map<long long, vertex*>::iterator i = V.begin(); i != V.end(); i++) {
12        color.insert(make_pair(i->first, 0));
13    }
14    long long cc = 0;
15    while (true) {
16        map<long long, vertex*>::iterator b = V.begin();
17        map<long long, vertex*>::iterator e = V.end();
18        if (find_if(b, e, [color](pair<long long, vertex*> i) {return
19            ↪ color.at(i.second->getId()) == 0; }) == e) {
20            break;
21        }
22        if (s.empty()) {
23            s.push(find_if(b, e, [color](pair<long long, vertex*> i) {return
24                ↪ color.at(i.second->getId()) == 0; })->first);
25            cc++;
26        }
27        map<long long, edge*> t = V[s.top()]->getOutEdges();
28        long long tv = s.top();
29        color[tv] = cc;
30        s.pop();
31        for (map<long long, edge*>::iterator i = t.begin(); i != t.end(); i++) {
32            if (color[i->second->getEnd()->getId()] == cc) {
33                return false;
34            }
35        }
36    }
37}
```

```

33         s.push(i->second->getEnd()->getId());
34     }
35 }
36 return true;
37 }
38 int main() {
39     graph g;
40     g = graphInterface();
41     cout << acycle(g);
42 }

```

5.3 Пример

```

1 open example1.txt
2 Result:
3 0 1 0
4 vertexes:
5 1,
6 2,
7 3,
8 4,
9 5,
10 6,
11 7.
12 edges:
13 1 2,
14 2 3,
15 1 4,
16 7 2,
17 5 7,
18 6 4.
19
20 -----
21 exit
22 1

```

6 Обходы графа II(2)

6.1 Задание

Выяснить, является ли граф связным.

6.2 Решение

```
1 #include <iostream>
2 #include "F:\5th semester\graphs\graph\graph\graph.h"
3 #include <fstream>
4 #include <stack>
5 #include "F:\5th semester\graphs\graph\graph\graphInterface.h"
6 using namespace std;
7 long long parts(graph g) {
8     map<long long, vertex*> V = g.getVertexes();
9     stack<long long> s = stack<long long>();
10    map<long long, long long> color = map<long long, long long>();
11    for (map<long long, vertex*>::iterator i = V.begin(); i != V.end(); i++) {
12        color.insert(make_pair(i->first, 0));
13    }
14    long long cc = 0;
15    while (true) {
16        map<long long, vertex*>::iterator b = V.begin();
17        map<long long, vertex*>::iterator e = V.end();
18        if (find_if(b, e, [color](pair<long long, vertex*> i) {return
19            ↪ color.at(i.second->getId()) == 0; }) == e) {
20            break;
21        }
22        if (s.empty()) {
23            s.push(find_if(b, e, [color](pair<long long, vertex*> i) {return
24                ↪ color.at(i.second->getId()) == 0; })->first);
25            cc++;
26        }
27        map<long long, edge*> t = V[s.top()]->getOutEdges();
28        long long tv = s.top();
29        color[tv] = cc;
30        s.pop();
31        for (map<long long, edge*>::iterator i = t.begin(); i != t.end(); i++) {
32            if (color[i->second->getEnd()->getId()] == cc) {
33                continue;
34            }
35        }
```

```

33     s.push(i->second->getEnd()->getId());
34 }
35 t = V[tv]->getInEdges();
36 for (map<long long, edge*>::iterator i = t.begin(); i != t.end(); i++) {
37     if (color[i->second->getStart()->getId()] == cc) {
38         continue;
39     }
40     s.push(i->second->getStart()->getId());
41 }
42 }
43 return cc;
44 }
45 int main() {
46     graph g;
47     g = graphInterface();
48     cout << parts(g);
49 }

```

6.3 Пример

```

1 open example1.txt
2 Result:
3 0 1 0
4 vertexes:
5 1,
6 2,
7 3,
8 4,
9 5,
10 6,
11 7.
12 edges:
13 1 2,
14 2 3,
15 1 4,
16 7 2,
17 5 7,
18 6 4.
19
20 -----

```

```
21 exit
22 1
```

7 Каркас III

7.1 Задание

Дан взвешенный неориентированный граф из N вершин и M ребер. Требуется найти в нем каркас минимального веса.

7.2 Решение

