**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
*Факультет інформатики та обчислювальної техніки*
*Кафедра обчислювальної техніки*

# Лабораторна робота №1

*з дисципліни «Програмне забезпечення комп'ютерних систем - 2»*

*на тему: «Проектування редакторів графів задач та комп'ютерної системи»*

**Виконав:**
студент 5-го курсу ФІОТ
групи ІО – 82мп
*Барабаш Т.А.*

**Перевірила:**
доцент
*Русанова О. В.*

**Київ - 2019**

# ЛАБОРАТОРНА РОБОТА №1

*Проектування редакторів графів задач та комп'ютерної системи*

**Мета:** Навчитися створювати інтерфейс програмної моделі для дослідження алгоритмів планування обчислень.

## I. Завдання

Створити інтерфейс програмної моделі для дослідження алгоритмів планування обчислень. У даній роботі необхідно створити меню програми, а також розробити редактор графа задачі і системи (крім SMP) з необхідними перевірками на будь-якій мові програмування.

## II. Приклад роботи програми

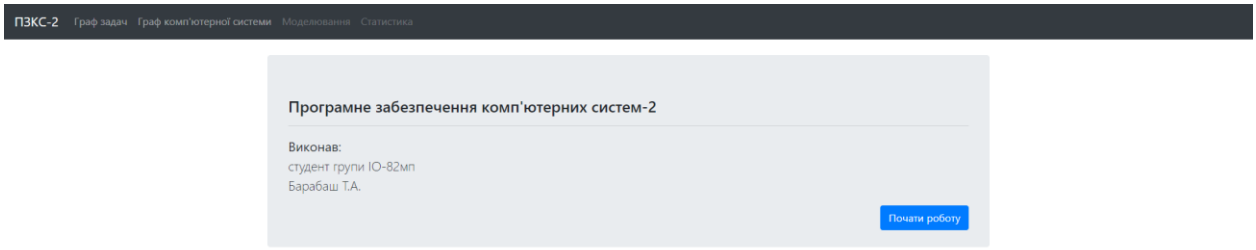


Рис.1. Стартовий екран програми

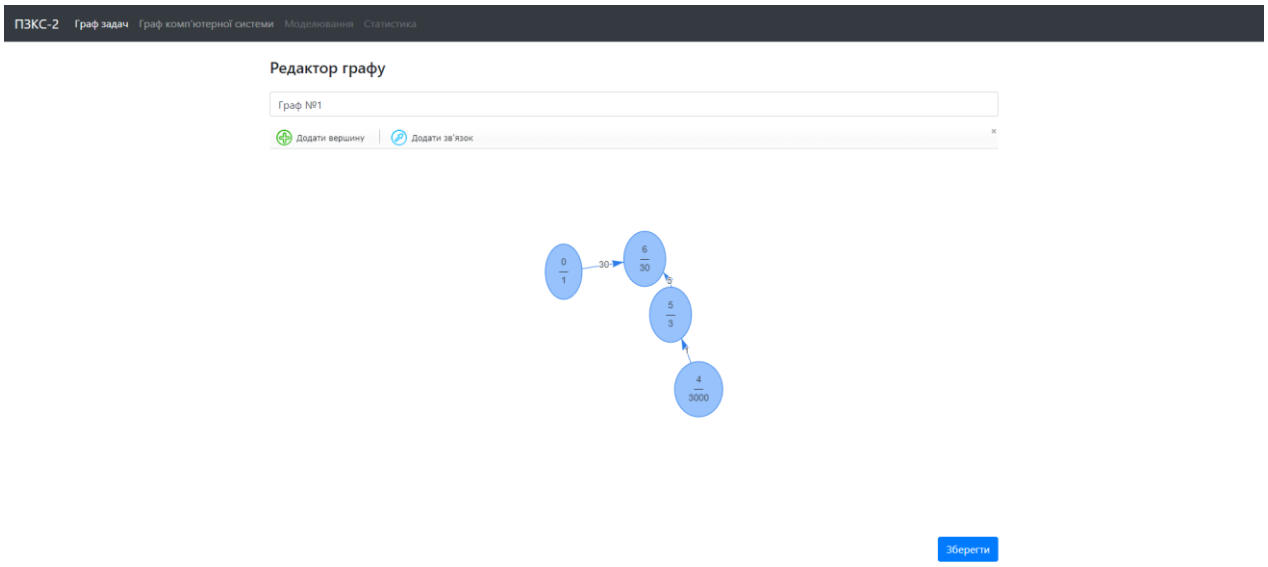


Рис. 2. Редактор графу задач

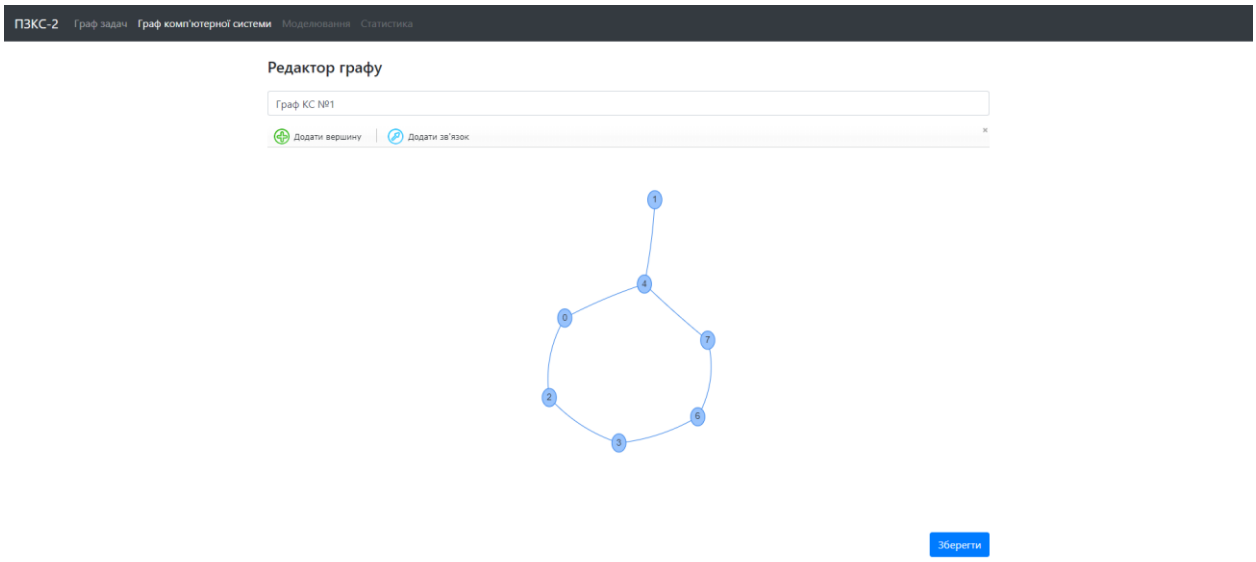


Рис.3. Редактор графу комп’ютерної системи

# III. Хід роботи

**При збережені графів відбувається перевірка:**

1. Для графу задач – чи присутні у графі цикли, - для кожної вершини знаходимо усі можливі переходи у інші вершини графу, та якщо було знайдено вершину, яка

повторно зустрічається при переході виводиться помилка про те, що граф циклічний.

2. Для графу комп’ютерної системи – чи зв’язний граф, тобто такий, що для кожної вершин існує хоча б один шлях до кожної із вершин системи. Алгоритм перевірки наступний: для кожної вершини шукаємо чи є можливість перейти у інші вершини. Якщо шлях не знайдено, виводиться помилка про те, що граф незв’язний.

