Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления

Кафедра Интеллектуальных информационных технологий

ОТЧЁТ

по расчетной работе

Выполнили:

Мухомедзянов Е. С.

Санюк М. И.

Заборовский И. В.

Студенты группы

421701

Проверил: Н. В. Зотов

Минск 2025

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Вариант 7. Составление графика работы сотрудников

Постановка задачи: Ресторан “Гурман” открыт ежедневно и обслуживает гостей в три смены (утреннюю, дневную, ночную). Всего задействовано 12 сотрудников различных профессий: повара, официанты, уборщики и администраторы. Каждый сотрудник может работать не более 5 дней в неделю, при этом для каждой смены требуется строго определённый состав команды: 1 повар, 2 официанта, 1 уборщик, 1 администратор. Для поваров и администраторов есть ограничения по доступным сменам: например, поварам запрещено работать ночью, а администраторы — только днём. Ресторану требуется каждый день соблюдать все кадровые ограничения и при этом равномерно распределить нагрузку среди сотрудников, а также предусмотреть резервные варианты на случай болезни ключевого работника.

Пример:

Штат сотрудников:

- повара: Антонов, Белов, Васильев, Гришин (не могут работать в ночную смену);
- официанты: Дмитриев, Егоров, Жуков, Зайцев;
- уборщики: Иванов, Козлов;
- администраторы: Львов, Михайлов (работают только в дневную смену).

Входные данные:

Список всех сотрудников (разделённых по профессиям) и ограничений по рабочим сменам для каждого, а также требования к составу смены.

Требуется:

1. Сформировать двудольный граф “сотрудники—смены” с учётом всех ограничений (работоспособность, максимальная нагрузка, запреты).
2. Найти максимальное паросочетание для оптимального распределения сотрудников по сменам.
3. Проверить, обеспечивает ли график выполнение всех требований к укомплектованию смен.
4. Сформировать недельный график работы для каждого сотрудника (календарь занятости).
5. Рассчитать загрузку каждого сотрудника (количество смен за неделю).

РЕАЛИЗАЦИЯ БАЗЫ ЗНАНИЙ

Описание разработанной предметной области

Для формализации задачи используются следующие ключевые элементы:

- Абсолютные понятия (классы*): concept_restaurant, concept_employee, concept_shift, concept_shift_type, concept_week_schedule, concept_staffing_issue, concept_employee_slot*.
- Роли сотрудников: *concept_cook, concept_waiter, concept_cleaner, concept_admin.*
- Отношения: *nrel_has_employee (ресторан -> сотрудник), nrel_has_role (сотрудник -> роль), nrel_available_shift_type (сотрудник -> тип смены), nrel_shift_type (смена -> тип), nrel_shift_day (смена -> день), nrel_max_shifts_per_week (сотрудник -> лимит), nrel_assigned_employee (смена -> назначенный сотрудник), nrel_reserve_employee (смена -> резерв), nrel_employee_schedule (сотрудник -> его расписание), nrel_shift_count (сотрудник -> количество смен), nrel_all_shifts_staffed (расписание -> флаг укомплектования).*
- Действие: *action_build_staff_schedule* (построение расписания персонала).

