**Лабораторная работа №3**

**«Графы»**

**Задание №1.**

Измените программу bfs.java (см. листинг 13.2) так, чтобы для поиска минимального остовного дерева применялся алгоритм обхода в ширину вместо обхода в глубину из программы mst.java (см. листинг 13.3). Создайте в методе main() граф с 9 вершинами и 12 ребрами, постройте его минимальное остовное дерево.

Листинг 13.2. Программа bfs.java

*// bfs.java*

*// Обход в ширину*

*// Запуск программы: C>java BFSApp*

*////////////////////////////////////////////////////////////////*

**class** Queue

{

**private** **final** **int** SIZE = 20;

**private** **int**[] queArray;

**private** **int** front;

**private** **int** rear;

*// -------------------------------------------------------------*

**public** Queue() *// Конструктор*

{

queArray = **new** **int**[SIZE];

front = 0;

rear = -1;

}

*// -------------------------------------------------------------*

**public** **void** insert(**int** j) *// Вставка элемента в конец очереди*

{

**if**(rear == SIZE-1)

rear = -1;

queArray[++rear] = j;

}

*// -------------------------------------------------------------*

**public** **int** remove() *// Извлечение элемента в начале очереди*

{

**int** temp = queArray[front++];

**if**(front == SIZE)

front = 0;

**return** temp;

}

*// -------------------------------------------------------------*

**public** **boolean** isEmpty() *// true, если очередь пуста*

{

**return** ( rear+1==front || (front+SIZE-1==rear) );

}

*// -------------------------------------------------------------*

} *// Конец класса Queue*

*////////////////////////////////////////////////////////////////*

**class** Vertex

{

**public** **char** label; *// Метка (например, 'A')*

**public** **boolean** wasVisited;

*// -------------------------------------------------------------*

**public** Vertex(**char** lab) *// Конструктор*

{

label = lab;

wasVisited = **false**;

}

*// -------------------------------------------------------------*

} *// Конец класса Vertex*

*////////////////////////////////////////////////////////////////*

**class** Graph

{

**private** **final** **int** MAX\_VERTS = 20;

**private** Vertex vertexList[]; *// Список вершин*

**private** **int** adjMat[][]; *// Матрица смежности*

**private** **int** nVerts; *// Текущее количество вершин*

**private** Queue theQueue;

*// ------------------*

**public** Graph() *// Конструктор*

{

vertexList = **new** Vertex[MAX\_VERTS];

*// Матрица смежности*

adjMat = **new** **int**[MAX\_VERTS][MAX\_VERTS];

nVerts = 0;

**for**(**int** j=0; j<MAX\_VERTS; j++) *// Матрица смежности*

**for**(**int** k=0; k<MAX\_VERTS; k++) *// заполняется нулями*

adjMat[j][k] = 0;

theQueue = **new** Queue();

}

*// -------------------------------------------------------------*

**public** **void** addVertex(**char** lab)

{

vertexList[nVerts++] = **new** Vertex(lab);

}

*// -------------------------------------------------------------*

**public** **void** addEdge(**int** start, **int** end)

{

adjMat[start][end] = 1;

adjMat[end][start] = 1;

}

*// -------------------------------------------------------------*

**public** **void** displayVertex(**int** v)

{

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(vertexList[v].label);

}

*// -------------------------------------------------------------*

**public** **void** bfs() *// Обход в ширину*

{ *// Алгоритм начинает с вершины 0*

vertexList[0].wasVisited = **true**; *// Пометка*

displayVertex(0); *// Вывод*

theQueue.insert(0); *// Вставка в конец очереди*

**int** v2;

**while**( !theQueue.isEmpty() ) *// Пока очередь не опустеет*

{

**int** v1 = theQueue.remove(); *// Извлечение вершины в начале очереди*

*// Пока остаются непосещенные соседи*

**while**( (v2=getAdjUnvisitedVertex(v1)) != -1 )

{ *// Получение вершины*

vertexList[v2].wasVisited = **true**; *// Пометка*

displayVertex(v2); *// Вывод*

theQueue.insert(v2); *// Вставка*

}

}

*// Очередь пуста, обход закончен*

**for**(**int** j=0; j<nVerts; j++) *// Сброс флагов*

vertexList[j].wasVisited = **false**;

