|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_***ИУК «Информатика и управление»*\_\_**\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №4**

**«Представление графов и операции над ними»**

**ДИСЦИПЛИНА: «Дискретная математика»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-32Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_Зудин Д.В.\_\_\_\_\_)  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_Никитенко У.В.\_\_\_)  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |
| Калуга, 2022 г.  **Цель**: изучение основ теории графов, базовых понятий и определений; освоение компьютерных способов представления графов и операций над ними.  **Задачи**:  Составить программу, которая:   * позволяет задавать пользователю граф в любом их трех видов (список смежности, матрица смежности или матрица инцидентности) и получать на выходе любое другое представление; * по запросу пользователя выводит характеристики графа (число вершин, число ребер, степенную последовательность, степень выбранной вершины); * позволяет пользователю выполнять операции (объединение, пересечение, кольцевой суммы, соединение графов, …); * визуализация графа – по желанию.   **Вариант №4**  **Формулировка общего задания**  Граф *G1* задан матрицей смежности, граф *G2* – матрицей инцидентности, орграф *G3* – матрицей инцидентности. В матрицах инцидентности по строкам располагаются номера вершин. Вывод результатов понимается как вывод их в файл и/или на экран (в отчете также представить результат). Если выполнить операцию невозможно – в отчете необходимо обоснование.  *Задание 1.* Для заданных графов и орграфа вывести (определить):   1. число вершин и число ребер (дуг); 2. списки смежности; 3. степенные последовательности (для орграфа – полустепени захода и исхода каждой вершины); 4. матрицу инцидентности *G1*, матрицы смежности *G2* и *G3*.   *Задание 2.*   1. Добавить в граф *G1* новые вершины *v1*, *v2* и удалить из него вершину *v3*. Вывести результат; 2. Добавить в полученный на предыдущем шаге граф ребра *e1, e2, e3, e4, e5* и удалить ребра *e6, e7.* Вывести результат в виде матрицы инцидентности; 3. Построить дополнение полученного на шаге b) графа. Пусть это граф *G4* (вывести его в виде матрицы смежности); 4. Добавить в орграф *G3* вершины *v4, v5, v6* и удалить *v7*. Добавить дуги *f1, f2, f3,* удалить дугу *f4*. Вывести результат; 5. Выполнить операции предыдущего пункта в обратном порядке (сначала дуги, потом вершины). Вывести результат и сравнить с предыдущим.   **Формулировка индивидуального задания**    **Результат выполнения программы**  for G1  vertices count  10  edge count  10  adjacency lists  [3]  [4]  [6, 7]  [0, 9]  [1, 5]  [4, 7, 8]  [2, 7, 8]  [2, 5, 6]  [5, 6]  [3]  power sequences  (0, 1)  (0, 1)  (0, 2)  (1, 1)  (1, 1)  (1, 2)  (1, 2)  (3, 0)  (2, 0)  (1, 0)  for G2  vertices count  6  edge count  6  adjacency lists  [4, 5]  [2]  [1, 3, 5]  [2]  [0, 4]  [0, 2]  power sequences  (0, 2)  (0, 1)  (1, 2)  (1, 0)  (2, 0)  (2, 0)  