算法特征提取程序

* 1. 程序介绍
     1. 功能

计算算法图中的关键路径，提取关键路径中的各种算子组合，归类统计输出

* + 1. 基本原理

算法AOE图：

用顶点表示算法的不同阶段，用边表示算法的的各项操作，边的权值表示各操作的电路延迟，边的方向表示数据流向。

关键路径算法：

定义Ve[i]为每一个点的最早发生时间；Vl[i]为每一个点的最迟发生时间；整个过程如下：

1. 拓扑排序，并求得ve[]：从源点V0出发，令Ve[0]=0，按拓扑有序求其余各顶点的最早发生时间Ve[i]。如果得到的拓扑有序序列中顶点个数小于网中顶点数n，则说明网中存在环，不能求关键路径，算法终止；否则执行步骤2。
2. 拓扑逆序，求得Vl[]：从汇点Vn出发，令Vl[n-1] = Ve[n-1]，按逆拓扑有序求其余各顶点的最迟发生时间Vl[i]。
3. 比较Ve[]和Vl[]，求得关键路径：根据各顶点的Ve[i]和Vl[i]值，求每条边的最早开始时间e(s)和最迟开始时间l(s)。若某条弧满足条件e(s) = l(s)，则为关键活动，其中：e(i) = ve(j)，l(i) = vl(k) - weight(<j,k>)。

关键路径上的算子组合：

对关键路径上的所有的算子进行临近组合，每一个有效组合作为一个组合元，所有的组合元构成有效组合的集合，遍历所有算法的关键路径统计这些组合元出现的频度。

* + 1. 开发环境

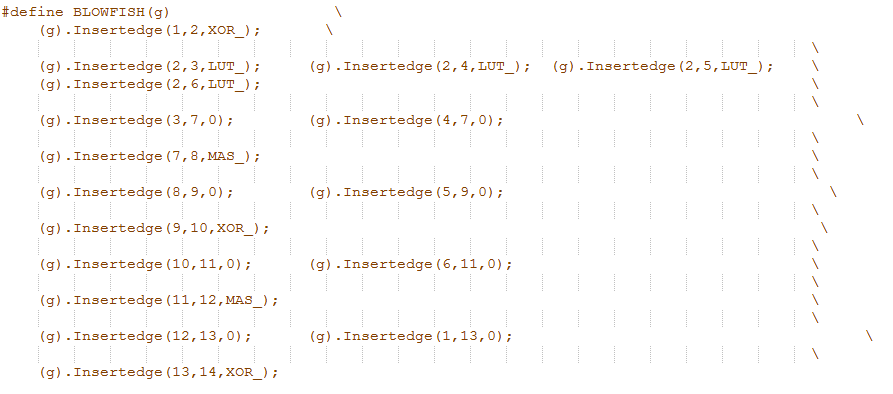
语言C++11

平台：Windows，开发环境VS2010以上版本

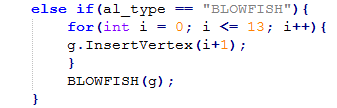
* + 1. 输入输出

输入：

以宏定义的形式输入算法AOE图，首先用宏定义算法的边输入，然后在init\_al\_graph函数中定义该算法，插入顶点，调用边插入的宏定义。



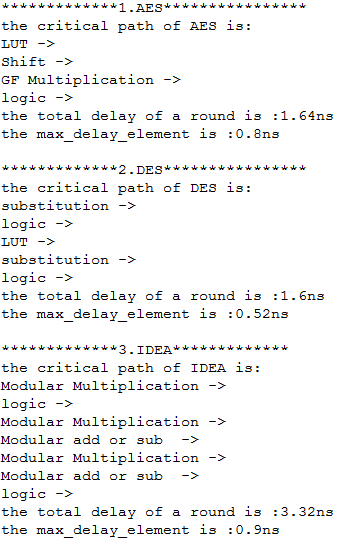
Blowfish算法AOE图输入宏定义



Blowfish算法AOE图顶点输入和图构建

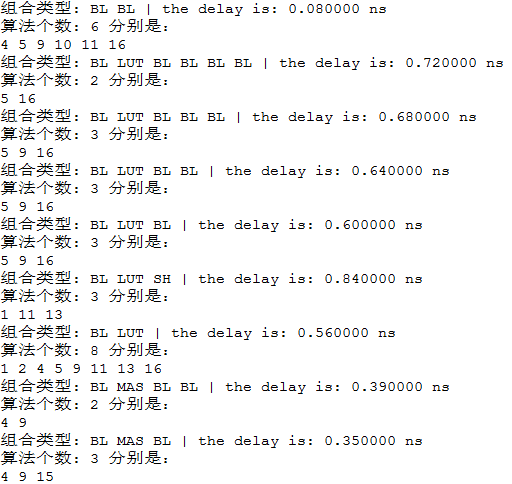
输出：

每一个算法的关键路径信息，包括关键路径组成，路径长度，路径中延迟最大的单元。



关键路径信息输出

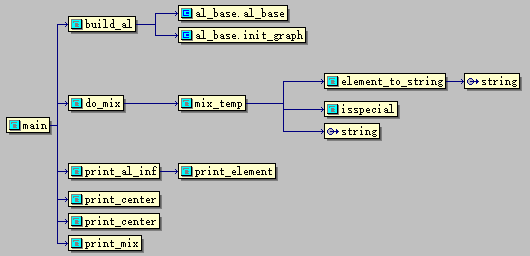
每一种算子的组合信息输出，包括每一种组合出现的频度，对应的算法。



组合信息输出

* + 1. 程序流程

1. 构建算法图、获取关键路径
2. 在关键路径上对算子进行组合
3. 输出算法基本信息
4. 输出每一种算子的前缀组合
5. 输出每一种算子的后缀组合
6. 输出算子的统计结果