}

*// -------------------------------------------------------------*

*// Метод возвращает непосещенную вершину, смежную по отношению к v*

**public** **int** getAdjUnvisitedVertex(**int** v)

{

**for**(**int** j=0; j<nVerts; j++)

**if**(adjMat[v][j]==1 && vertexList[j].wasVisited==**false**)

**return** j; *// Возвращает первую найденную вершину*

**return** -1; *// Таких вершин нет*

}

*// -------------------------------------------------------------*

} *// Конец класса Graph*

*////////////////////////////////////////////////////////////////*

**class** BFSApp

{

**public** **static** **void** main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args)

{

Graph theGraph = **new** Graph();

theGraph.addVertex('A'); *// 0 (исходная вершина)*

theGraph.addVertex('B'); *// 1*

theGraph.addVertex('C'); *// 2*

theGraph.addVertex('D'); *// 3*

theGraph.addVertex('E'); *// 4*

theGraph.addEdge(0, 1); *// AB*

theGraph.addEdge(1, 2); *// BC*

theGraph.addEdge(0, 3); *// AD*

theGraph.addEdge(3, 4); *// DE*

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("Visits: ");

theGraph.bfs(); *// Обход в ширину*

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println();

}

}

*////////////////////////////////////////////////////////////////*

Листинг 13.3. Программа mst.java

*// mst.java*

*// Построение минимального остовного дерева*

*// Запуск программы: C>java MSTApp*

*////////////////////////////////////////////////////////////////*

**class** StackX

{

**private** **final** **int** SIZE = 20;

**private** **int**[] st;

**private** **int** top;

*// -------------------------------------------------------------*

**public** StackX() *// Конструктор*

{

st = **new** **int**[SIZE]; *// Создание массива*

top = -1;

}

*// -------------------------------------------------------------*

**public** **void** push(**int** j) *// Размещение элемента в стеке*

{ st[++top] = j; }

*// -------------------------------------------------------------*

**public** **int** pop() *// Извлечение элемента из стека*

{ **return** st[top--]; }

*// -------------------------------------------------------------*

**public** **int** peek() *// Чтение с вершины стека*

{ **return** st[top]; }

*// -------------------------------------------------------------*

**public** **boolean** isEmpty() *// true, если стек пуст*

{ **return** (top == -1); }

*// -------------------------------------------------------------*

} *// Конец класса StackX*

*////////////////////////////////////////////////////////////////*

**class** Vertex

{

**public** **char** label; *// Метка (например, 'A')*

**public** **boolean** wasVisited;

*// -------------------------------------------------------------*

**public** Vertex(**char** lab) *// Конструктор*

{

label = lab;

wasVisited = **false**;

}

*// -------------------------------------------------------------*

} *// Конец класса Vertex*

*////////////////////////////////////////////////////////////////*

**class** Graph

{

**private** **final** **int** MAX\_VERTS = 20;

**private** Vertex vertexList[]; *// Список вершин*

**private** **int** adjMat[][]; *// Матрица смежности*

**private** **int** nVerts; *// Текущее количество вершин*

**private** StackX theStack;

*// -------------------------------------------------------------*

**public** Graph() *// Конструктор*

{

vertexList = **new** Vertex[MAX\_VERTS];

*// Матрица смежности*

adjMat = **new** **int**[MAX\_VERTS][MAX\_VERTS];

nVerts = 0;

**for**(**int** j=0; j<MAX\_VERTS; j++) *// Матрица смежности*

**for**(**int** k=0; k<MAX\_VERTS; k++) *// заполняется нулями*

adjMat[j][k] = 0;

theStack = **new** StackX();

}

*// -------------------------------------------------------------*

**public** **void** addEdge(**int** start, **int** end)

{

adjMat[start][end] = 1;

adjMat[end][start] = 1;