2.2 Код основных файлов

Файл ключевых узлов: problem-solver/cxx/staff-schedule-module/keynodes/staff_schedule_keynodes.hpp

```
action_build_staff_schedule
<- sc_node_class ;
<- concept_class ;
= > nrel_main_idtf : [ действие построения расписания персонала ] (* <- lang_ru
;; *);;

concept_employee
<- sc_node_class ;
<- concept_class ;
= > nrel_main_idtf : [ сотрудник ] (* <- lang_ru ;; *);;

concept_shift
<- sc_node_class ;
<- concept_class ;
= > nrel_main_idtf : [ смена ] (* <- lang_ru ;; *);;

concept_shift_type
<- sc_node_class ;
<- concept_class ;
= > nrel_main_idtf : [ тип смены ] (* <- lang_ru ;; *);;
```

```
nrel_has_employee
<- sc_node_non_role_relation ;
= > nrel_main_idtf : [ имеет сотрудника * ] (* <- lang_ru ;; *);;

nrel_has_role
<- sc_node_non_role_relation ;
= > nrel_main_idtf : [ имеет роль * ] (* <- lang_ru ;; *);;

nrel_available_shift_type
<- sc_node_non_role_relation ;
= > nrel_main_idtf : [ доступный тип смены * ] (* <- lang_ru ;; *);;

nrel_shift_type
<- sc_node_non_role_relation ;
= > nrel_main_idtf : [ тип смены * ] (* <- lang_ru ;; *);;

nrel_max_shifts_per_week
<- sc_node_non_role_relation ;
= > nrel_main_idtf : [ максимум смен в неделю * ] (* <- lang_ru ;; *);;

nrel_assigned_employee
<- sc_node_non_role_relation ;
= > nrel_main_idtf : [ назначенный сотрудник * ] (* <- lang_ru ;; *);;

nrel_reserve_employee
<- sc_node_non_role_relation ;
= > nrel_main_idtf : [ резервный сотрудник * ] (* <- lang_ru ;; *);;

nrel_employee_schedule
<- sc_node_non_role_relation ;
= > nrel_main_idtf : [ расписание сотрудника * ] (* <- lang_ru ;; *);;

nrel_shift_count
<- sc_node_non_role_relation ;
= > nrel_main_idtf : [ количество смен * ] (* <- lang_ru ;; *);;

nrel_all_shifts_staffed
<- sc_node_non_role_relation ;
= > nrel_main_idtf : [ все смены укомплектованы * ] (* <- lang_ru ;; *);;
```

РЕАЛИЗАЦИЯ АГЕНТОВ

Описание логики работы агентов

BuildStaffScheduleAgent:

- читает ресторан и список сотрудников, их роли, доступные типы смен и недельный лимит;
- читает список смен и их типы;
- строит двудольный граф назначений и расширяет его слотами для учета максимального числа смен;

- формирует требования по ролям на каждую смену (повар 1, официант 2, уборщик 1, администратор 1);
- решает задачу максимального потока (Dinic) и получает назначения;
- записывает расписание смен, персональные расписания, флаг укомплектования и причины недоукомплектования.

3.2 Листинг ключевых узлов

```
class StaffScheduleKeynodes : public ScKeynodes
{
  static inline ScKeynode const action_build_staff_schedule;
  static inline ScKeynode const concept_employee;
  static inline ScKeynode const concept_shift;
  static inline ScKeynode const concept_shift_type;
  static inline ScKeynode const concept_week_schedule;
  static inline ScKeynode const concept_restaurant;
  static inline ScKeynode const concept_staffing_issue;
  static inline ScKeynode const concept_cook;
  static inline ScKeynode const concept_waiter;
  static inline ScKeynode const concept_cleaner;
  static inline ScKeynode const concept_admin;
  static inline ScKeynode const nrel_has_employee;
  static inline ScKeynode const nrel_has_role;
  static inline ScKeynode const nrel_available_shift_type;
  static inline ScKeynode const nrel_shift_type;
  static inline ScKeynode const nrel_assigned_employee;
  static inline ScKeynode const nrel_reserve_employee;
  static inline ScKeynode const nrel_employee_schedule;
  static inline ScKeynode const nrel_shift_count;
  static inline ScKeynode const nrel_all_shifts_staffed;
};
```