for G3  vertices count  7  edge count  7  adjacency lists  [5]  [2, 3, 4, 6]  [1, 3]  [1, 2]  [1]  [0, 6]  [1, 5]  power sequences  (0, 1)  (0, 4)  (1, 1)  (2, 0)  (1, 0)  (1, 1)  (2, 0)  incidence matrix for G1  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]  [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]  [0, 0, 1, 1, 0, 0, 0, 0, 0, 0]  [1, 0, 0, 0, 1, 0, 0, 0, 0, 0]  [0, 1, 0, 0, 0, 1, 0, 0, 0, 0]  [0, 0, 0, 0, 0, 1, 1, 1, 0, 0]  [0, 0, 1, 0, 0, 0, 0, 0, 1, 1]  [0, 0, 0, 1, 0, 0, 1, 0, 1, 0]  [0, 0, 0, 0, 0, 0, 0, 1, 0, 1]  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  adjacency matrix for G2  [0, 0, 0, 0, 1, 1]  [0, 0, 1, 0, 0, 0]  [0, 1, 0, 1, 0, 1]  [0, 0, 1, 0, 0, 0]  [1, 0, 0, 0, 2, 0]  [1, 0, 1, 0, 0, 0]  adjacency matrix for G3  [0, 0, 0, 0, 0, 1, 0]  [0, 0, 1, 1, 1, 0, 1]  [0, 1, 0, 1, 0, 0, 0]  [0, 1, 1, 0, 0, 0, 0]  [0, 1, 0, 0, 0, 0, 0]  [1, 0, 0, 0, 0, 0, 1]  [0, 1, 0, 0, 0, 1, 0]  task\_2  new G1  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  [0, 0, 0, 0, 0, 0, 1, 1, 0, 0]  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1]  [0, 1, 0, 0, 0, 1, 0, 0, 0, 0]  [0, 0, 0, 0, 1, 0, 0, 1, 1, 0]  [0, 0, 1, 0, 0, 0, 0, 1, 1, 0]  [0, 0, 1, 0, 0, 1, 1, 0, 0, 0]  [0, 0, 0, 0, 0, 1, 1, 0, 0, 0]  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]  new G1  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]  [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]  [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]  [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0]  [0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]  [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]  [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0]  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  G4  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]  [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]  [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  [0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]  [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1]  [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0]  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  [1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0]  new G3  [1, 1, 0, 0, 0, 0, 0, 0]  [0, 0, 1, 1, 1, 0, 0, 0]  [0, 0, 1, 0, 0, 1, 1, 0]  [-1, 0, 0, 1, 0, 1, 0, 0]  [0, 1, 0, 0, 0, 0, -1, 1]  [0, 0, 0, 0, 0, 0, 0, 0]  [0, 0, 0, 0, 1, 0, 0, 1]  [0, 0, 0, 0, 0, 0, 0, 0]  [0, 0, 0, 0, 0, 0, 0, 0]  new G3  [1, 0, 0]  [0, 1, 1]  [0, 1, 0]  [0, 0, 1]  [0, 0, 0]  [1, 0, 0]  [0, 0, 0]  **Листинг программы**  *class* Graph:  *def \_\_init\_\_*(*self*):  *self*.gr = []   *def* from\_sm(*self*, i):  *self*.gr = i   *def* from\_in(*self*, i):   n = len(i)  *self*.gr = [[0 *for* \_ *in* range(n)] *for* \_ *in* range(n)]  *for* j *in* range(len(i[0])):  ver = []  ver\_id = []  *for* k *in* range(len(i)):  *if* i[k][j] != 0:  ver.append(i[k][j])  ver\_id.append(k)   *if* len(ver) == 1:  *if* ver[0] != 2:  *raise* ValueError("wrong edge nums")  *self*.gr[ver\_id[0]][ver\_id[0]] = 2   *elif* len(ver) == 2:  *if* ver[0] < ver[1]:  ver[0], ver[1] = ver[1], ver[0]  ver\_id[0], ver\_id[1] = ver\_id[1], ver\_id[0]   *if* ver[0] == 1 *and* ver[1] == -1:  *# print( ver\_id[0], ver\_id[1], n )  self*.gr[ver\_id[0]][ver\_id[1]] = 1  *elif* ver[0] == 1 *and* ver[1] == 1:  *self*.gr[ver\_id[0]][ver\_id[1]] = 1  *self*.gr[ver\_id[1]][ver\_id[0]] = 1  *else*:  *raise* ValueError("wrong edge nums")   *else*:  *raise* ValueError("wrong len of ver")   *def* to\_sm(*self*):  *return self*.gr   *def* to\_in(*self*):   n = len(*self*.gr)  inc = []  *for* i *in* range(n):  *for* j *in* range(n - i - 1):  *if self*.gr[i][1 + i + j] != 0 *or self*.gr[1 + i + j][i] != 0:  inc.append([0 *for* \_ *in* range(n)])  *if self*.gr[i][1 + i + j] == 0 *and self*.gr[1 + i + j][i] == 1:  inc[-1][i] = -1  inc[-1][1 + i + j] = 1  *elif self*.gr[i][1 + i + j] == 1 *and self*.gr[1 + i + j][i] == 0:  inc[-1][i] = 1  inc[-1][1 + i + j] = -1  *elif self*.gr[i][1 + i + j] == 1 *and self*.gr[1 + i + j][i] == 1:  inc[-1][i] = 1  inc[-1][1 + i + j] = 1   *for* i *in* range(n):  *if self*.gr[i][i] == 2:  inc.append([0 *for* \_ *in* range(n)])  inc[-1][i] = 2   inc\_t = [[inc[i][j] *for* i *in* range(len(inc))] *for* j *in* range(len(inc[0]))]   *return* inc\_t   *def* ver\_count(*self*):  *return* len(*self*.gr)   *def* edg\_count(*self*):  *return* len(*self*.to\_in()[0])   *def* sm\_list(*self*):  lines = []  *for* i *in* range(len(*self*.gr)):  l = []  *for* j *in* range(len(*self*.gr[0])):  *if self*.gr[i][j] != 0:  l.append(j)  lines.append(l)  *return* lines   *def* sm\_list\_r(*self*):  lines\_in = []  lines\_out = []  *for* i *in* range(len(*self*.gr)):  li = []  lo = []  *for* j *in* range(len(*self*.gr[0])):  *if self*.gr[i][j] != 0:  *if* i < j:  lo.append(j)  *else*:  li.append(j)  lines\_in.append(li)  lines\_out.append(lo)  *return* lines\_in, lines\_out   *def* is\_ograph(*self*):  n = len(*self*.gr)  *for* i *in* range(n):  *for* j *in* range(n - i - 1):  *if self*.gr[i][1 + i + j] != 0 *and self*.gr[1 + i + j][i] != 0:  *if self*.gr[i][1 + i + j] != *self*.gr[1 + i + j][i]:  *return False  return True   def* step\_posl(*self*):  *if self*.is\_ograph():  *return self*.polu\_step\_posl()  l = *self*.sm\_list()  c = [len(i) *for* i *in* l]  *return* c   *def* polu\_step\_posl(*self*):  li, lo = *self*.sm\_list\_r()  c1 = [len(i) *for* i *in* li]  c2 = [len(i) *for* i *in* lo]  c = [(c1[i], c2[i]) *for* i *in* range(len(c1))]  *return* c   *def* del\_ver(*self*, i):  i -= 1  *for* line *in self*.gr:  line.pop(i)  *self*.gr.pop(i)   *def* add\_ver(*self*, i):  n = len(*self*.gr)  i -= 1  new\_l = [0 *for* \_ *in* range(n + 1)]  *for* line *in self*.gr:  line.insert(i, 0)  *self*.gr.insert(i, new\_l)   *def* del\_edg(*self*, e):  a, b = e  a -= 1  b -= 1  *if self*.gr[a][b] == *self*.gr[b][a] == 1:  *self*.gr[a][b] = 0  *self*.gr[b][a] = 0  *if self*.gr[a][b] == 1:  *self*.gr[a][b] = 0   *def* add\_edg(*self*, e, pos=*False*):  a, b = e  a -= 1  b -= 1  *self*.gr[a][b] = 1  *if not* pos:  *self*.gr[b][a] = 1   *def* dop(*self*):  n = len(*self*.gr)  g = [[0 *for* \_ *in* range(n)] *for* \_ *in* range(n)]   *for* i *in* range(n):  *for* j *in* range(n):  *if self*.gr[i][j] != 0:  g[i][j] = 0  *else*:  g[i][j] = 1  *if* i == j:  g[i][j] = 2  *return* g   *def* out(*self*):  print\_matr(*self*.gr)   *def* print\_matr(m):  *for* l *in* m:  print(l)   *def* task\_1(g):  print("vertices count")  print(g.ver\_count())  print("edge count")  print(g.edg\_count())  print("adjacency lists")  print\_matr(g.sm\_list())  print("power sequences")  print\_matr(g.step\_posl())   *if* \_\_name\_\_ == "\_\_main\_\_":  G\_1 = [[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 0, 0, 1, 1, 0, 0],  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],  [0, 1, 0, 0, 0, 1, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 1, 1, 0],  [0, 0, 1, 0, 0, 0, 0, 1, 1, 0],  [0, 0, 1, 0, 0, 1, 1, 0, 0, 0],  [0, 0, 0, 0, 0, 1, 1, 0, 0, 0],  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0], ]  G\_2 = [[1, 1, 0, 0, 0, 0],  [0, 0, 1, 0, 0, 0],  [0, 0, 1, 1, 0, 1],  [0, 0, 0, 1, 0, 0],  [0, 1, 0, 0, 2, 0],  [1, 0, 0, 0, 0, 1], ]  G\_3 = [[1, 0, 0, 0, 0, 0, 0],  [0, 1, 1, 1, 0, 1, 0],  [0, 0, 0, 1, 1, 0, 0],  [0, 0, 0, 0, 1, 1, 0],  [0, 0, 1, 0, 0, 0, 0],  [1, 0, 0, 0, 0, 0, 1],  [0, 1, 0, 0, 0, 0, 1], ]   v\_i = [11, 12, 8, 8, 9, 7, 5]  e\_i = [(10, 11), (2, 11), (1, 11), (7, 11), (10, 11), (2, 5), (4, 10)]  f\_i = [(1, 4), (3, 4), (3, 5), (1, 7)]   G1 = Graph()  G1.from\_sm(G\_1)  G2 = Graph()  G2.from\_in(G\_2)  G3 = Graph()  G3.from\_in(G\_3)   *# G2.out()* print("for G1")  task\_1(G1)   print("for G2")  task\_1(G2)   print("for G3")  task\_1(G3)   print("incidence matrix for G1")  print\_matr(G1.to\_in())  print("adjacency matrix for G2")  print\_matr(G2.to\_sm())  print("adjacency matrix for G3")  print\_matr(G3.to\_sm())   print("task\_2")   print("new G1")  print\_matr(G1.to\_sm())   G1.add\_ver(v\_i[0])  G1.add\_ver(v\_i[1])  G1.del\_ver(v\_i[2])   print("new G1")  print\_matr(G1.to\_sm())   G1.add\_edg(e\_i[0])  G1.add\_edg(e\_i[1])  G1.add\_edg(e\_i[2])  G1.add\_edg(e\_i[3])  G1.add\_edg(e\_i[4])  G1.del\_edg(e\_i[5])  G1.del\_edg(e\_i[6])   print("G4")  print\_matr(G1.to\_sm())   G3.add\_ver(v\_i[3])  G3.add\_ver(v\_i[4])  G3.add\_ver(v\_i[5])  G3.del\_ver(v\_i[6])   G3.add\_edg(f\_i[0], *True*)  G3.add\_edg(f\_i[1], *True*)  G3.add\_edg(f\_i[2], *True*)  G3.del\_edg(f\_i[3])   print("new G3")  print\_matr(G3.to\_in())   G3.add\_edg(f\_i[3], *True*)  G3.del\_edg(f\_i[2])  G3.del\_edg(f\_i[1])  G3.del\_edg(f\_i[0])   G3.add\_ver(v\_i[6])  G3.del\_ver(v\_i[3])  G3.del\_ver(v\_i[4])  G3.del\_ver(v\_i[5])   print("new G3")  print\_matr(G3.to\_in())  **Выводы:**  В ходе работы были изучены основы теории графов, базовые понятия и определения; освоены компьютерные способы представления графов и операции над ними. | | |