# КП ППвИС(часть 1)

## Установка

Для выполнения первой части КП по курсу ППвИС разработан пример реализации волнового алгоритма поиска минимального пути в графе на базе библиотеки sc-memory.

На вашем компьютере должна быть установлена последняя версия ostis из репозитория: https://github.com/ShunkevichDV/ostis и QT Creator(версия больше 4.8).

Последние версии указанного примера и данного документа могут быть найдены в репозитории: https://github.com/ShunkevichDV/wave_find_path_sc_memory
или на сервере: Info/Studinfo/~Методическое обеспечение кафедры/~Учебные курсы/2 курс/ППвИС/1sem/КП 2014

1) Запустите терминал и перейдите в папку ostis, выполните команду git clone https://github.com/ShunkevichDV/wave_find_path_sc_memory

2) Перейдите в папку wave_find_path_sc_memory и скопируйте папку graph в ostis/kb

3) Запустите QT Creator. Запустив среду, необходимо выбрать пункт меню Open Project и в появившемся диалоговом окне указать файл CMakeLists.txt, находящийся в папке с примером. В следующем диалоговом окне среда предложит выбрать путь для сохранения файлов проекта (его можно не менять), необходимо нажать Run CMake и дождаться завершения обработки. В случае успешного завершения среда сообщит, что «Build files have been written to: <Указанный путь>»

4) После этого проект будет виден во вкладке Редактор(Edit). Перейдите в файл wavefindpath.cpp

5) Измените переменные в функции main

```
params.repo_path = "/home/<имя_пользователя>/ostis/kb.bin";
params.config_file = "/home/<имя_пользователя>/ostis/config/sc-web.ini";
params.ext_path = "/home/<имя_пользователя>/ostis/sc-machine/bin/extensions";
```

6) Перейдите в файл /home/<имя_компьютера>/ostis/config/sc-web.ini и измените в нем строчки Path и Directory на:

```
Path = /home/<имя_пользователя>/ostis/kb.bin
Directory = /home/<имя_пользователя>/ostis/sc-machine/bin/extensions
```

7) Пересоберите базу знаний (~/ostis/scripts$ ./build_kb.sh)

8) Нажмите на вкладку проект (Project) и перейдите во вкладку Run. Найтите строчку Run configuration и нажмите кнопку Add. Изменить конфигурацию в строчке Executable нажмите на кнопку Browse... выберите файл wave который должен находится по пути /home/<имя_компьютера>/ostis/sc-machine/bin/wave
*Примечание: Если файла wave нет, попробуйте выполнить пункт 7) еще раз*

9) Сохраните конфигурацию Ctrl+S и перейдите во вкладку Редактор(Edit)

10) Соберите и запустите проект. Сборка проекта — Shift+Ctrl+B, запуск проекта — Ctrl+R
Программа должна найти пути(если они есть) для графов которые находятся в папке ostis/kb/graph и завершится кодом 0

# Основные функции

**sc_addr** – основной тип данных, предназначенный для хранения адреса sc-элемента в памяти. По этому адресу мы можем хранить sc-узел (**sc_memory_node_new(context, sc_type_const)**), sc-ссылку (**sc_memory_link_new(context)**) или sc-дугу (**sc_memory_arc_new(context, sc_type_arc_pos_const_perm, v1, v2)**).

## Типы дуг:

- **sc_type_arc_pos_const_perm** - константная позитивная sc-дуга принадлежности
- **sc_type_arc_common** — sc-дуга общего вида (константность не задана, нужно задавать дополнительно)
- **sc_type_edge_common —** sc-ребро общего вида

## Функции для работы с sc-памятью:

- **sc_addr printed_vertex;**
  - объявляется переменная printed_vertex, которая содержит адрес узла printed_vertex в sc-памяти (sc-адрес)
- *printed_vertex* = **sc_memory_node_new(*context, sc_type_const*);**
  - в sc-памяти создается новый узел, sc-адрес которого возвращается переменной *printed_vertex*
- ***sc_memory_arc_new(context, type, printed_vertex, v1);***
  - функция создает дугу *с типом type (например, sc_type_arc_pos_const_perm) от узла printed_vertex к узлу v1*
- **sc_helper_resolve_system_identifier(*context, gr, &graph*);**
  -функция записывает в **graph* адрес элемента, имеющего системный идентификатор *gr*.
- **sc_memory_is_element(*context, label*);**
  -функция возвращает SC_TRUE, если элемент *label* существует в *sc-памяти*, и SC_FALSE, если элемент *lebel* удалён.
- **sc_memory_element_free(*context, new_wave*);**
  -функция удаляет элемент по адресу *new_wave* из sc-памяти.

