Search algorithm code library and instance generator Generated by Doxygen 1.8.17

1 File Documentation	1
1.1 /home/ultron/TCC-W/sources/CPUTimer.cpp File Reference	1
1.2 CPUTimer.cpp	1
1.3 /home/ultron/TCC-W/sources/GeneratorInstance.hpp File Reference	4
1.3.1 Function Documentation	5
1.3.1.1 LinearDecreasingDistribution()	5
1.3.1.2 LinearDecreasingDistribution_2D()	6
1.3.1.3 LinearDecreasingDistribution_3D()	7
1.3.1.4 LinearIncreasingDistribution()	8
1.3.1.5 LinearIncreasingDistribution_2D()	9
1.3.1.6 LinearIncreasingDistribution_3D()	10
1.3.1.7 LinearNormalDistribution()	11
1.3.1.8 LinearNormalDistribution_2D()	12
1.3.1.9 LinearNormalDistribution_3D()	13
1.4 GeneratorInstance.hpp	15
1.5 /home/ultron/TCC-W/sources/Main.cpp File Reference	18
1.5.1 Function Documentation	19
1.5.1.1 main()	19
1.5.1.2 test_D1()	21
1.5.1.3 test_D2()	24
1.5.1.4 test_D3()	26
1.6 Main.cpp	28
1.7 /home/ultron/TCC-W/sources/SearchAlgorithms.hpp File Reference	32
1.7.1 Function Documentation	34
1.7.1.1 binary_search()	35
1.7.1.2 binary_search_i()	36
1.7.1.3 binary_search_j()	37
1.7.1.4 binary_search_k()	38
1.7.1.5 exponential_search()	39
1.7.1.6 fibonaccian_search()	40
1.7.1.7 interpolation_search()	41
1.7.1.8 jump_search()	41
1.7.1.9 linear_search()	42
1.7.1.10 linialsaks_search() [1/2]	43
1.7.1.11 linialsaks_search() [2/2]	44
1.7.1.12 MAHL_e() [1/2]	46
1.7.1.13 MAHL_e() [2/2]	48
1.7.1.14 saddleback_ij()	50
1.7.1.15 saddleback_ik()	51
1.7.1.16 saddleback_jk()	52
1.7.1.17 saddleback_search() [1/2]	53
1.7.1.18 saddleback_search() [2/2]	54

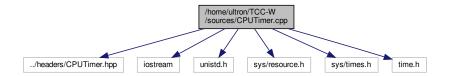
Index			65
1.8 SearchAlgorithms.hpp	 	 	56
1.7.1.20 shen_search() [2/2]	 	 	55
1.7.1.19 shen_search() [1/2]	 	 	55

Chapter 1

File Documentation

1.1 /home/ultron/TCC-W/sources/CPUTimer.cpp File Reference

#include "../headers/CPUTimer.hpp"
Include dependency graph for CPUTimer.cpp:



1.2 CPUTimer.cpp

```
00001 /*
00002 * CPUTimer.cpp
00003 *
          Created by Humberto Longo on 02/04/13. Instituto de Informatica - UFG
00004 *
00005 *
00006 *
00007
80000
00009 #include "../headers/CPUTimer.hpp"
00010
00011 //-
00013 CPUTimer::CPUTimer()
00014 {
00015
              started = false;
00016
00017
               CPUCurrSecs = 0;
00018
              CPUTotalSecs = 0;
00019
               CronoCurrSecs = 0;
00020
               CronoTotalSecs = 0;
00021 }
00022
00023 //-
00025 double CPUTimer::getCPUCurrSecs()
00026 {
00027
               return CPUCurrSecs;
00028 }
00029
00030 //
```

```
00032 double CPUTimer::getCPUTotalSecs()
00033 {
00034
             return CPUTotalSecs;
00035 }
00036
00037 //
00039 double CPUTimer::getCronoCurrSecs()
00040 {
00041
             return CronoCurrSecs;
00042 }
00043
00044 //---
00045
00046 double CPUTimer::getCronoTotalSecs()
00047 {
             return CronoTotalSecs;
00048
00049 }
00050
00051 //-
00052
00053 bool CPUTimer::start()
00054 {
00055
             bool status = true:
00056
00057
             CPUCurrSecs = 0;
00058
             CronoCurrSecs = 0;
00059
00060 CPUTStart = getCPUTime();
00061 CronoTStart = getRealTime();
00062
             gottime = false;
started = status;
00063
00064
00065
00066
             return ( status );
00067 }
00068
00069 //---
00070
00071 bool CPUTimer::stop()
00072 {
00073
             bool status = true;
00074
00075
             if (started)
00076
00077
         CPUTStop = getCPUTime();
00078
         CronoTStop = getRealTime();
00079
00080
         CPUTotalSecs += CPUTStop - CPUTStart;
00081
         CronoTotalSecs += CronoTStop - CronoTStart;
00082
             }
00083
             else
00084
             {
00085
                     std::cout « "CPUTimer::stop(): called without calling CPUTimer::start() first!\n";
00086
                     status = false;
00087
             }
00088
00089
             started = false;
00090
00091
             return status;
00092 }
00093
00094 //--
00095
00096 void CPUTimer::reset()
00097 {
             started = false;
00098
00099
00100
             CPUCurrSecs = 0;
             CPUTotalSecs = 0;
00101
             CronoCurrSecs = 0;
00102
00103
             CronoTotalSecs = 0;
00104 }
00105
00106 //---
00108 void CPUTimer::operator += ( CPUTimer t )
00109 {
             CPUCurrSecs += t.getCPUCurrSecs();
00110
             CPUTotalSecs += t.getCPUTotalSecs();
00111
             CronoCurrSecs += t.getCronoCurrSecs();
00112
             CronoTotalSecs += t.getCronoTotalSecs();
00113
00114 }
00115
00116 //-----
00117
00122 double CPUTimer::getCPUTime()
```

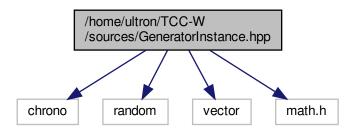
1.2 CPUTimer.cpp 3

```
00123 {
00124
       #if defined(_POSIX_TIMERS) && (_POSIX_TIMERS > 0)
00125
        /\star Prefer high-res POSIX timers, when available. \star/
00126
       clockid_t id;
00127
       struct timespec ts;
00128
        #if _POSIX_CPUTIME > 0
00129
00130
       /\star Clock ids vary by OS. Query the id, if possible. \star/
00131
        if ( clock_getcpuclockid( 0, &id ) == -1 )
00132
        #endif
       #if defined(CLOCK PROCESS CPUTIME ID)
00133
        /* Use known clock id for AIX, Linux, or Solaris. */
00134
00135
          id = CLOCK_PROCESS_CPUTIME_ID;
00136
       #elif defined(CLOCK_VIRTUAL)
        /* Use known clock id for BSD or HP-UX. */
00137
00138
         id = CLOCK_VIRTUAL;
00139
       #else
00140
         id = (clockid t) - 1;
00141
00142
       if ((id != (clockid_t) - 1) && (clock_gettime( id, &ts ) != -1))
    return ((double) ts.tv_sec + (double) ts.tv_nsec / 1000000000.0);
00143
00144
       #endif
00145
00146
00147
       #if defined(RUSAGE_SELF)
00148
       struct rusage rusage;
00149
00150
       if (getrusage( RUSAGE_SELF, &rusage ) != -1)
00151
         return ((double) rusage.ru_utime.tv_sec + (double) rusage.ru_utime.tv_usec / 1000000.0);
00152
       #endif
00153
00154
             /* Failed. */
00155
             return -1.0;
00156 }
00157
00158 //-----
00159
00164 double CPUTimer::getRealTime()
00165 {
00166
       #if defined(_POSIX_TIMERS) && (_POSIX_TIMERS > 0)
00167
             struct timespec ts;
       #if defined(CLOCK_MONOTONIC_PRECISE)
00168
00169
             /* BSD. -----
00170
              const clockid_t id = CLOCK_MONOTONIC_PRECISE;
00171
       #elif defined(CLOCK_MONOTONIC_RAW)
00172
             /* Linux. ----
00173
              const clockid_t id = CLOCK_MONOTONIC_RAW;
00174
       #elif defined(CLOCK_HIGHRES)
00175
             /* Solaris. -----
00176
              const clockid_t id = CLOCK_HIGHRES;
       #elif defined(CLOCK_MONOTONIC)
00177
00178
             /* AIX, BSD, Linux, POSIX, Solaris. ----- */
00179
              const clockid_t id = CLOCK_MONOTONIC;
00180
       #elif defined(CLOCK_REALTIME)
             /* AIX, BSD, HP-UX, Linux, POSIX. ---- */
const clockid_t id = CLOCK_REALTIME;
00181
00182
00183
00184
       /* Unknown. */
00185
              const clockid_t id = (clockid_t) - 1;
       #endif
00186
00187
             if ((id != (clockid_t) - 1) && (clock_gettime( id, &ts ) != -1))
00188
00189
                      return ((double) ts.tv_sec + (double) ts.tv_nsec / 1000000000.0);
00190
00191
       #elif defined(__MACH__) && defined(__APPLE__)
00192
              /* OSX. -
              static double timeConvert = 0.0;
00193
00194
00195
              if (timeConvert == 0.0)
00196
              {
00197
                      mach_timebase_info_data_t timeBase;
00198
                     (void) mach_timebase_info( &timeBase );
timeConvert = (double) timeBase.numer / (double) timeBase.denom / 1000000000.0;
00199
00200
00201
             }
00202
00203
             return (double) mach_absolute_time() * timeConvert;
00204
       #else
00205
00206
       /* Failed. */
             return -1.0;
00207
       #endif
00209 }
00210
00211 //----
```

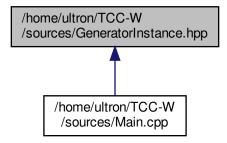
1.3 /home/ultron/TCC-W/sources/GeneratorInstance.hpp File Reference

#include <chrono>
#include <random>
#include <vector>
#include <math.h>

Include dependency graph for GeneratorInstance.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- template < class ForwardIt, class T >
 void LinearIncreasingDistribution (ForwardIt first, ForwardIt last, const T &min_value, const T &max_value)
 Function to generate an increasing uniform distribution.
- template < class ForwardIt, class T >
 void Linear Decreasing Distribution (ForwardIt first, ForwardIt last, const T &min_value, const T &max_value)
 Function to generate a decreasing uniform distribution.
- template < class ForwardIt, class T >
 void LinearNormalDistribution (ForwardIt first, ForwardIt last, const T &min_value, const T &max_value)
 Function to generate a normal uniform distribution.

template < class ForwardIt , class T >
 void LinearIncreasingDistribution_2D (ForwardIt first, ForwardIt last, const T &min_value, const T &max_←
 value)

Functions for generating a uniform distribution for a two-dimensional array.

template < class ForwardIt , class T >
 void LinearDecreasingDistribution_2D (ForwardIt first, ForwardIt last, const T &min_value, const T &max_←
 value)

Function to generate a decreasing uniform distribution for a two-dimensional array.

- template < class ForwardIt, class T >
 void LinearNormalDistribution_2D (ForwardIt first, ForwardIt last, const T &min_value, const T &max_value)
 Function to generate a normal uniform distribution for a two-dimensional array.
- template < class ForwardIt, class T >
 void LinearIncreasingDistribution_3D (ForwardIt first, ForwardIt last, const T &min_value, const T &max_
 value)

Functions for generating a uniform distribution for a three-dimensional array.

template < class ForwardIt, class T >
 void LinearDecreasingDistribution_3D (ForwardIt first, ForwardIt last, const T &min_value, const T &max_
 value)

Function to generate a decreasing uniform distribution for a three-dimensional array.

template < class ForwardIt, class T >
 void LinearNormalDistribution_3D (ForwardIt first, ForwardIt last, const T &min_value, const T &max_value)
 Function to generate a normal uniform distribution for a three-dimensional array.

1.3.1 Function Documentation

1.3.1.1 LinearDecreasingDistribution()

Function to generate a decreasing uniform distribution.

Parameters

first	iterator to start of array.
last	iterator to end of matrix.
min_value	initial value of sequence.
max_value	maximum sequence offset.

Definition at line 58 of file GeneratorInstance.hpp.

Referenced by test D1().

Here is the caller graph for this function:



1.3.1.2 LinearDecreasingDistribution_2D()

Function to generate a decreasing uniform distribution for a two-dimensional array.

Parameters

first	iterator to start of array.
last	iterator to end of matrix.
min_value	initial value of sequence.
max_value	maximum sequence offset.

Definition at line 136 of file GeneratorInstance.hpp.

```
00136
00137
             unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
         distribution.*/
00138
            std::mt19937_64 gen(seed);
             \verb|std::uniform_real_distribution| <> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00139
00140
             T offset;
            int M = (last - first); /* Size of the first dimension.*/
00141
             int N = (first[0].size()); /* Size of the second dimension.*/
offset = (max_value - min_value+1.) / (T) (M+N); /*The values are generated within this range and
00142
        added to the previous element of the sequence.*/
            for( int i = 0; i < M; ++i) {
    for( int j=0; j < N; ++j) {
        if( i == 0 || j == 0) {
            if( i == 0 && j == 0)
        }
}</pre>
00144
00145
00146
00147
00148
                                  first[i][j] = (T)((offset)*(1. - std::sqrt(1.-dis(gen)))) + min_value;
```

Referenced by test D2().

Here is the caller graph for this function:



1.3.1.3 LinearDecreasingDistribution_3D()

Function to generate a decreasing uniform distribution for a three-dimensional array.

Parameters

first	iterator to start of array.
last	iterator to end of matrix.
min_value	initial value of sequence.
max_value	maximum sequence offset.