```
1  #include <iostream>
2  #include "F:\5th semester\graphs\graph\graph\graph.h"
3  #include <fstream>
4  #include <stack>
5  #include <queue>
6  #include "F:\5th semester\graphs\graph\graph\graphInterface.h"
7  using namespace std;
8  graph prim(graph g) {
9      if (!(g.isWeighted() && !g.isMulty() && !g.isOriented())) {
10         return graph();
11     }
12     map<long long, vertex*> V = g.getVertexes();
13     map<long long, edge*> E = map<long long, edge*>();
14     map<long long, edge*> rest = map<long long, edge*>();
15     queue<long long> q = queue<long long>();
16     map<long long, long long> color = map<long long, long long>();
17     for (map<long long, vertex*>::iterator i = V.begin(); i != V.end(); i++) {
18         color.insert(make_pair(i->first, 0));
19     }
20     long long cc = 1;
21     while (true) {
22         map<long long, vertex*>::iterator b = V.begin();
23         map<long long, vertex*>::iterator e = V.end();
24         if (find_if(b, e, [color](pair<long long, vertex*> i) {return
25             ↪ color.at(i.second->getId()) == 0; }) == e) {
26             break;
27         }
28         if (q.empty()) {
29             q.push(find_if(b, e, [color](pair<long long, vertex*> i) {return
30                 ↪ color.at(i.second->getId()) == 0; })->first);
31             rest.clear();
32         }
33         map<long long, edge*> t1 = V[q.front()]->getOutEdges();
```

```

32     long long tv = q.front();
33     color[tv] = cc;
34     q.pop();
35     bool f = 0;
36     for (map<long long, edge*>::iterator i = t1.begin(); i != t1.end(); i++)
        ↪ {
37         if (color[i->second->getEnd()->getId()] != cc) {
38             rest.insert(*i);
39         }
40         f = f || color[i->second->getEnd()->getId()] != cc;
41         q.push(i->second->getEnd()->getId());
42     }
43     if (!f) {
44         break;
45     }
46     long long mn = LLONG_MAX;
47     long long mnId = 0;
48     for (map<long long, edge*>::iterator i = rest.begin(); i != rest.end();
        ↪ i++) {
49         if (i->second->getWeight() < mn) {
50             mn = i->second->getWeight();
51             mnId = i->first;
52         }
53     }
54     E.insert(make_pair(mnId, rest[mnId]));
55     rest.erase(mnId);
56 }
57 graph G = g;
58 G.setEdges(E);
59 return G;
60 }
61 int main() {
62     graph g;
63     g = graphInterface();
64     cout << prim(g).printForm();
65 }

```

7.3 Пример

```
1 open example3.txt
2 Result:
3 1 1 0
4 vertexes:
5 1,
6 2,
7 3,
8 4,
9 5,
10 6,
11 7.
12 edges:
13 1 2 3.000000,
14 2 3 4.000000,
15 3 1 0.000000.
16
17 -----
18 exit
19 0
```


8 Веса IV а

8.1 Задание

Определить множество вершин орграфа, расстояние от которых до заданной вершины не более N.

8.2 Решение

```
1 pair<map<long long, double>, map<long long, long long> > dijkstra(graph g,
  ↪ vertex v) {
2     map<long long, bool> used;
3     map<long long, double> d;
4     map<long long, long long> p;
5     auto t1 = g.getVertexes();
6     for (map<long long, vertex*>::iterator i = t1.begin(); i != t1.end(); i++)
  ↪ {
7         used.insert(make_pair(i->second->getId(), false));
8         d.insert(make_pair(i->second->getId(), 1e9));
9         p.insert(make_pair(i->second->getId(), -1));
10    }
11    vector<vector<edge> > ans = vector<vector<edge> >();
12    d[v.getId()] = 0;
13    set<long long> toProcess = set<long long>();
14    toProcess.insert(v.getId());
15    while (!toProcess.empty()) {
16        vertex cur;
17        long long curId;
18        long long mn = 1e18;
19        for (set<long long>::iterator i = toProcess.begin(); i !=
  ↪ toProcess.end(); i++) {
20            if (d[*i] < mn) {
21                mn = d[*i];
22                cur = *g.getVertexes()[*i];
23            }
24        }
25        curId = cur.getId();
26        toProcess.erase(curId);
27        if (used[curId]) {
28            continue;
29        }
30        used[curId] = true;
```

