ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И
ИНФОРМАТИКИ»

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине “Моделирование”

Выполнил студент Рудцких Владислав Евгеньевич
Ф.И.О.

Группы ИВ-222

Работу принял Ассистент кафедры ВС Уженцева А. В.
подпись

Защищена _____ Оценка

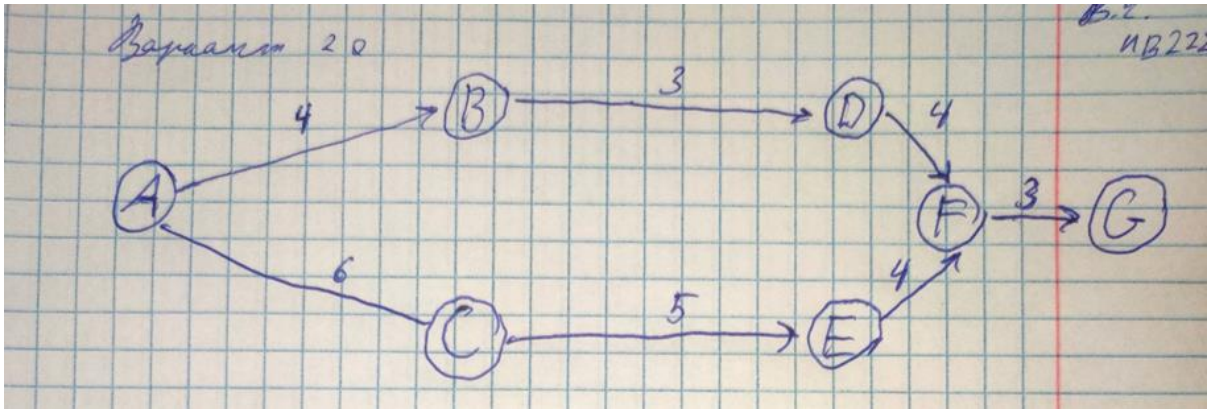
Новосибирск – 2026

Задание

Вариант 20:

Вариант 20

Вершина	Предшественник	Длительность
A	-	5
B	A	4
C	A	6
D	B	3
E	C	5
F	D,E	4
G	F	3



Реализовать построение сетевого графика и расчет критических точек по взвешенному графу событий.

Теория

Шифр - две вершины связанные ребром

t_ij - вес (длительность) ребра

t^РН_ij - самый ранний возможный момент времени, когда данная работа может быть начата при условии, что все предшествующие работы выполнены в строго запланированной последовательности

t^РО_ij - самый ранний из возможных сроков окончания работы

t^ПН_ij - самый поздний срок при котором может быть начата работа без нарушения общего срока работы

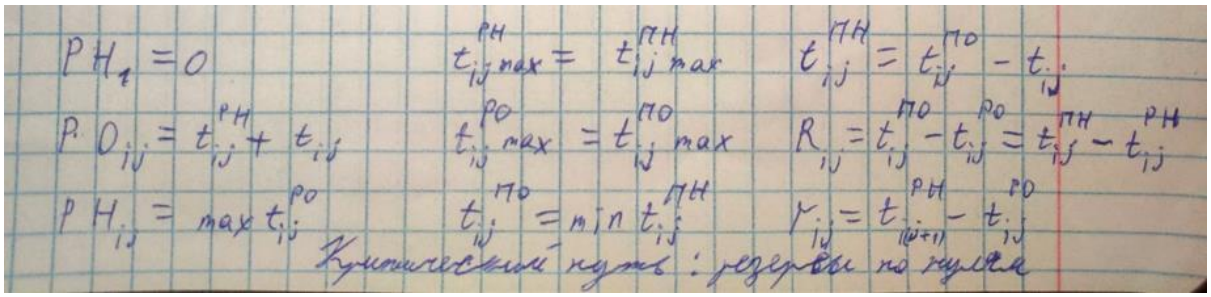
t^ПО_ij - самый поздний срок при котором можно закончить работу не срывая срок проекта