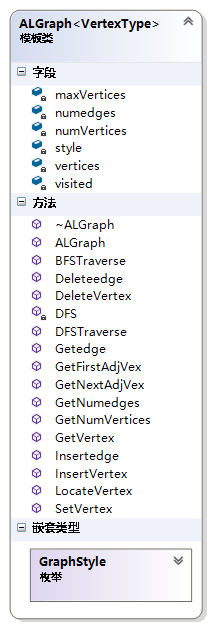
* + 1. 类继承关系

无类继承

* 1. 类介绍
     1. ALGraph类

类介绍

算法图类，包括图的基本数据成员、基本操作以及重要的几个算法用来完成图关键路径查找。



数据成员

|  |  |
| --- | --- |
| 数据成员 | 解释 |
| maxVertices | 最多顶点数 |
| numedges | 边数 |
| numVertices | 点数 |
| style | 图模式（有向or无向） |
| vertices | 节点存储 |
| visited | 访问标记 |

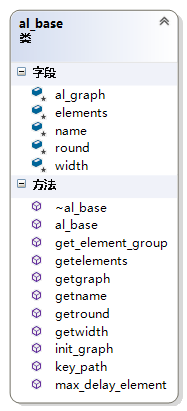
函数成员

|  |  |
| --- | --- |
| 成员函数 | 解释 |
| ALGraph | 构造函数 |
| BFSTraverse | 广度优先遍历 |
| Deleteedge | 删除边 |
| DeleteVertex | 删除节点 |
| DFS | 深度优先遍历 |
| DFSTraverse | 深度优先遍历 |
| Getedge | 获取边 |
| GetFirstAdjVex | 获取第一个邻近点 |
| GetNextAdjVex | 获取下一个邻近点 |
| GetNumedges | 获取边数 |
| GetNumVertices | 获取点数 |
| GetVertex | 获取点数 |
| Insertedge | 插入边 |
| InsertVertex | 插入点 |
| LocateVertex | 定位点 |
| SetVertex | 设置点 |
| FindInDegree | 入度查询 |
| TopologicalSort | 拓扑排序 |
| PrintCriticalPath | 输出关键路径 |
| SaveCriticalPath | 保存关键路径 |
| ~ALGraph | 析构函数 |

* + 1. al\_base类

类介绍

算法类，包括算法的基本属性，轮数、位宽、关键路径、算法图、算子等。



al\_base类结构

数据成员

|  |  |
| --- | --- |
| 数据成员 | 解释 |
| al\_graph | 算法图 |
| elements | 算子 |
| name | 算法名称 |
| round | 轮数 |
| width | 位宽 |

函数成员

|  |  |
| --- | --- |
| 成员函数 | 解释 |
| al\_base | 构造函数 |
| getelements | 获取算子 |
| getgraph | 获取算法图 |
| getname | 获取算法名称 |
| getround | 获取轮数 |
| getwidth | 获取位宽 |
| get\_element\_group | 获取算子组合 |
| init\_graph | 算法图初始化 |
| key\_path | 关键路径 |
| max\_delay\_element | 最大算子单元 |
| ~al\_base | 析构函数 |

* + 1. al\_protocol类

类介绍

基本电路信息定义



数据成员

无

函数成员

无

* 1. 全局数据、函数
     1. 全局数据

宏：

|  |  |
| --- | --- |
| 宏 | 解释 |
| AES | AES算法AOE图边插入宏定义 |
| ARIA | ARIA算法AOE图边插入宏定义 |
| BLOWFISH | BLOWFISH算法AOE图边插入宏定义 |
| C2 | C2算法AOE图边插入宏定义 |
| CAMELLIA | CAMELLIA算法AOE图边插入宏定义 |
| CAST128 | CAST128算法AOE图边插入宏定义 |
| CLEFIA | CLEFIA算法AOE图边插入宏定义 |
| CS\_CIPHER | CS\_CIPHER算法AOE图边插入宏定义 |
| DES | DES算法AOE图边插入宏定义 |
| E2 | E2算法AOE图边插入宏定义 |
| GOST | GOST算法AOE图边插入宏定义 |
| GRAND\_CRU | GRAND\_CRU算法AOE图边插入宏定义 |
| HIEROCRYPT\_3 | HIEROCRYPT\_3算法AOE图边插入宏定义 |
| HIEROCRYPT\_L1 | HIEROCRYPT\_L1算法AOE图边插入宏定义 |
| ICE | ICE算法AOE图边插入宏定义 |
| IDEA | IDEA算法AOE图边插入宏定义 |
| KASUMI | KASUMI算法AOE图边插入宏定义 |
| KHAZAD | KHAZAD算法AOE图边插入宏定义 |
| LUCIFER | LUCIFER算法AOE图边插入宏定义 |
| M6 | M6算法AOE图边插入宏定义 |
| MACGUFFIN | MACGUFFIN算法AOE图边插入宏定义 |
| NUSH | NUSH算法AOE图边插入宏定义 |
| PRESENT | PRESENT算法AOE图边插入宏定义 |
| Q | Q算法AOE图边插入宏定义 |
| RC5 | RC5算法AOE图边插入宏定义 |
| RC6 | RC6算法AOE图边插入宏定义 |
| SEED | SEED算法AOE图边插入宏定义 |
| SERPENT | SERPENT算法AOE图边插入宏定义 |
| SHARK | SHARK算法AOE图边插入宏定义 |
| SIMON | SIMON算法AOE图边插入宏定义 |
| SKIPJECT | SKIPJECT算法AOE图边插入宏定义 |
| SM4 | SM4算法AOE图边插入宏定义 |
| SPECK | SPECK算法AOE图边插入宏定义 |
| SQUARE | SQUARE算法AOE图边插入宏定义 |
| TEA | TEA算法AOE图边插入宏定义 |
| TWOFISH | TWOFISH算法AOE图边插入宏定义 |
| XTEA | XTEA算法AOE图边插入宏定义 |

