**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования «Рязанский государственный радиотехнический университет имени В.Ф. Уткина»

Кафедра «САПР»

Структуры и алгоритмы обработки данных

**Отчет о лабораторной работе № 2**

по теме

«Графы в прикладных алгоритмах»

**Выполнили:**

Студент гр. 045

Анохин В.А.

Вашкулатов Н.А.

**Проверили:**

Проф.Скворцов С.В

**Цель работы:** изучение и анализ структур данных, применяемых для описания графов в прикладных программах; анализ и программная реализация алгоритмов систематического обхода вершин графа; разработка и исследование прикладных программ, в которых применяются неориентированные и ориентированные графы.

**Ход работы:**

**Вариант задания:** представление графов в виде списков смежности.

На рисунке 1 показана схема алгоритма создания списка смежности.

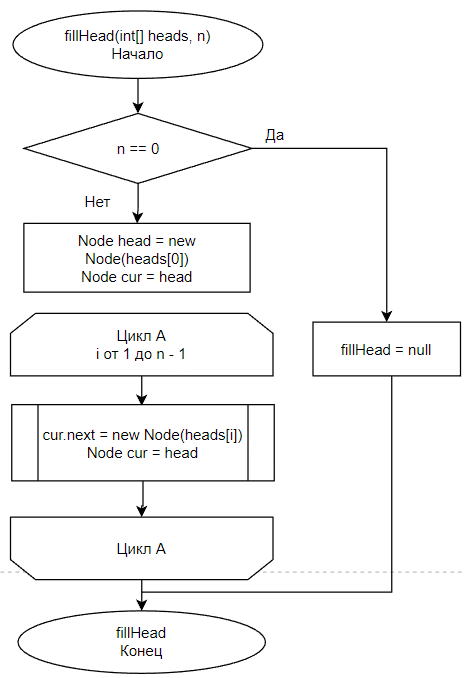


Рисунок 1 – Создание списка смежности

**Код программы:**

**class** Node {  
 **int num**;  
 Node **next**;

Node(**int** num) {  
 **this**.**num** = num;  
 **this**.**next** = **null**;  
 }  
 @Override  
 **public** String toString() {  
 **return num** + **"->"** + **next**;  
 }  
}

**public static** Node fillHead(**int**... heads) {  
 **if** (heads.**length** == 0) **return null**;  
 Node head = **new** Node(heads[0]);  
 Node cur = head;  
 **for** (**int** i = 1; i < heads.**length**; i++) {  
 cur.**next** = **new** Node(heads[i]);  
 cur = cur.**next**;  
 }  
 **return** head;  
}

**public static void** printGraph(Node[] heads) {  
 **for** (**int** i = 0; i < heads.**length**; i++) {  
 System.***out***.println(i + **": "** + heads[i]);  
 }  
}

Для алгоритма поиска в ширину используется очередь. На рисунке 2 показаны схемы алгоритмов для добавления элемента в очередь, извлечения элемента и проверки на пустую очередь.

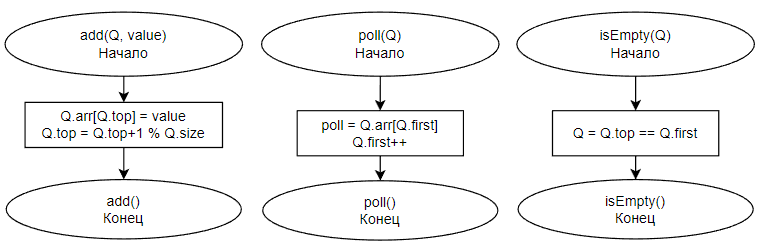


Рисунок 2 – Добавление, извлечение элемента и

проверка на пустую очередь

**Код программы:**

**class** Queue {  
 **int size**;  
 **int top** = 0;  
 **int first** = 0;  
 **int**[] **arr**;  
  
 **public** Queue(**int** size) {  
 **this**.**size** = size;  
 **arr** = **new int**[size];  
 }  
  
 **public void** add(**int** value) {  
 **arr**[**top**] = value;  
 **top**= **top**+1 % **size**;  
 }  
 **public boolean** isEmpty() {  
 **return top** == **first**;  
 }  
  
 **public int** poll() {  
 **return arr**[**first**++];  
 }  
}

Алгоритмы поиска в глубину и ширину показаны на рисунках 3-4.

