Описáние функций интерфейса пользователя - **ИССЛЕДОВАТЕЛЯ** системы SPF@home , позволяющих манипулировать представлéниями информационного графа алгоритма (ИГА)

I. Номерá операторов суть уникальные целые числа (32-х битовые со знаком) включая 0 (отрицательные числа использовать не рекомендуется).

II. Обозначéния boolean, integer, string суть типы переменных в Lua 5.3; т.к.корректные номерá операторов и ярусóв неотрицатéльны, при этом *возврáт отрицательного значения индицируéт ошибку* (см. далее).

III. Коды предусмóтренных ошибок:
ERR_RANGE_IN -1313 некорректный диапазон входных величин,
ERR_IN_DATA -888 некорректные входные данные
ERR_NOT_MASSIVE_EDGES -666 не сформирован массив Edges[][],
ERR_NOT_MASSIVE_TIERS -777 не сформирован массив Tiers[][],
ERR_NOT_MEMORY -133 нехвáтка памяти для массивов Edge[][] и/или Tiers[][],
ERR_METRIC -666.777 невозможно найти метрику вершины (оператора) или дуги ИГА,
ERR_CALC -777.666 код ошибки функций вычислений (семейство Calc...),
ERR_COMMON -123456789 общая ошибка.

IV. В ходе выполнения программа в текстовом окне может выдавать сообщения различного уровня: -I- информационный, -W- предупредительный, -E- уровень ошибки.

V. Основные данные для работы системы SPF@home находятся в файлах информационных зависимостей в алгоритме (ИГА). Файл текстовый, расширéние gv (программно данные располагаются во внутреннем динамическом 2D-массиве с именем Edges[][]). Формат gv соответствует DOT-стандарту описания графов. Пример описания напрáвленного (*direct*) ИГА нижеследует (здесь digraph – ключевое слово, GraphName – имя данного графа, a,b,c,d – целые числа, являющиеся идентификаторами вершин графа, симвóлика “b -> d ;” говорит о наличии дуги от вершины ‘a’ к вершине ‘b’; соответственно симвóлика “a -> b -> c -> ;” вещáет о наличии серии дуг (*путь в графе*) промежду вершинами a,b,c,d; в строке указанные сущности могут не разделяться пробелами или быть разделены их любым числом), после запятой могут располагаться любые комментарии:

```
// однострочный комментарий
# иной однострочный комментарий
/* многострочный
комментарий  к GV-файлу */
digraph GraphName {
 a -> b -> c -> d ; любой комментарий к описанным дýгам ИГА
 b -> d ; другой комментарий к данной дугé ИГА
}
```

ЯПФ информационного графа всегда строится на основе существующего массива Edges[][], причём собственно данные ЯПФ располагаются во внутреннем динамическом 2D-массиве именем Tiers[][]. Каждому Edges[][] может соответствовать множество Tiers[][]. Для правильного функционирования системы SPF@home важно в любом случае иметь корректное соответствие между представлением информационного графа в форме массивов Edges[][] и Tiers[][]. Массив Edges[][] вновь заполняется каждый раз при вызовах ReadEdges(), CreateTiersByEdges(), CreateTiersByEdges_Bottom() (аналоги F4, F5, Ctrl+F5 при "ручном" управлении, массив Tiers[][] – при вызовах CreateTiersByEdges(), CreateTiersByEdges_Bottom()).

VI. Формат (представле́ние) ИГА в форме ЯПФ включает ярусы (*Tiers*) от 0 до *N*. При этом узлы графа на нулевом ярусе не имеют входя́щих дуг и не соответствуют никаким операциям, а токмо определяют входные для данного алгоритма переменные (однако *не значения переменных*); при вычислении статистики (среднего по ярусам числа узлов, среднеквадратичного отклонения) узлы на 0-м ярусе не участвуют. С 0-го яруса нельзя переносить узлы вниз, под ним нельзя создать новый (пустой) ярус. При выводе ЯПФ номера узлов 0-го яруса снабжаются символом '«' слева от номера узла.

Ярусы 1÷*N* содержат узлы, соответствующие операторам – преобразователям информации, они участвуют в вычислении статистики, под каждым из них можно создавать новые ярусы. Узлы, не имеющие исходя́щих дуг (т.е. результат вычислений не используется в качестве входного операнда никаким иным узлом), считаются *выходными* и снабжаются символом '»' справа от номера узла.

VII. Файлы результатов расчётов выводятся файлами в подкаталог Out!Data относительно текущего каталога. Уникальность имён файлов для каждого расчёта достигается включением в имя файла метки времени с точностью до мсек:

- имена файлов вывода имеют вид: xxx!имя_Lua_скрипта!day-month-year_hour-min-sec-msec.txt. При автоматической обработке результатов большого количества экспериментов полное имя файла легко может быть разо́брано на составляющие с помощью строки формата "%s!%s!%d-%d-%d_%d-%d-%d-%d.txt", где xxx принимает значения protocol, stdin,stdout, stderr.

1. Информационные функции предоставляют работа́ющему в системе SPF@home ***клиенту*** (далее **ИССЛЕДОВАТЕЛЮ**) данные о *текущем* (в данный момент *обраба́тываемым*) ИГА

1.1 Информационные функции, предоставляющие данные о ИГА в целом (предполагается, что ИГА загру́жен из выбранного файла – для этого достаточно выполнить вызов ReadEdges(), тот же эффект дадут CreateTiersByEdges(), CreateTiersByEdges_Bottom(), см. ниже):

№ №	*Имя функции*	*Действие*	*Входные параметры*	*Выходные параметры*
1	integer GetCountEdges()	Возвращает число дуг (информационных связей операторов) в ИГА, становится известно сразу после считывания файла ИГА	Нет	Число дуг графа (*ENME)
2	integer GetCountOps()	Возвращает число операторов в данном ИГА (без входных операторов), становится известно сразу после считывания файла ИГА (*2)	Нет	Число операторов (*ENMT)
3	integer GetNumbOp	Возвращает номер оператора (начиная с 1) из общего числа GetCountOps() (*2)	Номер оператора по списку	Номер оператора (*ENMT)

	(integer Numb)			
4	integer GetCountOpsInput()	Возвращает число входных (исходные данные для алгоритма) операторов в данном ИГА (*2)	Нет	Число операторов (*ENMT)
5	integer GetNumbOpInput (integer Numb)	Возвращает номер оператора (начиная с 1) из списка входных операторов (числом GetCountOpsInput() (*2)	Номер оператора по списку	Номер оператора (*ENMT)
6	integer GetCountOpsOutput()	Возвращает число выходных (результат выполнения алгоритма) операторов в данном ИГА (*2)	Нет	Число операторов (*ENMT)
	integer GetNumbOpOutput (integer Numb)	Возвращает номер оператора (начиная с 1) из списка выходных операторов (числом GetCountOpsOutput() (*2)	Нет	Число операторов (*ENMT)
7	integer GetCountInEdgesByOp (integer Op)	Возвращает число *входящих* дуг для оператора Op (аналог величины "*степень захóда*" в теории графов)	Номер оператора Op	Число входящих дуг для оператора Op (для входных операторов ≡ 0, при несуществующем Op возвращается ERR_IN_DATA)
8	integer GetCountOutEdgesByOp (integer Op)	Возвращает число *выходящих* дуг для оператора Op (аналог величины "*степень исхóда*" в теории графов)	Номер оператора Op	Число выходящих дуг для оператора Op (для выходных операторов ≡ 0, при несуществующем Op возвращается ERR_IN_DATA)
9	integer GetNumbInEdgeByOp (integer Numb, integer Op)	Возвращает номер оператора, соответствующего началу Numb-той *входящей* относительно оператора Op дуге (*3)	Номер (начиная с 1) дуги Numb, номер оператора Op	Номер оператора (*ENME, *EC)
10	integer GetNumbOutEdgeByOp (integer Numb, integer Op)	Возвращает номер оператора, соответствующего концу Numb-той *выходящей* относительно оператора Op дуге (*3)	Номер (начиная с 1) дуги Numb, номер оператора Op	Номер оператора (*ENME, *EC)

1.2 Информационные функции, предоставляющие данные о ЯПФ информационного графа алгоритма (предполагается, что ИГА не только загру́жен из выбранного файла, но и создана ЯПФ – это достигается вызовами CreateTiersByEdges(), CreateTiersByEdges_Bottom(), см. ниже):

