| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [Calculators/src](http://docs.google.com/index.html) - TFOrderEnforcer.cpp (source / [functions](http://docs.google.com/TFOrderEnforcer.cpp.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 188 | 208 | 90.4 % | | Date: | 2012-03-23 |  | Functions: | 9 | 12 | 75.0 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 64 | 80 | 80.0 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : #include "Memory.h"  2 : : #include "TFOrderEnforcer.h"  3 : : #include "SubField.h"  4 : : #include "Index.h"  5 : : #include "ConverterParams.h"  6 : : #include "Logger.h"  7 : : #include "SDDB.h"  8 : : #include "print.h"  9 : : #include "RetryFunc.h"  10 : : #include "HealthCheckCounter.h"  11 : :   12 : : namespace nft {  13 : :   14 : :   15 : : TFOrderEnforcer::TFOrderEnforcer( ConverterParams \*inParams )  16 : 1094 : : mLocationOrder(),  17 : : mPriority(),  18 : 3599 : mCompare(NULL) {  19 : : const DPMANAGER\_MESSAGE\_CONVERTER \*pStartMsg  20 : 1094 : = inParams->getStartupMessage();  21 : 1094 : mLocationOrder = static\_cast<TFLocationOrder>(  22 : : pStartMsg->usTFLocationOrder);  23 : 1094 : mPriority = static\_cast<TFOrderingPriority>( pStartMsg->usTFPriority);  24 : 1094 : uint16\_t shapeOrder = pStartMsg->usTFShapeOrder;  25 [ + + ][ # # ]: 5470 : for( int32\_t i = 4; i >= 0; i--)  26 : : {  27 : 5470 : mShapeOrder[i]= (shapeOrder%10);  28 : 5470 : shapeOrder /= 10;  29 : : }  30 : : }  31 : :   32 : :   33 : 2188 : TFOrderEnforcer::~TFOrderEnforcer() {  34 : 1094 : nftFree(mCompare);  35 : 0 : mCompare = NULL;  36 : : }  37 : :   38 : :   39 : : TFOrderEnforcer::TFOrderEnforcer(  40 : : const TFOrderEnforcer & source)  41 : 0 : :mLocationOrder(source.mLocationOrder),  42 : 0 : mPriority(source.mPriority) {   43 : :   44 : 0 : memcpy(mShapeOrder, source.mShapeOrder, 5 \* sizeof(uint8\_t));  45 : : }  46 : :   47 : : TFOrderEnforcer &  48 : 0 : TFOrderEnforcer::operator=(const TFOrderEnforcer & source) {  49 : :   50 [ # # ]: 0 : if(this != &source)  51 : : {  52 : 0 : mLocationOrder = source.mLocationOrder;  53 : 0 : mPriority = source.mPriority;  54 : 0 : memcpy(mShapeOrder, source.mShapeOrder, 5 \* sizeof(uint8\_t));  55 : : }  56 : 0 : return \*this;  57 : : }  58 : :   59 : : void  60 : : TFOrderEnforcer::performOrdering(  61 : 2116 : SubField \*inoutSubField) throw(DPL::Error \*) {  62 : :   63 : 2116 : HealthCheckCounter::instance()->incrCounter(  64 : : HealthCheckCounter::eUpdateFrequent |  65 : : HealthCheckCounter::eUpdateInfrequent);  66 : :   67 : 2116 : try  68 : : {  69 : 2116 : Segment::Pointer seg = NULL;  70 : :   71 [ + + ]: 2116 : if(eShapeBeforeLocation == mPriority  72 : : || eShapeWithBucketSort == mPriority)  73 : : {  74 [ + + ]: 384 : for(int16\_t i = 0; i < 5; i++)  75 : : {  76 : 320 : seg = inoutSubField->getSegmentsForShape(mShapeOrder[i]);  77 : 320 : orderSegments(seg);  78 : : }  79 : : }  80 : : else  81 : : {  82 : 2052 : seg = inoutSubField->getSegmentsForLocationPriority();  83 : 2052 : orderSegments(seg);  84 : : }  85 : : }  86 [ + - ]: 18 : catch(DPL::Error \*pChild)  