```

31     auto t2 = cur.getOutEdges();
32     for (map<long long, edge*>::iterator i = t2.begin(); i != t2.end(); i++)
        ↪ {
33         edge* e = i->second;
34         long long neighbourId = e->getEnd()->getId();
35         long long res = d[curId] + e->getWeight();
36         if (d[neighbourId] > res) {
37             d[neighbourId] = res;
38             p[neighbourId] = curId;
39         }
40         toProcess.insert(neighbourId);
41     }
42 }
43 return make_pair(d, p);
44 }
45
46 vector<list<edge> > dijkstraRecovery(graph g, map<long long, long long> p,
        ↪ vertex v) {
47     vector<list<edge> > ans = vector<list<edge> >();
48     map<long long, vertex*> V = g.getVertexes();
49     map<long long, edge*> E = g.getEdges();
50     long long vi = v.getId();
51     for (map<long long, vertex*>::iterator i = V.begin(); i != V.end(); i++) {
52         long long k = i->first;
53         list<edge> t = list<edge>();
54         while (k != vi) {
55             long long pk = p[k];
56             map<long long, edge*>::iterator ns = find_if(E.begin(), E.end(), [pk,
                ↪ k](pair<long long, edge*> e)
57                 {return e.second->getStart()->getId() == pk &&
                ↪ e.second->getEnd()->getId() == k; });
58             if (ns == E.end()) {
59                 t.clear();
60                 ans.push_back(t);
61                 break;
62             }
63             k = pk;
64             t.push_back(*(ns->second));
65         }
66         ans.push_back(t);
67     }

```

```
68     return ans;
69 }
```

8.3 Пример

```
1  open example6.txt
2  Result:
3  1 1 0
4  vertexes:
5  1,
6  2,
7  3,
8  4,
9  5,
10 6,
11 7.
12 edges:
13 1 2 100.000000,
14 1 3 100.000000,
15 2 3 50.000000,
16 2 4 50.000000,
17 3 4 100.000000,
18 3 5 50.000000,
19 4 5 75.000000,
20 4 6 75.000000,
21 5 6 125.000000.
22
23 -----
24 exit
25 151
26 2 4 5 6
```

9 Веса IV b

9.1 Задание

Вывести все кратчайшие пути из вершины u.

9.2 Решение

```
1 vector<list<edge> > dijkstraRecovery(graph g, map<long long, long long> p,
   ↪ vertex v) {
2     vector<list<edge> > ans = vector<list<edge> >();
3     map<long long, vertex*> V = g.getVertexes();
4     map<long long, edge*> E = g.getEdges();
5     long long vi = v.getId();
6     for (map<long long, vertex*>::iterator i = V.begin(); i != V.end(); i++) {
7         long long k = i->first;
8         list<edge> t = list<edge>();
9         while (k != vi) {
10            long long pk = p[k];
11            map<long long, edge*>::iterator ns = find_if(E.begin(), E.end(), [pk,
   ↪ k](pair<long long, edge*> e)
12                {return e.second->getStart()->getId() == pk &&
   ↪ e.second->getEnd()->getId() == k; });
13            if (ns == E.end()) {
14                t.clear();
15                ans.push_back(t);
16                break;
17            }
18            k = pk;
19            t.push_back(*(ns->second));
20        }
21        ans.push_back(t);
22    }
23    return ans;
24 }
25
26 pair<map<long long, double>, map<long long, long long> > fordBellman(graph g,
   ↪ vertex v) {
27     map<long long, double> d;
28     map<long long, long long> p;
29     auto t1 = g.getVertexes();
30     for (map<long long, vertex*>::iterator i = t1.begin(); i != t1.end(); i++)
   ↪ {
```

```

31     d.insert(make_pair(i->second->getId(), 1e18));
32     p.insert(make_pair(i->second->getId(), -1));
33 }
34 vector<vector<edge> > ans = vector<vector<edge> >();
35 d[v.getId()] = 0;
36 map<long long, edge*> E = g.getEdges();
37 for (int k = 0; k < g.getVertexes().size(); k++) {
38     for (map<long long, edge*>::iterator i = E.begin(); i != E.end(); i++) {
39         edge* e = i->second;
40         long long curId = e->getStart()->getId();
41         long long neighbourId = e->getEnd()->getId();
42         long long res = d[curId] + e->getWeight();
43         if (d[curId] < 1e17 && d[neighbourId] > res) {
44             d[neighbourId] = res;
45             p[neighbourId] = curId;
46         }
47     }
48 }
49 return make_pair(d, p);
50 }

```

9.3 Пример