# IV. Код програми

```
import { Network } from "vis";
import React, { Component, createRef } from "react";
import "vis/dist/vis.css";

const TYPE_CS = 1;
const TYPE_TASK = 2;

class GraphView extends Component {
  constructor(props) {
    super(props);
    this.type = props.name === "tasks" ? TYPE_TASK : TYPE_CS;
    console.log(this.type);
    this.network = {};
    this.appRef = createRef();
    this.options = {
      locales: this.locales,
      locale: "uk",
      manipulation: {
        enabled: true,
        initiallyActive: true,
        addNode: (data, callback) => {
          this.params(data, callback, "add", "вершини");
        },
        // editNode: (data, callback) => {
        //   this.params(data, callback, "edit", "вершини");
        // },
        addEdge: (data, callback) => {
          this.params(data, callback, "add", "зв'язку");
        }
        // editEdge: (data, callback) => {
        //   this.params(data, callback, "edit", "зв'язку");
        // }
      },
      edges: {
        arrows: {
          to: {
            enabled: this.type === TYPE_TASK
          }
        }
```

```
    }
  };
  this.state = {
    data: props.data,
    type: this.type
  };
}

locales = {
  uk: {
    edit: "Редагувати",
    del: "Видалити обране",
    back: "Назад",
    addNode: "Додати вершину",
    addEdge: "Додати зв'язок",
    editNode: "Редагувати вершину",
    editEdge: "Редагувати зв'язок",
    addDescription: "Натисніть на пустому місці аби додати вершину.",
    edgeDescription:
      "Click on a node and drag the edge to another node to connect them.",
    editEdgeDescription:
      "Click on the control points and drag them to a node to connect to it.",
    createEdgeError: "Cannot link edges to a cluster.",
    deleteClusterError: "Clusters cannot be deleted.",
    editClusterError: "Clusters cannot be edited."
  }
};

params = (data, callback, mode, type) => {
  const isEdit = mode === "edit";
  const isEdge = type === "зв'язку";
  const { nodes } = this.state.data;
  const lastId = nodes.map(node => node.id).sort((a, b) => b - a)[0];
  const number = isEdit ? data.number : !isNaN(lastId) ? lastId + 1 : 0;
  const weight = this.getWeight(type, isEdit, data);
  if (!weight) return;
  const label = `${
    !isEdge
      ? this.state.type === TYPE_CS
        ? `${number}`
        : `${number}\n – \n${weight}`
      : this.state.type === TYPE_CS
      ? ``
      : `${weight}`
  }`;
  data.number = number;
  data.weight = weight;
  data.label = label;
  if (!isEdit && !isEdge) {
    data.id = number;
```

```
    }
    if (isEdge) {
      const { edges } = this.state.data;
      const { from, to } = data;
      console.log(edges);
      console.log(data);
      console.log(edges.get({ from, to })[0]);
    }
    try {
      return callback(data);
    } catch (e) {
      console.log(e);
    }
  };

  getWeight(type, isEdit, data) {
    if (this.state.type === TYPE_CS) return ` `;
    let weight = prompt(`Вага ${type}`, isEdit ? data.weight : ``);
    if (!weight) return alert(`"Вагу ${type} не задано!"`);
    weight = parseInt(weight);
    if (isNaN(weight)) return alert(`Невірний формат ваги ${type}!`);
    if (weight <= 0)
      return alert(`Вага не може бути негативною чи дорівнювати 0`);
    return weight;
  }

  getSnapshotBeforeUpdate(prevProps, prevState) {
    const { edges, nodes } = this.props.data;
    const { edges: edgesP, nodes: nodesP } = prevProps.data;
    if (!edgesP || !nodesP) return true;
    return edges.length !== edgesP.length || nodes.length !== nodesP.length;
  }

  onDoubleClick = params => {
    const { nodes, edges } = params;
    const {
      data: { nodes: nodesS, edges: edgesS },
      type
    } = this.state;
    if (type === TYPE_CS) return;
    if (nodes.length > 0) {
      const node = nodesS.get(nodes[0]);
      const weight = this.getWeight("вершини", true, { weight: node.weight });
      if (!weight) return;
      const label = `${node.id}\n—\n${weight}`;
      nodesS.update({ ...node, weight, label });
    } else {
      const edge = edgesS.get(edges[0]);
      const weight = this.getWeight("зв'язку", true, { weight: edge.weight });
      if (!weight) return;
```

```
      const label = `${weight}`;
      edgesS.update({ ...edge, weight, label });
    }
  };

  componentDidMount() {
    this.network = new Network(
      this.appRef.current,
      this.props.data,
      this.options
    );
    if (this.state.type === TYPE_TASK)
      this.network.on("doubleClick", params => this.onDoubleClick(params));
    this.setState({
      data: {
        nodes: this.network.body.data.nodes,
        edges: this.network.body.data.edges
      }
    });
  }

  componentDidUpdate(prevProps, prevState, snapshot) {
    if (!snapshot) return;
    this.network = new Network(
      this.appRef.current,
      this.props.data,
      this.options
    );
    if (this.state.type === TYPE_TASK)
      this.network.on("doubleClick", params => this.onDoubleClick(params));
    this.setState({
      data: {
        nodes: this.network.body.data.nodes,
        edges: this.network.body.data.edges
      }
    });
  }

  render() {
    return <div style={{ height: "600px", width: "100%" }} ref={this.appRef} />;
  }
}

export default GraphView;
```