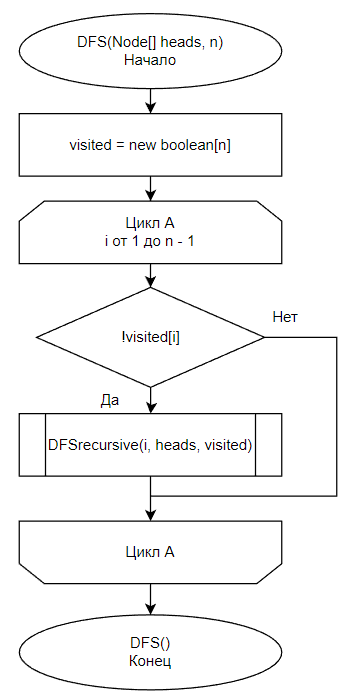
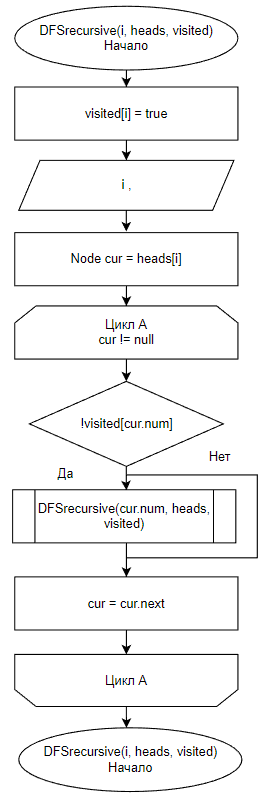
 

Рисунок 3 – Схема алгоритма поиска в глубину

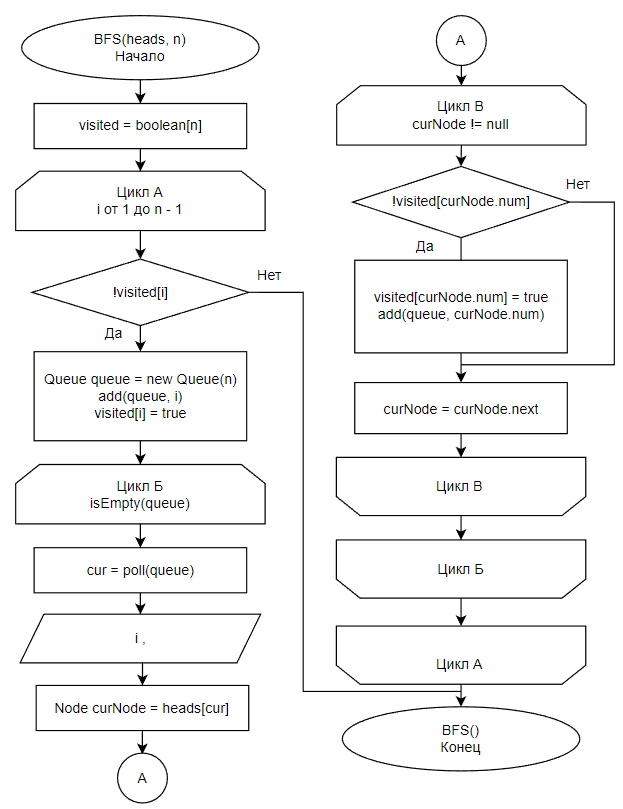


Рисунок 4 – Схема алгоритма поиска в ширину

**Код программы:**

**public static void** DFS(Node[] heads) {  
 **boolean**[] visited = **new boolean**[heads.**length**];  
 **for** (**int** i = 0; i < heads.**length**; i++) {  
 *DFSrecursive*(i, heads, visited);  
 }  
}