11	integer GetCountTiers()	Возвращает общее число (без нулевого) ярусóв ЯПФ для данного ИГА (*2)	Нет	Число ярусов ЯПФ (*ENMT)
12	integer GetCountOpsOnTier (integer Tier)	Возвращает число операторов на заданном ярусе Tier	Номер яруса Tier	Число операторов (*ENME)
13	integer GetOpByNumbOnTier (integer Numb, integer Tier)	Возвращает номер Numb-того по счёту оператора на ярусе Tier; диапазон Numb от 1 до GetCountOpsOnTier(Tier)	Номер (по счёту начиная с 1) оператора на ярусе Tier	Номер оператора (*ENME)
14	integer GetMaxTierMaybeOp (integer Op)	Возвращает *максимальный* (наиболее "нижний") номер яруса, на котором может находиться Op (при заданных информационных связях ИГА)	Номер оператора Op	Номер яруса, *первый и последний ярусы* не учитываются (*ENMT)
15	integer GetMinTierMaybeOp (integer Op)	Возвращает *минимальный* (наиболее "верхний") номер яруса, на котором может находиться Op (при заданных информационных связях ИГА)	Номер оператора Op	Номер яруса, *первый и последний ярусы* не учитываются (*ENMT)
16	integer GetTierFirstMinOps (integer Tier1, integer Tier2)	Возвращает *первый* (с min Tier - если минимумов несколько) номер яруса (в диапазоне Tier1-Tier2) с минимумом по ярусам числом операторов	Номера ярусов Tier1, Tier2	Номер яруса (*ENMT)
17	integer GetTierLastMinOps (integer Tier1, integer Tier2)	Возвращает *последний* (с max Tier - если минимумов несколько) номер яруса (в диапазоне Tier1-Tier2) с минимумом по ярусам числом операторов	Номера ярусов Tier1, Tier2	Номер яруса (*ENMT)
18	integer GetTierFirstMaxOps (integer Tier1, integer Tier2)	Возвращает *первый* (с max Tier - если максимумов несколько) номер яруса (в диапазоне Tier1-Tier2) с максимумом по ярусам числом операторов	Номера ярусов Tier1, Tier2	Номер яруса (*ENMT)
19	integer GetTierLastMaxOps (integer Tier1, integer Tier2)	Возвращает *последний* (с max Tier - если максимумов несколько) номер яруса (в диапазоне Tier1-Tier2) с максимумом по ярусам числом операторов	Номера ярусов Tier1, Tier2	Номер яруса (*ENMT)
20	integer GetTierByOp (integer Op)	Возвращает номер яруса, на котором находится оператор Op	Номер оператора Op	Номер яруса (*ENMT), при несуществующем Op возвращается ERR_COMMON

2. Акцио́нные функции позволяют **ИССЛЕДОВАТЕЛЮ** соверша́ть *эквивалентные* преобразования представле́ния информационного графа алгоритма:

21	boolean CreateTiersByEdges (string FileName)	Считывается ИГА-файл с именем FileName, организуется массив Edges[][], на его основе с создаётся первоначальная (с "подтя́нутостью" операторов *вверх*) ЯПФ (массив Tiers[][])	Имя ИГА-файла описания информационных связей в алгоритме (*.gv-файл)	Удачно - TRUE, нет - FALSE
22	boolean CreateTiersByEdges_Bottom (string FileName)	Считывается ИГА-файл с именем FileName, организуется массив Edges[][], на его основе с создаётся ЯПФ с "подтя́нутостью" операторов *вниз*	Имя ИГА-файла описания информационных связей в алгоритме (*.gv-файл)	Удачно - TRUE, нет - FALSE
23	integer AddTier (integer Tier)	Добавляет (пустой) ярус вслéдствие ("ниже") яруса Tier (кроме яруса 0)	Номер яруса Tier	(*ENME)
24	integer DelTier (integer Tier)	Уничтожает (пустой) ярус Tier	Номер яруса Tier	(*ENMH)
25	integer MoveOpTierToTier (integer Op, integer Tier)	Перено́с оператора Op на ярус Tier с учётом эквивалентности (сохранения информационных связей в ИГА) преобразований (*4)	Номер оператора Op, номер яруса Tier	При удаче возвращается 1, при неудаче <0 (*ENMT)
26	integer SwapOpsTierToTier (integer Op1, integer Op2)	Обмен (находящихся на *разных* ярусах) операторов Op1 и Op2 с учётом эквивалентности (сохранения информационных связей в ИГА) преобразований (*4)	Номера операторов Op1 и Op2	При удаче возвращается 1, при неудаче <0 (*ENMT)

3. Функции работы с гетерогенным полем параллельных вычислителей (*7)

3.1. Функции, относящиеся к параметрам ОПЕРАТОРОВ (эти параметры сравниваются с параметрами ВЫЧИСЛИТЕЛЕЙ для определения возможности выполнения данного оператора на поле параллельных вычислителей)

27	boolean LoadFileNameParamsOps (string FileName)	Задаёт и загружает файл FileName параметров ОПЕРАТОРОВ (если такой файл открыть не удалось, предпринимается попытка открыть файл по умолчанию OPERATORS.OPS)	Имя файла параметров ОПЕРАТОРОВ	Возвращает TRUE при удаче и FALSE при ошибке; в последнем случае также

				выдаёт предупреждение в текстовом окне вывода
28	string GetParamsByOp (integer Op)	Возвращает полную строку параметров ОПЕРАТОРА Op	Номер ОПЕРАТОРА Op	Поиск осуществляется среди начинающихся с '=n1/n2:' подстрок, далее среди '=Def:' (учитывается только первое вхожде́ние), при отсутствии возвращается пустая строка
29	integer GetCountParamsByOp (integer Op)	Возвращает число параметров ОПЕРАТОРА Op	Номер ОПЕРАТОРА Op	-.-.-
30	string GetNumbParamByOp (integer Numb, integer Op)	Возвращает подстроку номер Numb списка параметров ОПЕРАТОРА Op	Номер подстроки Numb (от 1 до GetCount ParamsByOp), номер оператора Op	-.-.-
31	string GetNameNumbParamByOp (integer Numb, integer Op)	Возвращает имя параметра номер Numb списка параметров ОПЕРАТОРА Op	Номер подстроки Numb, номер ОПЕРАТОРА Op	-.-.-
32	number GetValNumbParamByOp (integer Numb, integer Op)	Возвращает значение (*вещественное*) параметра номер Numb параметров ОПЕРАТОРА Op	Номер подстроки Numb, номер ОПЕРАТОРА Op	-.-.-

3.1. Функции, относящиеся к параметрам ВЫЧИСЛИТЕЛЕЙ (эти параметры сравниваются с параметрами ОПЕРАТОРОВ для определения возможности выполнения на данном вычислителе определённых операторов)

33	boolean LoadFileNameParamsCalcs (string FileName)	Задаёт и загружает файл FileName параметров ВЫЧИСЛИТЕЛЕЙ (если такой файл открыть не удалось, предпринимается попытка открыть файл по умолчанию CALCULATORS.CLS)	Имя файла параметров ВЫЧИСЛИТЕЛЕЙ	Возвращает TRUE при удаче и FALSE при ошибке; в последнем случае также выдаёт предупреждение в текстовом окне вывода

34	string GetParamsByCalc (integer Calc)	Возвращает полную строку параметров ВЫЧИСЛИТЕЛЯ номер Calc	Номер ВЫЧИСЛИТЕЛЯ Calc	Поиск осуществляется среди начинающихся с '=n1/n2:' подстрок, далее среди '=Def:' (учитывается только первое вхожде́ние), при отсутствии возвращается пустая строка
35	integer GetCountParamsByCalc (integer Calc)	Возвращает число параметров ВЫЧИСЛИТЕЛЯ Calc	Номер ВЫЧИСЛИТЕЛЯ Calc	-.-.-
36	string GetNumbParamByCalc (integer Numb, integer Calc)	Возвращает подстроку номер Numb параметров ВЫЧИСЛИТЕЛЯ Calc	Номер подстроки Numb (от 1 до GetCount ParamsByCalc), номер ВЫЧИСЛИТЕЛЯ Calc	-.-.-
37	string GetNameNumbParamByCalc (integer Numb, integer Calc)	Возвращает имя параметра номер Numb параметров вычислителя Calc	Номер подстроки Numb, номер ВЫЧИСЛИТЕЛЯ Calc	-.-.-
38	number GetMinValNumbParamByCalc (integer Numb, integer Calc)	Возвращает минимум (*вещественное*) диапазона значений параметра номер Numb параметров ВЫЧИСЛИТЕЛЯ Calc	Номер подстроки Numb, номер ВЫЧИСЛИТЕЛЯ Calc	-.-.-
39	number GetMaxValNumbParamByCalc (integer Numb, integer Calc)	Возвращает максимум (*вещественное*) диапазона значений параметра номер Numb параметров ВЫЧИСЛИТЕЛЯ Calc	Номер подстроки Numb, номер ВЫЧИСЛИТЕЛЯ Calc	-.-.-
40	integer GetCountCalcs()	Возвращает общее число вычислителей		Возвращает 0, если не загружен файл параметров вычислителей
41	integer CanExecOpCalc (integer Op, integer Calc)	При возможности выполнения оператора Op на вычислителе Calc возвращается число подтверждённых сравнений	Номер ОПЕРАТОРА Op, номер ВЫЧИСЛИТЕЛЯ Calc	Подробности см. в (*8)

4. Функции работы с метриками ДУГ и ВЕРШИН (операторов) (*8)