变量：

|  |  |
| --- | --- |
| 变量 | 解释 |
| al\_str | 算法名称 |
| element\_delay | 算子延迟 |

* + 1. 全局函数

|  |  |
| --- | --- |
| 全局函数 | 解释 |
| build\_al | 构建算法 |
| do\_mix | 算子组合（壳） |
| element\_to\_string | 元素到字符串转换（输出） |
| init\_al\_graph | 初始化算法图 |
| isspecial | 是否是特殊算子 |
| main | 主函数 |
| mix\_temp | 算子组合（完成组合工作） |
| print\_al\_inf | 打印算法的基本信息 |
| print\_center | 打印算子组合 |
| print\_center\_post | 打印算字后缀组合 |
| print\_center\_pre | 打印算子前缀组合 |
| print\_element | 打印算子（字符串输出） |
| print\_mix | 打印所有组合 |

算法映射程序

* 1. 程序介绍
     1. 功能

完成算法图到架构图的映射

* + 1. 基本原理

构建构建算法图、架构图，利用VF2子图同构完成算法图到架构图的同构映射，从多个映射结果中选出最优映射结果。

**算法图模型：**

ALG=(V, E)，其中V是节点集，表示算法中的操作；E是边集，表示操作之间的数据依赖关系。



算法图的AOV表示

如上图所示，图中(a)是算法中的数据流程图的一部分，它标示了算法的各种操作以及数据方向，(b)则是与数据流图对应的算法图模型。表3-2列出了对应节点的属性，其中操作属性中，xor表示异或，sh表示移位，lut表示查找表，au表示算术，通过给(b)的节点赋予这些属性，(b)就能够描述算法。

算法图结点属性

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 节点 | x | y | 操作属性 | 节点 | x | y | 操作属性 |
| v1 | 1 | 1 | xor | v5 | 3 | 3 | lut |
| v2 | 2 | 1 | sh | v7 | 4 | 1 | xor |
| v3 | 3 | 1 | lut | v8 | 5 | 1 | au |
| v4 | 3 | 2 | lut | v9 | 6 | 1 | xor |

上表中节点属性中的y属性标示了各个操作在并行上左右顺序，这个信息会被用到算法映射中的成本函数中用来找出没有功能交叉的映射。x属性对应纵向上的顺序，用来获取算子的次序信息。节点的操作属性用来完成映射时的功能匹配，在算法模型和架构模型中，操作属性都用字符串来描述，通过字串包含的方式来验证功能包含。

**架构图模型：**

ARG=(V，E)，其中V是节点集，表示架构中的PE；E是边集，表示架构中的数据互连。如下表所示在架构图中用字符串描述点的属性，PE1的功能属性的字符串描述了第一类PE的19种功能，表中的au表示算术功能，xor表示逻辑功能，sh表示移位功能，bn表示置换功能，lut表示S盒功能，gfm表示有限域乘法功能，字符串组合表示功能的组合，每一个功能字符串之间用“\_”字符隔开。架构图中的（x，y）位置属性标识了该节点对应的PE在架构中的位置。

架构图结点属性

|  |  |  |
| --- | --- | --- |
| 节点V | 位置坐标（x,y） | 功能属性 |
| PE1 | (1,1)、（1,2） | au\_xorau\_auxor\_xorauxor\_xorxorau\_xorxorauxor\_sh\_xorsh\_shxor\_xorshxor\_xor\_xorxor\_xorxorxor\_or\_and\_bn\_xorbn\_bnxor\_xorbnxor\_ |
| PE2 | (1,3)、（1,4） | au\_xorau\_auxor\_xorauxor\_xorxorau\_xorxorauxor\_sh\_xorsh\_shxor\_xorshxor\_xor\_xorxor\_xorxorxor\_or\_and\_ |
| PE3 | (2,1)、（2,2）  (2,3)、（2,4） | au\_xorau\_auxor\_xorauxor\_xorxorau\_xorxorauxor\_sh\_xorsh\_shxor\_xorshxor\_xor\_xorxor\_xorxorxor\_or\_and\_lut\_xorlut\_lutxor\_xorlutxor\_ |
| PE4 | (3,1)、（3,2） | au\_xorau\_auxor\_xorauxor\_xorxorau\_xorxorauxor\_sh\_xorsh\_shxor\_xorshxor\_xor\_xorxor\_xorxorxor\_or\_and\_bn\_xorbn\_bnxor\_xorbnxor\_gfm\_xorgfm\_gfmxor\_xorgfmxor\_ |
| PE5 | (3,3)、（3,4） | au\_xorau\_auxor\_xorauxor\_xorxorau\_xorxorauxor\_sh\_xorsh\_shxor\_xorshxor\_xor\_xorxor\_xorxorxor\_or\_and\_gfm\_xorgfm\_gfmxor\_xorgfmxor\_ |