**public static void** DFSrecursive(**int** i, Node[] heads, **boolean**[] visited) {  
 **if** (!visited[i]) {  
 visited[i] = **true**;  
 System.***out***.print(i + **", "**);  
 Node cur = heads[i];  
 **while** (cur != **null**) {  
 **if** (!visited[cur.**num**]) {  
 *DFSrecursive*(cur.**num**, heads, visited);  
 }  
 cur = cur.**next**;  
 }  
 }  
}

**public static void** BFS(Node[] heads) {  
 **boolean**[] visited = **new boolean**[heads.**length**];  
 **for** (**int** i = 0; i < heads.**length**; i++) {  
 **if** (!visited[i]) {  
 Queue queue = **new** Queue(heads.**length**);  
 queue.add(i);  
 visited[i] = **true**;  
 **while** (!queue.isEmpty()) {  
 **int** cur = queue.poll();  
 System.***out***.print(cur + **", "**);  
 Node curNode = heads[cur];  
 **while** (curNode != **null**) {  
 **if** (!visited[curNode.**num**]) {  
 visited[curNode.**num**] = **true**;  
 queue.add(curNode.**num**);  
 }  
 curNode = curNode.**next**;  
 }  
 }  
 }  
 }  
}

Используем граф на рисунке 5 для проверки работы программы. При поиске в глубину вершины будут проверяться в порядке: 0, 1, 2, 3, 4, 5, 6.

При поиске в ширину: 0, 1, 2, 4, 3, 5, 6.

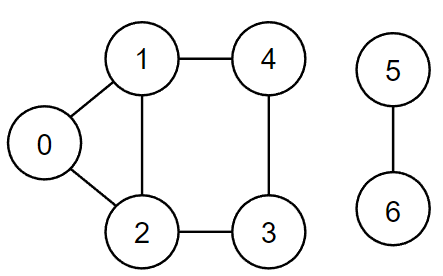
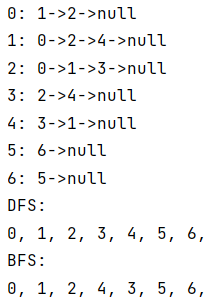


Рисунок 5 – Пример графа

**Код программы:**

Node[] notOriented = **new** Node[7];  
  
notOriented[0] = *fillHead*(1, 2);  
notOriented[1] = *fillHead*(0, 2, 4);  
notOriented[2] = *fillHead*(0, 1, 3);  
notOriented[3] = *fillHead*(2, 4);  
notOriented[4] = *fillHead*(3, 1);  
notOriented[5] = *fillHead*(6);  
notOriented[6] = *fillHead*(5);  
  
*printGraph*(notOriented);  
  
System.***out***.println(**"DFS: "**);  
*DFS*(notOriented);  
System.***out***.println(**"\nBFS:"**);  
*BFS*(notOriented);

**Окно вывода:**



При нахождение связных компонент неориентированного графа, используем граф на рисунке 5. Вершины разделятся на 2 группы: 0, 1, 2, 3, 4 и 5, 6.

**Код программы:**

**public static int**[] linkingByDFS(Node[] heads) {  
 **boolean**[] visited = **new boolean**[heads.**length**];  
 **int**[] comp = **new int**[heads.**length**];  
 **int** num = 0;  
 **for** (**int** i = 0; i < heads.**length**; i++) {  
 **if** (!visited[i]) {  
 num++;  
 *DFSrecursiveLinking*(i, heads, visited, comp, num);  
 }  
 }  
 **return** comp;  
}

**Окно вывода:**



На рисунке 6 показан граф, который используется для алгоритма топологической сортировки вершин ориентированного бесконтурного графа. Слева показан граф до начала сортировки, а справа – после.