}

*// -------------------------------------------------------------*

**public** **void** displayVertex(**int** v)

{

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(vertexList[v].label);

}

*// -------------------------------------------------------------*

**public** **void** mst() *// Построение минимального остовного дерева*

{ *//*

vertexList[0].wasVisited = **true**; *// Пометка*

theStack.push(0); *// Занесение в стек*

**while**( !theStack.isEmpty() ) *// Пока стек не опустеет*

{ *// Извлечение элемента из стека*

**int** currentVertex = theStack.peek();

*// Получение следующего соседа*

**int** v = getAdjUnvisitedVertex(currentVertex);

**if**(v == -1) *// Если соседей больше нет,*

theStack.pop(); *// извлечь элемент из стека*

**else** *// Сосед существует*

{

vertexList[v].wasVisited = **true**; *// Пометка*

theStack.push(v); *// Занесение в стек*

*// Вывод ребра*

displayVertex(currentVertex); *// От currentVertex*

displayVertex(v); *// к v*

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(" ");

}

}

*// Стек пуст, работа закончена*

**for**(**int** j=0; j<nVerts; j++) *// Сброс флагов*

vertexList[j].wasVisited = **false**;

}

*// -------------------------------------------------------------*

*// Метод возвращает непосещенную вершину, смежную по отношению к v*

**public** **int** getAdjUnvisitedVertex(**int** v)

{

**for**(**int** j=0; j<nVerts; j++)

**if**(adjMat[v][j]==1 && vertexList[j].wasVisited==**false**)

**return** j;

**return** -1;

}

*// -------------------------------------------------------------*

} *// Конец класса Graph*

*////////////////////////////////////////////////////////////////*

**class** MSTApp

{

**public** **static** **void** main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args)

{

Graph theGraph = **new** Graph();

theGraph.addVertex('A'); *// 0 (исходная вершина)*

theGraph.addVertex('B'); *// 1*

theGraph.addVertex('C'); *// 2*

theGraph.addVertex('D'); *// 3*

theGraph.addVertex('E'); *// 4*

theGraph.addEdge(0, 1); *// AB*

theGraph.addEdge(0, 2); *// AC*

theGraph.addEdge(0, 3); *// AD*

theGraph.addEdge(0, 4); *// AE*

theGraph.addEdge(1, 2); *// BC*

theGraph.addEdge(1, 3); *// BD*

theGraph.addEdge(1, 4); *// BE*

theGraph.addEdge(2, 3); *// CD*

theGraph.addEdge(2, 4); *// CE*

theGraph.addEdge(3, 4); *// DE*

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("Minimum spanning tree: ");

theGraph.mst(); *// Минимальное остовное дерево*

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println();

}

} *// Конец класса MSTApp*

*////////////////////////////////////////////////////////////////*

**Задание №2.**

Измените программу dfs.java (см. листинг 13.1), чтобы она выводила таблицу связности для направленного графа.

Листинг 13.1

*// dfs.java*

*// Обход в глубину*

*// Запуск программы: C>java DFSApp*

*////////////////////////////////////////////////////////////////*

**class** StackX

{

**private** **final** **int** SIZE = 20;

**private** **int**[] st;

**private** **int** top;

*// -----------------------------------------------------------*

**public** StackX() *// Конструктор*

{

st = **new** **int**[SIZE]; *// Создание массива*

top = -1;

}

*// -----------------------------------------------------------*

**public** **void** push(**int** j) *// Размещение элемента в стеке*

{ st[++top] = j; }

*// -----------------------------------------------------------*

**public** **int** pop() *// Извлечение элемента из стека*

{ **return** st[top--]; }

*// ------------------------------------------------------------*

**public** **int** peek() *// Чтение с вершины стека*

{ **return** st[top]; }

*// ------------------------------------------------------------*

**public** **boolean** isEmpty() *// true, если стек пуст*

{ **return** (top == -1); }

*// ------------------------------------------------------------*

} *// Конец класса StackX*

*////////////////////////////////////////////////////////////////*

**class** Vertex

{

**public** **char** label; *// метка (например, 'A')*

**public** **boolean** wasVisited;

*// ------------------------------------------------------------*

**public** Vertex(**char** lab) *// Конструктор*

{

label = lab;

wasVisited = **false**;