下图中，在PE组的图模型中，每一行的4个PE被抽象成四个点，每个点有与PE对应的功能属性，行与行之间的互连被抽象成从上一行到下一行的有向边。



PE组的图模型

**映射算法：**

映射算法伪代码

1) PROCEDURE Match(s)

2) 输入：中间状态s，初始状态s0满足

3) 输出：两个图之间的所有匹配

4) IF M(s) 覆盖了G2的所有结点

5) 输出M(s)

6) ELSE

7) 计算可以被包含到M(s)中的候选点对集合P(s)

8) FOREACH P(s)中的点对p

9) IF 候选点对满足加入M(s)的所有约束规则

10) 将p加入M(s)，计算s’

11) CALL Match(s’)

12) END IF

13) END FOREACH

14) 恢复数据状态

15) END IF

16) END PROCEDURE Match(s)

算法流程：

状态s存储搜索过程中的部分匹配，以及算法需要的其它数据。M(s)代表中间状态，M1(s)和M2(s)表示当前状态s的部分匹配中图G1和图G2中的点，初始化时状态是s0，M(s0)是空集，即还没有任何匹配；之后递归的进行搜索。如果当前状态s代表的部分匹配包含了G2中的所有节点，则已经找到了G2在G1中同构的子图，搜索结束；否则，在当前的局部匹配基础上，再匹配一个点。找出所以可能进行匹配点对集合P(s)；对于每一个匹配对p，检查加入匹配p是否可行，即加入p后，两个图是否保证同构，以及加入p之后，是否还有扩展的可能性；如果加入匹配p可行，则将p加入s，递归调用Match()，继续搜索。如果若干次调用Match()后都没有找到同构的子图，则说明当前状态不可能扩展出可行的子图同构匹配；所以将生成新状态时加入的匹配p从s中删除，回溯到上一个状态。

映射时的匹配约束规则：

先约定几个符号：N1和 N2表示图G1和图G2中的点集。n和m分别表示图G1和图G2中的点。Pred(G，n)表示点n在图G中的前驱，Succ(G，n)表示点n在图G中的后继。和表示状态s在图G1和图G2中指向当前已经匹配的点集的所有边的源点集合。和表示状态s在图G1和图G2中，从当前已经匹配的点集出发的所有边的终点集合。，表示当前状态s在图中已经匹配的点集的所有一步邻居。，表示图中，除了s中已经匹配的点和这些点的一步邻居以外的点。

 （4.1） （4.2）







 （4.6）

最优映射筛选规则：

 （4.7）

 （4.8）

 （4.9）

 （4.10）

 （4.11）

式（4.7）是映射最优选择的第一优先级约束，它保证选出的候选集的映射占用最少的资源行。首先统计每一个可行映射中行数的最大值，然后从这些最大值中选出行数最少的映射集合。算法在可重构阵列上按行映射，最小的资源单位是行，因此映射的行数最少表示使用的行资源最少。

式（4.8）是映射最优选择的第二优先级约束，它保证选出的候选集的映射占有最少的列数。首先统计每一个可行映射中列数的最大值，然后从这些最大值中选出列数最少的映射集合。对于很多算法，映射时不需要阵列上的所有列参与运算，对于这样的算法可以同时进行多路映射，找出列数最少的映射，总列数除以这个值就是这个算法允许同时执行的路数。

通过式（4.7）和式（4.8）这两个约束选择出同时满足行列最小化的映射，这已经是最优映射；而式（4.9）、（4.8）和（4.11）确定的是映射工具映射时使用资源的聚集倾向。式（4.9）和（4.8）分别约束在算法映射时的功能聚集方向，在能完成映射的前提下，功能总会向架构的左上聚集。式（4.11）约束功能乱序，保证在映射时并行功能在并行上的顺序和在算法本身中的顺序一致。式（4.9）、（4.8）和（4.11）虽然和最终的资源最优无关，但是这三个规则使所有算法在架构上映射时都有统一的映射风格，保证这些算法不会胡乱使用资源。

* + 1. 开发环境

语言C++11

平台：Windows，开发环境VS2010以上版本

* + 1. 输入输出

**输入：**

输入包括算法图输入和架构图输入

算法图输入：

包括算法图顶点文件（ALN\_xxx.txt）和算法图边文件（ALE\_xxx.txt），并且需要在程序中声明对应的文件路径，al\_n\_file存储算法顶点文件路径，al\_e\_file存储算法边文件路径。



顶点文件的格式如下图所示：

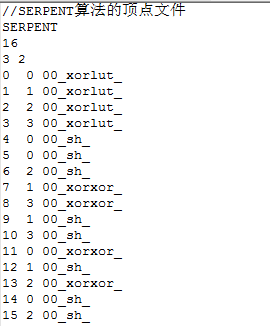
第一行：注释，任何信息

第二行：算法名

第三行：包含顶点数

第四行：加快映射速度的建议值，其中第一个数值是建议的开始映射的组数，第二个数字是建议的开始行，这两个值默认是1,1；有时一个算法规模比较大，为了加快映射速度跳过前面单个组映射尝试的时间，可以人为的给予映射建议。这个值不是必须的，如果忽略直接用默认值1,1。