Definition at line 254 of file GeneratorInstance.hpp.

```
00254
00255
            unsigned seed = std::chrono::system clock::now().time since epoch().count(); /* Random seed for
        distribution.*/
00256
            std::mt19937_64 gen(seed);
00257
            00258
            T offset;
00259
            int M = (last - first); /* Size of the first dimension.*/
            int N = (first[0].size()); /* Size of the second dimension.*/
int P = (first[0][0].size()); /* Size of the third dimension.*/
offset = (max_value - min_value+1.) / (T) (M+N+P); /*The values are generated within this range
00260
00261
00262
        and added to the previous element of the sequence.*/
           for( int i = 0; i < M; ++i) {
    for( int j = 0; j < N; ++j) {
        for( int k = 0; k < P; ++k) {
            if( i == 0 || j == 0 || k==0) {</pre>
00263
00264
00265
00266
00267
                                if(i == 0 \&\& j == 0 \&\& k==0)
```

```
00268
                                 first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) + min_value;
                            else if( i == 0) {
   if( k == 0)
00269
00270
                                     first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
00271
        first[i][j-1][k];
00272
                                 else if (j==0)
                                     first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
00273
        first[i][j][k-1];
00274
                                     first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
00275
        std::max(first[i][j-1][k], first[i][j][k-1]);
00276
                            else if(j == 0){
    if( i == 0)
00277
00278
00279
                                     first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
        first[i][j][k-1];
00280
                                 else if (k==0)
                                     first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
00281
        first[i-1][j][k];
00282
                                 else
00283
                                     first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
        std::max(first[i-1][j][k], first[i][j][k-1]);
00284
                             }else{
                                 if( i == 0)
00285
                                     first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(qen)))) +
00286
        first[i][j-1][k];
00287
                                 else if(j==0)
00288
                                     first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
        first[i-1][j][k];
00289
                                     first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
00290
       std::max(first[i-1][j][k], first[i][j-1][k]);
00291
00292
                        }else
        first[i][j][k] = (T) (offset * (1.-std::sqrt(1.-dis(gen)))) + std::max(first[i-1][j][k], std::max(first[i][j-1][k], first[i][j][k-1])); 
00293
00294
                   }
00295
00296
00297 }
```

Referenced by test_D3().

Here is the caller graph for this function:



1.3.1.4 LinearIncreasingDistribution()

Function to generate an increasing uniform distribution.

Parameters

first	iterator to start of array.
last	iterator to end of matrix.
min_value	initial value of sequence.
max_value	maximum sequence offset.

Definition at line 36 of file GeneratorInstance.hpp.

```
00036
00037
           unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
       distribution.*
           std::mt19937_64 gen(seed);
00038
           \verb|std::uniform_real_distribution| <> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00039
00040
           T offset:
          int N = (last - first); /* Array size.*/
offset = (max_value - min_value+1.) /(T)N; /*The values are generated within this range and added
00041
00042
       to the previous element of the sequence.*/
          for(int i = 0; i < N; ++i) {
   if(i == 0)</pre>
00043
00044
00045
                   first[i] = (T) (offset * std::sqrt(dis(gen))) + min_value;
00046
00047
                   first[i] = (T) (offset * std::sqrt(dis(gen))) + first[i-1];
00048
00049 }
```

Referenced by test D1().

Here is the caller graph for this function:



1.3.1.5 LinearIncreasingDistribution_2D()

Functions for generating a uniform distribution for a two-dimensional array.

Function to generate a growing uniform distribution for a two-dimensional array.

Parameters

first	iterator to start of array.	
last	iterator to end of matrix.	
min_value	initial value of sequence.	
max value maximum sequence offset.		

Definition at line 106 of file GeneratorInstance.hpp.

```
00107
           unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
        distribution.*/
00108
           std::mt19937_64 gen(seed);
           std::uniform_real_distribution<> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00109
00110
           T offset;
           int M = (last - first);  /* Size of the first dimension.*/
int N = (first[0].size()); /* Size of the second dimension.*/
offset = (max_value - min_value+1.) /(T)(M+N); /*The values are generated within this range and
00111
00112
00113
        added to the previous element of the sequence.*/
00114
           for(int i = 0; i < M; ++i){</pre>
00115
                for (int j = 0; j < N; ++j) {
00116
                     if( i == 0 || j == 0){
                          if( i == 0 && j == 0)
    first[i][j] = (T)(offset * std::sqrt(dis(gen))) + min_value;
00117
00118
00119
                          else if (i == 0)
00120
                              first[i][j] = (T)(offset * std::sqrt(dis(gen))) + first[i][j-1];
00121
                          else
00122
                              first[i][j] = (T)(offset * std::sqrt(dis(gen))) + first[i-1][j];
00123
                     }else
                          first[i][j] = (T) (offset * std::sqrt(dis(gen))) + std::max(first[i-1][j],
00124
        first[i][j-1]);
00125
00126
00127 }
```

Referenced by test_D2().

Here is the caller graph for this function:



1.3.1.6 LinearIncreasingDistribution 3D()

Functions for generating a uniform distribution for a three-dimensional array.

Function to generate a growing uniform distribution for a three-dimensional array.

Parameters

first	iterator to start of array.
last	iterator to end of matrix.
min_value	initial value of sequence.
max_value	maximum sequence offset.

Definition at line 201 of file GeneratorInstance.hpp.

```
00202
            unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
        distribution.*/
00203
            std::mt19937_64 gen(seed);
00204
            std:uniform\_real\_distribution <> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00205
            T offset;
00206
            int M = (last - first); /* Size of the first dimension.*/
            int N = (first[0].size()); /* Size of the second dimension.*/
int P = (first[0][0].size()); /* Size of the third dimension.*/
offset = (max_value - min_value+1.) /(T) (M+N+P); /*The values are generated within this range and
00207
00208
00209
        added to the previous element of the sequence.*/
00210
            for ( int i = 0; i < M; ++i) {
                 ( int i = 0; 1 < M; ++1){
  for( int j = 0; j < N; ++j) {
    for( int k = 0; k < P; ++k) {
      if( i == 0 || j == 0 || k==0) {
        if( i == 0 && j == 0 && k==0)
            first[i][j][k] = (T) (offset * std::sqrt(dis(gen))) + min_value;</pre>
00211
00212
00213
00214
00215
00216
                                else if( i == 0){
                                    if(k == 0)
00217
00218
                                         first[i][j][k] = (T) (offset * std::sqrt(dis(gen))) + first[i][j-1][k];
00219
                                     else if (j==0)
                                         first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) + first[i][j][k-1];
00220
00221
                                     else
                                         first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) +
00222
        std::max(first[i][j-1][k], first[i][j][k-1]);
00223
                               else if(j == 0){
    if( i == 0)
00224
00225
                                         first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) + first[i][j][k-1];
00226
00227
                                     else if (k==0)
00228
                                         first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) + first[i-1][j][k];
00229
00230
                                         first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) +
        \texttt{std::max(first[i-1][j][k], first[i][j][k-1]);}\\
00231
                                }else{
                                    if(i == 0)
00232
00233
                                         first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) + first[i][j-1][k];
00234
                                     else if (j==0)
00235
                                         first[i][j][k] = (T) (offset * std::sqrt(dis(gen))) + first[i-1][j][k];
00236
                                     else
                                         first[i][j][k] = (T) (offset * std::sqrt(dis(gen))) +
00237
        std::max(first[i-1][j][k], first[i][j-1][k]);
00238
00239
00240
                                first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) + std::max(first[i-1][j][k],
        \mathtt{std::max}(\mathtt{first[i][j-1][k],\ first[i][j][k-1]));}
00241
                     }
00242
                 }
00243
            }
00244 }
```

Referenced by test_D3().

Here is the caller graph for this function:



1.3.1.7 LinearNormalDistribution()

```
template<class ForwardIt , class T >
void LinearNormalDistribution (
```

```
ForwardIt first,
ForwardIt last,
const T & min_value,
const T & max_value )
```

Function to generate a normal uniform distribution.

Parameters

first	iterator to start of array.
last	iterator to end of matrix.
min_value	initial value of sequence.
max_value	maximum sequence offset.

Definition at line 80 of file GeneratorInstance.hpp.

```
00080
00081
            unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
        distribution.*/
00082
            std::mt19937_64 gen(seed);
00083
            std::uniform_real_distribution<> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00084
            T offset;
        int N = (last - first); /* Array size.*/
offset = (max_value - min_value+1.) / (T)N; /*The values are generated within this range and added to the previous element of the sequence.*/
00085
00086
           for( int i = 0; i < N; ++i) {
   if(i == 0)</pre>
00087
00088
                     first[i] = (T)((offset) * dis(gen)) + min_value;
00089
00090
                 else
00091
                     first[i] = (T)((offset) * dis(gen)) + first[i-1];
00092
            }
00093 }
```

Referenced by test_D1().

Here is the caller graph for this function:



1.3.1.8 LinearNormalDistribution 2D()

Function to generate a normal uniform distribution for a two-dimensional array.

Parameters

first	iterator to start of array.
last	iterator to end of matrix.
min_value	initial value of sequence.
max_value	maximum sequence offset.

Definition at line 167 of file GeneratorInstance.hpp.

```
00167
00168
         unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
      distribution.*
00169
         std::mt19937_64 gen(seed);
         \verb|std::uniform_real_distribution| <> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00170
00171
         T offset:
00172
         int M = (last - first); /* Size of the first dimension.*/
00173
         int N = (first[0].size()); /* Size of the second dimension.*/
00174
         offset = (\max_{value} - \min_{value} + 1.) / (T) (M+N); /*The values are generated within this range and
      added to the previous element of the sequence. \!\!\!\!\star/
         00175
00176
00177
00178
00179
00180
                     else if (i == 0)
00181
                        first[i][j] = (T)(offset * dis(gen)) + first[i][j-1];
00182
                     else
00183
                         first[i][j] = (T)(offset * dis(gen)) + first[i-1][j];
00184
                 }else
00185
                     first[i][j] = (T) (offset * dis(gen)) + std::max(first[i-1][j], first[i][j-1]);
00186
             }
00187
         }
00188 }
```

Referenced by test_D2().

Here is the caller graph for this function:



1.3.1.9 LinearNormalDistribution_3D()

Function to generate a normal uniform distribution for a three-dimensional array.

Parameters

first	iterator to start of array.
last	iterator to end of matrix.
min_value	initial value of sequence.
max_value	maximum sequence offset.

Definition at line 307 of file GeneratorInstance.hpp.

```
00307
00308
           unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
        distribution.*/
00309
           std::mt19937_64 gen(seed);
00310
           std:uniform\_real\_distribution <> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00311
            T offset;
           int M = (last - first); /* Size of the first dimension.*/ int N = (first[0].size()); /* Size of the second dimension.*/
00312
00313
           int P = (first[0][0].size()); /* Size of the third dimension.*/
offset = (max_value - min_value+1.) / (T) (M+N+P); /*The values are generated within this range and
00314
00315
        added to the previous element of the sequence.*/
00316
            for ( int i = 0; i < M; ++i) {
                for( int j = 0; j < N; ++j) {
   for( int k = 0; k < P; ++k) {
      if( i == 0 || j == 0 || k==0) {
       if( i == 0 && j == 0 && k==0)
      }
}</pre>
00317
00318
00319
00320
                                   first[i][j][k] = (T)(offset * dis(gen)) + min_value;
00321
00322
00323
                                   if(k == 0)
00324
                                        first[i][j][k] = (T)(offset * dis(gen)) + first[i][j-1][k];
00325
                                   else if (i==0)
                                       first[i][j][k] = (T)(offset * dis(gen)) + first[i][j][k-1];
00326
00327
                                   else
00328
                                        first[i][j][k] = (T)(offset * dis(gen)) + std::max(first[i][j-1][k],
        first[i][j][k-1]);
00329
                              else if(j == 0){
00330
                                   if( i == 0)
00331
                                       first[i][j][k] = (T)(offset * dis(gen)) + first[i][j][k-1];
00332
00333
                                   else
00334
                                        first[i][j][k] = (T)(offset * dis(gen)) + first[i-1][j][k];
00335
                                        first[i][j][k] = (T)(offset * dis(gen)) + std::max(first[i-1][j][k],
00336
        first[i][j][k-1]);
00337
                              }else{
00338
                                   if(i == 0)
00339
                                        first[i][j][k] = (T)(offset * dis(gen)) + first[i][j-1][k];
00340
                                   else if(j==0)
                                       first[i][j][k] = (T)(offset * dis(gen)) + first[i-1][j][k];
00341
00342
                                   else
                                        first[i][j][k] = (T)(offset * dis(gen)) + std::max(first[i-1][j][k],
00343
        first[i][j-1][k]);
00344
00345
00346
                              first[i][j][k] = (T)(offset * dis(gen)) + std::max(first[i-1][j][k],
        \mathtt{std::max}(\mathtt{first[i][j-1][k],\ first[i][j][k-1]));}
00347
                    }
00348
00349
00350 }
```

Referenced by test D3().

Here is the caller graph for this function:



1.4 GeneratorInstance.hpp

```
00001
00012 /*
                  GeneratorInstance.h
 00014 *
00015 *
                   Created by Walisson Pereira de Jesus on 05/12/19.
00016 *
                   Instituto de Informatica - UFG
00017 *
00018
00019
00020 #ifndef GeneratorInstance_hpp
00021 #define GeneratorInstance_hpp
00022
00023 #include <chrono>
00024 #include <random>
00025 #include <vector>
00026 #include <math.h>
00027
00035 template<class ForwardIt, class T>
00036 void LinearIncreasingDistribution(ForwardIt first, ForwardIt last, const T& min_value, const T&
             max value) {
00037
                   unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
             distribution.*/
00038
                   std::mt19937_64 gen(seed);
00039
                    std::uniform_real_distribution<> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00040
                    T offset;
00041
                   int N = (last - first); /* Array size.*/
offset = (\max_{value} - \min_{value+1.}) /(T)N; /*The values are generated within this range and added
00042
             to the previous element of the sequence.*/
00043
                  for (int i = 0; i < N; ++i) {
00044
                           if(i == 0)
00045
                                    first[i] = (T)(offset * std::sqrt(dis(gen))) + min_value;
00046
                            else
00047
                                    first[i] = (T) (offset * std::sqrt(dis(gen))) + first[i-1];
00048
00049 }
00057 template<class ForwardIt, class T>
00058 void LinearDecreasingDistribution(ForwardIt first, ForwardIt last, const T& min_value, const T&
              max_value) {
00059
                   unsigned seed = std::chrono::system clock::now().time since epoch().count(); /* Random seed for
             distribution.*
00060
                   std::mt19937_64 gen(seed);
00061
                    std::uniform real distribution<> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00062
                    T offset;
                   int N = (last - first); /* Array size.*/ offset = (max\_value - min\_value+1.) / (T)N; /*The values are generated within this range and
00063
00064
              added to the previous element of the sequence.*/
00065
                   for( int i = 0; i < N; ++i){
                           if( i == 0)
00066
00067
                                    first[i] = (T)((offset) * (1. - std::sqrt(1.-dis(gen)))) + min_value;
00068
00069
                                    first[i] = (T)((offset) * (1. - std::sqrt(1.-dis(gen)))) + first[i-1];
00070
00071 }
00079 template<class ForwardIt, class T>
00080 void LinearNormalDistribution(ForwardIt first, ForwardIt last, const T& min_value, const T&
              max_value) {
00081
                   unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
              distribution.*
                   std::mt19937_64 gen(seed);
00083
                    std::uniform_real_distribution<> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00084
                   int N = (last - first); /* Array size.*/
offset = (max_value - min_value+1.) / (T)N; /*The values are generated within this range and
00085
00086
              added to the previous element of the sequence.*/
00087
                   for ( int i = 0; i < N; ++i) {
                            if(i == 0)
00088
00089
                                    first[i] = (T)((offset) * dis(gen)) + min_value;
00090
00091
                                    first[i] = (T)((offset) * dis(gen)) + first[i-1];
00092
                    }
00093 }
00094
00095
00097
00105 template<class ForwardIt, class T>
00106 void LinearIncreasingDistribution_2D(ForwardIt first, ForwardIt last, const T& min_value, const T&
             max_value) {
00107
                    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
              distribution.*/
00108
                   std::mt19937_64 gen(seed);
00109
                    \texttt{std::uniform\_real\_distribution} <> \; \texttt{dis} \; (0.0, \; 1.0); \; / \star \; \texttt{Uniform} \; \texttt{value} \; \; \texttt{generated} \; \; \texttt{between} \; \; 0 \; \; \texttt{and} \; \; 1. \star / \; \texttt{value} \; \; \texttt{val
00110
                    T offset;
00111
                    int M = (last - first);
                                                                      /* Size of the first dimension.*/
00112
                    int N = (first[0].size()); /* Size of the second dimension.*/
00113
                    offset = (\max_{value} - \min_{value} + 1.) / (T) (M+N); /*The values are generated within this range and
```

```
added to the previous element of the sequence.*/
          for(int i = 0; i < M; ++i){
00114
               for(int j = 0; j < N; ++j) {
    if( i == 0 || j == 0) {
00115
00116
                       if( i == 0 && j == 0)
00117
                           first[i][j] = (T)(offset * std::sqrt(dis(gen))) + min_value;
00118
                       else if( i == 0)
00119
00120
                           first[i][j] = (T) (offset * std::sqrt(dis(gen))) + first[i][j-1];
00121
00122
                           first[i][j] = (T)(offset * std::sqrt(dis(gen))) + first[i-1][j];
00123
                   lelse
00124
                       first[i][j] = (T)(offset * std::sqrt(dis(qen))) + std::max(first[i-1][j],
       first[i][j-1]);
00125
00126
00127 }
00135 template<class ForwardIt, class T>
00136 void LinearDecreasingDistribution_2D(ForwardIt first, ForwardIt last, const T& min_value, const T&
       max_value) {
00137
          unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
       distribution.*
          std::mt19937_64 gen(seed);
00138
00139
           std::uniform_real_distribution<> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00140
          T offset;
           int M = (last -
00141
                           first); /* Size of the first dimension.*/
           int N = (first[0].size()); /* Size of the second dimension.*/
00142
           offset = (max\_value - min\_value+1.) / (T) (M+N); /*The values are generated within this range and
00143
       00144
          for ( int i = 0; i < M; ++i) {
              for( int j=0; j < N; ++j) {
    if( i == 0 || j == 0) {
        if( i == 0 && j ==
00145
00146
00147
00148
                           first[i][j] = (T)((offset)*(1. - std::sqrt(1.-dis(gen)))) + min_value;
00149
                       else if( i == 0)
00150
                           first[i][j] = (T)((offset)*(1. - std::sqrt(1.-dis(gen)))) + first[i][j-1];
00151
                       else
00152
                           first[i][j] = (T)((offset)*(1. - std::sgrt(1.-dis(gen)))) + first[i-1][j];
00153
                   lelse
00154
                       first[i][j] = (T)((offset)*(1. - std::sqrt(1.-dis(gen)))) + std::max(first[i-1][j],
       first[i][j-1]);
00155
00156
          }
00157 }
00158
00166 template<class ForwardIt, class T>
00167 void LinearNormalDistribution_2D(ForwardIt first, ForwardIt last, const T& min_value, const T&
       max_value) {
00168
          unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
       distribution.*/
00169
          std::mt19937_64 gen(seed);
00170
          std::uniform_real_distribution<> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00171
           T offset;
00172
           int M = (last - first); /* Size of the first dimension.*/
          int N = (first[0].size()); /* Size of the second dimension.*/
offset = (max_value - min_value+1.) / (T) (M+N); /*The values are generated within this range and
00173
00174
       added to the previous element of the sequence.*/
00175
          for ( int i = 0; i < M; ++i) {
               for ( int j = 0; j < N; ++ j) {
00176
00177
                   if( i == 0 || j == 0){
                       if( i == 0 && j == 0)
00178
00179
                           first[i][j] = (T)(offset * dis(gen)) + min_value;
00180
                       else if (i == 0)
00181
                           first[i][j] = (T)(offset * dis(gen)) + first[i][j-1];
00182
                       else
00183
                           first[i][j] = (T)(offset * dis(gen)) + first[i-1][j];
00184
                   }else
00185
                       first[i][j] = (T)(offset * dis(gen)) + std::max(first[i-1][j], first[i][j-1]);
00186
               }
00187
          }
00188 }
00189
00190
00192
00200 template<class ForwardIt, class T>
00201 void LinearIncreasingDistribution 3D(ForwardIt first, ForwardIt last, const T& min value, const T&
       max value) {
00202
          unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
       distribution.*/
          std::mt19937_64 gen(seed);
00203
00204
          std::uniform real distribution<> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00205
          T offset;
           int M = (last -
00206
                           first); /* Size of the first dimension.*/
           int N = (first[0].size()); /* Size of the second dimension.*/
00207
00208
                    (first[0][0].size()); /* Size of the third dimension.*/
00209
           \text{offset = (max\_value - min\_value+1.) / (T) (M+N+P); /*The values are generated within this range and } \\ 
       added to the previous element of the sequence.*/
for( int i = 0; i < M; ++i) {</pre>
00210
```

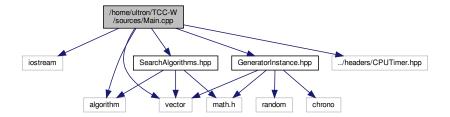
```
00211
                for ( int j = 0; j < N; ++ j) {
                    for( int k = 0; k < P; ++k) {
    if( i == 0 || j == 0 || k==0) {
        if( i == 0 && j == 0 && k==0)
00212
00213
00214
00215
                                 first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) + min_value;
00216
                             else if (i == 0) {
                                  if(k == 0)
00217
00218
                                      first[i][j][k] = (T) (offset * std::sqrt(dis(gen))) + first[i][j-1][k];
00219
                                  else if (j==0)
00220
                                      first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) + first[i][j][k-1];
00221
                                  else
                                     first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) +
00222
        std::max(first[i][j-1][k], first[i][j][k-1]);
00223
                             else if(j == 0){
   if( i == 0)
00224
00225
                                      first[i][j][k] = (T) (offset * std::sqrt(dis(gen))) + first[i][j][k-1];
00226
00227
                                  else if (k==0)
00228
                                     first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) + first[i-1][j][k];
00229
                                  else
                                      first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) +
00230
        std::max(first[i-1][j][k], first[i][j][k-1]);
00231
                             }else{
                                  if(i == 0)
00232
00233
                                      first[i][j][k] = (T)(offset * std::sqrt(dis(qen))) + first[i][j-1][k];
00234
                                  else if(j==0)
00235
                                      first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) + first[i-1][j][k];
00236
00237
                                      first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) +
        std::max(first[i-1][j][k], first[i][j-1][k]);
00238
00239
                         }else
                             first[i][j][k] = (T)(offset * std::sqrt(dis(gen))) + std::max(first[i-1][j][k],
00240
        std::max(first[i][j-1][k], first[i][j][k-1]));
00241
00242
               }
00243
           }
00244 }
00245
00253 template<class ForwardIt, class T>
00254 void LinearDecreasingDistribution_3D(ForwardIt first, ForwardIt last, const T& min_value, const T&
        max value) {
00255
           unsigned seed = std::chrono::system clock::now().time since epoch().count(): /* Random seed for
        distribution.*/
           std::mt19937_64 gen(seed);
00256
00257
           std::uniform_real_distribution<> dis(0.0, 1.0); /* Uniform value generated between 0 and 1.*/
00258
           T offset;
           int M = (last - first); /* Size of the first dimension.*/ int N = (first[0].size()); /* Size of the second dimension.*/ int P = (first[0][0].size()); /* Size of the third dimension.*/
00259
00260
00261
           offset = (max_value - min_value+1.) / (T) (M+N+P); /*The values are generated within this range
00262
        and added to the previous element of the sequence.*/
00263
           for ( int i = 0; i < M; ++i) {
               for( int j = 0; j < N; ++j){
  for( int k = 0; k < P; ++k){
    if( i == 0 || j == 0 || k==0){
        if( i == 0 && j == 0 && k==0)
00264
00265
00266
00267
                                 first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) + min_value;
00268
00269
                             else if( i == 0){
00270
                                  if(k == 0)
00271
                                      first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
        first[i][i-1][k];
00272
                                  else if (j==0)
00273
                                      first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
        first[i][j][k-1];
00274
                                      first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
00275
        std::max(first[i][j-1][k], first[i][j][k-1]);
00276
00277
                             else if(j == 0){
00278
                                  if(i == 0)
00279
                                      first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
        first[i][j][k-1];
00280
                                  else if (k==0)
                                      first[i][j][k] = (T)(offset * (1.-std::sgrt(1.-dis(gen)))) +
00281
        first[i-1][j][k];
00282
                                  else
00283
                                      first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
        \texttt{std::max(first[i-1][j][k], first[i][j][k-1]);}\\
00284
                             lelse(
00285
                                 if(i == 0)
00286
                                      first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
        first[i][j-1][k];
00287
                                  else if (j==0)
00288
                                      first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
        first[i-1][j][k];
00289
                                  else
```

```
first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
              std::max(first[i-1][j][k], first[i][j-1][k]);
00291
00292
                                           }else
00293
                                                  first[i][j][k] = (T)(offset * (1.-std::sqrt(1.-dis(gen)))) +
              std::max(first[i-1][j][k], std::max(first[i][j-1][k], first[i][j][k-1]));
00294
                                }
00295
                           }
00296
00297 }
00298
00306 template<class ForwardIt, class T>
00307 void LinearNormalDistribution_3D(ForwardIt first, ForwardIt last, const T& min_value, const T&
              max_value) {
00308
                   unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); /* Random seed for
              distribution.*/
                   std::mt19937_64 gen(seed);
00309
00310
                   \verb|std::uniform_real_distribution| <> | dis(0.0, 1.0); /* | Uniform_real_distribution| <> | dis(0.0, 1.0); /* | dis(0.0, 1.0); /* | Uniform_real_distribution| <> | dis(0.0, 1.0); /* | Uniform_real_distribution| <> | dis(0.0, 1.0); /* | dis(0.0, 
00311
                   T offset;
00312
                                                  first); /* Size of the first dimension.*/
00313
                    int N = (first[0].size()); /* Size of the second dimension.*/
00314
                   int P = (first[0][0].size()); /* Size of the third dimension.*/
                   offset = (max\_value - min\_value+1.) / (T) (M+N+P); /*The values are generated within this range and
00315
             added to the previous element of the sequence. \!\!\!\!\star/
                   for( int i = 0; i < M; ++i){
    for( int j = 0; j < N; ++j) {
        for( int k = 0; k < P; ++k) {</pre>
00316
00317
00318
                                           if( i == 0 || j == 0 || k==0) {
    if( i == 0 && j == 0 && k==0)
00319
00320
00321
                                                          first[i][j][k] = (T)(offset * dis(gen)) + min_value;
00322
                                                   else if (i == 0) {
00323
                                                           if(k == 0)
00324
                                                                  first[i][j][k] = (T)(offset * dis(gen)) + first[i][j-1][k];
00325
                                                           else if(j==0)
00326
                                                                  first[i][j][k] = (T)(offset * dis(gen)) + first[i][j][k-1];
00327
                                                           else
                                                                   first[i][j][k] = (T)(offset * dis(gen)) + std::max(first[i][j-1][k],
00328
              first[i][j][k-1]);
00329
00330
                                                   else if(j == 0){
00331
                                                           if(i == 0)
                                                                  first[i][j][k] = (T) (offset * dis(gen)) + first[i][j][k-1];
00332
00333
                                                           else if (k==0)
00334
                                                                  first[i][j][k] = (T)(offset * dis(gen)) + first[i-1][j][k];
00335
00336
                                                                   first[i][j][k] = (T)(offset * dis(gen)) + std::max(first[i-1][j][k],
              first[i][j][k-1]);
00337
                                                   }else{
00338
                                                           if(i == 0)
                                                                  first[i][j][k] = (T) (offset * dis(gen)) + first[i][j-1][k];
00339
00340
                                                           else if (j==0)
00341
                                                                  first[i][j][k] = (T)(offset * dis(gen)) + first[i-1][j][k];
00342
00343
                                                                   first[i][j][k] = (T)(offset * dis(gen)) + std::max(first[i-1][j][k],
              first[i][j-1][k]);
00344
00345
                                                   first[i][j][k] = (T)(offset * dis(gen)) + std::max(first[i-1][j][k],
00346
              std::max(first[i][j-1][k], first[i][j][k-1]));
00347
00348
                           }
00349
00350 }
00352 #endif
```

1.5 /home/ultron/TCC-W/sources/Main.cpp File Reference

```
#include <iostream>
#include <vector>
#include <algorithm>
#include "SearchAlgorithms.hpp"
#include "GeneratorInstance.hpp"
#include "../headers/CPUTimer.hpp"
```

Include dependency graph for Main.cpp:



Functions

- void test_D1 (int N, int ld, int min_value, int interval)
- · void test D2 (int M, int N, int Id, int min value, int interval)
- void test_D3 (int M, int N, int P, int Id, int min_value, int interval)
- int main ()

Function main.

1.5.1 Function Documentation

1.5.1.1 main()

```
int main ( )
```

Function main.