}

*// ------------------------------------------------------------*

} *// Конец класса Vertex*

*////////////////////////////////////////////////////////////////*

**class** Graph

{

**private** **final** **int** MAX\_VERTS = 20;

**private** Vertex vertexList[]; *// Список вершин*

**private** **int** adjMat[][]; *// Матрица смежности*

**private** **int** nVerts; *// Текущее количество вершин*

**private** StackX theStack;

*// -----------------------------------------------------------*

**public** Graph() *// Конструктор*

{

vertexList = **new** Vertex[MAX\_VERTS];

*// Матрица смежности*

adjMat = **new** **int**[MAX\_VERTS][MAX\_VERTS];

nVerts = 0;

**for**(**int** j=0; j<MAX\_VERTS; j++) *// Матрица смежности*

**for**(**int** k=0; k<MAX\_VERTS; k++) *// заполняется нулями*

adjMat[j][k] = 0;

theStack = **new** StackX();

}

*// -----------------------------------------------------------*

**public** **void** addVertex(**char** lab)

{

vertexList[nVerts++] = **new** Vertex(lab);

}

*// -----------------------------------------------------------*

**public** **void** addEdge(**int** start, **int** end)

{

adjMat[start][end] = 1;

adjMat[end][start] = 1;

}

*// ------------------------------------------------------------*

**public** **void** displayVertex(**int** v)

{

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(vertexList[v].label);

}

*// ------------------------------------------------------------*

**public** **void** dfs() *// Обход в глубину*

{ *// Алгоритм начинает с вершины 0*

vertexList[0].wasVisited = **true**; *// Пометка*

displayVertex(0); *// Вывод*

theStack.push(0); *// Занесение в стек*

**while**( !theStack.isEmpty() ) *// Пока стек не опустеет*

{

*// Получение непосещенной вершины, смежной к текущей*

**int** v = getAdjUnvisitedVertex( theStack.peek() );

**if**(v == -1) *// Если такой вершины нет,*

theStack.pop(); *// элемент извлекается из стека*

**else** *// Если вершина найдена*

{

vertexList[v].wasVisited = **true**; *// Пометка*

displayVertex(v); *// Вывод*

theStack.push(v); *// Занесение в стек*

}

}

*// Стек пуст, работа закончена*

**for**(**int** j=0; j<nVerts; j++) *// Сброс флагов*

vertexList[j].wasVisited = **false**;

}

*// ------------------------------------------------------------*

*// Метод возвращает непосещенную вершину, смежную по отношению к v*

**public** **int** getAdjUnvisitedVertex(**int** v)

{

**for**(**int** j=0; j<nVerts; j++)

**if**(adjMat[v][j]==1 && vertexList[j].wasVisited==**false**)

**return** j; *// Возвращает первую найденную вершину*

**return** -1; *// Таких вершин нет*

}

*// ------------------------------------------------------------*

} *// Конец класса Graph*

*////////////////////////////////////////////////////////////////*

**class** DFSApp

{

**public** **static** **void** main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args)

{

Graph theGraph = **new** Graph();

theGraph.addVertex('A'); *// 0 (исходная вершина)*

theGraph.addVertex('B'); *// 1*

theGraph.addVertex('C'); *// 2*

theGraph.addVertex('D'); *// 3*

theGraph.addVertex('E'); *// 4*

theGraph.addEdge(0, 1); *// AB*

theGraph.addEdge(1, 2); *// BC*

theGraph.addEdge(0, 3); *// AD*

theGraph.addEdge(3, 4); *// DE*

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("Visits: ");

theGraph.dfs(); *// Обход в глубину*

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println();

}

} *// Конец класса DFSApp*

*////////////////////////////////////////////////////////////////*