42	boolean LoadFileNameParamsEdges (string FileName)	Задаёт и загружает файл FileName метрик ДУГ графа (если такой файл открыть не удалось, предпринимается попытка открыть файл по умолчанию EDGES.MED)	Имя файла метрик ДУГ	Возвращает TRUE при удаче и FALSE при ошибке; в последнем случае также выдаёт предупреждение в текстовом окне вывода
43	boolean LoadFileNameParamsVertices (string FileName)	Задаёт и загружает файл FileName метрик ВЕРШИН графа (ОПЕРАТОРОВ); (если такой файл открыть не удалось, предпринимается попытка открыть файл по умолчанию VERTICES.MVR)	Имя файла метрик ВЕРШИН	Возвращает TRUE при удаче и FALSE при ошибке; в последнем случае также выдаёт предупреждение в текстовом окне вывода
44	number GetMetricOpByName (integer Op, string Name)	Возвращает значение метрики с именем Name вершины (оператора) Op, используя данные из файлов *.mvr	Номер вершины (оператора) Op, имя метрики Name	Если не найдено имя метрики Name в последовательностях номеров операторов и в последовательности =Def, то возвращается ERR_METRIC
45	number GetMetricEdgeByName (integer from_Op, Integer to_Op, string Name)	Возвращает значение метрики с именем Name дуги от вершины (оператора) from_Op до to_Op, используя данные из файлов *.med	Номера вершин (операторов) from_Op и to_Op, имя метрики Name	Если не найдено имя метрики Name в последовательностях номеров операторов и в последовательности =Def, то возвращается ERR_METRIC

5. Целевые функции работы с ОПЕРАТОРАМИ и ВЫЧИСЛИТЕЛЯМИ

46	integer CanExecOpCalc (integer Op, integer Calc)	При возможности выполнения ОПЕРАТОРА Op на ВЫЧИСЛИТЕЛЕ Calc возвращается (*положительное*) число подтверждённых сравнений по параметрам	Номер ОПЕРАТОРА Op, номер ВЫЧИСЛИТЕЛЯ Calc	Дополнительно см. (*8)
47	boolean PutParamsAll()	Заново считывает файлы параметров, корректирует синтаксис их содержимого и выводит в текстовый фрейм строки параметров ВЫЧИСЛИТЕЛЕЙ, ОПЕРАТОРОВ, метрик ДУГ и ВЕРШИН для	Нет	-.-.-

		визуального контроля (как исходный вариант, так и скорректированный)		
48	boolean TestCanExecAllOpsCalcs (integer Rule)	На основе загруженных данных тестирует возможность выполнения *всех* ОПЕРАТОРОВ на *всех* ВЫЧИСЛИТЕЛЯХ (последнее выдаётся в списке пар N|F, где N - номер ВЫЧИСЛИТЕЛЯ, F - метрика выполнения данного ОПЕРАТОРА на данном ВЫЧИСЛИТЕЛЕ).	При Rule#0 в текстовое окно выдаётся список ОПЕРАТОРОВ с ВЫЧИСЛИТЕЛЯМИ, на которых эти ОПЕРАТОРЫ могут быть выполнены; при Rule=0 список не выдаётся	При недостатке данных выдаётся предупреждение и работа подпрограммы заканчивается; дополнительно см. в (*9)

6. Функции определения “времени жизни данных” внутри текущего ЯПФ (используются для подготовки параметров для задачи определения правил временного хранения данных (напр., при оптимизации использования регистров общего назначения или др.)

49	integer GetOpByMaxTierLowerPreset (integer Op)	Возвращает оператор, информационно зависящий от Op и находящийся на ярусе с максимальным номером (если таких операторов несколько, возвращается последний из списка) ☞ Для получения общего числа операторов, информационно зависящих от оператора Op, используется вызов GetCountOutEdgesByOp, для перечисления этих зависимостей – GetNumbOutEdgeByOp	Номер оператора Op	Номер оператора; при отсутствии Tiers[][] возможен (*ENMT)
50	integer PutTimeLiveDataToTextFrame()	Создаёт и выводит в текстовое окно диаграмму времён жизни внутренних данных (результатов выполнения одних операторов и операндов иных операторов) для текущего ЯПФ	Нет	-.-.-
51	integer DrawDiagrTLD()	Отображает (вертикальную) диаграмму числа временных данных между ярусами ЯПФ среднеарифметическое число операторов указывается алым пунктиром	Нет	(*ENMT)
52	integer SaveTileLiveData (string FileName)	Создаёт и выводит в файл FileName диаграмму времён жизни внутренних данных (результатов выполнения одних операторов и операндов иных операторов) для текущего ЯПФ (по умолчанию расшире́ние полного имени файла TLD)	Имя файла FileName	-.-.-

7. Системные команды: диалоговые окна ввода/вывода, функции запуска внешних программ:

53	string InputDialog (string Caption, string Prompt, string Text)	Выводит стандартное диалоговое Windows-окно InputBox с двумя кнопками (OK и Cancel) 💣 Данный вызов возвращает строку, для преобразования строки в число можно использовать Lua-функцию tonumber, для обратного преобразования - tostring или string.format ☛ **В** режиме с командной строкой окно диалога не отображается и немедленно возвращается пустая строка	Caption - заголовок окна, Prompt - текст приглашения, Text - текст для ввода по умолчанию	Пользователь может редактировать текст, при нажатии кнопки OK возвращается изменённый текст, Cancel - исходный текст
54	integer MessageDialog (string Caption, string Text, string Buttons, integer Pictogram)	Выводит диалоговое Windows-окно CreateMessageDialog 🕮 Buttons - строка из 8 символов, причём любой символ кроме нулевого слева направо в этой строке вво́дит в набор кнопки Yes (6), No (7), OK (1), Cancel (2), Abort (3), Retry (4), Ignore (5) и All (8) соответственно (в скобках приведён код возвра́та при нажатии на данную кнопку). 🕮 Pictogram задаёт символ в окне - "Подтверждение" (0), "Информация" (1), "Ошибка" (2), "Предупреждение" (3) и отсутствие символа (4) ☛ **В** режиме с командной строкой окно диалога не отображается и немедленно возвращается (-1)	Caption - заголовок окна, Text - информационный текст, Buttons задаёт набор кнопок, Pictogram - рисунок в окне	Возвращает код нажатой кнопки (см. ранее). При любых некорректностях в задании параметров принимается наличие кнопки Yes и символа "Подтверждение".
55	integer lWinExec (string cmdLine, int cmdShow)	🖋 Практически полный аналог системного WINDOWS-вызова WinExec. Запуска́ет указанное строкой cmdLine приложение в режиме cmdShow (полный аналог системного вызова WinExec). 🛠 lWinExec("calc.exe", 9) откроет приложение “Калькулятор” в режиме совме́стной (конкуре́нтной) многозадачности 🛠 lWinExec("notepad.exe history.txt", 9) в текстовом редакторе notepad откроет файл history.txt.	Значение cmdShow задаётся ***числом*** (напр., число 9 соответствует SW_RESTORE); см. документацию по системному вызову WinExec	Возвращаемое значении >32 свидетельствует об успешном выполнении, в ином случае – ошибка (пояснение кодов ошибок см. в документации по системному вызову WinExec)
56	Integer lShellExecute (string Operation, string File, string Parameters, string Directory, integer cmdShow)	🖋 Практически полный аналог системного WINDOWS-вызова ShellExecute, 🛠 Напр., вызов lShellExecute("", "http://vbakanov.ru/spf@home/spf@home.htm", 0, 0, 1) в InterNet-броузере по умолчанию откроет WEB-страницу http://vbakanov.ru/spf@home/spf@home.htm описания данного проекта на WEB-сайте автора данной разработки	--.-.-	Возвращаемое значении >32 свидетельствует об успешном выполнении, иное – ошибка (пояснение кодов ошибок см. в документации по системному вызову ShellExecute)

57	integer lCreateProcess (string cmdLine, integer RuleParent, integer Priority, integer RuleMessage)	∾ Надстройка на системным WINDOWS-вызовом CreateProcess. ▶ При RuleParent=0 процесс-родитель ждет окончания работы потомка (при этом позволяя работать другим приложениям посредством вызовов ProcessMessages ▶ При RuleParent=1 процесс-родитель не ждёт окончания работы потомка ▶ При всех других значениях RuleParent процесс-родитель после запуска процесса-потомка завершается ➤ Priority=0/1/2/3 соответствует приоритетам запускаемого приложения REALTIME / HIGH / NORMAL / IDLE соответственно (все другие значения Priority соответствуют IDLE) ➡При RuleMessage#0 ошибки выполнения CreateProcess выдаются в стандартных всплывающих окнах Windows 🛠 Напр. вызов lShellExecute("Calc", 0, 2, 1) запускает на исполнение программу-калькулятор, при этом блоки́руя выполнение родительского приложения (до момента закрытия калькулятора)	Возвращает #0 при удачном запуске приложения-потомка

8. Прочие функции:

58	boolean AddLineToTextFrame (string str)	Добавляет строку str во текстовый фрейм (подокно) вывода текста (строка str может содержать символы '\n' перехода на новую строку); имеется укоро́ченный вариант этой функции - OutLine	Текстовая строка str	-
59	integer PutEdgesToTextFrame()	Выводит матрицу информационных связей (дуг графа) в текстовый фрейм	Нет	(*ENMT)
60	integer PutTiersToTextFrame()	Выводит ЯПФ в текстовый фрейм	Нет	(*ENMT)
61	integer PutParamsTiers()	Выводит в главное окно приложения-клиента и в файл протокола параметры ИГА и его ЯПФ; автоматически вызывается при исполнении CreateTiersByEdges, CreateTiersByEdges_Bottom и PutTiersToTextFrame (*5)	Нет	(*ENMT)
62	boolean ClearTextFrame()	Очищает текстовый фрейм	Нет	-
63	integer DrawDiagrTiers()	Отображает (вертикальную) диаграмму числа операторов на ярусах ЯПФ, среднеарифметическое (без нулевого яруса) число операторов указывается алым пунктиром	Нет	(*ENMT)