R_ij - максимальное время на которое можно увеличить время работы без нарушения позднего срока конечного события сохраняя продолжительность критического пути

r_ij - количество времени на которое можно увеличить продолжительность работы или перенести ее начало так, чтобы не изменилось ранее начало последующих работ

T_кр - сумма длительностей всех операций критического пути

Сетевой график - графическая модель проекта описывающего логические взаимосвязи и последовательности между операциями



Ход работы

Выполнение вычислений:

1) Построение графа на основе таблицы

2) Построение сетевого графика

3) Нахождение критического пути

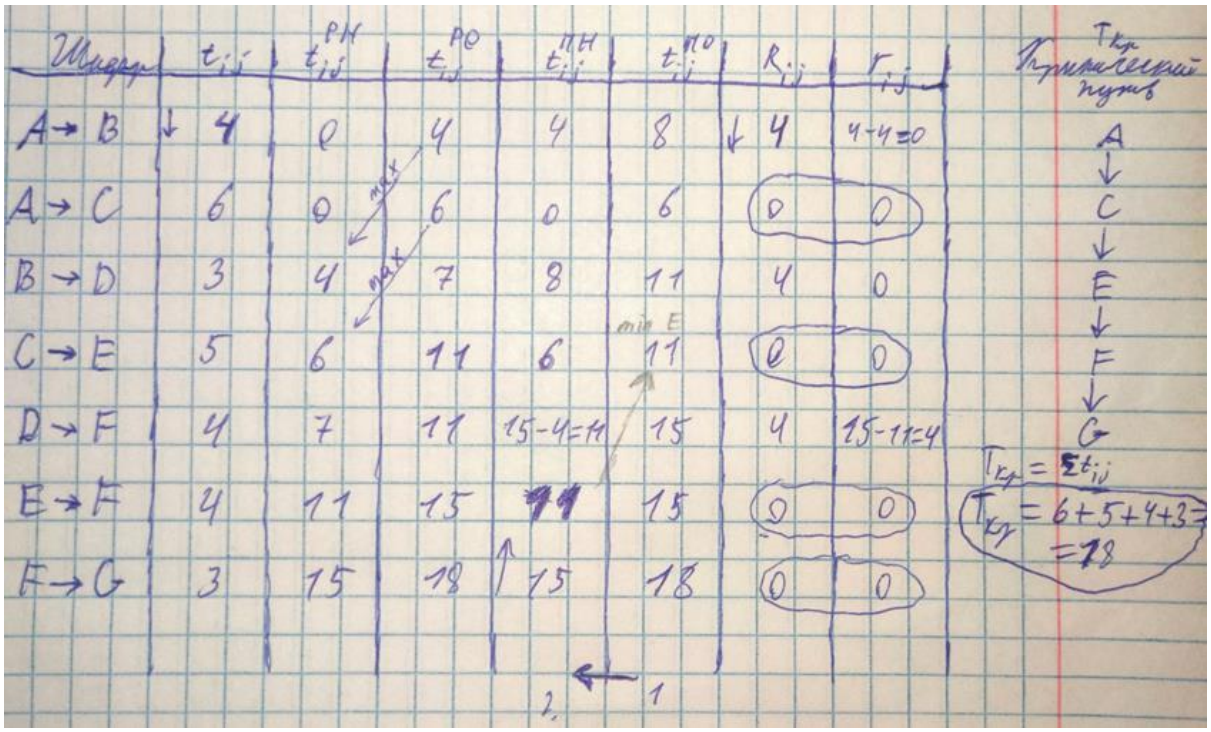
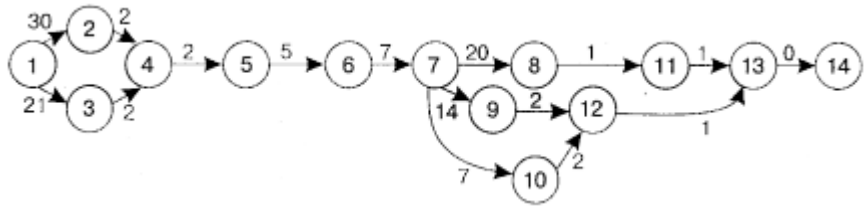
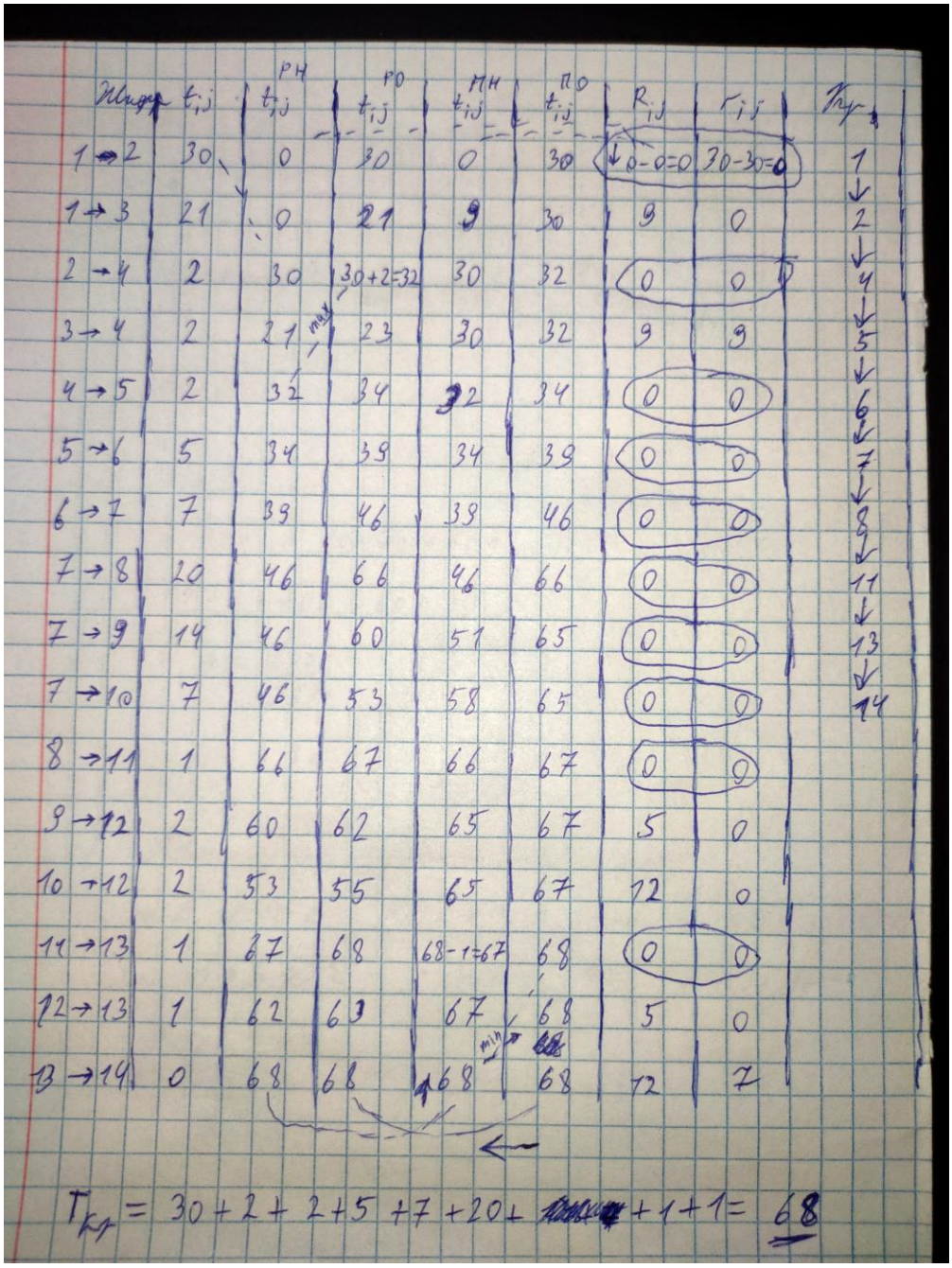