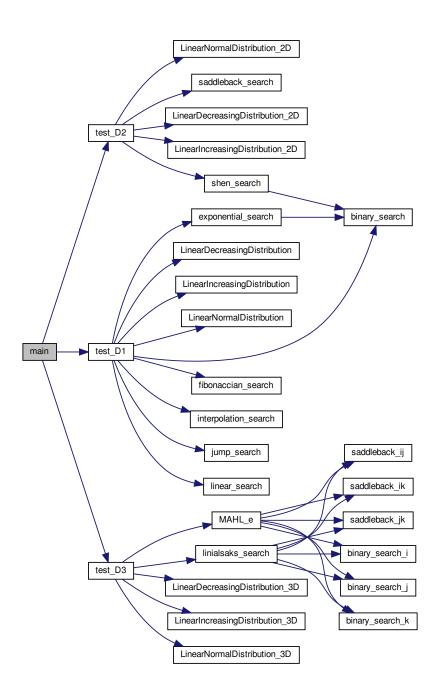
Definition at line 297 of file Main.cpp.

```
00297
00298
                                          int option;
00299
                                        do{
00300
                                                          printf("Test search algorithms for: \\ \\ n1 - one-dimensional \\ \\ n2 - two-dimensional \\ \\ n3 - two-dimensional \\ \\ n4 - two-dimensional \\ \\ n5 - two-dimensional \\ \\ n5 - two-dimensional \\ \\ n6 - two-dimensional \\ \\ n7 - two-dimensional \\ \\ n8 - two-dimensional \\ \\ \\ n8 - two-dimensional \\ \\ \\ n8 - two-dimensional \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\
                             three-dimensional\nAnother value to leave:\n");
00301
                                                          scanf(" %d", &option);
00302
00303
                                                          switch(option){
00304
                                                                         case 1:{
00305
                                                                                          int N, ld, min_value, interval;
                                                                                          printf("What kind of distribution?\n1 - LID\n2 - LDD \n3 - LND\n"); scanf(" %d", &ld);
00306
00307
                                                                                          printf("What is the size of the array? ");
scanf(" %d", &N);
printf("Start of break: ");
00308
00309
00310
00311
00312
                                                                                           scanf(" %d", &min_value);
00313
                                                                                          printf("End of break: ");
00314
00315
00316
                                                                                           scanf(" %d", &interval);
                                                                                           test_D1(N, ld, min_value, interval);
00317
00318
00319
00320
                                                                           case 2:{
                                                                                          int M, N, ld, min_value, interval; printf("What kind of distribution?\n1 - LID\n2 - LDD \n3 - LND\n"); scanf(" %d", &ld);
00321
00322
00323
00324
                                                                                          printf("What is the size of the array? n");
```

```
printf("M: ");
scanf(" %d", &M);
printf("N: ");
scanf(" %d", &N);
printf("Start of break: ");
scanf(" %d", &min_value);
printf("End of break: ");
scanf(" %d", &interval);
test D2(M. N. ld. min value)
00326
00327
00328
00329
00330
00331
00332
                                             test_D2(M, N, ld, min_value, interval);
00333
00334
                                             break;
00335
                                    case 3:{
   int M, N, P, ld, min_value, interval;
   printf("What kind of distribution?\n1 - LID\n2 - LDD \n3 - LND\n");
00336
00337
00338
                                          printf("What kind of distribution?\n1 - LID'
scanf(" %d", &ld);
printf("What is the size of the array? \n");
printf("M: ");
scanf("%d", &M);
printf("N: ");
scanf("%d", &N);
printf("P: ");
scanf(" %d", &P);
printf("Start of break: ");
scanf("%d", &min_value);
printf("End of break: ");
scanf("%d", &interval);
test_D3(M, N, P, ld, min_value, interval);
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
                                             test_D3(M, N, P, ld, min_value, interval);
00352
                                             break;
00353
                                     default:
00354
00355
                                           break:
00356
00357
                     }while(option > 0 && option < 4);</pre>
00358
                     return 0;
00359 }
```

References test_D1(), test_D2(), and test_D3().

Here is the call graph for this function:



1.5.1.2 test_D1()

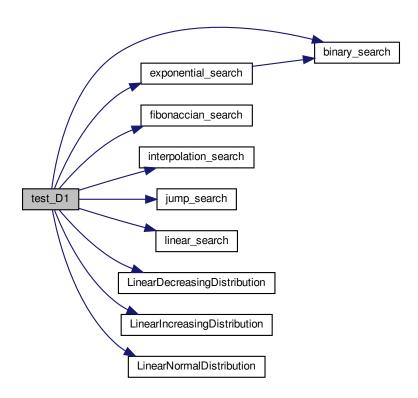
```
int min_value,
int interval )
```

```
Definition at line 13 of file Main.cpp.
                                                                {
00014
          vector<int> A(N);
00015
          CPUTimer timer;
00016
          if(ld == 1){
              printf("Linear Increasing Distribution\n");
00017
00018
              LinearIncreasingDistribution(A.begin(), A.end(), min_value, interval);
00019
00020
00021
             printf("Linear Decreasing Distribution\n");
              LinearDecreasingDistribution(A.begin(), A.end(), min_value, interval);
00022
00023
00024
          else if(ld == 3){
00025
             printf("Linear Normal Distribution\n");
00026
              LinearNormalDistribution(A.begin(), A.end(), min_value, interval);
00027
00028
          else
00029
              return:
00030
00031
          int query;
          printf("How many queries in the range: ");
scanf(" %d", &query);
while(query > 0){
00032
00033
00034
00035
              getchar();
00036
              bool found;
              int key;
printf("Search key value: ");
00037
00038
00039
              scanf(" %d", &key);
00040
              /* ----- */
00041
00042
             printf("---
00043
                                                                  ----\n\n");
00044
             printf("Key: %d\n", key);
00045
             printf("N = %d\n", N);
00046
00047
00048
00049
              timer.reset();
00050
              found = false;
00051
              timer.start();
00052
              found = linear_search(A.begin(), A.end(), key);
00053
              timer.stop();
              printf("Linear search: ");
00054
00055
              if (found == true) {
00056
                  printf("YES\n");
00057
00058
              else{
00059
                 printf("NO\n");
00060
00061
              printf("timer: %.10lf\n", timer.getCronoTotalSecs());
00062
00063
00064
00065
              timer.reset();
00066
              found = false:
00067
              timer.start();
00068
              found = jump_search(A.begin(), A.end(), key);
00069
              timer.stop();
00070
              printf("Jump search: ");
              if(found == true) {
    printf("YES\n");
00071
00072
00073
00074
              else{
00075
                  printf("NO\n");
00076
00077
              printf("timer: %.10lf\n", timer.getCronoTotalSecs());
00078
00079
00080
00081
              timer.reset();
00082
00083
              timer.start();
00084
              found = binary_search(A.begin(), A.end(), key);
00085
              timer.stop();
              printf("Binary search: ");
00086
              if(found == true){
00087
                 printf("YES\n");
00088
00089
00090
              else{
00091
                  printf("NO\n");
00092
00093
               printf("timer: %.10lf\n", timer.getCronoTotalSecs());
00094
```

```
00095
00096
00097
             timer.reset();
00098
             found = false;
00099
             timer.start();
00100
             found = interpolation_search(A.begin(), A.end(), key);
00101
             timer.stop();
00102
             printf("Interpolation search: ");
00103
             if(found == true){
00104
                 printf("YES\n");
00105
00106
             else{
                printf("NO\n");
00107
00108
00109
             printf("timer: %.101f\n", timer.getCronoTotalSecs());
00110
00111
00112
00113
             timer.reset();
00114
             found = false;
00115
             timer.start();
00116
             found = exponential_search(A.begin(), A.end(), key);
00117
             timer.stop();
             printf("Exponential search: ");
00118
00119
             if(found == true){
00120
                printf("YES\n");
00121
00122
                 printf("NO\n");
00123
00124
             printf("timer: %.101f\n", timer.getCronoTotalSecs());
00125
00126
00127
00128
00129
             timer.reset();
00130
             found = false:
00131
             timer.start();
00132
             found = fibonaccian_search(A.begin(), A.end(), key);
00133
             timer.stop();
00134
             printf("Fibonaccian search: ");
00135
             if(found == true){
                printf("YES\n");
00136
00137
00138
             else{
               printf("NO\n");
00139
00140
00141
             printf("timer: %.101f\n", timer.getCronoTotalSecs());
                                                                   ----\n\n");
00142
             printf("--
00143
             query--;
00144
00145
         printf("\n");
00146 }
```

References binary_search(), exponential_search(), fibonaccian_search(), interpolation_search(), jump_search(), linear_search(), LinearDecreasingDistribution(), LinearIncreasingDistribution(), and LinearNormalDistribution().

Here is the call graph for this function:



Here is the caller graph for this function:



1.5.1.3 test_D2()

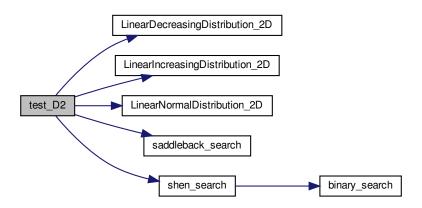
```
void test_D2 (
          int M,
          int ld,
          int min_value,
          int interval )
```

```
Definition at line 149 of file Main.cpp.
00150
00151
00152
00153
          vector<vector<int> > A(M, vector<int>(N));
00154
00155
          CPUTimer timer;
00156
00157
          if (1d == 1) {
              \label{eq:printf("Linear Increasing Distribution $$n");}
00158
00159
              LinearIncreasingDistribution_2D(A.begin(), A.end(), min_value, interval);
00160
00161
          else if ( ld == 2) {
00162
              printf("Linear Decreasing Distribution \n");\\
00163
              LinearDecreasingDistribution_2D(A.begin(), A.end(), min_value, interval);
00164
00165
          else if( ld==3){
             printf("Linear Normal Distribution\n");
00166
00167
              LinearNormalDistribution_2D(A.begin(), A.end(), min_value, interval);
00168
00169
          else
00170
              return;
00171
          int query;
         printf("How many queries in the range: ");
scanf(" %d", &query);
00172
00173
00174
          while (query>0) {
00175
            bool found;
             int key;
printf("Search key value: ");
00176
00177
              scanf(" %d", &key);
00178
00179
00180
00181
              printf("----
00182
                                                               -----\n\n");
              printf("Key: %d\n", key);
00183
              printf("M x N = %d x %d\n", M, N);
00184
00185
              timer.reset();
              found = false;
00186
00187
              timer.start();
00188
              found = saddleback_search(A.begin(), A.end(), key);
00189
              timer.stop();
              printf("Saddleback search: ");
00190
00191
00192
              if(found == true){
                 printf("YES\n");
00193
00194
00195
              else{
                  printf("NO\n");
00196
00197
00198
              printf("timer: %.10lf\n", timer.getCronoTotalSecs());
00199
00200
              timer.reset();
00201
              found = false;
00202
              timer.start();
00203
              found = shen_search(A.begin(), A.end(), key);
00204
              timer.stop();
00205
00206
              printf("Shen search: ");
00207
              if(found == true) {
                 printf("YES\n");
00208
00209
00210
              else{
00211
                 printf("NO\n");
00212
00213
              printf("timer: %.10lf\n", timer.getCronoTotalSecs());
              printf("--
                                                                       ----\n\n");
00214
00215
              query--;
00216
00217
         printf("\n");
00218 }
```

References LinearDecreasingDistribution_2D(), LinearIncreasingDistribution_2D(), LinearNormalDistribution_2D(), saddleback search(), and shen search().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



1.5.1.4 test_D3()

```
void test_D3 (
          int M,
          int P,
          int ld,
          int min_value,
          int interval )
```

Definition at line 220 of file Main.cpp.

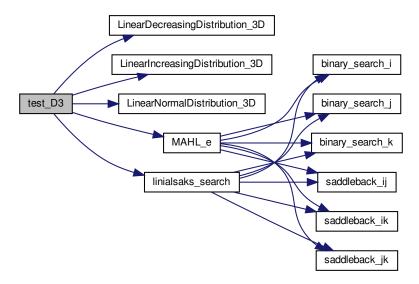
```
00220
00221
00222
00223
          vector<vector<int> > > A(M, vector<vector<int> > (N, vector<int>(P)));
00224
          CPUTimer timer;
00225
00226
00227
              printf("Linear Increasing Distribution\n");
00228
              LinearIncreasingDistribution_3D(A.begin(), A.end(), min_value, interval);
00229
00230
          else if(ld == 2){
              \label{eq:printf}  \mbox{printf("Linear Decreasing Distribution$\n");} 
00231
00232
               LinearDecreasingDistribution_3D(A.begin(), A.end(), min_value, interval);
```

```
00233
00234
          else if(ld == 3){
              printf("Linear Normal Distribution\n");
00235
              LinearNormalDistribution_3D(A.begin(), A.end(), min_value, interval);
00236
00237
00238
          else
00239
              return;
00240
00241
          int query;
         printf("How many queries in the range: ");
scanf("%d", &query);
while(query > 0){
00242
00243
00244
           bool found;
int key;
00245
00246
              printf("Search key value: ");
scanf(" %d", &key);
00247
00248
00249
00250
00251
              printf("----
00252
              printf("Key: %d\n", key);
printf("M x N x P = %d x %d x %d\n", M, N, P);
if( M == N && M == P){
00253
00254
00255
00256
              timer.reset();
found = false;
00257
00258
                  timer.start();
00259
                  found = linialsaks_search(A.begin(), A.end(), key);
00260
                 timer.stop();
00261
                 printf("LinialSaks search: ");
00262
00263
                  if (found == true) {
                     printf("YES\n");
00264
00265
00266
                  else{
                  printf("NO\n");
}
00267
00268
00269
                  printf("timer: %.101f\n", timer.getCronoTotalSecs());
00270
             }
00271
00272
              /* ----- */
00273
00274
              timer.reset();
00275
              found = false:
00276
              timer.start();
00277
              found = MAHL_e(A.begin(), A.end(), key);
00278
              timer.stop();
00279
             printf("Shen3D search: ");
00280
              if(found == true){
                 printf("YES\n");
00281
00282
00283
              else{
00284
                printf("NO\n");
00285
00286
              printf("timer: %.101f\n", timer.getCronoTotalSecs());
00287
00288
              printf("-
                                                                    ----\n\n");
00289
              query--;
00290
00291
          printf("\n");
00292 }
```

References LinearDecreasingDistribution_3D(), LinearIncreasingDistribution_3D(), LinearNormalDistribution_3D(), linialsaks_search(), and MAHL_e().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