64	boolean ClearDiagrArea()	Очищает поле (вертикальной) диаграммы графической иллюстрации данных	Нет	-.-.-.-
65	boolean DelayMS (integer Sleep)	Ожидает Sleep миллисекунд (при зада́нии отрицательного значения - секунд)	Число миллисекунд (или секунд) ожидания Sleep	-
66	boolean SoundPlay (string FileName)	Проигрывает звуковой файл FileName в синхронном режиме (останов выполнения программы на время проигрывания)	Текстовая строка FileName	-
67	integer CountMovesZeroing()	Обнуляет текущее значение счётчика числа переносов операторов между уровнями функцией MoveOpTierToTier. *Пользователь по аналогии может определить любое количество подобных счётчиков (глобальных для Lua), задать це́ну каждого при́за/про́маха для заданной операции с целью количественного анализа применения данной стратегии*.	Нет	-
68	integer GetOpsMoves()	Возвращает текущее состояние счётчика числа перено́сов операторов между уровнями функцией MoveOpTierToTier	Нет	Текущее состояние счётчика
69	boolean ReadEdges (string FileName)	Читает файл описа́ния связей ИГА в массив Edges[][] (по умолчанию расшире́ние полного имени файла GV)	Имя ИГА-файла описания информационных связей в алгоритме (*.gv-файл)	Удачно - TRUE, нет - FALSE
70	boolean SaveEdges (string FileName)	Создаёт файл описания связей ИГА из массива Edges[][], существующий файл переписывается (по умолчанию расшире́ние полного имени файла GV, создаваемыё файл доступен для считывания ReadEdges)	-.-.-.-	-.-.-.-
71	boolean ReadTiers (string FileName)	Читает файл описания ЯПФ графа в массив Tiers[][] (по умолчанию расшире́ние полного имени файла TRS)	Имя файла описания ярусов	-.-.-.-
72	boolean SaveTiers (string FileName)	Создаёт файл описания ярусов ЯПФ из массива Tiers[][], существующий файл переписывается (по умолчанию расшире́ние полного имени файла TRS, создаваемый файл доступен для считывания ReadTiers)	-.-.-.-	-.-.-.-

73	integer SaveEdgesVizu (string FileName)	Создаёт (удобочита́емый) файл описания связей ИГА из массива Edges[][], существующий файл переписывается (по умолчанию расшире́ние полного имени файла GV_VZ)	Имя файла	(*ENME)
74	integer SaveTiersVizu (string FileName)	Создаёт (удобочита́емый) файл описания ярусов ЯПФ из массива Tiers[][], существующий файл переписывается (по умолчанию расшире́ние полного имени файла TRS_VZ)	-.-.-.-	(*ENMT)
75	integer SaveInOutOpVizu (string FileName)	Создаёт (удобочита́емый) файл списка входных и выходных дуг для каждого оператора ИГА из массивов Edges[][] и Tiers[][], существующий файл переписывается (по умолчанию расшире́ние полного имени файла INO_VZ) (*6)	-.-.-.-	(*ENME, *ENMT)
74	integer SaveParamsVizu (string FileName)	Создаёт (удобочита́емый) файл усреднённых параметров ИГА из массива Edges[][] (по умолчанию расшире́ние полного имени файла PRM_VZ)	-.-.-.-	(*ENME)

9. Асинхронные функции:

75	Integer LuaCallByTimer (string CommandLine, integer d_Ticks)	Запускает на выполнение строку CommandLine (*функция обратного вызова*) текста Lua через d_Ticks "ти́ков" (интервалов времени, величина тика в секундах задаётся параметром Interval секции [Tick_Interval] в файле SPF_CLIENT.INI	d_Ticks должно быть ≥ 0, CommandLine не должна быть пустой строкой	0 – всё нормально, -1 – задано отрицательное или нулевое d_Ticks, -2 – задана пустая строка. Дополнительно см. в (*10)

10. "Вычислительные" функции (семейство Calc…):

76	number CalcAverMeanOpsOnTiers()	Вычисляет средне-арифметическое чисел операторов по ярусам ЯПФ (без нулевого яруса)	Нет	При невозможности вычислить возвращается значение ERR_CALC
77	number CalcStdDevOpsOnTiers()	Вычисляет стандартное отклонение (несмещённое) чисел операторов по ярусам ЯПФ (без нулевого яруса)	Нет	-.-.-.-

* (*ENME) или (*ENMT) - возможен возврат значения ERR_NOT_MASSIVE_EDGES или ERR_NOT_MASSIVE_TIERS ("не существует массива Edges[][] или Tiers[][] для обработки"); (*EC) - ERR_COMMON - общая ошибка (обычно "возвращаемое значение может быть некорректным").

(*2) Пример:

В случае такого содержимого файла ИГА:	*Иллюстрация функционирования GetCountOpsInput(), GetNumbOpInput()*	*Иллюстрация функционирования GetCountOps(), GetNumbOp()*	*Иллюстрация функционирования GetCountOpsOutput(), GetNumbOpOutput()*	*При этом ЯПФ такая:*
17 12 14 12 3 11 2 11 1 13 4 14 9 14 10 3 4 2 7 2 8 1 6 4 6 6 5 5 7 7 9 5 8 8 10	for i=1,GetCountOpsInput() do AddLineToTextFrame (i.. ": ".. GetNumbOpInput(i)) end 1: 12 2: 11 3: 13	for i=1,GetCountOps() do AddLineToTextFrame (i.. ": "..GetNumbOp(i)) end 1: 14 2: 3 3: 2 4: 1 5: 4 6: 9 7: 10 8: 7 9: 8 10: 6 11: 5	for i=1,GetCountOpsOutput() do AddLineToTextFrame (i.. ": "..GetNumbOpOutput(i)) end 1: 9 2: 10	0/3: «12 «11 «13 1/4: 14 3 2 1 2/1: 4 3/1: 6 4/1: 5 5/2: 7 8 6/2: 9» 10»

Функции GetCountOpsInput(),GetCountOps() и GetCountOpsOutput() возвращают правильный результат сразу после загрузки файла ИГА функцией ReadEdges() даже если ещё не построена ЯПФ графа (все три обновляются после постройки ЯПФ).

Функции GetNumbOpInput(),GetNumbOp() и GetNumbOpOutput() возвращают номера операторов в порядке просмотра файла ИГА - по столбцам слева направо и по строкам сверху вниз (как принято при чтении в Европе).

В число ярусóв ЯПФ входят я́русы с 1-го по последний (номером nTiers), на нулевом ≡ входном ярусе свершается тóкмо объявление переменных; общее число операций (возвращается GetCountOps) также суть сумма операций на ярусáх [1÷nTiers], нулевой ≡ входной - ярус не учитывается такóжде. При выводе ЯПФ в текстовое окно (по F5 или Ctrl-F5 или вы́зовом PutTiersToTextFrame() *входные* и *выходные* операторы индицируются символом "двойные угловые скобки" слева или справа относительно номера оператора соответственно.

(*3) Возврат значения ERR_COMMON функциями GetNumbInEdgeByOp() и GetNumbOutEdgeByOp() означает, что оператор Op является входным и выходным (соответственно) для данного информационного графа.

(*4) Перено́с операторов "от" и "на" нулевой≡входной ярус невозможен (ибо этот ярус представляет собой лишь процедуры присваивания исходных данных переменным программы). Вызов MoveOpTierToTier(Op,Tier) проверяет значение Tier на принадлежность диапазону от GetMaxTierMaybeOp(Op) до GetMaxTierMaybeOp(Op); при выходе за эти границы MoveOpTierToTier() выполнен не будет.

(*5) Т.к. вывод ЯПФ в текстовый фрейм посредством вызова PutTiersToTextFrame() требует определённого времени, при длительных преобразованиях ЯПФ неразумно многократно вызывать PutTiersToTextFrame(); с целью регистрации параметров ЯПФ достаточно каждый раз вызывать PutParamsTiers(). Данный вызов обновляет текст в нижней части окна текстового вывода (см. ниже) и добавляет соответствующие строки в файл протокола выполнения сценария (файл protocol!*.txt в подката́логе Out!Data относительно текущего каталога); при PutParamsTiersOnTextFrame#0 (один из параметров файла настроек spf_client.ini) эта же информация добавляется в виде строк и в сам текстовый фрейм окна текстового вывода.