Рисунок 1. Сетевой график

Пример №2:



Решение:



Программная реализация

Алгоритм программы:

1. Структура Work — хранит данные о каждой работе:
    a. Начальное и конечное событие (start, end)
    b. Продолжительность (duration)
    c. Временные параметры: раннее/позднее начало/окончание
    d. Резервы времени: полный (R_ij) и частный (r_ij)
2. Класс Graph — основной вычислительный модуль:
    a. Загрузка графа из файла (формат: вершина, предшественник, вес)
    b. Построение списков смежности и предшественников
    c. Расчет ранних сроков (прямой проход по графу)
    d. Расчет поздних сроков (обратный проход)
    e. Вычисление резервов времени
    f. Определение критического пути (работы с нулевым полным и частным резервом)
3. Вывод результатов:
    a. Таблица со всеми рассчитанными параметрами
    b. Длина критического пути
    c. Последовательность работ критического пути

Результат

Пример №1:

```
Использовать данные с моей карточки? (Вариант 20) (y/n): y
Шифр       t(i,j)        t^РН_ij      t^РО_ij      t^ПН_ij      t^ПО_ij       R_ij      r_ij       Кр.
1-2           4             0            4            4            8            4         0
1-3           6             0            6            0            6            0         0          Критическая точка
2-4           3             4            7            8            11           4         0
3-5           5             6            11           6            11           0         0          Критическая точка
4-6           4             7            11           11           15           4         4
5-6           4             11           15           11           15           0         0          Критическая точка
6-7           3             15           18           15           18           0         0          Критическая точка
Длина критического пути: 18

Критический путь: 1 -> 3 -> 5 -> 6 -> 7
```

Рисунок 2. Сетевой график (программная реализация)

Пример №2

```
Использовать данные с моей карточки? (Вариант 20) (y/n): n
Введите имя файла с данными: test_graph.txt
Шифр       t(i,j)        t^РН_ij      t^РО_ij      t^ПН_ij      t^ПО_ij       R_ij      r_ij       Кр.
1-2           30            0            30           0            30           0         0          Критическая точка
1-3           21            0            21           9            30           9         0
2-4           2             30           32           30           32           0         0          Критическая точка
3-4           2             21           23           30           32           9         9
4-5           2             32           34           32           34           0         0          Критическая точка
5-6           5             34           39           34           39           0         0          Критическая точка
6-7           7             39           46           39           46           0         0          Критическая точка
7-8           20            46           66           46           66           0         0          Критическая точка
7-9           14            46           60           51           65           5         0
7-10          7             46           53           58           65           12        0
8-11          1             66           67           66           67           0         0          Критическая точка
9-12          2             60           62           65           67           5         0
10-12         2             53           55           65           67           12        7
11-13         1             67           68           67           68           0         0          Критическая точка
12-13         1             62           63           67           68           5         5
13-14         0             68           68           68           68           0         0          Критическая точка
Длина критического пути: 68

Критический путь: 1 -> 2 -> 4 -> 5 -> 6 -> 7 -> 8 -> 11 -> 13 -> 14
```

:

Рисунок 3. Сетевой график (программаня реализация) №2

Вывод

В ходе выполнения работы была разработана программа для расчета временных параметров сетевого графика на основе взвешенного ориентированного графа.