1.6 Main.cpp

```
00001 #include <iostream>
00002 #include <vector>
00003 #include <algorithm>
00004
00005
00006 #include "SearchAlgorithms.hpp"
00007 #include "GeneratorInstance.hpp"
00008 #include "../headers/CPUTimer.hpp"
00009
00010 using namespace std;
00011
00012
00013 void test_D1(int N, int 1d, int min_value, int interval){
00014
              vector<int> A(N);
00015
              CPUTimer timer;
if(ld == 1){
00016
                    printf("Linear Increasing Distribution\n");
LinearIncreasingDistribution(A.begin(), A.end(), min_value, interval);
00017
00018
00019
              printf("Linear Decreasing Distribution\n");
   LinearDecreasingDistribution(A.begin(), A.end(), min_value, interval);
00020
00021
00022
00023
00024
              else if(ld == 3){
```

1.6 Main.cpp 29

```
00025
              printf("Linear Normal Distribution\n");
00026
              LinearNormalDistribution(A.begin(), A.end(), min_value, interval);
00027
00028
          else
00029
             return;
00030
00031
         int query;
         printf("How many queries in the range: ");
scanf(" %d", &query);
while(query > 0){
00032
00033
00034
00035
              getchar();
00036
              bool found;
             int key;
printf("Search key value: ");
00037
00038
              scanf(" %d", &key);
00039
00040
              /* ----- */
00041
00042
00043
            printf("-----
                                       -----\n\n");
00044
            printf("Key: %d\n", key);
00045
            printf("N = %d\n", N);
00046
00047
00048
00049
              timer.reset();
00050
              found = false;
              timer.start();
00051
00052
              found = linear_search(A.begin(), A.end(), key);
00053
              timer.stop();
              printf("Linear search: ");
00054
              if(found == true){
00055
00056
                 printf("YES\n");
00057
00058
              else{
00059
                 printf("NO\n");
00060
00061
              printf("timer: %.10lf\n", timer.getCronoTotalSecs());
00062
00063
00064
00065
              timer.reset();
00066
              found = false;
00067
              timer.start();
00068
              found = jump_search(A.begin(), A.end(), key);
00069
              timer.stop();
00070
              printf("Jump search: ");
00071
              if(found == true){
                 printf("YES\n");
00072
00073
00074
              else{
00075
                 printf("NO\n");
00076
00077
              printf("timer: %.101f\n", timer.getCronoTotalSecs());
00078
00079
              /* ----- */
00080
00081
              timer.reset();
00082
              found = false;
00083
              timer.start();
00084
              found = binary_search(A.begin(), A.end(), key);
00085
              timer.stop();
              printf("Binary search: ");
if(found == true) {
00086
00087
00088
                 printf("YES\n");
00089
00090
              else{
00091
                 printf("NO\n");
00092
              }
              printf("timer: %.10lf\n", timer.getCronoTotalSecs());
00093
00094
00095
00096
00097
              timer.reset();
00098
              found = false:
00099
              timer.start();
00100
              found = interpolation_search(A.begin(), A.end(), key);
00101
              timer.stop();
00102
              printf("Interpolation search: ");
00103
              if(found == true){
                 printf("YES\n");
00104
00105
00106
              else{
00107
                 printf("NO\n");
00108
00109
              printf("timer: %.10lf\n", timer.getCronoTotalSecs());
00110
00111
```

```
00113
              timer.reset();
00114
              found = false;
00115
             timer.start();
00116
              found = exponential_search(A.begin(), A.end(), key);
00117
             timer.stop();
             printf("Exponential search: ");
00118
00119
             if(found == true){
                printf("YES\n");
00120
00121
              else{
00122
                printf("NO\n");
00123
00124
00125
             printf("timer: %.10lf\n", timer.getCronoTotalSecs());
00126
00127
00128
00129
              timer.reset();
00130
              found = false;
00131
              timer.start();
00132
              found = fibonaccian_search(A.begin(), A.end(), key);
00133
              timer.stop();
             printf("Fibonaccian search: ");
00134
00135
              if (found == true) {
                printf("YES\n");
00136
00137
00138
             else{
               printf("NO\n");
00139
00140
00141
              printf("timer: %.10lf\n", timer.getCronoTotalSecs());
             printf("--
00142
                                                                         ----\n\n");
00143
             querv--;
00144
00145
         printf("\n");
00146 }
00147
00148
00149 void test_D2(int M, int N, int ld, int min_value, int interval){
00150
00151
00152
00153
         vector<vector<int> > A(M, vector<int>(N));
00154
00155
         CPUTimer timer;
00156
00157
          if (1d == 1) {
00158
             printf("Linear Increasing Distribution\n");
00159
              LinearIncreasingDistribution_2D(A.begin(), A.end(), min_value, interval);
00160
00161
         else if ( ld == 2) {
             printf("Linear Decreasing Distribution\n");
00162
00163
              LinearDecreasingDistribution_2D(A.begin(), A.end(), min_value, interval);
00164
00165
          else if( ld==3){
             printf("Linear Normal Distribution\n");
00166
00167
             LinearNormalDistribution_2D(A.begin(), A.end(), min_value, interval);
00168
00169
00170
             return;
00171
         int query;
         printf("How many queries in the range: ");
scanf(" %d", &query);
00172
00173
00174
         while (query>0) {
           bool found;
00175
             int key;
00176
             printf("Search key value: ");
scanf(" %d", &key);
00177
00178
00179
00180
00181
00182
             printf("-----
                                             -----\n\n");
              printf("Key: %d\n", key);
00183
              printf("M x N = %d x %d\n", M, N);
00184
00185
              timer.reset();
00186
              found = false;
00187
              timer.start();
00188
              found = saddleback_search(A.begin(), A.end(), key);
00189
              timer.stop();
             printf("Saddleback search: ");
00190
00191
00192
              if(found == true){
                printf("YES\n");
00193
00194
00195
              else{
                printf("NO\n");
00196
00197
00198
             printf("timer: %.10lf\n", timer.getCronoTotalSecs());
```

1.6 Main.cpp 31

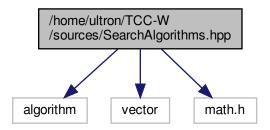
```
00199
              timer.reset();
00200
00201
              found = false;
00202
              timer.start();
00203
              found = shen_search(A.begin(), A.end(), key);
00204
              timer.stop();
00206
              printf("Shen search: ");
              if(found == true) {
    printf("YES\n");
00207
00208
00209
00210
              else{
                 printf("NO\n");
00211
00212
00213
              printf("timer: %.101f\n", timer.getCronoTotalSecs());
00214
              printf("--
                                                                          -----\n\n");
00215
              query--;
00216
00217
          printf("\n");
00218 }
00219
00220 void test_D3(int M, int N, int P, int ld, int min_value, int interval) {
00221
00222
00223
          vector<vector<iint> > > A(M, vector<vector<iint> > (N, vector<iint> (P)));
00224
          CPUTimer timer;
00225
00226
          if ( ld == 1) {
00227
              printf("Linear Increasing Distribution\n");
              \label{linear_IncreasingDistribution_3D(A.begin(), A.end(), min_value, interval);} \\
00228
00229
00230
          else if(ld == 2){
00231
             printf("Linear Decreasing Distribution\n");
00232
              LinearDecreasingDistribution_3D(A.begin(), A.end(), min_value, interval);
00233
          else if(ld == 3){
00234
             printf("Linear Normal Distribution\n");
00235
00236
              LinearNormalDistribution_3D(A.begin(), A.end(), min_value, interval);
00237
00238
          else
00239
              return;
00240
00241
          int query;
00242
          printf("How many queries in the range: ");
          scanf("%d", &query);
00243
00244
          while(query > 0){
            bool found;
00245
             int key;
printf("Search key value: ");
scanf(" %d", &key);
00246
00247
00248
00249
00250
00251
              printf("-----
                                       -----\n\n");
00252
              printf("Key: %d\n", key);
printf("M x N x P = %d x %d x %d\n", M, N, P);
00253
00254
00255
              if ( M == N && M == P) {
00256
                  timer.reset();
00257
                  found = false;
00258
                  timer.start();
                  found = linialsaks_search(A.begin(), A.end(), key);
00259
00260
                  timer.stop();
00261
00262
                  printf("LinialSaks search: ");
00263
                  if(found == true) {
                     printf("YES\n");
00264
00265
00266
                  elsef
00267
                     printf("NO\n");
00268
00269
                  printf("timer: %.101f\n", timer.getCronoTotalSecs());
00270
              }
00271
00272
00273
00274
              timer.reset();
00275
              found = false;
00276
              timer.start();
00277
              found = MAHL_e(A.begin(), A.end(), key);
00278
              timer.stop():
00279
              printf("Shen3D search: ");
00280
              if(found == true){
                 printf("YES\n");
00281
00282
00283
              else{
                  printf("NO\n");
00284
00285
              }
```

```
printf("timer: %.101f\n", timer.getCronoTotalSecs());
00288
                 printf("----\n\n");
00289
                query--;
00290
           printf("\n");
00291
00292 }
00293
00297 int main(){
00298
           int option;
00299
           do {
                 printf("Test search algorithms for:\n1 - one-dimensional\n2 - two-dimensional\n3 -
00300
        three-dimensional\nAnother value to leave:\n");
00301
00302
                 scanf(" %d", &option);
00303
                 switch(option){
00304
                     case 1:{
                         int N, ld, min_value, interval;
00305
                          printf("What kind of distribution?\n1 - LID\n2 - LDD \n3 - LND\n");
00306
00307
                          scanf(" %d", &ld);
                         printf("What is the size of the array? ");
scanf(" %d", &N);
printf("Start of break: ");
00308
00309
00310
00311
00312
                          scanf(" %d", &min_value);
00313
                          printf("End of break: ");
00314
00315
                          scanf(" %d", &interval);
00316
                          test_D1(N, ld, min_value, interval);
00317
00318
                          break:
00319
00320
00321
                          int M, N, ld, min_value, interval;
                         int M, N, Id, min_value, interval;
printf("What kind of distribution?\n1 - LID\n2 - LDD \n3 - LND\n");
scanf(" %d", &ld);
printf("What is the size of the array? \n");
printf("M: ");
scanf(" %d", &M);
printf("N: ");
scanf(" %d", &N);
printf("Start of break: ");
scanf(" %d", &min value);
00322
00323
00324
00325
00326
00327
00328
00329
                          scanf(" %d", &min_value);
00330
                         printf("End of break:
00331
00332
                          scanf(" %d", &interval);
00333
                          test_D2(M, N, ld, min_value, interval);
00334
                          break;
00335
00336
                     case 3:{
00337
                         int M, N, P, ld, min_value, interval;
                          printf("What kind of distribution?\n1 - LID\n2 - LDD \n3 - LND\n"); scanf(" %d", &ld);
00338
00339
00340
                          printf("What is the size of the array? n");
                         printf("M: ");
scanf("%d", &M);
printf("N: ");
00341
00342
00343
                         scanf("%d", &N);
printf("P: ");
scanf(" %d", &P);
00344
00345
00346
                          printf("Start of break: ");
00347
                         scanf("%d", &min_value);
printf("End of break: ");
00348
00349
00350
                          scanf("%d", &interval);
00351
                          test_D3(M, N, P, ld, min_value, interval);
00352
                          break;
00353
00354
                     default:
00355
                          break:
00356
            }while(option > 0 && option < 4);</pre>
00358
00359 }
```

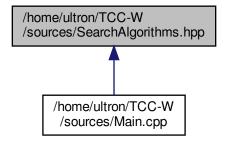
1.7 /home/ultron/TCC-W/sources/SearchAlgorithms.hpp File Reference

```
#include <algorithm>
#include <vector>
#include <math.h>
```

Include dependency graph for SearchAlgorithms.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- template < class ForwardIt, class T >
 bool linear_search (ForwardIt first, ForwardIt last, const T &value)
 One-dimensional search functions.
- template < class ForwardIt, class T >
 bool jump_search (ForwardIt first, ForwardIt last, const T &value)
 Linear search function.
- template < class ForwardIt , class T >
 bool interpolation_search (ForwardIt first, ForwardIt last, const T &value)
 Interpolation search function.
- template < class ForwardIt, class T >
 bool exponential_search (ForwardIt first, ForwardIt last, const T &value)
 Exponential search function.
- template < class ForwardIt, class T >
 bool fibonaccian_search (ForwardIt first, ForwardIt last, const T &value)
 Fiboncci search function.

```
• template<class ForwardIt , class T >
  bool saddleback search (ForwardIt first, int i1, int j1, int in, int jn, const T &value)
      Two-dimensional search functions.
• template < class ForwardIt , class T >
  bool saddleback_search (ForwardIt first, ForwardIt last, const T &value)
      Saddleback search function.

    template < class ForwardIt , class T >

  bool binary_search (ForwardIt first, int i1, int j1, int in, int jn, const T &value)
      Binary search function.
• template<class ForwardIt , class T >
  bool shen search (ForwardIt first, int i1, int j1, int in, int jn, const T &value)
      Shen search function.
• template<class ForwardIt , class T >
  bool shen search (ForwardIt first, ForwardIt last, const T &value)
      Shen search function.
• template < class ForwardIt , class T >
  bool saddleback_ij (ForwardIt first, int i1, int in, int j1, int jn, int k, const T &value)
      Saddleback search function for array ij face.

    template < class ForwardIt . class T >

  bool saddleback_ik (ForwardIt first, int i1, int in, int j, int k1, int kn, const T &value)
      Saddleback search function for array ik face.

    template < class ForwardIt , class T >

  bool saddleback jk (ForwardIt first, int i, int j1, int jn, int k1, int k2, const T &value)
      Saddleback search function for array ik face.
• template<class ForwardIt , class T >
  int binary search i (ForwardIt first, int i1, int in, int j, int k, const T &value)
      Binary search function in subvetor i.
• template < class ForwardIt , class T >
  int binary_search_j (ForwardIt first, int i, int j1, int jn, int k, const T &value)
      Binary search function in subvetor j.
• template<class ForwardIt , class T >
  int binary_search_k (ForwardIt first, int i, int j, int k1, int kn, const T &value)
      Binary search function in subvetor k.

    template < class ForwardIt , class T >

  bool linialsaks search (Forwardlt first, int i1, int j1, int k1, int in, int jn, int kn, const T &value)
      Linial and Saks search function.
• template<class ForwardIt , class T >
  bool linialsaks search (ForwardIt first, ForwardIt last, const T &value)
      Linial and Saks search function.
• template<class ForwardIt , class T >
  bool MAHL_e (ForwardIt first, int i1, int j1, int k1, int im, int jn, int kp, const T &value)
      MAHL e function.
• template<class ForwardIt , class T >
  bool MAHL_e (ForwardIt first, ForwardIt last, const T &value)
      MAHL e function.
```

1.7.1 Function Documentation

1.7.1.1 binary_search()

Binary search function.