3.3 Листинг кода агента

```
#include "build_staff_schedule_agent.hpp"
#include "keynodes/staff_schedule_keynodes.hpp"

#include <sc-memory/sc_memory.hpp>
#include <sc-memory/sc_iterator.hpp>

#include <algorithm>
#include <functional>
#include <string>
#include <vector>

using namespace std;

namespace
{
struct EmployeeInfo
{
  ScAddr addr;
  ScAddr role;
  vector<ScAddr> availableShiftTypes;
  size_t assignedCount = 0;
```

```
  size_t maxShifts = 5;
  vector<ScAddr> assignedShifts;
};

struct ShiftInfo
{
  ScAddr addr;
  ScAddr shiftType;
  ScAddr day;
};

struct ShiftSlot
{
  ScAddr shift;
  ScAddr role;
};

bool HasAddr(vector<ScAddr> const & list, ScAddr const & addr)
{
  for (auto const & item : list)
  {
    if (item == addr)
      return true;
  }
  return false;
}
}

ScAddr BuildStaffScheduleAgent::GetActionClass() const
{
  return StaffScheduleKeynodes::action_build_staff_schedule;
}

ScResult BuildStaffScheduleAgent::DoProgram(ScAction & action)
{
  m_logger.Debug("BuildStaffScheduleAgent started");

  try
  {
    auto const & [restaurantAddr] = action.GetArguments<1>();
    if (!m_context.IsElement(restaurantAddr))
    {
      m_logger.Error("Restaurant not specified.");
      return action.FinishWithError();
    }

    // Собираем типы смен один раз, чтобы использовать при проверке
доступности.
    vector<ScAddr> allShiftTypes;
    ScIterator3Ptr itShiftTypes = m_context.CreateIterator3(
        StaffScheduleKeynodes::concept_shift_type,
        ScType::ConstPermPosArc,
        ScType::ConstNode);
    while (itShiftTypes->Next())
    {
      allShiftTypes.push_back(itShiftTypes->Get(2));
```

```
    }

    vector<EmployeeInfo> employees;
    ScIterator5Ptr itEmployees = m_context.CreateIterator5(
        restaurantAddr,
        ScType::ConstCommonArc,
        ScType::ConstNode,
        ScType::ConstPermPosArc,
        StaffScheduleKeynodes::nrel_has_employee);

    while (itEmployees->Next())
    {
      EmployeeInfo info;
      info.addr = itEmployees->Get(2);

      // У каждого сотрудника должна быть роль; некорректные записи пропускаем.
      ScIterator5Ptr itRole = m_context.CreateIterator5(
          info.addr,
          ScType::ConstCommonArc,
          ScType::ConstNode,
          ScType::ConstPermPosArc,
          StaffScheduleKeynodes::nrel_has_role);
      if (itRole->Next())
      {
        info.role = itRole->Get(2);
      }
      else
      {
        m_logger.Warning("Employee without role skipped");
        continue;
      }

      // Если доступные типы смен не указаны, считаем, что сотрудник доступен
для всех типов.
      ScIterator5Ptr itShiftType = m_context.CreateIterator5(
          info.addr,
          ScType::ConstCommonArc,
          ScType::ConstNode,
          ScType::ConstPermPosArc,
          StaffScheduleKeynodes::nrel_available_shift_type);
      while (itShiftType->Next())
      {
        info.availableShiftTypes.push_back(itShiftType->Get(2));
      }
      if (info.availableShiftTypes.empty())
      {
        info.availableShiftTypes = allShiftTypes;
      }

      // Читаем недельный лимит; если его нет или он неверный, используем 5.
      ScIterator5Ptr itMax = m_context.CreateIterator5(
          info.addr,
          ScType::ConstCommonArc,
          ScType::ConstNodeLink,
          ScType::ConstPermPosArc,
          StaffScheduleKeynodes::nrel_max_shifts_per_week);
```

```
      if (itMax->Next())
      {
        ScAddr const & linkAddr = itMax->Get(2);
        string value;
        if (m_context.GetLinkContent(linkAddr, value))
        {
          try
          {
            info.maxShifts = static_cast<size_t>(stoi(value));
          }
          catch (exception const &)
          {
            info.maxShifts = 5;
          }
        }
      }

      employees.push_back(info);
    }

    if (employees.empty())
    {
      m_logger.Error("No employees found for restaurant");
      return action.FinishWithError();
    }
```

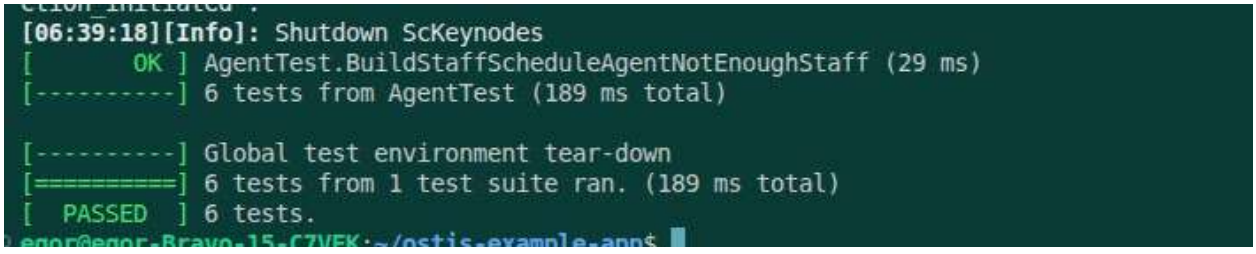
ТЕСТИРОВАНИЕ

Описание тестов

Тесты реализованы в файле ***problem-solver/cxx/staff-schedule-module/test/build_staff_schedule_agent_tests.cpp***.