后面每一行表示一个顶点，其中每行的第一个数字表示顶点序号，第二个数字表示y属性，后面的字符串表示顶点操作属性，其中前缀为00，表示这是个算法顶点（与架构顶点区分），后面的字符串表示操作具体的操作属性，前后用”\_”标识。



顶点文件格式

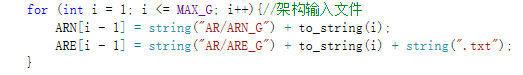
边文件格式中每一行表示一条边，用数字对标识（x, y），表示从x标识的顶点到y标识的顶点的一条有向边。



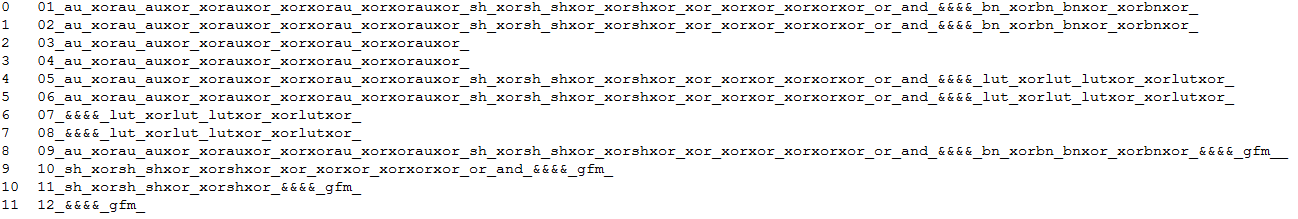
边文件格式

架构图输入：

包括架构图顶点文件（ARN\_xxx.txt）和架构图边文件（ARE\_xxx.txt），并且需要在程序中声明对应的文件路径，ARN存储架构顶点文件路径，ARE存储架构边文件路径。



架构图顶点文件中每一行表示一个顶点，其中第一个数字表示顶点编号，后面的这是对应的属性，其中前缀数字表示对应顶点在架构的异构组中的位置，后续的字符串表示顶点对应的功能，其中每一个功能（功能组合）间用“\_”隔开。