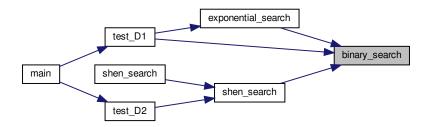
Parameters

first	iterator to start of array.
j1	leftmost j position of the array.
jn	rightmost j position of the array.
value	is the search key.

Definition at line 221 of file SearchAlgorithms.hpp.

```
00221
00222
            int lower, high;
           if((in-i1+1) < 4) {
    for( int i = i1; i <= in; i++) {</pre>
00223
00224
                     if(std::binary_search(first[i].begin(), first[i].end(), value)){
00225
00226
                          return true;
00227
00228
00229
           }else{
               for( int i = j1; i <= jn; i++) {
    lower = i1;
    high = in;</pre>
00230
00231
00232
00233
                     while ( lower <= high) {</pre>
00234
                         int mid = (lower+high) »1;
00235
                          if( value == first[mid][i])
00236
                              return true;
00237
                          else if( value < first[mid][i])</pre>
                              high = mid-1;
00238
00239
                          else
00240
                               lower = mid+1;
00241
                     }
00242
                }
00243
00244
            return false;
00245 }
```

Referenced by exponential_search(), shen_search(), and test_D1().



1.7.1.2 binary_search_i()

Binary search function in subvetor i.

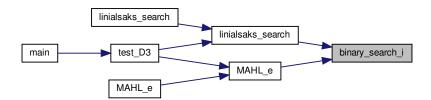
Parameters

first	iterator to start of array.
i1	leftmost i position of the array.
in	rightmost i position of the array.
j	array j position.
k	array k position.
value	is the search key.

Definition at line 409 of file SearchAlgorithms.hpp.

```
00410
               int lo, hi;
lo = i1;
hi = in;
00411
00412
               mi = in,
while(lo <= hi) {
   int mid = (lo+hi) »1;
   if( first[mid][j][k] < value)</pre>
00413
00414
00415
                      lo = mid+1;
else if( first[mid][j][k] > value)
hi = mid-1;
00416
00417
00418
00419
                      else
00420
                            return mid;
00421
00422
                return hi;
00423 }
```

Referenced by linialsaks_search(), and MAHL_e().



1.7.1.3 binary_search_j()

Binary search function in subvetor j.

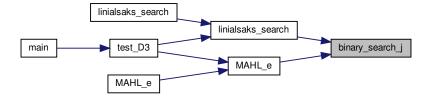
Parameters

first	iterator to start of array.
i	array i position.
j1	leftmost j position of the array.
jn	rightmost j position of the array.
k	array k position.
value	is the search key.

Definition at line 437 of file SearchAlgorithms.hpp.

```
00438
               int lo, hi;
              inc in;
lo = j1;
hi = jn;
while(lo <= hi){
   int mid = (lo+hi)*1;
   if( first[i][mid][k] < value)</pre>
00439
00440
00441
00442
00443
                    lo = mid+1;
else if( first[i][mid][k] > value)
hi = mid-1;
00444
00445
00446
00447
                    else
00448
                          return mid;
00449
00450
               return hi;
00451 }
```

Referenced by linialsaks_search(), and MAHL_e().



1.7.1.4 binary_search_k()

Binary search function in subvetor k.

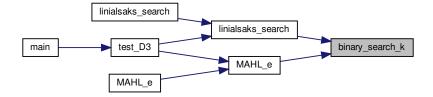
Parameters

i	array i position.
j	array j position.
k1	leftmost k position of the array.
kn	rightmost k position of the array.
value	is the search key.

Definition at line 465 of file SearchAlgorithms.hpp.

```
00465
                 int lo, hi;
lo = k1;
hi = kn;
00466
00468
                 while (lo <= hi) {
   int mid = (lo+hi) »1;
   if ( first[i][j][mid] < value)
       lo = mid+1;
   else if ( first[i][j][mid] > value)
00469
00470
00471
00472
00474
                             hi = mid-1;
00475
00476
                              return mid;
00477
00478
                 return hi;
00479 }
```

Referenced by linialsaks_search(), and MAHL_e().



1.7.1.5 exponential_search()

Exponential search function.

Parameters

first	iterator to start of array.
last	iterator to end of array.
value	is the search key.

Definition at line 115 of file SearchAlgorithms.hpp.

```
00116
       int n;
00117
       n = last - first;
       if(first[0] == value)
00118
       return true;
int i = 1;
00119
00120
       while(i < n && value > first[i]){
00122
         i *= 2;
00123
       00124
00125 }
```

References binary_search().

Referenced by test_D1().

Here is the call graph for this function:





1.7.1.6 fibonaccian_search()

Fiboncci search function.

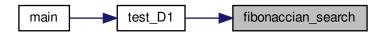
Parameters

first	iterator to start of array.
last	iterator to end of array.
value	is the search key.

Definition at line 135 of file SearchAlgorithms.hpp.

```
int f, f1, f2, n, offset, p;
n = last - first;
f2 = 0;
f1 = 1;
f = f1 + f2;
00136
00137
00138
00139
00140
00141
             while(f < n){</pre>
                f2 = f1;
f1 = f;
f = f1 + f2;
00142
00143
00144
00145
00146
             offset = -1;
            while(f > 1){
    p = (offset + f2) < n-1? offset+f2 : n-1;</pre>
00147
00148
00149
                  if(value > first[p]){
                  f = f1;
f1 = f2;
f2 = f - f1;
offset = p;
}else if(value < first[p]){</pre>
00150
00151
00152
00153
00154
                      f = f2;
f1 -= f2;
00155
00156
                       f2 = f - f1;
00157
00158
                  }else{
00159
                       return true;
00160
                  }
00161
             }
00162
             if(f > 0 && first[offset+1] == value){}
00163
00164
                  return true;
00165
00166
             return false;
00167 }
```

Referenced by test_D1().



1.7.1.7 interpolation_search()

Interpolation search function.

Parameters

first	iterator to start of array.
last	iterator to end of array.
value	is the search key.

Definition at line 88 of file SearchAlgorithms.hpp.

```
00089
              int i, n, j;
             i = 0;
n = last - first;
00090
00091
00092
              j = n-1;
             while(i <= j && first[i] != first[j] && value >= first[i] && value <= first[j]){
   int p = i + (((double)(j-i) / (first[j]-first[i]))*(value - first[i]));
   if( value == first[p]){</pre>
00093
00094
00095
00096
                        return true;
00097
00098
                   else if(value < first[p])</pre>
00099
                   j = p-1;
else
00100
00101
                        i = p+1;
00102
              if(i < n-1 && value == first[i])</pre>
00103
00104
                  return true;
              return false;
00105
00106 }
```

Referenced by test_D1().

Here is the caller graph for this function:



1.7.1.8 jump_search()

Linear search function.

Parameters

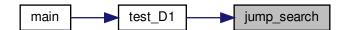
first	iterator to start of array.
last	iterator to end of array.
value	is the search key.

Definition at line 54 of file SearchAlgorithms.hpp.

```
00054
              int i, n, step, j;
i = 0;
n = last - first;
00055
00056
00057
00058
               step = std::sqrt(n);
              step = state;
j = step;
while(j < n){
    if( value == first[j]){
        return true;
}</pre>
00059
00060
00061
00062
00063
                     else if(value > first[j]){
    i = j;
    j += step;
}
00064
00065
00066
00067
                     }else
00068
                          break;
00069
               j = j<n? j : n-1;
while( i <= j) {</pre>
00070
00071
00072
00073
                    if(value == first[i])
    return true;
00074
                     else
00075
                           ++i;
00076
00077
               return false;
00078 }
```

Referenced by test_D1().

Here is the caller graph for this function:



1.7.1.9 linear_search()

One-dimensional search functions.

Function.

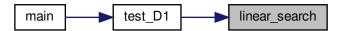
Parameters

first	iterator to start of array.
last	iterator to end of array.
value	is the search key.

Definition at line 39 of file SearchAlgorithms.hpp.

Referenced by test_D1().

Here is the caller graph for this function:



{

1.7.1.10 linialsaks_search() [1/2]

Linial and Saks search function.

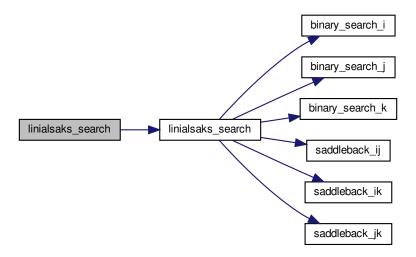
Parameters

first	iterator to start of array.
last	iterator to end of array.
value	is the search key.

Definition at line 582 of file SearchAlgorithms.hpp.

References linialsaks_search().

Here is the call graph for this function:



1.7.1.11 linialsaks_search() [2/2]

Linial and Saks search function.

Parameters

first	iterator to start of array.
i1	leftmost i position of the array.
in	rightmost i position of the array.
j1	leftmost j position of the array.
jn	rightmost j position of the array.
k1	leftmost k position of the array.
kn	rightmost k position of the array.
value	is the search key.

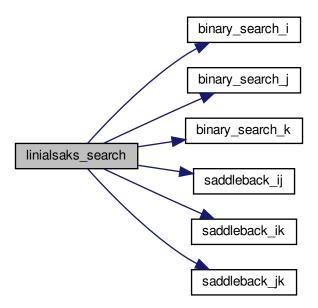
Definition at line 494 of file SearchAlgorithms.hpp.

```
00494
00495
             if(i1 > in || j1 > jn || k1 > kn)
00496
                 return false;
             if(i1 == in ||j1 == jn || k1 == kn) {
00497
                  if(i1 == in \&\& j1 == jn \&\& k1 == kn) {
00498
                            if(value == first[i1][j1][k1])
00499
00500
                            return true;
00501
                            else
00502
                            return false;
00503
00504
                  else(
00505
                            if(i1 == in) {
00506
                                      return saddleback_jk(first, i1, j1, jn, k1, kn, value);
00507
00508
                            else if (j1 == jn) {
                                       return saddleback ik(first, il, in, il, kl, kn, value);
00509
00510
00511
                            else
00512
                                      return saddleback_ij(first, i1, in, j1, jn, k1, value);
00513
00514
             }
00515
00516
            int u1, u2, w1, w2, v1, v2;
00517
00518
             /\star Binary search in subarray ul. Variable ul is the position returned in binary search. \star/
00519
            u1 = binary_search_k(first, i1, jn, k1, kn, value);
            if(u1>= 0 && first[i1][jn][u1] == value)
00520
00521
                   return true;
             /\star Binary search in subarray w1. Variable w1 is the position returned in binary search. \!\star/
00522
            w1 = binary_search_i(first, i1, in, j1, kn, value);
if(w1 >= 0 && first[w1][j1][kn] == value)
00523
00524
00525
00526
            /* Binary search in subarray u2. Variable u2 is the position returned in binary search.*/ u2 = binary_search_k(first, in, j1, k1, kn, value); if (u2 >= 0 && first[in][j1][u2] == value)
00527
00528
00529
                 return true;
00531
00532
             /\star Binary search in subarray w2. Variable w2 is the position returned in binary search. \star/
00533
            w2 = binary_search_i(first, i1, in, jn, k1, value);
            if (w2 >= 0 && first[w2][jn][k1] == value)
00534
00535
                 return true:
00536
            /* Binary search in subarray v1. Variable v1 is the position returned in binary search.*/v1 = binary_search_j(first, in, j1, jn, k1, value); if (v1 >= 0 && first[in][v1][k1] == value)
00537
00538
00539
                 return true;
00540
00541
            /* Binary search in subarray v2. Variable v2 is the position returned in binary search.*/ v2 = binary_search_j(first, i1, j1, jn, kn, value); if (v2 >= 0 && first[i1][v2][kn] == value)
00542
00543
00544
00545
                  return true;
00546
            /* saddleback v2 u1... (i1, v2+1, u1+1) to (i1, jn, kn) */ if(v2+1 <= jn && u1+1 <= kn && saddleback_jk(first, i1, v2+1, jn, u1+1, kn, value) == true)
00547
00548
00549
00550
            /* saddleback w2 v1... (w2+1, v1+1, k1) to (in, jn, k1) */ if (w2+1 <= in && v1+1<= jn && saddleback_ij(first, w2+1, in, v1+1, jn, k1, value) == true)
00551
00552
00553
                 return true:
00554
            /* saddleback u1 w2... (i1, jn, k1) to (w2, jn, u1) */ if (w2 >= 0 && u1 >= 0 && saddleback_ik(first, i1, w2, jn, k1, u1, value) == true)
00555
00556
00557
                  return true;
00558
            /* saddleback u2 w1 ... (w1+1, j1, u2+1) to (in, j1, kn) */ if (w1+1 <= in && u2+1 <= kn && saddleback_ik(first, w1+1, in, j1, u2+1, kn, value) == true)
00559
00560
00561
                 return true;
00562
00563
                                           (in, j1, k1) to (in, v1, u2) */
             /* saddleback u2 v1...
00564
             if(v1 >= 0 && u2 >= 0 && saddleback_jk(first, in, j1, v1, k1, u2, value) == true)
00565
                 return true;
00566
            /* saddleback w1 v2... (i1, j1, kn) to (w1, v2, kn) */ if (w1 >= 0 && v2 >= 0 && saddleback_ij(first, i1, w1, j1, v2, kn, value) == true)
00567
00568
00569
00570
             return linialsaks_search(first, il+1, jl+1, kl+1, in-1, jn-1, kn-1, value);
00571
00572 }
```

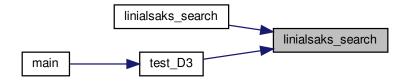
References binary_search_i(), binary_search_j(), binary_search_k(), saddleback_ij(), saddleback_ik(), and saddleback ik().

Referenced by linialsaks_search(), and test_D3().

Here is the call graph for this function:



Here is the caller graph for this function:



1.7.1.12 MAHL_e() [1/2]

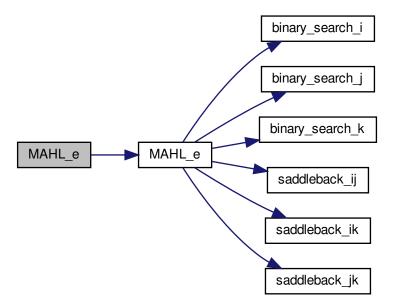
MAHL_e function.