Приложение

main.cpp:

```
#include <algorithm>
#include <climits>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <queue>
#include <sstream>
#include <vector>

using namespace std;

struct Work {
    int start;
    int end;
    int duration;

    int t_early_start;
    int t_early_finish;
    int t_late_start;
    int t_late_finish;
    int full_reserve;
    int reserve;

    Work(int i, int j, int d)
        : start(i),
          end(j),
          duration(d),
          t_early_start(0),
          t_early_finish(0),
          t_late_start(0),
          t_late_finish(0),
          full_reserve(0),
          reserve(0)
    {
    }
};

class Graph {
private:
    int num_events;
    vector<Work> works;
    vector<vector<int>> adj_list;
    vector<vector<int>> pred_list;
```

```
map<int, string> event_names;

void calculateEarlyTimes()
{
   vector<int> early_time(num_events + 1, 0);

   for (int i = 1; i <= num_events; i++) {
      for (int work_idx : adj_list[i]) {
         Work& w = works[work_idx];
         int new_time = early_time[w.start] + w.duration;
         if (new_time > early_time[w.end]) {
            early_time[w.end] = new_time;
         }
      }
   }

   for (Work& w : works) {
      w.t_early_start = early_time[w.start];
      w.t_early_finish = w.t_early_start + w.duration;
   }
}

void calculateLateTimes()
{
   int critical_time = 0;
   for (const Work& w : works) {
      critical_time = max(critical_time, w.t_early_finish);
   }

   vector<int> late_time(num_events + 1, critical_time);

   for (int i = num_events; i >= 1; i--) {
      for (int work_idx : pred_list[i]) {
         Work& w = works[work_idx];
         int new_time = late_time[w.end] - w.duration;
         if (new_time < late_time[w.start]) {
            late_time[w.start] = new_time;
         }
      }
   }

   for (Work& w : works) {
      w.t_late_finish = late_time[w.end];
      w.t_late_start = w.t_late_finish - w.duration;
   }
}

void calculateFloats()
```

```
    {
        for (Work& w : works) {
            w.full_reserve = w.t_late_start - w.t_early_start;

            int min_early_start_next = INT_MAX;
            for (int next_idx : adj_list[w.end]) {
                const Work& next_work = works[next_idx];
                min_early_start_next
                        = min(min_early_start_next, next_work.t_early_start);
            }

            if (min_early_start_next != INT_MAX) {
                w.reserve = min_early_start_next - w.t_early_finish;
            } else {
                w.reserve = 0;
            }
        }
    }

public:
    Graph() : num_events(0)
    {
    }

    bool loadFromFile(const string& filename)
    {
        ifstream file(filename);
        if (!file.is_open()) {
            cerr << "ERROR: Не удалось открыть файл " << filename << endl;
            return false;
        }

        works.clear();
        adj_list.clear();
        pred_list.clear();
        event_names.clear();

        string line;
        int max_event = 0;
        vector<tuple<int, int, int>> temp_works;

        while (getline(file, line)) {
            if (line.empty())
                continue;

            istringstream iss(line);
            int ver, predecessor, weight;
```

```
        if (iss >> ver >> predecessor >> weight) {
            max_event = max(max_event, ver);
            if (predecessor > 0) {
                max_event = max(max_event, predecessor);
            }
            temp_works.push_back(make_tuple(predecessor, ver, weight));
        }
    }

    file.close();

    if (temp_works.empty()) {
        cerr << "ERROR: Файл не содержит данных" << endl;
        return false;
    }

    num_events = max_event;
    adj_list.resize(num_events + 1);
    pred_list.resize(num_events + 1);

    for (const auto& [pred, ver, weight] : temp_works) {
        if (pred > 0) {
            addWork(pred, ver, weight);
        } else {
            if (ver != 1) {
            }
            addWork(1, ver, weight);
        }
    }