顶点文件格式

边文件格式与算法图一样，每一行表示一条边，用数字对标识（x, y），表示从x标识的顶点到y标识的顶点的一条有向边。



边文件格式

**输出：**

**指定输出文件**

****

**输出格式**

算法映射方案格式：

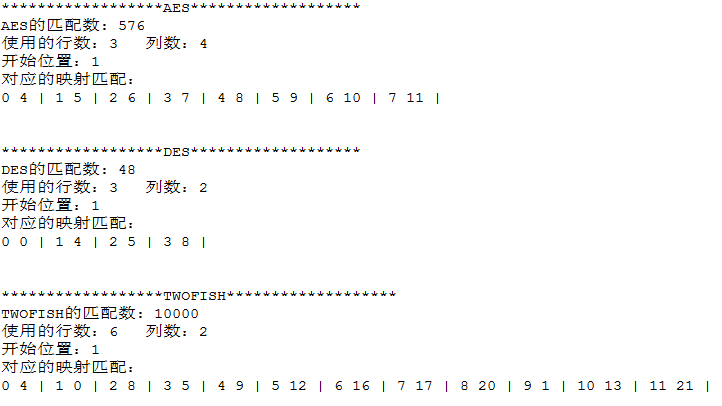
第一行为算法名称

第二行输出原始的映射方案个数

第三行输出映射所需的行数和列数

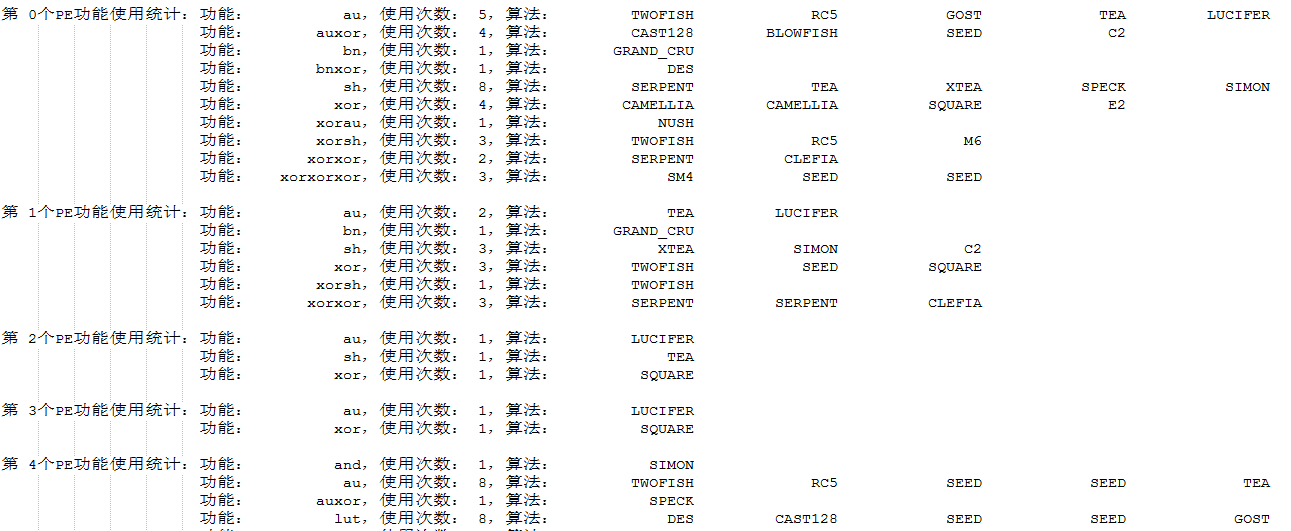
第四行输出开始映射的位置

第五行输出经过最优筛选的算法图到架构图的映射方案，每一个点对表示一个算法图顶点到架构图顶点的映射，不同的映射点对用“|”隔开



算法映射结果输出

对PE组的12个PE的功能使用进行统计，包括每一个PE中使用到的功能，这些功能使用的次数以及对应的使用算法

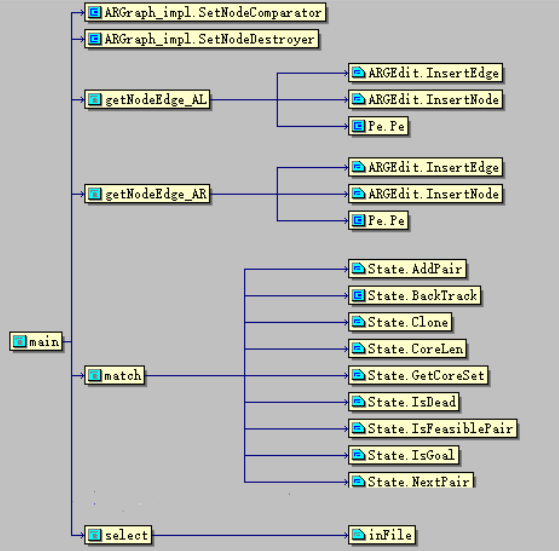


架构功能使用频度统计输出

* + 1. 程序流程

1. 设置节点的比较和销毁函数（在PE类里定义）
2. 算法图文件输入，构建算法图
3. 架构图文件输入，构建架构图
4. 算法图到架构图匹配：映射流程，包括点对加入，约束检查，递归下一个状态，结果查询等
5. 对映射结果进行最优筛选