Операторов= 271, дуг= 451, ярусов= 30 ‡‡ средн. опер./ярус= 9.033, СКО шир.ЯПФ= 4.716, CV= 0.5221 ‡‡ операторов на ярусе/ярус (min:max)= 1/13:15/15 ‡‡ неравном. ширины ЯПФ (по ярусам 1-30)= 15.000 ‡‡ вариативность ЯПФ: Vo|Vt|Vot= 0.3173|16.6|5.268 ‡‡ средняя длина дуги: 3.756 ярусов ЯПФ	

Здесь (расшифровка величин):

- ”средн. опер./ярус=” – среднеарифметическое число операторов по всем ярусам ($\overline{W}$) ;
- “СКО шир.ЯПФ=” – среднеквадратичное отклонение числа операторов по ярусам ($\sigma=\sqrt{\frac{1}{N}\sum_{i=1}^{i=N}(W_i-\overline{W})^2}$), где N – число ярусов ЯПФ;
- “CV=” – коэффициент вариации числа операторов по ярусам ($CV = {}^{\sigma}/_{\overline{W}}$) ;
- “операторов на ярусе/ярус (min:max)=” – минимальное число операторов по всем ярусам/номер яруса, где этот минимум достигается и то же для максимального числа операторов;
- “неравном. ширины ЯПФ (по ярусам 1-N)=” – отношение j_{max}/j_{min}, где j_{max} и j_{min} – максимальное и минимальное число операторов среди всех ярусов ЯПФ ;
- “вариативность ЯПФ: Vo|Vt|Vot=” – где $Vo=\sum\frac{O_i}{O}$; $Vt=\frac{1}{N}\sum\left(T_i^{max}-T_i^{min}\right)$, $Vot= Vo\times Vt$, O_i - число обладающих вариативностью операторов, O - общее число операторов, T_i^{max} и T_i^{min} - максимальный и минимальный номера ярусов данной ЯПФ, на которых может располагаться i-тый оператор (фактически диапазон возможного их расположения), N – общее число операторов. Vo учитывает вклад числа операторов со свойством вариабельности, Vt - величину собственно вариабельности, оба они дают ноль при отсутствии вариабельности и монотонно возрастают с её увеличением, *Vot* - суперпозиция предыдущих параметров в форме их произведения ;
- “средняя длина дуги: XXX ярусов ЯПФ” – среднеарифметическая длина дуг в единицах ярусов ЯПФ.

(*6) Выполнение SaveInOutOpVizu() возможно только после создания ЯПФ (массива Tiers[][]).

(*7) Эти функции используют информацию из файлов XXX.OPS и XXX.CLS (OPS, CLS являются *рекомендуемыми* расширениями файлов, XXX - обычно имя проекта, при отсутствии таких файлов используются имена файлов по умолчанию OPERATORS.OPS и CALCULATORS.CLS).

Файлы типов OPS и CLS используются всегда вместе и служат для определения, на каких ВЫЧИСЛИТЕЛЯХ (Calcs) может быть выполнен данный ОПЕРАТОР (Ops) в случае гетерогенного поля параллельных вычислителей.

Формат файлов *.OPS параметров ОПЕРАТОРОВ (*Operators*) следующий (допускаются переносы со строки на строку и др.):

^=n1/n2:^-nameParameter1^Val1^-nameParameter2^Val2^ ... ; комментарий до конца строки

где:

n1,n2	- диапазон ("от/до" включительно) номеров ОПЕРАТОРОВ, для которых задаются далее следующие параметры (n1,n2 - целые числа, символы равенства, прямого слэша и двоеточия обязательны);
nameParameter1	- имя параметра - начинается с символа дефис, первый символ после дефиса обязательно буква на латинице, далее любые символы (кроме пробела), имя параметра регистрозави́симо; после =n1/n2: должен следовать хотя бы один параметр;
Val1	- значение данного параметра (интерпретируется как *вещественное*, разделитель целой и дробной частей - точка);
^	- один или сколько угодно пробелов.

Формат файлов *.CLS параметров то же для ВЫЧИСЛИТЕЛЕЙ (*Calculators*):

^=n1/n2:^-nameParameter1^minVal1^maxVal1^-nameParameter2^minVal2^maxVal2^ ... ; комментарий до конца строки

где:

n1, n2	- диапазон ("от/до" включительно) номеров ВЫЧИСЛИТЕЛЕЙ, для которых задаются далее следующие параметры (n1,n2 - целые числа, символы равенства, прямого слэша и двоеточия обязательны);

nameParameter1	- имя параметра - начинается с символа дефис, первый символ после дефиса обязательно буква на латинице, далее любые символы (кроме пробела), имя параметра регистрозави́симо; после =n1/n2: должен следовать хотя бы один параметр;
minVal1 и maxVal1	- минимум и максимум значений данного параметра (интерпретируется как *вещественные*, разделитель целой и дробной частей - точка);
^	- один или сколько угодно пробелов.

(*8) Эти функции используют информацию из файлов XXX.MVR и XXX.MED (MVR, MED являются *рекомендуемыми* расширениями файлов, XXX - обычно имя проекта, при отсутствии таких файлов используются имена файлов по умолчанию VERTICES.MVR и EDGES.MED).

Формат файлов *.MVR параметров ме́трик ВЕРШИН (mvr) определяет значения метрик для заданного диапазона номеров вершин графа (в нашем случае вершины ассоциируются с ОПЕРАТОРАМИ):

^=n1/n2:^-nameMetric1^Val1^-nameMetric^Val2^ ... ; комментарий до конца строки

где:

n1,n2	- диапазон ("от/до" включительно) номеров ОПЕРАТОРОВ, для которых задаются далее следующие параметры (n1,n2 - целые числа, символы равенства, прямого слэша и двоеточия обязательны);
nameMetric1	- имя метрики - начинается с символа дефис, первый символ после дефиса обязательно буква на латинице, далее любые символы (кроме пробела), имя параметра регистрозави́симо; после =n1/n2: должен следовать хотя бы один параметр;
Val1	- значение данной метрики (интерпретируется как *вещественное*, разделитель целой и дробной частей - точка);
^	- один или сколько угодно пробелов.

Формат файлов *.MED параметров ме́трик ДУГ (edg) определяет значения метрик для заданного диапазона дуг (каждая дуга задаётся номерами двух вершин):

^=n1/n2|m1/m2:^-nameMetric1^Val1^-nameMetric2^Val2^ ... ; комментарий до конца строки

где:

n1,n2	- диапазон ("от/до" включительно) номеров ОПЕРАТОРОВ, для которых дуга является исходя́щей; n1,n2 - целые числа;
m1,m2	- диапазон ("от/до" включительно) номеров ОПЕРАТОРОВ, для которых дуга является входя́щей; m1,m2 - целые числа, символы равенства, прямого слэша и двоеточия обязательны;
nameMetric1	- имя метрики - начинается с символа дефис, первый символ после дефиса обязательно буква на латинице, далее любые символы (кроме пробела), имя параметра регистрозави́симо; после =n1/n2|m1/m2: должен следовать хотя бы один параметр;
Val1	- значение данной метрики (интерпретируется как *вещественное*, разделитель целой и дробной частей - точка);
^	- один или сколько угодно пробелов.

Для проверки корректности зада́ния параметров служит вы́зов PutParamsAll(), выводящий в текстовый фрейм ИСХОДНЫЕ и ОТКОРРЕКТИРОВАННЫЕ (синтаксически неверные подстроки удаляются) строки параметров ВЫЧИСЛИТЕЛЕЙ, ОПЕРАТОРОВ и СООБЩЕНИЙ (эти строки заключены в вертикальные черты). Если строка пустая (символ ||), то параметры не загружались или файлы параметров пустые.

Напр., запомнить в переменной M значение метрики с именем Times вершины (оператора) N можно с помощью вызова M=GetMetricOpByName(N, “Times”). Следует помнить, что при поиске всегда возвращается значение, соответствующее ***первому вхожде́нию*** заданного номера вершины (оператора) и имени метрики в файле; в противном случае возвращается ERR_METRIC. Например, при нижеприведённом содержимом *.mvr-файла вызов GetMetricOpByName(100, “Times”) вернёт ERR_METRIC (а не 100, как казалось бы); таким же образом функционируют вызовы GetMetricEdgeByName (обрабатывает файлы *.med), CanExecOpCalc, PutParamsAll, TestCanExecAllOpsCalcs (обрабатывают файлы *.ops и *.cls).

=1/200: -Time 12.221 -Time1 13.3
=Def: -Time 12.331 -Time1 13.33
=100/100: -Times 0

Исходя из сказанного, удобной конструкцией может быть такая:

projectName = "e313_o206_t32" – зададим имя проекта
--

```
LoadFileNameParamsCalcs( projectName .. ".cls" ) -- или LoadFileNameParamsCalcs( projectName )
LoadFileNameParamsOps( projectName .. ".ops" ) -- или LoadFileNameParamsOps( projectName )
LoadFileNameParamsVertices( projectName .. ".mvr" ) -- или LoadFileNameParamsVertices( projectName )
LoadFileNameParamsEdges( projectName .. ".med" ) -- или LoadFileNameParamsEdges( projectName )
--
--
ReadEdges( projectName .. ".gv" ) -- или ReadEdges( projectName )
```

Общее число ВЫЧИСЛИТЕЛЕЙ от 1 до max(n1,n2) по всем диапазонам (определяется вызовом GetCountCalcs()); вариантом является описание =n1/n2: с последующим (заве́домо уникальным) именем параметра ВЫЧИСЛИТЕЛЯ (ВЫЧИСЛИТЕЛЕЙ), не совпадающим ни с одним из параметров ОПЕРАТОРОВ.