87 : : {  88 : 18 : Logger::instance()->log(LogMessageTable::eTF\_INFO,  89 : : "TF ordering failed");  90 : 18 : ErrorHandler \*pErr = ErrorHandler::instance();  91 : 18 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  92 : : ErrorHandler::eErrFunction,  93 : : pErr->getErrorMsg(  94 : : ErrorHandler::eErrFunction,  95 : : "TFOrderEnforcer::performOrdering"));  96 : : }  97 : : }  98 : :   99 : : void  100 : 1094 : TFOrderEnforcer::constructComparators() throw(DPL::Error \*) {  101 : :   102 : 1094 : try  103 : : {   104 : 1094 : mCompare = reinterpret\_cast<TFComparator \*>(  105 : : nftMalloc( 2 \* sizeof(TFComparator)));  106 : :   107 [ + - ][ + + : 1094 : switch( mLocationOrder )   + + + + +   + + + + +   + + + +   - ]  108 : : {  109 : : case eTForderUwdUwdFwd:  110 : : {  111 : 973 : mCompare[0] = TFComparator( eUwdFwdSegmentTFOrder,  112 : : mShapeOrder,mPriority );  113 : 973 : mCompare[1] = TFComparator(eUwdFwdSegmentTFOrder,  114 : : mShapeOrder, mPriority );  115 : 973 : break;  116 : : }  117 : : case eTForderUwdDwdFwd:  118 : : {  119 : 9 : mCompare[0] = TFComparator( eUwdFwdSegmentTFOrder,  120 : : mShapeOrder,mPriority );  121 : 9 : mCompare[1] = TFComparator(eDwdFwdSegmentTFOrder,  122 : : mShapeOrder, mPriority );  123 : 9 : break;  124 : : }  125 : : case eTForderDwdDwdFwd:  126 : : {  127 : 8 : mCompare[0] = TFComparator( eDwdFwdSegmentTFOrder,  128 : : mShapeOrder,mPriority );  129 : 8 : mCompare[1] = TFComparator(eDwdFwdSegmentTFOrder,  130 : : mShapeOrder, mPriority );  131 : 8 : break;  132 : : }  133 : : case eTForderDwdUwdFwd:  134 : : {  135 : 8 : mCompare[0] = TFComparator(eDwdFwdSegmentTFOrder ,  136 : : mShapeOrder,mPriority );  137 : 8 : mCompare[1] = TFComparator(eUwdFwdSegmentTFOrder,  138 : : mShapeOrder, mPriority );  139 : 8 : break;  140 : : }  141 : : case eTForderUwdUwdBwd:  142 : : {  143 : 8 : mCompare[0] = TFComparator( eUwdBwdSegmentTFOrder,  144 : : mShapeOrder,mPriority );  145 : 8 : mCompare[1] = TFComparator(eUwdBwdSegmentTFOrder,  146 : : mShapeOrder, mPriority );  147 : 8 : break;  148 : : }  149 : : case eTForderUwdDwdBwd:  150 : : {  151 : 8 : mCompare[0] = TFComparator( eUwdBwdSegmentTFOrder,  152 : : mShapeOrder,mPriority );  153 : 8 : mCompare[1] = TFComparator(eDwdBwdSegmentTFOrder,  154 : : mShapeOrder, mPriority );  155 : 8 : break;  156 : : }  157 : : case eTForderDwdUwdBwd:  158 : : {  159 : 8 : mCompare[0] = TFComparator( eDwdBwdSegmentTFOrder,  160 : : mShapeOrder,mPriority );  161 : 8 : mCompare[1] = TFComparator(eUwdBwdSegmentTFOrder,  162 : : mShapeOrder, mPriority );  163 : 8 : break;  164 : : }  165 : : case eTForderDwdDwdBwd:  166 : : {  167 : 8 : mCompare[0] = TFComparator( eDwdBwdSegmentTFOrder,  168 : : mShapeOrder,mPriority );  169 : 8 : mCompare[1] = TFComparator(eDwdBwdSegmentTFOrder,  170 : : mShapeOrder, mPriority );  171 : 8 : break;  172 : : }  173 : : case eTForderFwdFwdUwd:  174 : : {  175 : 8 : mCompare[0] = TFComparator( eFwdUwdSegmentTFOrder,  176 : : mShapeOrder,mPriority );  177 : 8 : mCompare[1] = TFComparator(eFwdUwdSegmentTFOrder,  178 : : mShapeOrder, mPriority );  179 : 8 : break;  180 : : }  181 : : case eTForderFwdBwdUwd:  182 : : {  183 : 8 : mCompare[0] = TFComparator(eFwdUwdSegmentTFOrder ,  184 : : mShapeOrder,mPriority );  185 : 8 : mCompare[1] = TFComparator(eBwdUwdSegmentTFOrder,  186 : : mShapeOrder, mPriority );  187 : 8 : break;  188 : : }  189 : : case eTForderBwdBwdUwd:  190 : : {  191 : 8 : mCompare[0] = TFComparator(eBwdUwdSegmentTFOrder,  192 : : mShapeOrder,mPriority );  193 : 8 : mCompare[1] = TFComparator(eBwdUwdSegmentTFOrder,  194 : : mShapeOrder, mPriority );  195 : 8 : break;  196 : : }  197 : : case eTForderBwdFwdUwd:  198 : : {  199 : 8 : mCompare[0] = TFComparator(eBwdUwdSegmentTFOrder,  200 : : mShapeOrder,mPriority );  201 : 8 : mCompare[1] = TFComparator(eFwdUwdSegmentTFOrder,  202 : : mShapeOrder, mPriority );  203 : 8 : break;  204 : : }  205 : : case eTForderFwdFwdDwd:  206 : : {  207 : 8 : mCompare[0] = TFComparator(eFwdDwdSegmentTFOrder,  208 : : mShapeOrder,mPriority );  209 : 8 : mCompare[1] = TFComparator(eFwdDwdSegmentTFOrder,  210 : : mShapeOrder, mPriority );  211 : 8 : break;  212 : : }  213 : : case eTForderFwdBwdDwd:  214 : : {  215 : 8 : mCompare[0] = TFComparator(eFwdDwdSegmentTFOrder,  216 : : mShapeOrder,mPriority );  217 : 8 : mCompare[1] = TFComparator(eBwdDwdSegmentTFOrder,  218 : : mShapeOrder, mPriority );  219 : 8 : break;  220 : : }  221 : : case eTForderBwdBwdDwd:  222 : : {  223 : 8 : mCompare[0] = TFComparator( eBwdDwdSegmentTFOrder,  224 : : mShapeOrder,mPriority );  225 : 8 : mCompare[1] = TFComparator(eBwdDwdSegmentTFOrder,  226 : : mShapeOrder, mPriority );  227 : 8 : break;  228 : : }  229 : : case eTForderBwdFwdDwd:  230 : : {  231 : 8 : mCompare[0] = TFComparator(eBwdDwdSegmentTFOrder,  232 : : mShapeOrder,mPriority );  233 : 8 : mCompare[1] = TFComparator(eFwdDwdSegmentTFOrder,  234 : : mShapeOrder, mPriority );  235 : 8 : break;  236 : : }  237 : : }  238 : : }  239 [ # # ]: 0 : catch(DPL::Error \*pChild)  240 : : {  241 : 0 : Logger::instance()->log(LogMessageTable::eTF\_INFO,  242 : : "Comparator construction failed");  243 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  244 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  245 : : ErrorHandler::eErrFunction,  246 : : pErr->getErrorMsg(  247 : : ErrorHandler::eErrFunction,  248 : : "TFOrderEnforcer::constructComparators"));  249 : : }  250 : : }  251 : :   252 : : void  253 : 1411 : TFOrderEnforcer::registerSegmentWriter( SegmentWriter inSegmentWriter ) {  254 : 1411 : mSegmentWriter = inSegmentWriter;  255 : : }  256 : :   257 : : void  258 : 2372 : TFOrderEnforcer::orderSegments(Segment::Pointer inSegs) {  259 : 2372 : ConverterParams \*pParams = ConverterParams::instance();  260 : : const DPMANAGER\_MESSAGE\_CONVERTER \*pStartMsg  261 : 2372 : = pParams->getStartupMessage();  262 : 2372 : uint16\_t segmentCountX = pStartMsg->usSegmentCountX;  263 : 2372 : uint16\_t segmentCountY = pStartMsg->usSegmentCountY;  264 : :   265 : 2372 : int st1 = 0;  266 : 2372 : int st2 = 0;  267 : 2372 : int end1 = 0;  268 : 2372 : int end2 = 0;  269 : 2372 : bool revOrder = false;  270 [ + - ][ + + : 2372 : switch(mLocationOrder)   + + + + +   + - ]  271 : : {  272 : : // col major, 0 to col, 0 to row  273 : : case(eTForderUwdUwdFwd) :  274 : : case(eTForderUwdDwdFwd) :  275 : : {  276 : 2036 : st1 = 0;  277 : 2036 : end1 = segmentCountX;  278 : 2036 : st2 = 0;  279 : 2036 : end2 = segmentCountY;  280 : :   281 [ + + ]: 2036 : if((eTForderUwdDwdFwd == mLocationOrder))   282 : : {  283 : 25 : revOrder = true;  284 : : }  285 : 25 : break;  286 : : }  287 : : case(eTForderDwdDwdFwd) :  288 : : case(eTForderDwdUwdFwd) :  289 : : {  290 : 48 : st1 = 0;  291 : 48 : end1 = segmentCountX;  292 : 48 : st2 = segmentCountY;  293 : 48 : end2 = 0;  294 : :   295 [ + + ]: 48 : if((eTForderDwdUwdFwd == mLocationOrder))   296 : : {  297 : 24 : revOrder = true;  298 : : }  299 : 24 : break;  300 : : }  301 : : //col major col to 0, 0 to row  302 : : case(eTForderUwdUwdBwd) :  303 : : case(eTForderUwdDwdBwd) :  304 : : {  305 : 48 : st1 = segmentCountX;  306 : 48 : end1 = 0;  307 : 48 : st2 = 0;  308 : 48 : end2 = segmentCountY;  309 : :   310 [ + + ]: 48 : if((eTForderUwdDwdBwd == mLocationOrder))   311 : : {  312 : 24 : revOrder = true;  313 : : }  314 : 24 : break;  315 : : }  316 : : case(eTForderDwdUwdBwd) :  317 : : case(eTForderDwdDwdBwd) :  318 : : {  319 : 48 : st1 = segmentCountX;  320 : 48 : end1 = 0;  321 : 48 : st2 = segmentCountY;  322 : 48 : end2 = 0;  323 : :   324 [ + + ]: 48 : if((eTForderDwdUwdBwd == mLocationOrder))   325 : : {  326 : 24 : revOrder = true;  327 : : }  328 : 24 : break;  329 : : }  330 : : //row major, 0 to row, 0 to col  331 : : case(eTForderFwdFwdUwd) :  332 : : case(eTForderFwdBwdUwd) :  333 : : {  334 : 48 : st1 = 0;  335 : 48 : end1 = segmentCountY;  336 : 48 : st2 = 0;  337 : 48 : end2 = segmentCountX;  338 : :   339 [ + + ]: 48 : if((eTForderFwdBwdUwd== mLocationOrder))   340 : : {  341 : 24 : revOrder = true;  342 : : }  343 : 24 : break;  344 : : }  345 : : case(eTForderBwdBwdUwd) :  346 : : case(eTForderBwdFwdUwd) :  347 : : {  348 : 48 : st1 = 0;  349 : 48 : end1 = segmentCountY;  350 : 48 : st2 = segmentCountX;  351 : 48 : end2 = 0;  352 : :   353 [ + + ]: 48 : if((eTForderBwdFwdUwd == mLocationOrder))   354 : : {  355 : 24 : revOrder = true;  356 : : }  357 : 24 : break;  358 : : }  359 : : //row major, row to 0, 0 to col  360 : : case(eTForderFwdFwdDwd) :  361 : : case(eTForderFwdBwdDwd) :  362 : : {  363 : 48 : st1 = segmentCountY;  364 : 48 : end1 = 0;  365 : 48 : st2 = 0;  366 : 48 : end2 = segmentCountX;  367 : :   368 [ + + ]: 48 : if((eTForderFwdBwdDwd == mLocationOrder))   369 : : {  370 : 24 : revOrder = true;  371 : : }  372 : 24 : break;  373 : : }  374 : : case(eTForderBwdBwdDwd) :  375 : : case(eTForderBwdFwdDwd) :  376 : : {  377 : 48 : st1 = segmentCountY;  378 : 48 : end1 = 0;  379 : 48 : st2 = segmentCountX;  380 : 48 : end2 = 0;  381 : :   382 [ + + ]: 48 : if((eTForderBwdFwdDwd == mLocationOrder))   383 : : {  384 : 24 : revOrder = true;  385 : : }  386 : 2372 : break;  387 : : }  388 : : }   389 : :   390 : 2372 : Index idx1;  391 : 2372 : Index idx2;  392 : 2372 : idx1.initialize(st1, end1);  393 : :   394 : 2372 : bool segEmpty = false;  395 : 2372 : bool first = true;  396 : 2372 : int16\_t index = 1;  397 : 2372 : int endIdx1 = idx1.end();  398 : 2372 : SDDB \*pSDDB = SDDB::instance();  399 : 2372 : pSDDB->resetSD81();  400 [ + + ]: 11535 : for(; idx1 < endIdx1; idx1++) // traverse over col  401 : : {  402 [ + + ]: 9163 : if(!segEmpty)  403 : : {  404 [ + + ]: 5426 : if( first )  405 : : {  406 : 5297 : idx2.initialize(st2, end2);  407 : : }  408 : : else  409 : : {  410 : : //idx2.initialize(st2, end2);  411 : 129 : idx2.initialize( end2, st2 );  412 : : }  413 [ + + ]: 5426 : if(revOrder)  414 : : {  415 : 386 : first = !first;  416 : : }  417 : 5426 : index = (index + 1) % 2;  418 : 5426 : segEmpty = true;  419 : : }  420 : : else  421 : : {  422 : 3737 : idx2.resetIndex();  423 : : }  424 : :   425 : 9163 : int endIdx2 = idx2.end();  426 [ + + ]: 42924 : for(; idx2 < endIdx2; idx2++) // traverse over row  427 : : {  428 : : Segment::Pointer s = inSegs  429 : : + (idx1.getVal() \* idx2.getCount()  430 : 33779 : + idx2.getVal());  431 : 33779 : TFBuffer::Pointer bufPtr = s->getTFs();  432 : :   433 [ + + ]: 33779 : if(bufPtr)  434 : : {  435 : 4796 : TF::Pointer \*TFs = bufPtr->getTFs();  436 : 4796 : if(0 == bufPtr->getSize())  437 : : {  438 : : //segEmpty = true;  439 : 4796 : continue;  440 : : }  441 : 4796 : segEmpty = false;  442 [ + + ]: 4796 : if( eLocationBeforeShape == mPriority  443 : : || eShapeBeforeLocation == mPriority )  444 : : {  445 : : //Sorting is not done in bucket modes  446 : 1754 : std::sort( TFs, TFs + bufPtr->getSize(), mCompare[index] );  447 : : }  448 : 4796 : try  449 : : {  450 : 4796 : pSDDB->segmentBuffGenerated(bufPtr);  451 : : }  452 [ # # ]: 0 : catch(DPL::Error \*pChild)  453 : : {  454 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  455 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  456 : : ErrorHandler::eErrFunction,  457 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  458 : : "SDDB::segmentBuffGenerated"));  459 : : }  460 : 4796 : try {  461 : 4796 : mSegmentWriter( s );  462 : : }  463 [ + - ]: 18 : catch(DPL::Error \*pChild)  464 : : {  465 : 18 : ErrorHandler \*pErr = ErrorHandler::instance();  466 : 18 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  467 : : ErrorHandler::eErrFunction,  468 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  469 : : "outputgenerator::writeSegment"));  470 : : }  471 : :   472 : : }  473 : : else  474 : : {  475 : : //segEmpty = true;  476 : 33761 : continue;  477 : : }  478 : : }  479 : 9145 : try  480 : : {  481 : 9145 : pSDDB->segmentMajorSwitch();  482 : : }  483 [ # # ]: 0 : catch(DPL::Error \*pChild)  484 : : {  485 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  486 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  487 : : ErrorHandler::eErrFunction,  488 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  489 : : "SDDB::segmentMajorSwitch"));  490 : : }  491 : : }  492 : : }  493 : :   494 : 1215 : } //namespace nft |

|  |
| --- |
| Generated by: [LCOV version 1.9](http://ltp.sourceforge.net/coverage/lcov.php) |