整个流程如下图所示

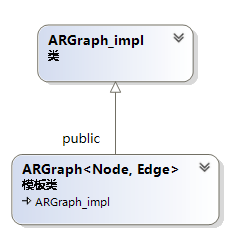


主程序函数调用流程

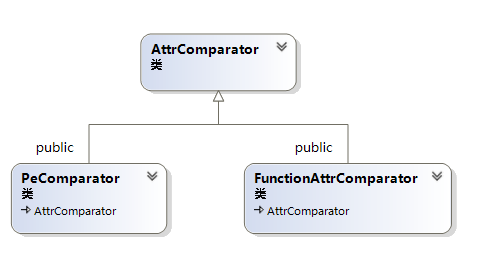
* + 1. 类继承关系

整个程序中有四个类继承关系：

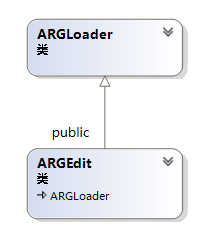
1. 图类继承关系



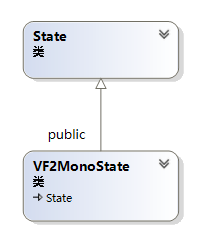
1. 属性比较类继承关系



1. 图操作类继承关系



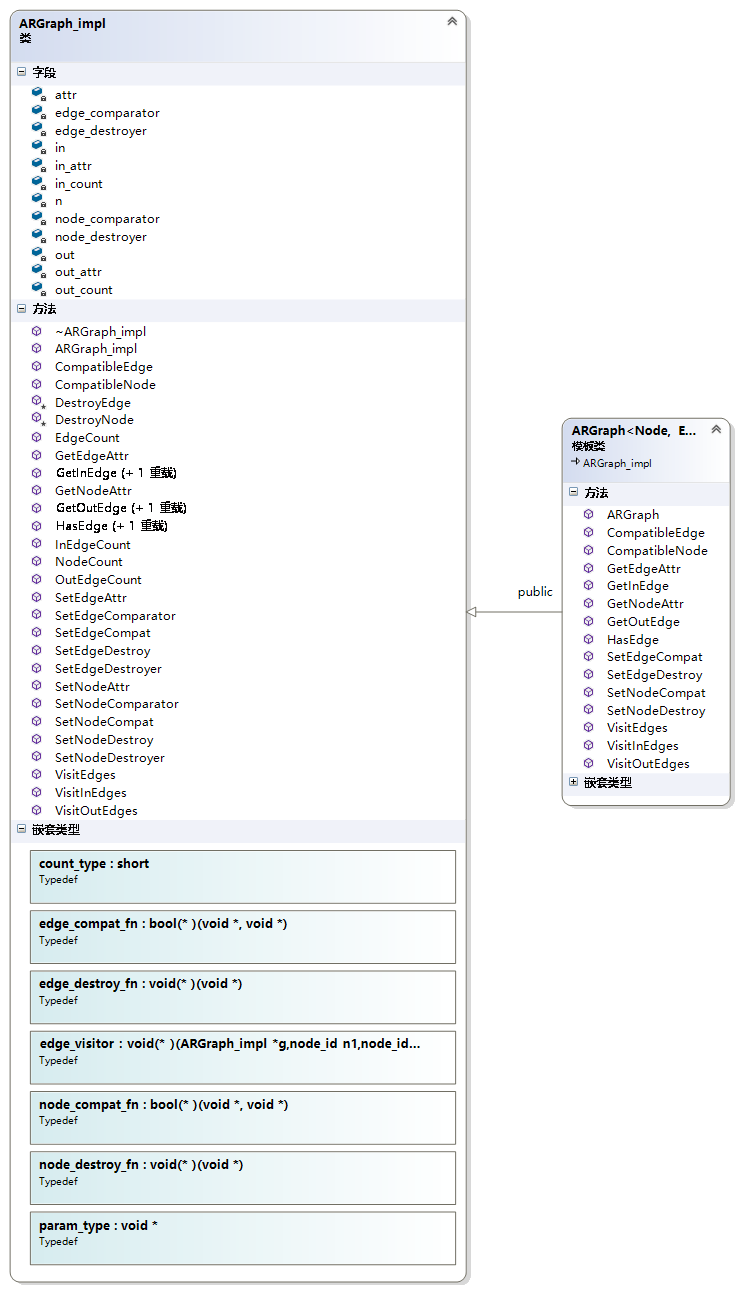
1. VF2匹配状态类继承关系



* 1. 类介绍
     1. ARGraph类

类介绍

ARGraph类是一个带属性的图类，包括图的各种属性以及本程序的映射算法使用到的出点入点属性，类的方法里包括属性相关的操作，属性设置，比较，销毁等，但是不包括对图的操作，图的操作会用一个专门的类来实现以降低操作与图类之间的耦合度。类的整体结构如下图所示。



ARGraph类整体结构

数据成员

|  |  |
| --- | --- |
| 数据成员 | 注释 |
| attr | 节点属性 |
| edge\_comparator | 边属性比较 |
| edge\_destroyer | 边属性消除 |
| in | 每一个节点入边的起点 |
| in\_attr | 入边属性 |
| in\_count | 每一个节点的出边 |
| n | 节点个数 |
| node\_comparator | 节点属性比较 |
| node\_destroyer | 节点属性清除 |
| out | 出边的终点 |
| out\_attr | 出边的属性 |
| out\_count | 出边的个数 |

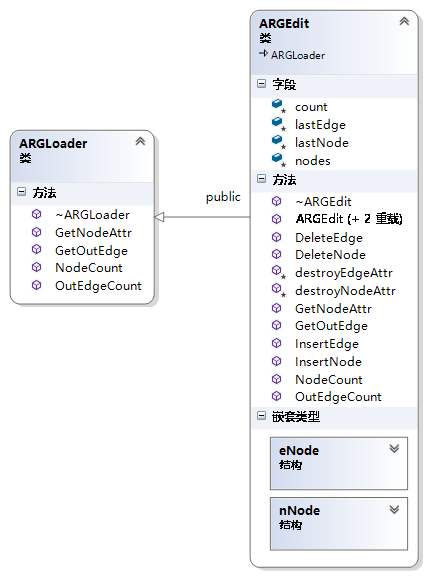
函数成员

|  |  |
| --- | --- |
| 成员函数 | 解释 |
| ARGraph\_impl | 构造函数 |
| CompatibleEdge | 边比较函数 |
| CompatibleNode | 节点比较函数 |
| DestroyEdge | 边属性销毁函数 |
| DestroyNode | 顶点属性销毁函数 |
| EdgeCount | 边数 |
| GetEdgeAttr | 获取边属性 |
| GetInEdge | 获取入边 |
| GetNodeAttr | 获取节点属性 |
| GetOutEdge | 获取出边 |
| HasEdge | 边查询 |
| InEdgeCount | 获取入边数 |
| NodeCount | 获取顶点数 |
| OutEdgeCount | 获取边数 |
| SetEdgeAttr | 设置边属性 |
| SetEdgeComparator | 设置边属性比较器 |
| SetEdgeCompat | 设置边属性比较器 |
| SetEdgeDestroy | 设置边销毁器 |
| SetEdgeDestroyer | 设置边销毁器 |
| SetNodeAttr | 设置节点属性 |
| SetNodeComparator | 设置节点比较器 |
| SetNodeCompat | 设置节点比较器 |
| SetNodeDestroy | 设置节点销毁器 |
| SetNodeDestroyer | 设置节点销毁器 |
| VisitEdges | 访问边 |
| VisitInEdges | 访问入边 |
| VisitOutEdges | 访问出边 |
| ~ARGraph\_impl | 析构函数 |

* + 1. ARGEdit类

类介绍

图操作类，负责图构建销毁，节点和边的插入和删除，整体结构如下图所示



ARGEdit类结构

数据成员

|  |  |
| --- | --- |
| 数据成员 | 解释 |
| count | 节点个数 |
| lastEdge | 最后的边序号 |
| lastNode | 最后的顶点序号 |
| nodes | 节点存储 |

函数成员

|  |  |
| --- | --- |
| 成员函数 | 解释 |
| ARGEdit | 构造函数 |
| ARGEdit | 构造函数 |
| ARGEdit | 构造函数 |
| DeleteEdge | 删除指定边 |
| DeleteNode | 删除指定节点 |
| destroyEdgeAttr | 销毁边属性 |
| destroyNodeAttr | 销毁节点属性 |
| GetNodeAttr | 获取顶点属性 |
| GetOutEdge | 获取出边 |
| InsertEdge | 插入边 |
| InsertNode | 插入顶点 |
| NodeCount | 获取节点数目 |
| OutEdgeCount | 获取出边数目 |
| ~ARGEdit | 析构函数 |