В случае перекры́тия имён параметров в разных строках в расчёт принимается ***первое вхожде́ние*** (в порядке чтения CLS, OPS, MVR и MED-файлов. Для просмотра (вывода в текстовое окно) параметров всех операторов можно использовать такую конструкцию:

```
for i=1,GetCountOps() do  -- по общему числу операторов
  Op = GetNumbOp( i ) -- номер ("имя") оператора
  AddLineToTextFrame ( "#Op= "..Op.." ; "..GetParamsByOp(Op) )
end -- конец цикла по операторам
```

То самое для вычислителей:

```
for Calc=1,GetCountCalcs() do  -- по номерам вычислителей
  AddLineToTextFrame ( "#Calc= "..Calc.." ; "..GetParamsByCalc(Calc))
end -- конец цикла по вычислителям
```

(*8) При проверке возможности выполнения данного ОПЕРАТОРА Op на заданном ВЫЧИСЛИТЕЛЕ Calc с помощью вызова CanExecOpCalc(Op,Calc) ОПЕРАТОР фактически является *клиентом*, который запрашивает возможность обслуживания (выполнение) у *сервера* (ВЫЧИСЛИТЕЛЯ). В соответствии с этим условие выполнимости заключается в удовлетворении всех запросов *клиента* Op данным *сервером* Calc (фактически список параметров ОПЕРАТОРА связан логикой "И"). Если численное значение n каждого из запросов *клиента* попадает в диапазон (включая границы) n1-n2 численных значений *сервера* (с тем же именем параметра, естественно), вызов CanExecOpCalc(Op,Calc) возвращает число больше нуля (число удовлетворяющих условию выбора параметров - оно равно, конечно, GetCountParamsByOp(Op) для данного ОПЕРАТОРА); в противном случае возвращается (для информации) значение меньше нуля, равное числу совпадений (оно по модулю меньше GetCountParamsByOp(Op).

Если строка параметров ОПЕРАТОРА Op пустая - он может выполняться на любом ВЫЧИСЛИТЕЛЕ; если строка параметров ВЫЧИСЛИТЕЛЯ Calc пустая - на нём не может выполняться ни один ОПЕРАТОР.

Нижеприведён фрагмент Lua-кода для проверки выпо́лнимости *всех* ОПЕРАТОРОВ на *всех* ВЫЧИСЛИТЕЛЯХ (вызов PutParamsAll() даст бо́льшую информацию):

```
for Calc=1,GetCountCalcs() do  -- по общему числу вычислителей
for iOp=1,GetCountOps() do  -- по общему числу операторов
  Op = GetNumbOp( iOp ) -- номер ("имя") конкретного  оператора
  AddLineToTextFrame (string.format("#Op=%d #Calc=%d |%s|%s| !%d!",  -- результат сравнения в поле !x!
                                Op,Calc,GetParamsByOp(Op),GetParamsByCalc(Calc),CanExecOpCalc(Op,Calc)))
end -- конец по операторам
end -- конец по вычислителям
```

(*9) Весьма практически-полезный сервис предоставляет API-функция TestCanExecAllOpsCalcs, которая при нулевом значении параметра возвращает TRUE, если каждый ОПЕРАТОР может быть выполнен хотя бы на одном ВЫЧИСЛИТЕЛЕ и FALSE, ежели хоть один ОПЕРАТОР не может быть выполнен ни на одном ВЫЧИСЛИТЕЛЕ. При ненулевом входном параметре выдаётся (для каждого ОПЕРАТОРА) список ВЫЧИСЛИТЕЛЕЙ, на которых данный ОПЕРАТОР может быть выполнен. В (проблемном) случае возвра́та FALSE полезно вызвать также PutParamsAll() для понимания проблемы. В общем случае практично использовать такой Lua-шаблон:

```
if( not TestCanExecAllOpsCalcs(0) ) then -- имеются проблемы с выполни́мостью части ОПЕРАТОРОВ..!
  AddLineToTextFrame( "\nВнимание! Проблемы с выполни́мостью ОПЕРАТОРОВ на ВЫЧИСЛИТЕЛЯХ..\n" ) -- предупреждение в текстовом фрейме
  TestCanExecAllOpsCalcs(1) -- выдать полную информацию о выполнИмости ОПЕРАТОРОВ на ВЫЧИСЛИТЕЛЯХ
  PutParamsAll() -- распечатать строки параметров для проверки корректности их (строк параметров) определения
  return -- или end etc с целью прекращения дальнейшего выполнения Lua-программы для избежа́ния критических ошибок
end
```

(*10) LuaCallByTimer реализуется вызовом нового состояния Lua, прикреплённого к основному Lua-состоянию (не является отдельным потоком операционной системы и не может физически выполняться одновременно с другими). Примером вызова может служить вызов LuaCallByTimer("Fun(100)",10) в тексте Lua-программы, при этом должна быть описана Lua-функция (*функция обратного вызова*) Fun(i), которая и вызовется чрез 10 тиков (формально строка CommandLine может содержать любую корректную последовательность команд Lua):

```
function Fun( i )
  print("-- Вызвана функция Fun(" .. i .. ") --")
  OutLine("\n-- Lua: Вызвана функция Fun(" .. i .. ") ")
end
```

Т.к. реальной многозадачности не имеется (она лишь *имитируется*), излишне большое число таким образом вызываемых функций может привести к нестабильности работы системы SPF@home. С целью недопуще́ния этого рекомендуется минимизировать время выполнения функции обратного вызова и увеличение значения Interval.

“Ручнойt” вариант выполнения основных функций системы SPF@home.

Некоторые (базовые) действия в системе SPF@home могут быть выполнены “вручную” (без выполнения Lua-скриптов). В основном эта возможность применяется для уточнения исходных и промежуточных данных расчёта.

Нажатие клавиши F4 позволяет выбрать файл ИГА, запомнить его в массиве Edges[][] и вывести в текстовое окно (данное действие фактически равноценно оследовательности вызовов ReadEdges() и PutEdgesToTextFrame(), при этом ЯПФ не создаётся). Здесь первое число – номер дуги, далее следуют разделённые символами “->” два числа, определяющие напра́вленную дугу (пример вывода ИГА-файла squa_equ_2.gv приведён ниже):

```
-=- Дуг ИГА = 21 -=-
#1: 100 -> 109
#2: 100 -> 110
#3: 101 -> 104
#4: 102 -> 107
#5: 102 -> 108
#6: 103 -> 105
#7: 104 -> 105
#8: 105 -> 106
#9: 106 -> 107
#10: 106 -> 108
#11: 107 -> 109
#12: 108 -> 110
#13: 111 -> 100
#14: 111 -> 101
#15: 112 -> 102
#16: 112 -> 103
#17: 113 -> 104
#18: 114 -> 100
#19: 114 -> 103
#20: 115 -> 101
#21: 116 -> 102
```

ЯПФ графа в “верхней” канонической форме строится на основе прочитанного ИГА-файла и выводится в текстовое окно после нажатия клавиши F5 в следующей форме (реализуется последовательностью вызовов ReadEdges(), CreateTiersByEdges(), ClearDiagrTiers(), DrawDiagrTiers(), PutTiersToTextFrame()):

```
-=- Ярусов ЯПФ = 6 -=-
0|6:  «111 «112 «113 «114 «115 «116
1|4:  100 101 102 103
2|1:  104
3|1:  105
4|1:  106
5|2:  107 108
6|2:  109» 110»
```

Здесь каждая строка соответствует ярусу ЯПФ; строка начинается символами вида N|M (где N – номер яруса, M – число операторов на нём, далее следует список принадлежащих ярусу операторов, причём входные данные предваряются символом ‘«‘, а выходные завершаются символом ‘»’).

При нажатии Ctrl+F5 выдаётся ЯПФ в “нижней” форме (вместо CreateTiersByEdges() вызывается CreateTiersByEdges_Bottom()); в обоих случаях отрисовывается ленточный график шири́н ЯПФ и в нижней части окна текстового вывода распечатываются параметры построенной ЯПФ (вызов PutParamsTiers()). При использовании F5 или Ctrl+F5 содержимое массивов Edges[][] и Tiers[][] обновляется.