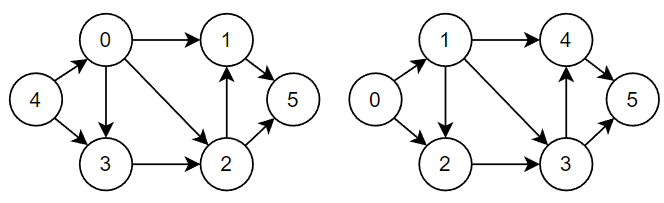


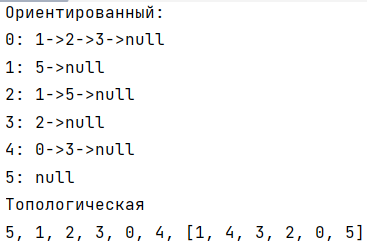
Рисунок 6 – Ориентированный граф

**Код программы:**

**public static int**[] topologicalSort(Node[] heads) {  
 **boolean**[] visited = **new boolean**[heads.**length**];  
 **int**[] labels = **new int**[heads.**length**];  
 **int** num = heads.**length**;  
 **for** (**int** i = 0; i < heads.**length**; i++) {  
 **if** (!visited[i]) {  
 num = *DFStopologicalSort*(i, heads, visited, labels, num);  
 }  
 }  
 **return** labels;  
}

**private static int** DFStopologicalSort(**int** i, Node[] heads, **boolean**[] visited, **int**[] labels, **int** num) {  
 visited[i] = **true**;  
 Node cur = heads[i];  
 **while** (cur != **null**) {  
 **if** (!visited[cur.**num**]) {  
 num = *DFStopologicalSort*(cur.**num**, heads, visited, labels, num);  
 }  
 cur = cur.**next**;  
 }  
 num--;  
 labels[i] = num;  
 System.***out***.print(i + **", "**);  
 **return** num;  
}

**Окно вывода:**



Модифицированный текст программы топологической сортировки вершин произвольного ориентированного графа

**Код программы:**

**public static int**[] topologicalModifiedSort(Node[] heads) {  
 **boolean**[] visited = **new boolean**[heads.**length**];  
 **int**[] labels = **new int**[heads.**length**];  
 **int** num = heads.**length**;  
 **for** (**int** i = 0; i < heads.**length**; i++) {  
 **if** (!visited[i]) {  
 num = *DFStopologicalModifiedSort*(i, heads, visited, labels, num);  
 }  
 }  
 **return** labels;  
}

**private static int** DFStopologicalModifiedSort(**int** i, Node[] heads, **boolean**[] visited, **int**[] labels, **int** num) {  
 visited[i] = **true**;  
 Node cur = heads[i];  
 **while** (cur != **null**) {  
 **if** (!visited[cur.**num**]) {  
 num = *DFStopologicalModifiedSort*(cur.**num**, heads, visited, labels, num);  
 } **else** {  
 **if**(labels[cur.**num**] == 0) {  
 **throw new** RuntimeException(**"Контур через вершины: "** + cur.**num** + **" и "** + i);  
 }  
 }  
 cur = cur.**next**;  
 }  
 num--;  
 labels[i] = num;  
 System.***out***.print(i + **", "**);  
 **return** num;  
}

**Окно вывода:**



Для замкнутого графа, как на рисунке 7, топологическая сортировка невозможна.

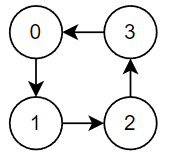


Рисунок 7 – Граф с контуром

**Вывод:** мы изучили и провели анализ структур данных, применяемых для описания графов в прикладных программах; анализ и программная реализация алгоритмов систематического обхода вершин графа; разработка и исследование прикладных программ, в которых применяются неориентированные и ориентированные графы.