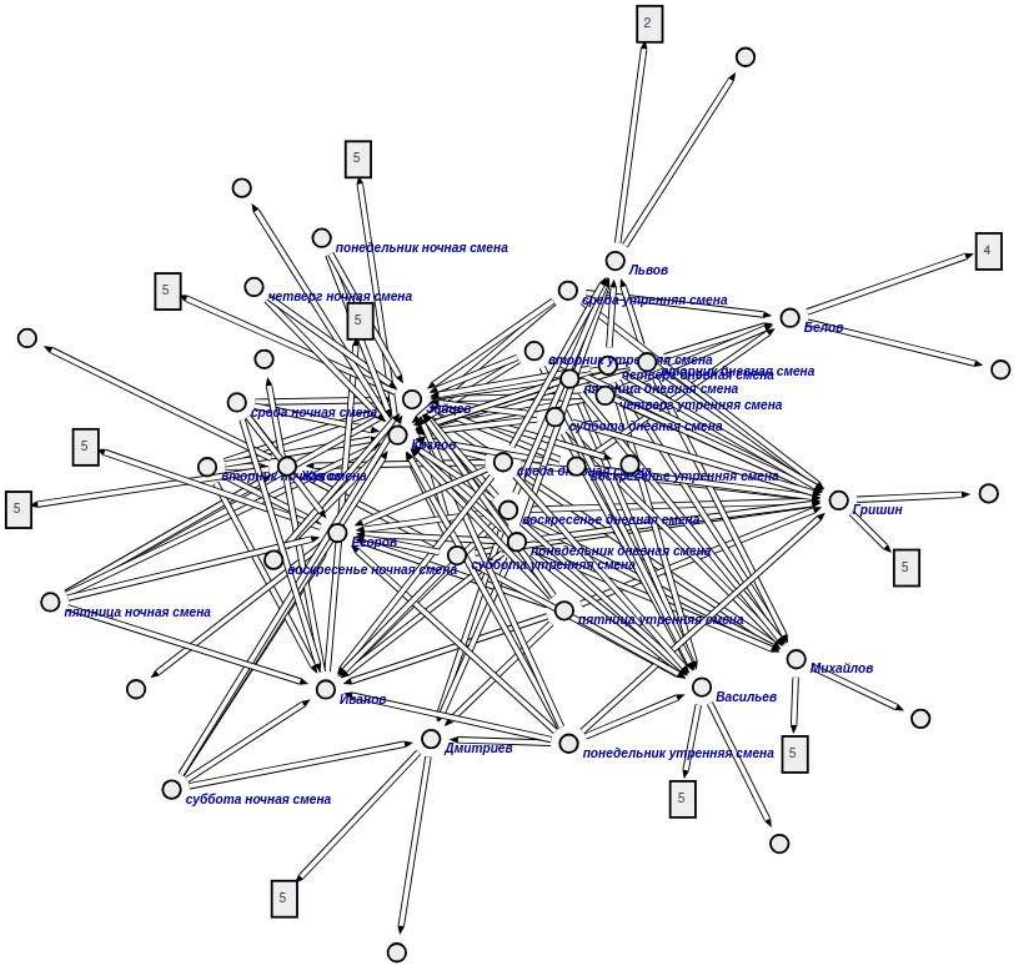
Проверяется: базовое построение расписания, учет типа смены, учет максимального числа смен, работа can_work, сценарии без смен и при нехватке персонала.

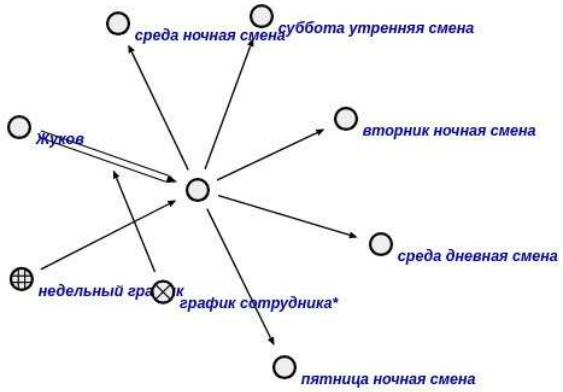
Тестирование с помощью *google-тестов*:

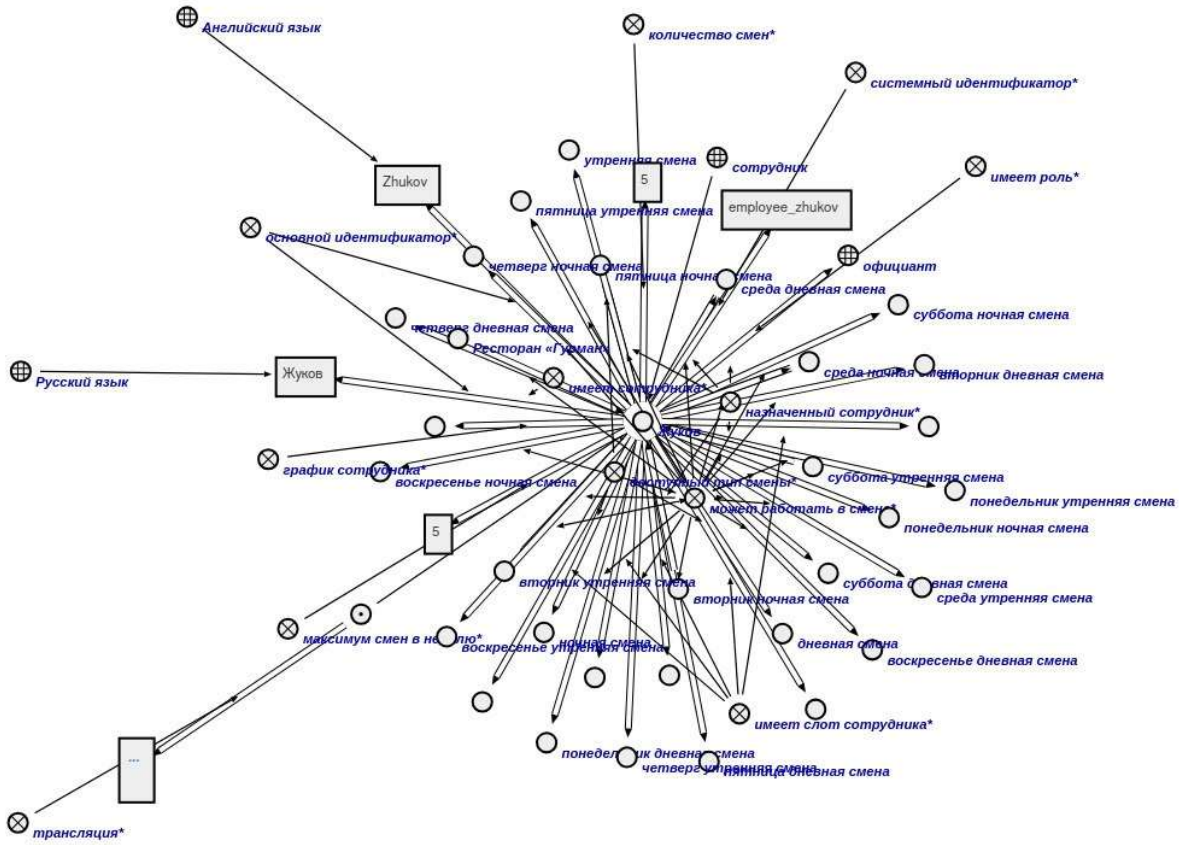


Работа системы через web интерфейс

Приведём несколько примеров графических визуализаций понятий:







ЗАКЛЮЧЕНИЕ

В ходе выполнения расчётной работы была разработана интеллектуальная система на базе технологии OSTIS для формирования недельного графика смен персонала ресторана. Предметная область полностью формализована: введены понятия «ресторан», «сотрудник», «смена», «тип смены», «недельное расписание» и «проблема укомплектования», а также необходимые отношения для ролей, доступности и ограничений по нагрузке. На языке C++ реализован агент BuildStaffScheduleAgent. Агент извлекает данные из sc-памяти, строит граф возможных назначений с учетом ролей и доступных типов смен, применяет алгоритм максимального потока для подбора назначений и формирует результат в виде семантической структуры с расписанием, резервами и флагом укомплектованности. Проведено тестирование: написаны и успешно пройдены модульные тесты, проверяющие базовые сценарии, учет типов смен, лимитов и случаев недостатка персонала. Решение интегрировано в OSTIS и готово к использованию.