Вызов PutDataLiveDiagrTotextFrame() позволяет построить (по текущему ЯПФ) и вы́вести в текстовое окно т.н. “Диаграмму времени жизни данных” в программе, исполняемой в соответствие с заданным ЯПФ расписанием (для сохранения этих данных в файл служит вызов SaveDataLiveDiagr). При этом принимается, что данные создаются в результате выполнения конкретного оператора и должны сохраняться некоторым образом (обычно в регистрах общего назначения - РОН) до момента их использования (в качестве операндов иных операторов); в дальнейшем эти данные не нужны. Это является идеологе́мой для постано́вки и решения задач по оптимизации использования РОН в практике. Форма вывода данных вызовом PutDataLiveDiagrTotextFrame() следующая:

```
-=- Интервалов ЯПФ = 7 -=-
0/1|6:  «111|0–›1 «112|0–›1 «113|0–›2 «114|0–›1 «115|0–›1 «116|0–›1
1/2|5:  «113|0–›2 100|1–›6 101|1–›2 102|1–›5 103|1–›3
2/3|4:  100|1–›6 102|1–›5 103|1–›3 104|2–›3
3/4|3:  100|1–›6 102|1–›5 105|3–›4
4/5|3:  100|1–›6 102|1–›5 106|4–›5
5/6|3:  100|1–›6 107|5–›6 108|5–›6
6/$|2:  109»|6–›$ 110»|6–›$
```

Здесь каждая строка представляет собой интервал между двумя ярусами ЯПФ и начинается форматом n1/n2|m:, где n1 и n2 – номера образу́ющих интервал ярусов (условно “верхний” и “нижний”), m – число данных (без конкретизации размера), актуальных в этом интервале. Далее (через пробел) следует список параметров этих данных в формате m|n1–›n2, где m – номер оператора, результатом выполнения которого является данное, n1 и n2 – номера ярусов ЯПФ, на которых это данное произведено (в виде результата выполнения оператора m) и на котором *последний раз* использовано (в качестве операнда) соответственно; после этого данное не нужно. Для определения физического размера данных (напр., в байтах) можно использовать вызов GetMetricEdgeByName(), возвращающий значения метрик дуг ИГА (конечно, предварительно настроив соответствующий MED-файл).

В целом вызов PutDataLiveDiagrTotextFrame/SaveDataLiveDiagr позволяет произвести начальную оценку параметров файла РОН, предназначенных для временного хранения данных между операциями. Более сложную работу (общая задача распределения регистров и оптимизация функционирования РОН) предлагается проводить на уровне скриптового языка системы SPF@home или сторо́нней программой, которая может быть вызвана в качестве процесса-потомка; здесь может быть использована информати́вная функция GetOpByMaxTierLowerPreset() или считанные данные из записанного с помощью SaveDataLiveDiagr() файла. При использовании вызова SaveDataLiveDiagr() для упроще́ния распознавания отдельных параметров в первой строке этого файла указывается общее число межъярусных промежутков, а символы “»” и “»” не выводятся .

Как нача́ть писа́ть (*разрабатывать*) скрипты на языке Lua?

1. Манипуляции (набивка, редактирование и др.) исходных текстов на Lua происходит в центральном фрейме главного окна клиентской части системы SPF@home. **ИССЛЕДОВАТЕЛЮ** доступны стандартные функции работы с текстом – прочитать Lua-скрипт из файла (Ctrl-O), сохранить скрипт в файл (Ctrl-S), сохранить с текущим именем (F2), передать текст скрипта в Notepad или MS Word (для анализа или дополнительного просмотра), печать текста скрипта, изменить начертание текста (параметры запоминаются в файле конфигурации), поиск/замена фрагментов текста, запуск скрипта на исполнение (F9) в виде кнопок на панели сверху этого фрейма; текущие координаты текстового курсора отображаются там же. Кроме того, доступны стандартные манипуляции по обмену данными через ClipBoard посредством выбора фрагмента текста и использования клавиш Ctrl+X, Ctrl+C, Ctrl+V (уничтожить выделенный фрагмент текста, скопировать его и взять из ClipBoard соответственно, Ctrl-Z отменяет последнюю выполненную команду). Дополнительные возможности предоставляются через главное меню.

2. При старте в *режиме командной строки* (КС, пакетный режим) приложение пытается загрузить указанный в командной строке файл (должен быть единственным параметром КС), интерпретируя его как файл проекта (предположительное расширение PRJ) и сразу же начинает исполнение заданного скрипта с определёнными в файле параметрами (допускается одновременное выполнение множества экземпляров программы). Корректный файл проекта состоит из 6 строк и имеет вид (пример):

```
GeteroCalcs_00.lua ; файл скрипта на Lua
e17_o11_t6.gv ; файл исходного ИГА
e17_o11_t6.ops ; файл параметров ОПЕРАТОРОВ
e17_o11_t6.cls ; файл параметров ВЫЧИСЛИТЕЛЕЙ
e17_o11_t6.mvr ; файл параметров (метрик) ВЕРШИН
e17_o11_t6.med ; файл параметров (метрик) ДУГ
```

В этом файле должны присутствовать все указанные 6 строк, после символа ';' следуют произвольные комментарии. При работе в пакетном режиме главное окно не показывается совсем (редактирование файла скрипта излишне), результат расчётов находится в файле протокола с уникальным именем.

Естественно, загрузка в тексте Lua-скрипта файла ИГА, параметров ОПЕРАТОРОВ, ВЫЧИСЛИТЕЛЕЙ, параметров (мер) ВЕРШИН и ДУГ (вызовами ReadEdges, LoadFileNameParamsOps, LoadFileNameParamsCalcs, LoadFileNameParamsVertices, LoadFileNameParamsEdges) приведёт к переопределению загруженных значений (в режиме командной строки эти вызовы в тексте Lua-скрипта делать не надо).

3. Для начала работы необходимо прочитать файл описания графа (по умолчанию это файл EdgesData.gv) или прочитать его и построить по нему первоначальную ("верхнюю") ЯПФ, для этого следует вызвать:

```
ReadEdges("EdgesData.gv") – только прочитать файл EdgesData.gv в массив Edges[][]
-- или --
CreateTiersByEdges("EdgesData.gv") -- прочитать файл EdgesData.gv в массив Edges[][]  и создать массив Tiers[][] по массиву Edges[][]
```

При желании контролировать возвращаемые значения (коды ошибок) можно использовать такую нотацию (выдава́емые функцией print на stdout значения появятся в нижнем левом фрейме):

```
print( ReadEdges("InputDataEdges") ) -- то же с контролем кода возврата
print( CreateTiersByEdges("InputDataEdges") ) -- -.-.-.-
```

4. Далее можно прочитать (и запомнить в переменных nOps и nTiers) общее число операторов и ярусов ЯПФ вызовами:

```
nOps = GetCountOps()
nTiers = GetCountTiers()
```

5. Полезно (с целью последующего самостоятельного анализа распределе́ния числа операторов по ярусам ЯПФ) занести в массив OpsOnTiers[] число операторов на каждом ярусе:

```
OpsOnTiers = {} -- пустой Lua-массив OpsOnTiers (нумерация элементов в Lua с 1 !!! )

-- заполняем массив OpsOnTiers
nTiers = GetCountTiers() -- число ярусов
print( "nTiers="  ..  nTiers )

for iTier=1, nTiers, 1 -- цикл по ярусам ЯПФ - нумерация с 1 по nTiers= GetCountTiers()
```

```
do -- начало тела цикла for iTier
 table.insert(OpsOnTiers,  GetCountOpsOnTier( iTier ) ) -- число операторов на ярусе iTier
 print( "iTier=" .. iTier .. ";nOps=" .. GetCountOpsOnTier( iTier ) )
end -- конец for iTier
```

Распечатать (для проверки) значения в массиве OpsOnTiers можно так:

```
for i=1, #OpsOnTiers do -- символ # суть длина OpsOnTiers
 print( i .. "-" .. OpsOnTiers[i] )
end
```

Понятно, что два дефиса являются признаком далее следующего комментария. В этом примере также показано использование оператора (две точки с пробелами слева и справа от них) конкатенации строк.

6. Номер jOp-того (jOp=1...GetCountOpsOnTier(iTier)) по счёту слева направо оператора на ярусе iTier (iTier=1...nTier) можно получить вызовом:

```
Op = GetOpByNumbOnTier( jOp, iTier ) -- номер оператора jOp на ярусе iTier
```

Важно! Lua-массивы начинаются с индекса 1 (***единица***) !!! Так же нумеруются яруса ЯПФ - *первый* имеет индекс 1, а *последний* - индекс nTiers; на каждом ярусе iTier находится GetCountOpsOnTier(iTier) операторов, их номера́ Op можно получить с помощью вызова Op=GetOpByNumbOnTier(iOp,iTier), где iOp - номер (слева направо) оператора на ярусе iTier. Номера дуг также лежат в диапазоне от 1 до nEdges.

В текстовое окно может быть выведена строковая информация любой сложности при использовании AddLineToTextFrame(str), причём в строке могут встречаться символы ‘\n’ (переход на новую строку). Доступна более короткая форма этого вызова – OutLine(str).

Все стандартные функции манипуляции с текстом доступны **ИССЛЕДОВАТЕЛЮ** и в текстовом фрейме второго окна, реализуются они через щелчок правой кнопки “мыши’ и выбор из всплывающего меню (сочетания клавиш Ctrl+X, Ctrl+C и Ctrl+V не задействованы вследствие вывода текста в режиме ReadOnly).