    // cout << "Количество событий: " << num_events << endl;
    // cout << "Количество работ: " << works.size() << endl;

    return true;
}

void addWork(int i, int j, int duration)
{
    works.push_back(Work(i, j, duration));
    int index = works.size() - 1;
    adj_list[i].push_back(index);
    pred_list[j].push_back(index);
}

void calculateAll()
{
    if (works.empty()) {
        cerr << "ERROR: Нет данных для расчета" << endl;
```

```
        return;
    }
    calculateEarlyTimes();
    calculateLateTimes();
    calculateFloats();
}

void printTable()
{
    if (works.empty()) {
        cout << "Нет данных для отображения" << endl;
        return;
    }
    cout << left << setw(15) << "Шифр" << setw(15) << "t(i,j)" << setw(15)
         << "t^РН_ij" << setw(15) << "t^РО_ij" << setw(15) << "t^ПН_ij"
         << setw(15) << "t^ПО_ij" << setw(13) << "   R_ij" << setw(13)
         << "r_ij"
         << "Кр." << endl;

    int critical_path_length = 0;
    for (const Work& w : works) {
        critical_path_length = max(critical_path_length, w.t_early_finish);
    }

    for (size_t k = 0; k < works.size(); k++) {
        const Work& w = works[k];

        bool is_critical = (w.full_reserve == 0);

        string code = to_string(w.start) + '-' + to_string(w.end);
        ;

        cout << left << setw(15) << code << setw(15) << w.duration
             << setw(13) << w.t_early_start << setw(13) << w.t_early_finish
             << setw(13) << w.t_late_start << setw(13) << w.t_late_finish
             << setw(10) << w.full_reserve << setw(10) << w.reserve;

        if (is_critical)
            cout << " Критическая точка";

        cout << endl;
    }

    cout << "Длина критического пути: " << critical_path_length << endl;

    cout << "\nКритический путь: ";
    findAndPrintCriticalPath();
    cout << endl;
```

```
}

void findAndPrintCriticalPath()
{
   vector<int> path;
   vector<bool> visited(num_events + 1, false);

   int current = 1;
   path.push_back(current);

   while (current != num_events) {
      bool found = false;
      for (int work_idx : adj_list[current]) {
         Work& w = works[work_idx];
         if (w.full_reserve == 0 && w.start == current) {
            if (!visited[w.end]) {
               current = w.end;
               path.push_back(current);
               visited[current] = true;
               found = true;
               break;
            }
         }
      }
      if (!found)
         break;
   }

   for (size_t i = 0; i < path.size(); i++) {
      if (event_names.count(path[i])) {
         cout << event_names[path[i]];
      } else {
         cout << path[i];
      }
      if (i < path.size() - 1)
         cout << " -> ";
   }
}

// void printGraphInfo() {
//    cout << "\nИнформация о графе:" << endl;
//    cout << "Количество событий: " << num_events << endl;
//    cout << "Количество работ: " << works.size() << endl;
//    cout << "\nСписок работ:" << endl;
//    for (const Work& w : works) {
//       cout << w.start << " -> " << w.end << " : " << w.duration <<
//       endl;
//    }
```

```
    // }
};

int main()
{
    Graph graph;

    cout << "Использовать данные с моей карточки? (Вариант 20) (y/n): ";
    char choice;
    cin >> choice;

    if (choice == 'y' || choice == 'Y') {
        Graph test_graph;

        ofstream test_file("test_graph.txt");
        test_file << "2 1 4\n";
        test_file << "3 1 6\n";
        test_file << "4 2 3\n";
        test_file << "5 3 5\n";
        test_file << "6 4 4\n";
        test_file << "6 5 4\n";
        test_file << "7 6 3\n";
        test_file.close();

        if (test_graph.loadFromFile("test_graph.txt")) {
            test_graph.calculateAll();
            test_graph.printTable();
        }
    } else {
        string filename;

        cout << "Введите имя файла с данными: ";
        cin >> filename;

        if (graph.loadFromFile(filename)) {
            graph.calculateAll();
            graph.printTable();
        }
    }

    return 0;
}
```