```

1 open example6.txt
2 Result:
3 1 1 0
4 vertexes:
5 1,
6 2,
7 3,
8 4,
9 5,
10 6,
11 7.
12 edges:
13 1 2 100.000000,
14 1 3 100.000000,
15 2 3 50.000000,
16 2 4 50.000000,
17 3 4 100.000000,

```

```
18 3 5 50.000000,  
19 4 5 75.000000,  
20 4 6 75.000000,  
21 5 6 125.000000.  
22  
23 -----  
24 exit  
25 1 2  
26  
27 1 3
```

10 Веса IV с

10.1 Задание

Вывести кратчайший путь из вершины u до вершины v .

10.2 Решение

```
1 pair<map<long long, map<long long, double> >, map<long long, map<long long,
  ↪ long long> > > floydWarshall(graph g, vertex v) {
2   map<long long, map<long long, double> > d;
3   map<long long, map<long long, long long> > p;
4   auto V = g.getVertexes();
5   for (map<long long, vertex*>::iterator i = V.begin(); i != V.end(); i++) {
6     for (map<long long, vertex*>::iterator j = V.begin(); j != V.end(); j++)
7       ↪ {
8         d[i->second->getId()][j->second->getId()] = 1e18;
9         p[i->second->getId()][j->second->getId()] = -1;
10        if (i->second->getId() == j->second->getId()) {
11          d[i->second->getId()][j->second->getId()] = 0;
12        }
13      }
14    }
15    auto E = g.getEdges();
16    for (map<long long, edge*>::iterator i = E.begin(); i != E.end(); i++) {
17      d[i->second->getStart()->getId()][i->second->getEnd()->getId()] =
18        ↪ i->second->getWeight();
19      p[i->second->getStart()->getId()][i->second->getEnd()->getId()] =
20        ↪ i->second->getStart()->getId();
21    }
22    for (map<long long, vertex*>::iterator k = V.begin(); k != V.end(); k++) {
23      map<long long, map<long long, double> > newD = d;
24      for (map<long long, vertex*>::iterator i = V.begin(); i != V.end(); i++)
25        ↪ {
26          for (map<long long, vertex*>::iterator j = V.begin(); j != V.end();
27            ↪ j++) {
28            if (d[i->second->getId()][j->second->getId()] >
29              d[i->second->getId()][k->second->getId()] +
30              ↪ d[k->second->getId()][j->second->getId()])
31            {
32              newD[i->second->getId()][j->second->getId()] =
33                d[i->second->getId()][k->second->getId()] +
34                ↪ d[k->second->getId()][j->second->getId()];
35            }
36          }
37        }
38      }
39    }
```

```

28         p[i->second->getId()][j->second->getId()] = k->second->getId();
29     }
30     else
31     {
32         newD[i->second->getId()][j->second->getId()] =
33             ↪ d[i->second->getId()][j->second->getId()];
34     }
35 }
36 d = newD;
37 }
38 for (map<long long, edge*>::iterator i = E.begin(); i != E.end(); i++) {
39     p[i->second->getStart()->getId()][i->second->getEnd()->getId()] = -1;
40 }
41 return make_pair(d, p);
42 }
43
44 list<edge> floydWarshallRecovery(graph g, map<long long, map<long long, long
45     ↪ long> > p, vertex* v1, vertex* v2) {
46     vector<list<edge> > ans = vector<list<edge> >();
47     map<long long, vertex*> V = g.getVertexes();
48     map<long long, edge*> E = g.getEdges();
49     long long v1Id = v1->getId();
50     long long v2Id = v2->getId();
51     long long midId = (p[v1Id])[v2Id];
52     if (midId != -1) {
53         vertex* mid = V[midId];
54         list<edge> t1 = floydWarshallRecovery(g, p, v1, mid);
55         list<edge> t2 = floydWarshallRecovery(g, p, mid, v2);
56         int t1s = t1.size();
57         t1.insert(t1.end(), t2.begin(), t2.end());
58         if (t1s != 0 && t2.size() != 0) {
59             return t1;
60         }
61     }
62     else {
63         return list<edge>();
64     }
65 }
66 else {
67     map<long long, edge*>::iterator t = find_if(E.begin(), E.end(), [v1Id,
68         ↪ v2Id](pair<long long, edge*> x)

```