## Макросы sc-памяти:

- **SC_ADDR_IS_EQUAL(*vertex, v1*);**
  - возвращает *true*, если *vertex* эквивалентен *v1*, и *false*, если *vertex* не эквивалентен *v1*
- **SC_ADDR_IS_NOT_EQUAL(*loc, element*);**
  - функция возвращает *true*, если *loc* не эквивалентен *element*, и *false*, если *loc* эквивалентен *element*
- **SC_ADDR_IS_EMPTY(*addr*);**
  - возвращает *true*, если адрес *addr* пустой, и *false*, если адрес *addr* непустой.

- **SC_ADDR_IS_NOT_EMPTY(*addr*);**
  - возвращает *true*, если адрес *addr* непустой, и *false*, если адрес *addr* пустой.

- **SC_ADDR_MAKE_EMPTY(*addr*); -** делает addr пустым (не ссылающимся ни на какой элемент в памяти).

## Функции, реализованные в примере:

- **find_vertex_in_set(*v1, printed_vertex*);**
  - функция проверяет, есть ли элемент *v1* во множестве *printed_vertex*. Возвращает SC_FALSE если элемента нет, и SC_TRUE если элемент есть.

- **printEl(*context, t_node*);**
  - функция выводит в консоль идентификатор узла *t_node*

- **get_edge_vertexes(*t_arc, v1, v2*);**
  - функция записывает начало дуги t_arc в v1, конец дуги t_arc в v2

- other_vertex = **get_other_vertex_incidence_edge(*t_arc, vertex*);**
  -функция возвращает переменной *other_vertex* вершину графа G0, которая связана с вершиной *vertex* дугой *t_arc*.

- **Функции для работы с sc-ссылками:**

  - **printContent(*context, element*);**
    - выводит на консоль содержимое узла *elemen*t

  - **getInt(*context, element*);**
    - возвращает содержимое узла как целое число

  - **getFloat(*context, element*);**
    - возвращает содержимое узла как число с плавающей точкой

  - **t_node = genIntNode(*context, number*);**
    - устанавливает целое число *number* в качестве содержимого узла *t_node*

  - **t_node = genFloatNode(*context, number*);**
    - устанавливает число с плавающей точкой *number* в качестве содержимого узла *t_node*

  - **Пример работы этих функций:**

```
sc_addr link = sc_memory_link_new(context);
//создаём sc-ссылку link, то есть sc-элемент с типом sc-ссылка

link = genIntNode(context, 123);
// устанавливаем число 123 в качестве содержимого ссылки link

cout << getInt(context, link) << endl;
// выводим содержимое ссылки link на консоль
```

## Представление графа в SCs-коде

В SCs и SCg граф G0 состоящий из трех вершин можно записать следующим образом:

SCs:

*graph -> G0;;*
*G0 -> rrel_arcs: ..arcs;;*
*G0 -> rrel_nodes: ..nodes;;*

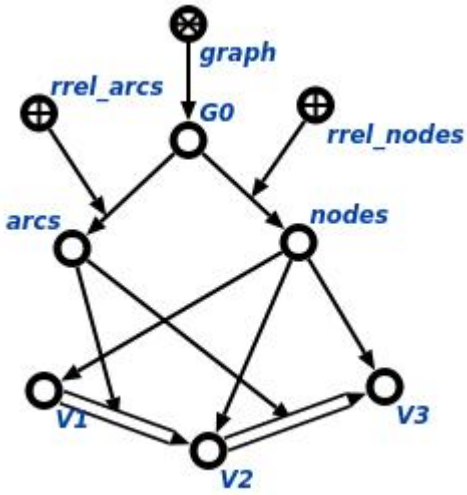
*..nodes -> V1;;*
*..nodes -> V2;;*
*..nodes -> V3;;*

