

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования  
**«Дальневосточный федеральный университет»**

**ШКОЛА ЕСТЕСТВЕННЫХ НАУК  
  
Кафедра прикладной математики, механики, управления и программного обеспечения**

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2 ПО ДИСЦИПЛИНЕ  
«СТРУКТУРЫ И АЛГОРИТМЫ КОМПЬЮТЕРНОЙ ОБРАБОТКИ ДАННЫХ»**

Направление 02.03.03 «Математическое обеспечение и администрирование  
информационных систем»

Выполнил студент гр. Б8117-02.03.03  
Волков Владимир Денисович  
Проверил:  
Доцент, к.т.н С.Н.Остроухова

Владивосток 2019

**Неформальная постановка задачи**

Реализовать класс неориентированный граф на основе упорядоченного по возрастанию списка смежности, имеющий следующие методы:

1. Конструктор
2. Деструктор
3. Добавление вершины
4. Добавление ребра
5. Удаление вершины
6. Удаление ребра
7. Печать

**Структура класса и используемых структур:**

**struct** V {

int name;

V \*next;

adj \*E = nullptr;

};

**struct** adj {

V \*name;

adj \*next;

};

**class** Graf {

V \*H;

bool **isInV**(V \*Head, int name);

bool **isInAdj**(adj \*Head, int name);

V \***searchV**(V \*Head, int name);

**public**:

**Graf**();

**~Graf**();

int **addV**(int name);

int **addE**(int v1, int v2);

int **delE**(int v1, int v2);

int **delV**(int v1);

void **print**();

}

**Спецификация публичных методов класса**

**Graf() –** конструктор – инициализирует структуру графа – односвязный упорядоченный по возрастанию список.

**~Graf()**– деструктор – освобождает память, занятую элементами списка вершин и элементами списка ребер.

**int addV(int name) –** операция добавления вершины – в случае наличия вершины *name* в списке вершин возвращает 1, иначе ищет место вставки в список (*i-1.name < name < i+1.name*), добавляет вершину *name* в список и возвращает 0.

**int addE(int v1, int v2)** – операция добавление ребра – если нет хотя бы одной из вершин *v1* или *v2* в списке вершин графа, возвращает 1, если вершины *v1* и *v2* есть в списке вершин и в списках ребер *v1* и *v2* уже существует ребро *(v1,v2)* и *(v2,v1)* соответственно, возвращает 2, иначе ищет место вставки в список (*v1i-1.name.name < name < v1i+1.name.name*), добавляет ребро *name* в список ребер *v1*, аналогично для вершины *v2* и возвращает 0.

**int delE(int v1, int v2) –** операция удаления ребра – если нет хотя бы одной из вершин *v1* или *v2* в списке вершин графа, возвращает 1, если вершины *v1* и *v2* есть в списке вершин и в списках ребер *v1* и *v2* нет ребра *(v1,v2)* и *(v2,v1)* соответственно, возвращает 2, иначе ищет ребро *(v1,v2)* в списке ребер вершины *v1* и удаляет его (освобождая при этом память), аналогично для вершины *v2* и возвращает 0.

**int delV(int v1) –** операция удаления вершины – если вершины *v1* нет в списке вершин графа, возвращает 1, иначе для каждой вершины графа вызывает операцию удаления ребра **delE(“текучая вершина”, v1)**, ищет вершину *v1* в списке вершин графа, удаляет ее (освобождая при этом память) и возвращает 0.

**void print()** – операция печати – выводит граф в консоль.

**Тесты**

\*Сквозные тестовые ситуации.

|  |  |  |
| --- | --- | --- |
| Тестовая ситуация | Вход | Выход |
| 1. Добавление вершины в пустой список вершин | Graf graf1;  graf1.addV(2) == 1;  graf1.print(); | 2 : |
| 2. Добавление вершины в конец списка вершин | graf1.addV(4) == 1;  graf1.print(); | 2 :  4 : |
| 3. Добавление вершины в начало списка вершин | graf1.addV(1) == 1;  graf1.print(); | 1 :  2 :  4 : |
| 4. Добавление вершины в середину списка вершин | graf1.addV(3) == 1;  graf1.print(); | 1 :  2 :  3 :  4 : |
| 5. Добавление повторяющейся вершины | if (graf1.addV(2) == 1) {  cout << "Duplicate nodes" << endl;  }; | Duplicate nodes |
| 6. Добавление ребра в пустой список ребер | graf1.addE(1, 2);  graf1.print(); | 1 : 2  2 : 1  3 :  4 : |
| 7. Добавление ребра в конец списка ребер | graf1.addE(1, 4);  graf1.print(); | 1 : 2 4  2 : 1  3 :  4 : 1 |
| 8. Добавление ребра в начало списка ребер | graf1.addE(1, 1);  graf1.print(); | 1 : 1 2 4  2 : 1  3 :  4 : 1 |
| 9. Добавление ребра в середину списка ребер | graf1.addE(1, 3);  graf1.print(); | 1 : 1 2 3 4  2 : 1  3 : 1  4 : 1 |
| 10. Добавление ребра с несуществующей вершиной v2 | if (graf1.addE(1, 5) == 1) {  cout << "Haven't nodes v1 or v2" << endl;  }; | Haven't nodes v1 or v2 |
| 11. Добавление ребра с несуществующей вершиной v1 | if (graf1.addE(5, 1) == 1) {  cout << "Haven't nodes v1 or v2" << endl;  }; | Haven't nodes v1 or v2 |
| 12. Добавление ребра с несуществующими вершинами v1 и v2 | if (graf1.addE(5, 5) == 1) {  cout << "Haven't nodes v1 or v2" << endl;  }; | Haven't nodes v1 or v2 |
| 13. Добавление повторяющегося ребра | if (graf1.addE(1, 2) == 2) {  cout << "The edge (v1,v2) is already in the graph" << endl;  } | The edge (v1,v2) is already in the graph |
| 14. Удаление ребра из начала списка и не из начала | graf1.delE(1, 4);  graf1.print(); | 1 : 1 2 3  2 : 1  3 : 1  4 : |
| 15. Удаление ребра с несуществующей вершиной v2 | if (graf1.delE(1, 5) == 1) {  cout << "Haven't nodes v1 or v2" << endl;  } | Haven't nodes v1 or v2 |
| 16. Удаление ребра с несуществующей вершиной v1 | if (graf1.delE(5, 1) == 1) {  cout << "Haven't nodes v1 or v2" << endl;  } | Haven't nodes v1 or v2" |
| 17. Удаление несуществующего ребра | if (graf1.delE(1, 4) == 2) {  cout << "No edges (v1,v2) in the graph" << endl;  } | No edges (v1,v2) in the graph |
| 18. Удаление существующей вершины | graf1.delV(3);  graf1.print(); | 1 : 1 2  2 : 1  4 : |
| 19. Удаление не существующей вершины | if (graf1.delV(3) == 1) {  cout << "Haven't node v1" << endl;  } | Haven't node v1 |
| 20. Печать | graf1.print(); | 1 : 1 2  2 : 1  4 : |