```

66         {return x.second->getStart()->getId() == v1Id &&
           ↪ x.second->getEnd()->getId() == v2Id; });
67     if (t != E.end()) {
68         return list<edge>(1, *(t->second));
69     }
70     else {
71         return list<edge>();
72     }
73 }
74 }

```

10.3 Пример

```

1 open example6.txt
2 Result:
3 1 1 0
4 vertexes:
5 1,
6 2,
7 3,
8 4,
9 5,
10 6,
11 7.
12 edges:
13 1 2 100.000000,
14 1 3 100.000000,
15 2 3 50.000000,
16 2 4 50.000000,
17 3 4 100.000000,
18 3 5 50.000000,
19 4 5 75.000000,
20 4 6 75.000000,
21 5 6 125.000000.
22
23 -----
24 exit
25 1 2
26 2 4
27 4 6

```

11 V Максимальный поток

11.1 Задание

Решить задачу на нахождение максимального потока любым алгоритмом.

11.2 Решение

```
1 list<list<pair<edge, double> > > dfs(graph g, vertex s, vertex e, vertex v,
  ↪ map<long long, bool> used, list<pair<edge, double> > path) {
2   if (v.getId() == e.getId()) {
3     return list<list<pair<edge, double> > >(1, path);
4   }
5   auto oe = v.getOutEdges();
6   if (oe.size() == 0) {
7     return list<list<pair<edge, double> > >();
8   }
9   used[v.getId()] = true;
10  list<list<pair<edge, double> > > ans = list<list<pair<edge, double> > >();
11  for (auto i = oe.begin(); i != oe.end(); i++) {
12    vertex* newV = i->second->getEnd();
13    if (!used[newV->getId()]) {
14      if (v.getId() != s.getId()) {
15        path.push_back(make_pair(*i->second, min(i->second->getWeight(),
  ↪ path.back().second)));
16      }
17      else {
18        path.push_back(make_pair(*i->second, i->second->getWeight()));
19      }
20      list<list<pair<edge, double> > > newPathes = dfs(g, s, e, *newV, used,
  ↪ path);
21      ans.insert(ans.end(), newPathes.begin(), newPathes.end());
22      path.pop_back();
23    }
24  }
25  return ans;
26 }
27
28 double fordFalkerson(graph g, vertex s, vertex e) {
29   map<long long, vertex*> V = g.getVertexes();
30   map<long long, edge*> E = g.getEdges();
31   map<long long, bool> used = map<long long, bool>();
```

```

32   for (auto i = V.begin(); i != V.end(); i++) {
33       used.insert(make_pair(i->first, false));
34   }
35   double ans = 0;
36   map<long long, double> c = map<long long, double>();
37   for (auto i = E.begin(); i != E.end(); i++) {
38       c.insert(make_pair(i->first, i->second->getWeight()));
39   }
40   list<list<pair<edge, double> > > pathes = dfs(g, s, e, s, used,
    ↪   list<pair<edge, double> >());
41   for (auto i = pathes.begin(); i != pathes.end(); i++) {
42       double f = i->back().second;
43       for (auto j = i->begin(); j != i->end(); j++) {
44           f = min(f, c[j->first.getId()]);
45       }
46       for (auto j = i->begin(); j != i->end(); j++) {
47           c[j->first.getId()] -= f;
48       }
49       ans += f;
50   }
51   return ans;
52 }

```

11.3 Пример

```

1  open example6.txt
2  Result:
3  1 1 0
4  vertexes:
5  1,
6  2,
7  3,
8  4,
9  5,
10 6,
11 7.
12 edges:
13 1 2 100.000000,
14 1 3 100.000000,
15 2 3 50.000000,

```

```
16 2 4 50.000000,  
17 3 4 100.000000,  
18 3 5 50.000000,  
19 4 5 75.000000,  
20 4 6 75.000000,  
21 5 6 125.000000.  
  
22  
23 -----  
24 exit  
25 200
```