**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра АПУ**

**ОТЧЕТ**

**по лабораторной работе №6**

**на тему: «АВЛ-дерево»**

**по дисциплине «Топологическая сортировка»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3391 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Николаев В.Ю. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Ряскова Е.Б. |

Санкт-Петербург

2024 г.

## Цель работы

Целью данной лабораторной работы является разработка программы для составления учебных планов с использованием алгоритма топологической сортировки, а также проверка возможности корректного размещения образовательных дисциплин в соответствии с заданными зависимостями.

## Задачи

Создать программу для составления учебных планов, в которой:

* реализован диалог с пользователем для ввода числа образовательных дисциплин и связей между ними (пар связанных дисциплин, например, запись [2,1], означает, что для освоения дисциплины 2 требуется освоить дисциплину 1),
* осуществлена проверка возможности корректного размещения всех дисциплин в учебном плане (без нарушения заданной последовательности освоения дисциплин).
* Для взаимодействия с пользователем реализуйте в программе меню, содержащее следующие разделы:
  + Ввод данных о связанных дисциплинах
  + Проверка возможности корректного размещения дисциплин в учебном плане
  + Выход из программы.

## Описание программы

Программа предоставляет пользователю возможность вводить дисциплины и их зависимости, а затем проверяет возможность корректного составления учебного плана с помощью топологической сортировки.

Листинг кода программы приведён в Приложении 1.

## Алгоритмы

### Алгоритм топологической сортировки

Топологическая сортировка — это упорядочение вершин ориентированного ациклического графа (DAG) таким образом, что для каждого ребра (*u*,*v*) вершина *u* предшествует вершине *v*.

Алгоритм реализован с использованием очереди и подсчёта степеней захода (in-degree). Основные шаги:

1. **Построение графа**: Каждая дисциплина представлена в виде вершины, а зависимости между ними — рёбрами. Вершины с нулевой степенью захода добавляются в очередь.
2. **Удаление вершин**: В процессе удаления вершины проверяются её соседи, и если после удаления у соседа остаётся нулевая степень захода, то он добавляется в очередь.
3. **Проверка циклов**: Если после завершения работы остались вершины с ненулевой степенью захода, это указывает на цикл в графе, что делает невозможным корректное составление учебного плана.

### Ввод данных

Пользователь вводит количество дисциплин и пары зависимостей между ними. Каждая зависимость вводится как пара чисел, где первое число — это дисциплина, требующая предварительного освоения второй

## Результаты работы программы

Для проверки работы программы используем следующие примеры:

|  |  |
| --- | --- |
| Пример 1: Успешная топологическая сортировка | Пример 2: Обнаружение цикла |
| Число дисциплин: 5.  Зависимости:   1. [2, 1] — для дисциплины 2 требуется освоить дисциплину 1. 2. [3, 2] — для дисциплины 3 требуется освоить дисциплину 2. 3. [4, 3] — для дисциплины 4 требуется освоить дисциплину 3. 4. [5, 4] — для дисциплины 5 требуется освоить дисциплину 4. | Число дисциплин: 4.  Зависимости:   1. [1, 2] — для дисциплины 1 требуется освоить дисциплину 2. 2. [2, 3] — для дисциплины 2 требуется освоить дисциплину 3. 3. [3, 4] — для дисциплины 3 требуется освоить дисциплину 4. 4. [4, 1] — для дисциплины 4 требуется освоить дисциплину 1. |

## Выводы

В ходе выполнения лабораторной работы были приобретены навыки работы с алгоритмом топологической сортировки и его применением для проверки корректности зависимостей между образовательными дисциплинами. Программа успешно реализует задачи ввода данных, проверки учебного плана и выводит корректный порядок дисциплин.

## Приложение 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| graph.h  |  | | --- | | #ifndef GRAPH\_H  #define GRAPH\_H  #include <iostream>  #include <vector>  #include <queue>  class Graph {  private:      int numSubjects;      std::vector<std::vector<int>> adjList;      std::vector<int> inDegree;  public:      Graph(int num);      void addEdge(int subject1, int subject2);      bool topologicalSort(std::vector<int>& result);      void displayGraph();  };  #endif // GRAPH\_H |  graph.cpp  |  | | --- | | #include "graph.h"  Graph::Graph(int num) : numSubjects(num), adjList(num), inDegree(num, 0) {}  void Graph::addEdge(int subject1, int subject2) {      adjList[subject2].push\_back(subject1);      inDegree[subject1]++;  }  bool Graph::topologicalSort(std::vector<int>& result) {      std::queue<int> q;      for (int i = 0; i < numSubjects; i++) {          if (inDegree[i] == 0) {              q.push(i);          }      }      int visitedCount = 0;      while (!q.empty()) {          int subject = q.front();          q.pop();          result.push\_back(subject);          for (int next : adjList[subject]) {              inDegree[next]--;              if (inDegree[next] == 0) {                  q.push(next);              }          }          visitedCount++;      }      return visitedCount == numSubjects;  }  void Graph::displayGraph() {      for (int i = 0; i < numSubjects; ++i) {          std::cout << "Subject " << i << ": ";          for (int j : adjList[i]) {              std::cout << j << " ";          }          std::cout << std::endl;      }  } | | utils.h  |  | | --- | | #ifndef UTILS\_H  #define UTILS\_H  #include "graph.h"  void displayMenu();  int getUserInput(const std::string& prompt);  void enterDependencies(Graph& graph);  #endif // UTILS\_H |  utils.cpp  |  | | --- | | #include "utils.h"  #include "graph.h"  #include <iostream>  void displayMenu() {      std::cout << "===== Меню =====" << std::endl;      std::cout << "1. Ввод данных о связанных дисциплинах" << std::endl;      std::cout << "2. Проверка возможности корректного размещения дисциплин" << std::endl;      std::cout << "3. Выход из программы" << std::endl;  }  int getUserInput(const std::string& prompt) {      int value;      std::cout << prompt;      std::cin >> value;      return value;  }  void enterDependencies(Graph& graph) {      int numEdges = getUserInput("Введите количество зависимостей: ");      for (int i = 0; i < numEdges; i++) {          int subject1 = getUserInput("Введите номер первой дисциплины: ") - 1;          int subject2 = getUserInput("Введите номер второй дисциплины: ") - 1;          graph.addEdge(subject1, subject2);      }  } |  main.cpp  |  | | --- | | #include <iostream>  #include "graph.h"  #include "utils.h"  int main() {      int numSubjects = getUserInput("Введите количество дисциплин: ");      Graph graph(numSubjects);      while (true) {          displayMenu();          int choice = getUserInput("Выберите пункт меню: ");          if (choice == 1) {              enterDependencies(graph);          } else if (choice == 2) {              std::vector<int> result;              if (graph.topologicalSort(result)) {                  std::cout << "Корректное размещение дисциплин возможно. Порядок: ";                  for (int subject : result) {                      std::cout << subject + 1 << " ";                  }                  std::cout << std::endl;              } else {                  std::cout << "Корректное размещение дисциплин невозможно!" << std::endl;              }          } else if (choice == 3) {              std::cout << "Выход из программы." << std::endl;              break;          } else {              std::cout << "Неверный выбор, попробуйте снова." << std::endl;          }      }      return 0;  } | |