7. Пример подпрограмм (текст подпрограмм помещается до основной программы):

```
function AllOpsDown()
-- перемещает все операторы как можно НИЖЕ по уровням ЯПФ
local  iTier,iOp -- локальные переменные (действуют только внутри функции)
```

```
CountMovesZeroing() -- обнулили счётчик переносов
--
for iTier=GetCountTiers(),1,-1 do -- по ярусам снизу вверх
 for iOp=GetCountOpsOnTier(iTier),1,-1 do -- по #операторов на ярусе справа налево
  Op=GetOpByNumbOnTier(iOp,iTier) -- номер оператора
  MoveOpTierToTier(Op,GetMaxTierMaybeOp(Op)) -- переносим Op
 end -- конец цикла по iOp
end -- конец цикла по iTier
--
return GetOpsMoves() -- число переносов операторов
--
end -- конец функции AllOpsDown

--============================================================

function AllOpsTop()
-- перемещает все операторы как можно ВЫШЕ по уровням ЯПФ
local  iTier,iOp -- локальные переменные (действуют только внутри функции)
CountMovesZeroing() -- обнулили счётчик переносов
--
for iTier=1,GetCountTiers(),1 do -- по ярусам сверху вниз
 for iOp=GetCountOpsOnTier(iTier),1,-1 do --по #операторов на ярусе справа налево
  Op=GetOpByNumbOnTier(iOp,iTier) -- номер оператора
  MoveOpTierToTier(Op,GetMinTierMaybeOp(Op)) -- переносим Op
 end end -- конец цикла по iOp / конец цикла по iTier
--
return GetOpsMoves() -- число переносов операторов
--
end -- конец функции AllOpsTop
```

Приведены две функции (без входных параметров) для перемещения всех операторов в ЯПФ "как можно ниже" (функция AllOpsDown) и "как можно выше" (функция AllOpsTop), переносы производятся с учётом ненарушéния информационных связей (за это отвечает вызов GetMinTierMaybeOp). Напомним, что вызов CreateTiersByEdges (так же как и нажатие F5) строят ЯПФ в "верхней форме" (все операторы расположены на ярусах, непосредственно следующих за теми ярусами, на

которых определяются их операнды); т.о. применение AllOpsTop приводит ЯПФ к начальной "верхней" форме. Заметим, что операторы нулевого яруса перемещены быть не могут.

Обе функции возвращают одно значение - число перено́сов операторов с яруса на ярус GetOpsMoves, потребовавшееся для свершéния заданного (удобно в начале функции обнулить счётчик вызовом CountMovesZeroing). Т.к. возможность переноса каждого оператора с яруса на ярус зависит от каждого свершившегося переноса, однократный проход по всем операторам может не привести к заданному результату, при этом необходимо вызывать AllOpsDown и AllOpsTop итерационно до момента возврата нулевого значения (на число итераций влияет последовательность обхода ярусов ЯПФ (сверху вниз или снизу вверх) и последовательность перебора операторов на каждом ярусе (слева направо или справа налево). Разработчик алгоритма может, напр., модифицировать тела функций AllOpsDown и AllOpsTop путём заключения их в блок repeat ... until(CountMovesZeroing==0) для достижения желаемого.

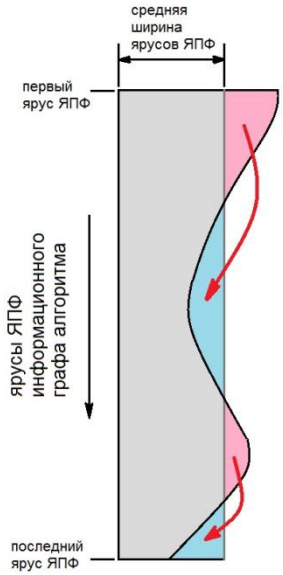
8. Следующий пример демонстрирует работу вышеописанных функций AllOpsDown и AllOpsTop в режиме вызова из управляющей программы (аналога main в C/C++):

```
--===здесь функция AllOpsDown()====================================

--===здесь функция AllOpsTop()=====================================

--===начало MAIN-программы==========================================

ReadEdges("InputDataEdges") -- читать файл описания графа
-- или --
CreateTiersByEdges("InputDataEdges") -- строим ЯПФ в "верхней"  форме
ClearTextFrame() -- очищаем окно вывода текста
PutTiersToTextFrame() -- выдадим ЯПФ в текстовое окно
ClearDiagrTiers() -- очистим область вывода ЯПФ в графике
DrawDiagrTiers() -- выводим ЯПФ в графике
--
--===переносим все операторы "вниз" по ярусам ЯПФ====================
--
for i=1,100,1 do -- ци́клим по вызовам функции AllOpsDown
  ClearDiagrTiers() -- очистим область вывода ЯПФ в графике
  DrawDiagrTiers() -- выводим ЯПФ в графике
--
--DelayMS( -5 ) -- ждём 5 сек для просмотра графика
--
```

```
out=AllOpsDown() -- вызываем целевую функцию
--
AddLineToTextFrame(" ") -- пустая строка вывод строки в текстовое окно
AddLineToTextFrame("Цикл=" .. tonumber(i) .. " ; переносов вниз=" .. tonumber(out))
--
if out == 0 then -- если функция AllOpsDown успешно сработала...
 break -- конец программы
end -- конец if out
--
end -- конец цикла по i

AddLineToTextFrame("") -- вывод пустой строки
AddLineToTextFrame("==============") -- строка-разделитель

PutTiersToTextFrame() -- вывод ЯПФ в текстовом виде

DelayMS( -5 ) -- ждать 5 сек

--
--===переносим все операторы "вверх" по ярусам ЯПФ==================
--
for i=1,100,1 do -- ци́клим по вызовам AllOpsTop
--
PutTiersToTextFrame()
ClearDiagrTiers()
DrawDiagrTiers()
--
--DelayMS( -1 ) -- ждать 1 сек
--
out=AllOpsTop() -- вызываем целевую функцию
--
AddLineToTextFrame(" ") -- пустая строка
AddLineToTextFrame("Цикл=" .. tonumber(i) .. " ; переносов вверх=" .. tonumber(out))
--
```

```
if out == 0 then
 break -- конец программы
end -- конец if
--
end -- конец цикла по i
--
--

print( PutTiersToTextFrame() )

-- конец основной программы
```

6. Теперь можно подумать о подходах к решению главной задачи - преобразованию ЯПФ в нужном направлении. Почти всегда "нужное направление" это:



1. Желание "сбалансировать" (добиться наиболее равной ширины всех ярусов) граф при неувеличении ярусов (сохранение общего времени выполнения алгоритма); фактически это требование максимального использования выделенных ресурсов - параллельно работающих вычислителей и/или

8. Требование выполнить алгоритм на заданном (обычно много меньшем текущей ширины ЯПФ) числе параллельных вычислителей; для этого придётся пойти на увеличение ярусов ЯПФ (увеличение времени выполнения алгоритма).

В общем случае полезно представить график распределения ширины ярусов ЯПФ в виде "холмов и впа́дин" (см. рис. слева). Красными стрелками при этом показано общее направление переноса операторов (стратегия "срыва́ния холмов и заполне́ния впадин"). Автором реализован один из вариантов такой стратегии, однако предлагаю желающим подумать и предложить свои собственные. Одним из работающих критериев "балансировки" ЯПФ служить среднеквадратичное отклонение (или дисперсия) ширины ярусов ЯПФ.

Вспомним, что изначально ЯПФ стро́ится в "верхней канонической" форме (имеющие возможность перемещения с яруса на ярус операторы расположена на наиболее высоком - ближайшему к первому - ярусах), поэтому в это время их можно только "опускать" (переносить в сторону последнего яруса).

Всё бы хорошо, но среди операторов обязательно встретятся такие, переносить которые на более нижние уровни невозможно (согласно их информационным связям), поэтому после первого этапа получаем довольно "ровную" ЯПФ с некоторым числом "пи́ков" (соответствующим ранее упомянутым "непереноси́мым" операторам). Т.о. складывается ситуации, когда нет иного варианта, чем увеличить число ярусов ЯПФ; это делается с использованием вызовов AddTier и MoveOpTierToTier (описание см. выше) API системы SPF@home (в подразделе 7.1 подробно разобран пример реализации такой стратегии).

По всей вероятности лучшие результаты можно получить при стратегии, включающей совместную (в разных последовательностях) реализацию обоих рассмотренных подходов.

9. Итак, разбираем пример реализации стратегии "сжатия" ЯПФ по ширине до заданной величины (напр., реально имеющегося или выделенного на данную задачу числа ядер процессора) при возрастании высоты ЯПФ (времени работы алгоритма).

Стратегия довольно проста - внутри цикла (длящегося, пока максимум числа операторов на всех ярусах кроме нулевого и последнего не станет менее Width) выбирается ярус с мах числом операторов, под ним создаётся новый пустой и на него переносится *половина* операторов с данного яруса). Вызов CountMovesZeroing() обнуляет счётчик числа перемещений операторов с яруса на ярус, а GetOpsMoves() печатает общее число перемещений. После каждого цикла преобразований ЯПФ выдаётся в текстовое окно вызовом PutTiersToTextFrame() и отображается в графике функцией DrawDiagrTiers().

Данная стратегия действительно работает (можно поэкспериментировать с разными Width), однако неравномерность в распределении операторов по ярусам ЯПФ остаётся значительной. Можно попробовать сгладить её перед запуском этой программы, а можно и после.

Интересно поэкспериментировать с до́лей операторов, переноси́мых каждый раз на нижележащий ярус (кто гарантирует, что 50% - оптимум?)... Кстати, переносить на нижеследующий уровень, наверное, надо не просто "операторы по счёту" (формально перенести можно каждый), а в соответствие с их информационными связями... а может быть, при этом заданной цели удастся достичь за меньшее число перестано́вок?

В общем - работайте своими мозговы́ми "маленькими серыми клеточками" (как любил гова́ривать незабве́нный Эркюль Пуаро)..!