* + 1. Pe 类

类介绍

PE（包括算法操作）节点类，这个数据类型是图的存储类型，主要定义基本的节点属性，以及在属性设置和销毁。



Pe类结构

数据成员

|  |  |
| --- | --- |
| 数据成员 | 解释 |
| peattr | 节点属性 |

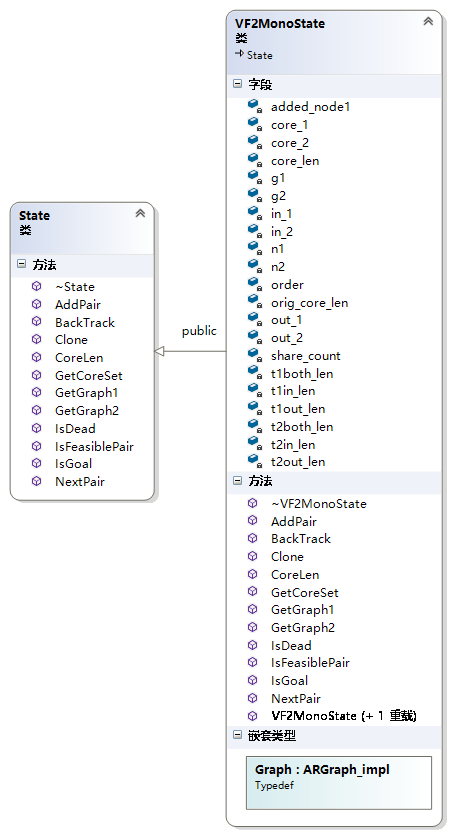
函数成员

|  |  |
| --- | --- |
| 成员函数 | 解释 |
| getPeAttr | 获取Pe属性 |
| Pe | 构造函数 |
| setPeAttr | 设置Pe属性 |

* + 1. state类

类介绍

映射算法的中间映射状态类，包括映射阶段保存，新匹配对加入，约束检查，终结检查，图复制，回溯等。



State类结构

数据成员

|  |  |
| --- | --- |
| 数据成员 | 解释 |
| added\_node1 | 新加入的节点 |
| core\_1 | 图1的匹配节点 |
| core\_2 | 图2的匹配节点 |
| core\_len | 匹配进度 |
| g1 | 图1 |
| g2 | 图2 |
| in\_1 | 图1的入点 |
| in\_2 | 图2的入点 |
| order | 节点顺序 |
| orig\_core\_len | 初始匹配进度 |
| out\_1 | 图1的出点 |
| out\_2 | 图2的出点 |
| t1both\_len | Core nodes 计数 |
| t1in\_len |
| t1out\_len |
| t2both\_len |
| t2in\_len |
| t2out\_len |

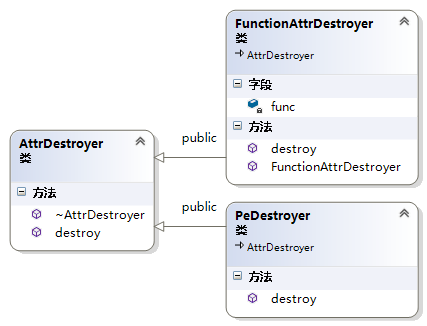
函数成员

|  |  |
| --- | --- |
| 成员函数 | 解释 |
| AddPair | 加入新节点对 |
| BackTrack | 回溯 |
| Clone | 复制部分图 |
| CoreLen | 获取进度 |
| GetCoreSet | 获取匹配集合 |
| GetGraph1 | 获取图1 |
| GetGraph2 | 获取图2 |
| IsDead | 终结查询 |
| IsFeasiblePair | 约束检查 |
| IsGoal | 检查是否到达匹配结果要求 |
| NextPair | 生成下一个点对 |
| VF2MonoState | 构造函数 |
| ~VF2MonoState | 析构函数 |

* + 1. AttrDestroy类

类介绍

节点的属性销毁类，每一个自定义的顶点都需要从AttrDestroyer基类里派生出自己的销毁类。



AttrDestroy类结构

数据成员

无

函数成员

|  |  |
| --- | --- |
| 成员函数 | 解释 |
| destroyer | 属性消除器 |

* 1. 全局数据、函数
     1. 全局数据

常量：

|  |  |
| --- | --- |
| 常量 | 解释 |
| AL\_NUM\_NOW | 映射算法个数 |
| MAXNODES | 最多节点数 |
| MAX\_G | 映射的最多PE组数 |

变量：

|  |  |
| --- | --- |
| 变量 | 解释 |
| matchingCnt | 记录匹配个数 |
| al\_n\_file | 算法图配置节点输入文件路径 |
| al\_e\_file | 算法图配置边输入文件路径 |
| col | 节点列信息保存 |
| al | 算法名称 |
| Node\_num | 算法节点数 |
| result | 映射结果保存 |
| al\_pe\_func | 不同算法的节点功能保存 |
| m\_row，m\_col | 映射的行、列结果 |
| pe\_util | pe的功能使用统计 |
| ARN | 架构输入节点文件 |
| ARE | 架构输入边文件 |

* + 1. 全局函数

|  |  |
| --- | --- |
| 全局函数 | 解释 |
| clear | 对指定指针的内存清零 |
| error | 自定义错误输出 |
| getNodeEdge\_AL | 从输入文件中构建算法图 |
| getNodeEdge\_AR | 从输入文件中构建架构图 |
| graph\_visitor | 匹配控制函数（控制匹配的输出格式和终止控制） |
| main | 主函数 |
| match | 匹配函数 |
| nodeInfoComp1 | 节点属性比较 |
| nodeInfoComp2 | 节点属性比较 |
| ptrcheck | 指针检查 |
| select | 最优映射筛选函数 |