*..arcs -> (V1 => V2);;*
*..arcs -> (V2 => V3);;*

*//arcs — множество дуг*
*//nodes — множество вершин*

*//rrel_arcs — дуга'*
*//rrel_nodes — вершина'*

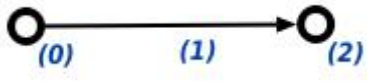
SCg:



# Итераторы

## Трёхэлементный итератор (итератор 3)

sc_iterator3 ищет конструкции типа

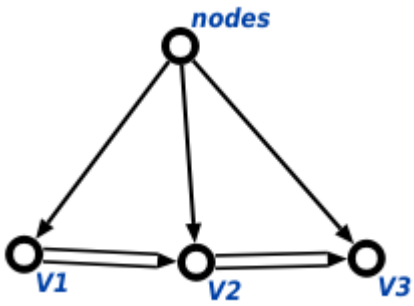



sc_iterator3 задаётся тремя параметрами, которые могут быть известными (**fixed**) и неизвестными (**assigned**).
При задании итератора известные элементы sc-памяти обозначаются буквой f, а неизвестные — буквой a.
В зависимости от того, какие узлы известны в Вашей задаче и какие необходимо найти, Вы можете модифицировать порядок параметров при задании трёхэлементного итератора.
Трёхэлементный итератор поддерживает следующие модификации:

- sc_iterator3_f_a_a_new(context, <список параметров>);
- sc_iterator3_a_a_f_new(context, <список параметров>);
- sc_iterator3_f_a_f_new(context, <список параметров>);

sc_iterator3 *nodes_it = **sc_iterator3_f_a_a_new(*context,***
***nodes,*** ***// (0) известный элемент***
***sc_type_arc_pos_const_perm,*** ***// (1) тип дуги***
***0*****);** ***// (2) тип элемента (0 - любой)***

- функция создает трёхэлементный итератор с названием nodes_it . Итератор будет искать для нашего G0 вершины V*1, V2* и *V3*, в данном случае эти вершины — неизвестное (2)




**sc_iterator3_next(nodes_it);**
- функция переходит к следующему элементу *nodes_it* и возвращает SC_TRUE, если такой элемент есть, в противном случае она возвращает SC_FALSE (при первом использовании находит первую конструкцию).

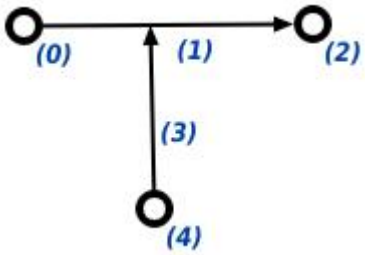
t_node = **sc_iterator3_value(*it, 2)*;**
- функция заносит в узел *t_node* элемент итератора *nodes_it* под номером 2 (наше неизвестное).

**sc_iterator3_free(nodes_it);**
-функция освобождает память от трёхэлементного итератора с адресом *nodes_it.*
***Всегда очищайте память!***

В переменной nodes хранится множество всех вершин графа G0. Итератор *nodes_it* найдет три удовлетворяющие условию конструкции (смотрите SCg). В переменную *t_node* на каждом шаге занесется вершина, на которую указывает итератор *nodes_it.*

## Пятиэлементный итератор (итератор 5)

sc_iterator5 ищет конструкции типа

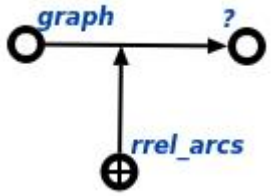



sc_iterator5 задаётся пятью параметрами, которые могут быть известными (**fixed**) и неизвестными (**assigned**).
При задании итератора известные узлы sc-памяти обозначаются буквой f, а неизвестные — буквой a.
В зависимости от того, какие узлы известны в Вашей задаче и какие необходимо найти, Вы можете модифицировать порядок параметров при задании пятиэлементного итератора.
Пятиэлементный итератор поддерживает следующие модификации:

- sc_iterator5_f_a_a_a_f_new(context, <список параметров>);
- sc_iterator5_a_a_f_a_f_new(context, <список параметров>);
- sc_iterator5_a_a_f_a_a_new(context, <список параметров>);
- sc_iterator5_f_a_a_a_a_new(context, <список параметров>);
- sc_iterator5_f_a_f_a_f_new(context, <список параметров>);
- sc_iterator5_f_a_f_a_a_new(context, <список параметров>);

sc_iterator5 *it = **sc_iterator5_f_a_a_a_f_new(*context,*
***graph,*** ***// (0)***
***sc_type_arc_pos_const_perm,*** ***// (1)***
***0,*** ***// (2)***
***sc_type_arc_pos_const_perm,*** ***// (3)***
***rrel_arcs*****);** ***// (4)***

-функция создает пятиэлементный итератор с названием *it.* Итератор будет искать пятиэлементные конструкции, *(?)* - неизвестное задано параметром *0*, чтобы учесть все типы sc-элементов




**sc_iterator5_next(*it*);**
- функция переходит к следующему элементу *it* и возвращает SC_TRUE, если есть такой элемент, в противном случае она возвращает SC_FALSE (при первом использовании находит первую конструкцию).

arcs = **sc_iterator5_value(*it, 2*);**
- функция заносит в узел arcs элемент итератора *it* под номером 2 (наше неизвестное).

**sc_iterator5_free(*it*);**

-функция освобождает память от пятиэлементного итератора с адресом *it.*
***Всегда очищайте память!***

В переменной graph хранится название нашего графа, в первом тесте это G0. Итератор it найдет единственную удовлетворяющую конструкцию и занесет в переменную arcs - узел с названием arcs (смотрите SCg) — наше неизвестное (2).

## Пример работы итераторов на основе функции вывода всех вершин

```
void showAllNodes() // Функция выводит на экран все вершины графа graph
{
   cout << "Vertex : ";
   sc_iterator5 *it = sc_iterator5_f_a_a_a_f_new(context,
         // Создаём итератор с названием it с пятью элементами (0, 1, 2, 3, 4)
                        graph,                                  // 0 - известный узел
                        sc_type_arc_pos_const_perm,     // 1 - неизвестная  дуга принадлежности
                        0,                                      // 2 - неизвестное
                        sc_type_arc_pos_const_perm,  // 3 - неизвестная дуга принадлежности
                        rrel_nodes);                        // 4 - известный узел

   if (SC_TRUE == sc_iterator5_next(it)) {  // Переходим к первому элементу итератора it, если
он есть
      nodes = sc_iterator5_value(it, 2);
      // В переменную nodes заносим 2 элемент, на который указывает итератор it
      // В переменной nodes хранится объект nodes, т. к.  nodes <- graph связаны отношением
         rrel_nodes

      sc_iterator3 *nodes_it = sc_iterator3_f_a_a_new(context,
      // Создаем итератор с названием nodes_it с тремя элементами (0, 1, 2)
                        nodes,                                  // 0 - известный узел
                        sc_type_arc_pos_const_perm,    // 1 - неизвестная дуга принадлежности
                        0);                                     // 2 - неизвестное

      while (SC_TRUE == sc_iterator3_next(nodes_it)) {
         // Если функция использована 1 раз: Переходим к первому элементу итератора
            nodes_it, если он есть
         // Если функция использована 2 раз: Переходим к следующему элементу итератора
            nodes_it, если он есть

         sc_addr t_node = sc_iterator3_value(nodes_it, 2);
         // В переменную t_node заносим 2 элемент, на который указывает итератор it
         // В переменной t_node хранится вершина графа graph (вершина может быть любой)

         printEl(context, t_node);  // Выводим в консоль название этой вершины
         cout << " ";
         }                                  // Возвращаемся к циклу while
      sc_iterator3_free(nodes_it);  // Очищаем память от трёхэлементного итератора
      }
   cout << endl;
   sc_iterator5_free(it);               // Очищаем память от пятиэлементного итератора
}
```