Parameters

first	iterator to start of array.
last	iterator to end of array.
i1	leftmost i position of the array.
in	rightmost i position of the array.
j1	leftmost j position of the array.
jn	rightmost j position of the array.
k1	leftmost k position of the array.
kn	rightmost k position of the array.
value	is the search key.

Definition at line 698 of file SearchAlgorithms.hpp.

References MAHL_e().



1.7.1.13 MAHL_e() [2/2]

```
template<class ForwardIt , class T >
bool MAHL_e (
             ForwardIt first,
             int i1,
             int j1,
             int k1,
             int im,
             int jn,
             int kp,
             const T & value )
```

MAHL e function.

Parameters

first	iterator to start of array.
i1	leftmost i position of the array.
in	rightmost i position of the array.
j1	leftmost j position of the array.
jn	rightmost j position of the array.
k1	leftmost k position of the array.
kn	rightmost k position of the array.
value	is the search key.

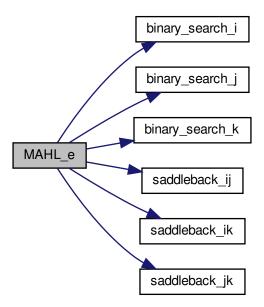
Definition at line 618 of file SearchAlgorithms.hpp.

```
00618
00619
            if(i1 > im || j1 > jn || k1 > kp)
00620
                return false;
            int diff_i = im - i1 + 1;
int diff_j = jn - j1 + 1;
int diff_k = kp - k1 + 1;
00621
00622
00623
00624
            /\star If dimension i is less than 3 and smaller than dimensions j and k, apply the saddleback
        algorithm to it. */
            if(diff_i <= 3 && diff_i <= diff_j && diff_i <= diff_k){</pre>
00625
                 for ( int i = i1; i <= im; ++i)
00626
00627
                     if(saddleback_jk(first, i, j1, jn, k1, kp, value))
00628
                           return true;
00629
                      return false:
00630
        /*If dimension j is less than 3 and smaller than dimensions i and k, apply the saddleback algorithm to it.*/
00631
00632
           if(diff_j <= 3 && diff_j <= diff_i && diff_j <= diff_k){</pre>
                 for( int j = j1; j <= jn; ++j)
    if(saddleback_ik(first, i1, im, j, k1, kp, value))</pre>
00633
00634
00635
                           return true;
                      return false;
00636
00637
            /\starIf dimension k is less than 3 and smaller than dimensions i and j, apply the saddleback
00638
        algorithm to it.*/
   if(diff_k <= 3 && diff_k <= diff_i && diff_k <= diff_j) {</pre>
00639
                 for( int k = k1; k <= kp; ++k)
    if(saddleback_ij(first, i1, im, j1, jn, k, value))</pre>
00640
00641
00642
                           return true;
00643
                      return false;
00644
00645
            /*If dimension i is larger, apply the algorithm to it.*/ if(diff_i >= diff_j && diff_i >= diff_k) {    int mid_j = (j1 + jn)   1; /* floor of N/2 */ int mid_k = (k1 + kp)   1; /* floor of P/2 */
00646
00647
00648
00649
00650
00651
                 int index_i = binary_search_i(first, i1, im, mid_j, mid_k, value);
00652
                 if( index_i >= 0 && first[index_i][mid_j][mid_k] == value)
00653
                      return true;
00654
00655
                 return MAHL_e(first, index_i+1, j1, k1, im, mid_j, kp, value) ||
00656
                 MAHL_e(first, i1, j1, mid_k, index_i, jn, kp, value) ||
```

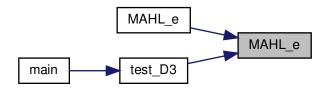
```
MAHL_e(first, i1, mid_j+1, k1, im, jn, mid_k-1, value);
00658
00659
               /*If dimension j is larger, apply the algorithm to it.*/
              else if(diff_j >= diff_i && diff_j >= diff_k) {
   int mid_i = (i1 + im) » 1; /* floor of M/2 */
   int mid_k = (k1 + kp) » 1; /* floor of P/2 */
00660
00661
00662
00663
00664
                    int index_j = binary_search_j(first, mid_i, j1, jn, mid_k, value);
00665
                    if(index_j >= 0 && first[mid_i][index_j][mid_k] == value)
00666
                           return true;
                    return MAHL_e(first, mid_i, j1, k1, im, index_j, kp, value) ||
MAHL_e(first, i1, j1, mid_k, mid_i-1, jn, kp, value) ||
MAHL_e(first, i1, index_j+1, k1, im, jn, mid_k-1, value);
00667
00668
00669
00670
00671
               /*If dimension k is larger, apply the algorithm to it.*/
00672
                    int mid_i = (i1 + im) » 1; /* floor of M/2 */
00673
00674
                    int mid_j = (j1 + jn) \gg 1; /* foor of N/2 */
00675
                    int index_k = binary_search_k(first, mid_i, mid_j, k1, kp, value);
if(index_k >= 0 && first[mid_i][mid_j][index_k] == value)
00676
00677
00678
                          return true;
                    return MAHL_e(first, mid_i, j1, k1, im, mid_j, kp, value) ||
MAHL_e(first, i1, j1, index_k+1, mid_i-1, jn, kp, value) ||
MAHL_e(first, i1, mid_j+1, k1, im, jn, index_k, value);
00679
00680
00681
00683 }
```

References binary_search_i(), binary_search_j(), binary_search_k(), saddleback_ij(), saddleback_ik(), and saddleback jk().

Referenced by MAHL_e(), and test_D3().



Here is the caller graph for this function:



1.7.1.14 saddleback_ij()

Saddleback search function for array ij face.

Parameters

first	iterator to start of array.	
i1	leftmost i position of the array.	
in	rightmost i position of the array.	
j1	leftmost j position of the array.	
jn	rightmost j position of the array.	
k	array k postition.	
value	is the search key.	

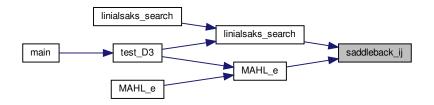
Definition at line 330 of file SearchAlgorithms.hpp.

```
00330
             int x, y; x = i1;
00331
00332
00333
             y - jn;
while (x <= in && y >= j1) {
    if (first[x][y][k] == value)
00334
00335
00336
                   return true;
if(first[x][y][k] > value)
00337
                   y--;
else
00338
00339
00340
00341
                         x++;
00342
              return false;
00343 }
```

Referenced by linialsaks_search(), and MAHL_e().

{

Here is the caller graph for this function:



1.7.1.15 saddleback_ik()

Saddleback search function for array ik face.

Parameters

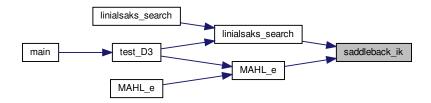
first	iterator to start of array.
i1	leftmost i position of the array.
in	rightmost i position of the array.
j	array j postition.
k1	leftmost k position of the array.
kn	rightmost k position of the array.
value	is the search key.

Definition at line 357 of file SearchAlgorithms.hpp.

```
00357
00358
           int x, z;
00359
          x = in;
z = k1;
while(x >= i1 && z <= kn){</pre>
00360
00361
00362
           if(first[x][j][z] == value)
00363
               if(first[x][j][z] > value)
00364
00365
               x--;
else
00366
00367
00368
00369
           return false;
00370 }
```

Referenced by linialsaks_search(), and MAHL_e().

Here is the caller graph for this function:



1.7.1.16 saddleback_jk()

Saddleback search function for array ik face.

Parameters

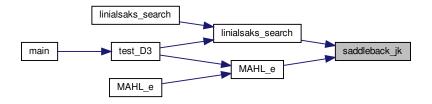
first	iterator to start of array.
i	array i position.
j1	leftmost j position of the array.
jn	rightmost j position of the array.
k1	leftmost k position of the array.
k2	rightmost k position of the array.
value	is the search key.

Definition at line 383 of file SearchAlgorithms.hpp.

```
00383
00384
           int y, z;
y = jn;
z = k1;
00385
00386
           while (y >= j1 && z <= k2) {</pre>
00387
00388
              if(first[i][y][z] == value)
00389
00390
               if(first[i][y][z] > value)
00391
               y--;
else
00392
00393
00394
00395
           return false;
00396 }
```

Referenced by linialsaks_search(), and MAHL_e().

Here is the caller graph for this function:



1.7.1.17 saddleback_search() [1/2]

Saddleback search function.

Parameters

first iterator to start of a		iterator to start of array.
	last	iterator to end of array.
	value	is the search key.

Definition at line 207 of file SearchAlgorithms.hpp.

```
00207

00208 int jn = first[0].size()-1;

00209 int in = last - first - 1;

00210 return saddleback_search(first, 0, 0, in, jn, value);

00211 }
```

References saddleback_search().



1.7.1.18 saddleback_search() [2/2]

Two-dimensional search functions.

Saddleback search function.

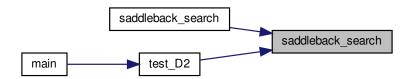
Parameters

first	iterator to start of array.	
j1	leftmost j position of the array.	
jn	rightmost j position of the array.	
value	is the search key.	

Definition at line 183 of file SearchAlgorithms.hpp.

```
00183
00184
             int i, j;
i = i1;
j = jn;
while(i <= in && j >= j1){
    if( first[i][j] == value){
        return true;
00185
00186
00187
00188
00189
00190
00191
                    if( first[i][j] > value)
00192
                          j--;
00193
00194
                          i++;
00195
00196
               return false;
00197 }
```

Referenced by saddleback_search(), and test_D2().



1.7.1.19 shen_search() [1/2]

Shen search function.

Parameters

first	iterator to start of array.
last	iterator to end of array.
value	is the search key.

Definition at line 288 of file SearchAlgorithms.hpp.

```
00288
00289         int j = (int)first[0].size()-1;
00290         int in = last - first - 1;
00291         return shen_search(first, 0, 0, in, j, value);
00292 }
```

References shen_search().

Here is the call graph for this function:



1.7.1.20 shen_search() [2/2]

Shen search function.

Parameters

first	iterator to start of array.	
j1	leftmost j position of the array.	
Geinerated byrightingst j position of the array		
value	is the search key.	

Definition at line 255 of file SearchAlgorithms.hpp.

```
if((in - i1+1) < 4 | | (jn-j1+1) < 4){
00256
00257
             return binary_search(first, i1, j1, in, jn, value);
00258
          int i = (i1+in) »1;
00259
00260
          if(value == first[i][j1])
00261
              return true;
00262
00263
              if( value < first[i][j1])</pre>
                      return shen_search(first, i1, j1, i-1, jn, value);
00264
00265
              elsef
00266
                      if( value > first[i][jn])
00267
                          return shen_search(first, i+1, j1, in, jn, value);
00268
                      else{
00269
                          int j;
                           j = lower_bound(first[i].begin() + j1, first[i].begin()+jn+1, value) -
00270
       first[i].begin();
00271
                          if( first[i][j] != value)
00272
                              return shen_search(first, i+1, j1, in, j, value) || shen_search(first, i1,
       j+1, i-1, jn, value);
00273
00274
                              return true;
00275
00276
                  }
00277
          }
00278 }
```

References binary_search().

Referenced by shen_search(), and test_D2().

Here is the call graph for this function:



Here is the caller graph for this function:



1.8 SearchAlgorithms.hpp

```
00001
00005 /*
00006 * SearchAlgorithms.h
00007 *
```

```
00008 * Created by Walisson Pereira de Jesus on 05/12/19.
00009 * Instituto de Informatica - UFG
00010 *
00011 */
00012
00023 #ifndef SearchAlgorithms_hpp
00024 #define SearchAlgorithms_hpp
00025
00026 #include <algorithm>
00027 #include <vector>
00028 #include <math.h>
00029
00031
00038 template<class ForwardIt, class T>
00039 bool linear_search(ForwardIt first, ForwardIt last, const T& value){
00040
       for(; first != last; ++first){
   if( *first == value)
00041
00042
                  return true;
00043
          }
00044
          return false;
00045 }
00046
00053 template<class ForwardIt, class T>
00054 bool jump_search(ForwardIt first, ForwardIt last, const T& value){
00055
         int i, n, step, j;
00056
           i = 0;
00057
          n = last - first;
00058
          step = std::sqrt(n);
          j = step;
while(j < n){</pre>
00059
00060
              if( value == first[j]) {
00061
00062
                   return true;
00063
00064
               else if(value > first[j]){
00065
                  i = j;
                   j += step;
00066
00067
               }else
00068
                  break;
00069
00070
           j = j<n? j : n-1;
           while ( i <= j) {
00071
             if (value == first[i])
00072
00073
                  return true;
00074
              else
00075
                  ++i;
00076
           }
00077
           return false;
00078 }
00079
08000
00087 template<class ForwardIt, class T>
00088 bool interpolation_search(ForwardIt first, ForwardIt last, const T& value){
00089
         int i, n, j;
00090
          i = 0;
          n = last - first;
00091
00092
          \dot{\eta} = n-1;
00093
          while(i <= j && first[i] != first[j] && value >= first[i] && value <= first[j]){
              int p = i + (((double)(j-i) / (first[j]-first[i]))*(value - first[i]));
if( value == first[p]){
00094
00095
00096
                   return true;
00097
00098
              else if(value < first[p])</pre>
00099
                  j = p-1;
00100
               else
00101
                   i = p+1;
00102
00103
          if(i < n-1 && value == first[i])</pre>
00104
              return true;
00105
           return false;
00106 }
00107
00114 template<class ForwardIt, class T>
00115 bool exponential_search(ForwardIt first, ForwardIt last, const T& value){
00116
         int n;
n = last - first;
00117
00118
          if(first[0] == value)
00119
              return true;
          int i = 1;
00120
00121
          while(i < n && value > first[i]){
             i *= 2;
00122
00123
00124
          return std::binary_search( first+(i/2+1), first + ((i < n)? i+1 : n), value);</pre>
00125 }
00126
00127
00134 template<class ForwardIt, class T>
00135 bool fibonaccian_search(ForwardIt first, ForwardIt last, const T& value){
```

```
int f, f1, f2, n, offset, p;
00137
           n = last - first;
          f2 = 0;
f1 = 1;
f = f1 + f2;
00138
00139
00140
00141
           while(f < n){</pre>
00142
             f2 = f1;
00143
               f1 = f;
00144
              f = f1 + f2;
00145
           offset = -1:
00146
          while(f > 1){
    p = (offset + f2) < n-1? offset+f2 : n-1;</pre>
00147
00148
00149
               if(value > first[p]){
                   f = f1;
f1 = f2;
f2 = f - f1;
00150
00151
00152
                   offset = p;
00153
               }else if(value < first[p]){</pre>
00154
00155
                   f = f2;
00156
                   f1 -= f2;
                   f2 = f - f1;
00157
               }else{
00158
00159
                   return true;
00160
               }
00161
          }
00162
00163
           if(f > 0 && first[offset+1] == value){
          . . o && firs return true;
00164
00165
00166
           return false:
00167 }
00168
00169
00171
00172
00173
00182 template<class ForwardIt, class T>
00183 bool saddleback_search(ForwardIt first, int i1, int j1, int in, int jn, const T& value){
          int i, j;
i = i1;
j = jn;
00184
00185
00186
          while( i <= in && j >= j1) {
00187
00188
              if( first[i][j] == value){
00189
                    return true;
00190
               if( first[i][j] > value)
00191
00192
                   j--;
00193
               else
00194
00195
00196
           return false;
00197 }
00198
00199
00206 template<class ForwardIt, class T>
00207 bool saddleback_search(ForwardIt first, ForwardIt last, const T& value){
        int jn = first[0].size()-1;
int in = last - first - 1;
00208
00209
         return saddleback_search(first, 0, 0, in, jn, value);
00210
00211 }
00212
00220 template<class ForwardIt, class T>
00221 bool binary_search(ForwardIt first, int i1, int j1, int in, int jn, const T& value){
00222
          int lower, high;
           if((in-i1+1) < 4) {
   for( int i = i1; i <= in ; i++) {</pre>
00223
00224
00225
                   if(std::binary_search(first[i].begin(), first[i].end(), value)){
00226
                        return true;
00227
00228
00229
          }else{
               for( int i = j1; i <= jn; i++) {</pre>
00230
                   lower = i1;
high = in;
00231
00232
00233
                    while ( lower <= high) {</pre>
00234
                      int mid = (lower+high) »1;
00235
                        if( value == first[mid][i])
00236
                            return true;
                        else if( value < first[mid][i])</pre>
00237
00238
                            high = mid-1;
00239
                        else
00240
                            lower = mid+1;
00241
                   }
00242
               }
00243
           }
```

```
00244
           return false;
00245 }
00246
00254 template<class ForwardIt, class T>
00255 bool shen_search(ForwardIt first, int i1,int j1, int in, int jn, const T& value){
00256    if((in - i1+1) < 4 || (jn-j1+1) < 4){
              return binary_search(first, i1, j1, in, jn, value);
00258
00259
           int i = (i1+in) \gg 1;
00260
           if(value == first[i][j1])
               return true;
00261
00262
           else{
00263
               if( value < first[i][j1])</pre>
00264
                        return shen_search(first, i1, j1, i-1, jn, value);
00265
               else{
00266
                        if( value > first[i][jn])
                            return shen_search(first, i+1, j1, in, jn, value);
00267
00268
                        else{
00269
00270
                            j = lower_bound(first[i].begin() + j1, first[i].begin()+jn+1, value) -
       first[i].begin();
00271
                            if( first[i][j] != value)
00272
                                return shen_search(first, i+1, j1, in, j, value) || shen_search(first, i1,
       j+1, i-1, jn, value);
00273
                            else
00274
                                return true;
00275
00276
                   }
00277
           }
00278 }
00279
00280
00287 template<class ForwardIt, class T>
00288 bool shen_search(ForwardIt first, ForwardIt last, const T& value){
               int j = (int)first[0].size()-1;
int in = last - first - 1;
00289
00290
00291
           return shen_search(first, 0, 0, in, j, value);
00292 }
00293
00294
00295 /\star Three-dimensional search functions \star/
00296
00297 /*
00298
                                 _v*2_
00299
00300
                                 / w1
                                         u1 I
00301 (0,0,0)
00302
                                        / (n, n, n)
                                    w2 /
00303
                             u2
00304
00305
00306 vector u1 = (i1 j2, )
00307 binary search in u1, w1, u2, w2, v1 end v2.
00308 saddleback face in the 6 faces:
00309 u1 and v2
00310 v1 and w2
00311 u1 and w2
00312 w1 and u2
00313 v1 and u2
00314 w1 and v2
00315
00316
00317
00318
00329 template<class ForwardIt, class T>
00330 bool saddleback_ij(ForwardIt first, int i1, int in, int j1, int jn, int k, const T& value){
00331
          int x, y;
00332
          x = i1:
          y = jn;
00333
           while (x <= in && y >= j1) {
00334
00335
              if(first[x][y][k] == value)
00336
                    return true:
               if(first[x][y][k] > value)
00337
00338
               y--;
else
00339
00340
                   x++;
00341
00342
           return false;
00343 }
00344
00345
00356 template<class ForwardIt, class T>
00357 bool saddleback_ik(ForwardIt first, int i1, int in, int j, int k1, int kn, const T& value){
00358
          int x, z;
          x = in;

z = k1;
00359
00360
           while ( x >= i1 && z <= kn) {
00361
```

```
if(first[x][j][z] == value)
00363
                                                    return true;
00364
                                       if(first[x][j][z] > value)
00365
                                                x--;
00366
                                       else
00367
                                                  z++;
00368
00369
                            return false;
00370 }
00371
00382 template<class ForwardIt, class T>
00383 bool saddleback_jk( ForwardIt first, int i, int j1, int jn, int k1, int k2, const T& value){
                           int y, z;
y = jn;
z = k1;
00384
00385
00386
00387
                           while (y \ge j1 \&\& z \le k2) {
                                       if(first[i][y][z] == value)
00388
00389
                                                  return true;
                                       if(first[i][y][z] > value)
00390
00391
                                                y--;
00392
                                       else
00393
                                                  z++;
00394
00395
                            return false;
00396 }
00397
00398
00408 template<class ForwardIt, class T>
00409 int binary_search_i(ForwardIt first, int i1, int in, int j, int k, const T& value) {
00410
                           int lo, hi;
                           lo = i1;
hi = in;
00411
00412
00413
                            while (lo <= hi) {</pre>
00414
                                      int mid = (lo+hi) \gg 1;
                                       if(first[mid][j][k] < value)
00415
00416
                                                 lo = mid+1;
                                      else if( first[mid][j][k] > value)
00417
00418
                                               hi = mid-1;
00419
                                       else
00420
                                               return mid;
00421
00422
                            return hi;
00423 }
00424
00425
00426
00436 template<class ForwardIt, class T>
00437 int binary_search_j(ForwardIt first, int i, int j1, int jn, int k, const T& value){
00438
                           int lo, hi:
                           lo = j1;
hi = jn;
00439
00440
00441
                            while(lo <= hi) {</pre>
00442
                                      int mid = (lo+hi) \gg 1;
                                       if( first[i][mid][k] < value)</pre>
00443
00444
                                                 lo = mid+1;
00445
                                       else if( first[i][mid][k] > value)
00446
                                               hi = mid-1;
00447
                                       else
00448
                                              return mid;
00449
                            }
00450
                            return hi:
00451 }
00452
00453
00454
00464 template<class ForwardIt, class T> \,
00465 \text{ int binary\_search\_k} \\ \text{(ForwardIt first, int i, int j, int k1, int kn, const T\& value)} \\ \text{(}\\ \text{(ForwardIt first, int i, int j, int k1, int kn, const T\& value)} \\ \text{(}\\ \text{(}\\ \text{(ForwardIt first, int i, int j, int k1, int kn, const T\& value)} \\ \text{(}\\ \text{(}\\ \text{(ForwardIt first, int i, int j, int k1, int kn, const T\& value)} \\ \text{(}\\ \text{(}\\ \text{(}\\ \text{(ForwardIt first, int i, int i, int k1, int kn, const T\& value)} \\ \text{(}\\ \text{(}\\
00466
                           int lo, hi:
00467
                            lo = k1;
                           hi = kn;
00468
00469
                            while(lo <= hi){</pre>
                                      int mid = (lo+hi) \gg 1;
00470
                                       if(first[i][j][mid] < value)
00471
00472
                                                 lo = mid+1;
                                       else if( first[i][j][mid] > value)
00473
00474
                                                hi = mid-1;
00475
                                       else
00476
                                               return mid;
00477
00478
                            return hi;
00479 }
00480
00493 template<class ForwardIt, class T>
00494 bool linialsaks_search(ForwardIt first, int i1, int j1, int k1, int in, int jn, int kn, const T&
                   value) {
00495
                           if(i1 > in || j1 > jn || k1 > kn)
    return false;
00496
```

```
00497
            if(i1 == in ||j1 == jn || k1 == kn){
                 if(i1 == in && j1 == jn && k1 == kn) {
    if(value == first[i1][j1][k1])
00498
00499
00500
                            return true;
00501
                            else
00502
                            return false;
00503
00504
                  else{
                            if(i1 == in) {
00505
00506
                                      return saddleback_jk(first, i1, j1, jn, k1, kn, value);
00507
00508
                            else if (j1 == jn) {
00509
                                      return saddleback_ik(first, i1, in, j1, k1, kn, value);
00510
00511
                            else
00512
                                      return saddleback_ij(first, i1, in, j1, jn, k1, value);
00513
                 }
00514
            }
00515
00516
            int u1, u2, w1, w2, v1, v2;
00517
00518
            /\star Binary search in subarray ul. Variable ul is the position returned in binary search. \star/
            u1 = binary_search_k(first, i1, jn, k1, kn, value);
if(u1>= 0 && first[i1][jn][u1] == value)
00519
00520
00521
                  return true;
            /* Binary search in subarray w1. Variable w1 is the position returned in binary search.*/w1 = binary_search_i(first, i1, in, j1, kn, value);
00522
00523
00524
            if(w1 >= 0 && first[w1][j1][kn] == value)
00525
                  return true;
00526
            /* Binary search in subarray u2. Variable u2 is the position returned in binary search.*/ u2 = binary_search_k(first, in, j1, k1, kn, value); if (u2 >= 0 && first[in][j1][u2] == value)
00527
00528
00529
00530
                  return true;
00531
            /* Binary search in subarray w2. Variable w2 is the position returned in binary search.*/ w2 = binary_search_i(first, i1, in, jn, k1, value); if (w2 >= 0 && first[w2][jn][k1] == value)
00532
00533
00535
                  return true;
00536
00537
            /\star \ {\tt Binary \ search \ in \ subarray \ v1. \ Variable \ v1 \ is \ the \ position \ returned \ in \ binary \ search.*/}
            v1 = binary_search_j(first, in, j1, jn, k1, value);
if(v1 >= 0 && first[in][v1][k1] == value)
00538
00539
00540
                  return true;
00541
00542
             /\star \ {\tt Binary \ search \ in \ subarray \ v2. \ Variable \ v2 \ is \ the \ position \ returned \ in \ binary \ search.} \star/
00543
            v2 = binary_search_j(first, i1, j1, jn, kn, value);
00544
            if(v2 >= 0 \&\& first[i1][v2][kn] == value)
00545
                  return true:
00546
            /* saddleback v2 u1... (i1, v2+1, u1+1) to (i1, jn, kn) */ if(v2+1 <= jn && u1+1 <= kn && saddleback_jk(first, i1, v2+1, jn, u1+1, kn, value) == true)
00547
00548
00549
                  return true;
00550
00551
            /\star saddleback w2 v1... (w2+1, v1+1, k1) to (in, jn, k1) \star/
00552
            if(w2+1 <= in && v1+1<= jn && saddleback_ij(first, w2+1, in, v1+1, jn, k1, value) == true)
00553
                 return true;
00554
            /* saddleback u1 w2... (i1, jn, k1) to (w2, jn, u1) */    if(w2 >= 0 && u1 >= 0 && saddleback_ik(first, i1, w2, jn, k1, u1, value) == true)
00555
00556
00557
                  return true:
00558
            /* saddleback u2 w1 ... (w1+1, j1, u2+1) to (in, j1, kn) */ if (w1+1 <= in && u2+1 <= kn && saddleback_ik(first, w1+1, in, j1, u2+1, kn, value) == true)
00559
00560
00561
                  return true;
00562
00563
            /* saddleback u2 v1... (in, j1, k1) to (in, v1, u2)*/    if(v1 >= 0 && u2 >= 0 && saddleback_jk(first, in, j1, v1, k1, u2, value) == true)
00564
00565
                 return true:
00566
00567
                                           (i1, j1, kn) to (w1, v2, kn) \star/
             /* saddleback w1 v2...
00568
            if(w1 \ge 0 \&\& v2 \ge 0 \&\& saddleback_ij(first, i1, w1, j1, v2, kn, value) == true)
00569
                 return true;
00570
00571
            return linialsaks_search(first, i1+1, j1+1, k1+1, in-1, jn-1, kn-1, value);
00572 }
00573
00574
00581 template<class ForwardIt, class T>
00582 bool linialsaks_search(ForwardIt first, ForwardIt last, const T& value){
           int in, jn, kn;
in = (last - first) -1;
00583
             jn = first[0].size() - 1;
00585
00586
            kn = first[0][0].size() - 1;
00587
            return linialsaks_search(first, 0, 0, 0, in, jn, kn, value);
00588 }
00589
```

```
00591 /*
00592
00593
        (N, *0, P)
00594
                                                              (M, N, P)
00595
        (M, 0, 0)
00597
00598
                                                             (0, N, P)
00599
00600
                                                 (0, N, 0)
00601
        (0.0.0)
00602
00603
00604
00605
00617 template<class ForwardIt, class T>
return false;
00620
00621
            int diff_i = im - i1 + 1;
            int diff_j = jn - j1 + 1;
00622
            int diff_k = kp - k1 + 1;
00623
            /*If dimension i is less than 3 and smaller than dimensions j and k, apply the saddleback
00624
        algorithm to it. */
00625
           if(diff_i <= 3 && diff_i <= diff_j && diff_i <= diff_k){</pre>
00626
                for ( int i = i1; i <= im; ++i)</pre>
00627
                    if(saddleback_jk(first, i, j1, jn, k1, kp, value))
00628
                         return true;
00629
                     return false:
00630
00631
            /*If dimension j is less than 3 and smaller than dimensions i and k, apply the saddleback
        algorithm to it.*/
00632
            for( int j = j1; j <= jn; ++j)
    if(saddleback_ik(first, i1, im, j, k1, kp, value))</pre>
00633
00634
00635
                         return true;
00636
                     return false;
00637
            /\star {	t If} dimension k is less than 3 and smaller than dimensions i and j, apply the saddleback
00638
        algorithm to it.*/
            if(diff_k <= 3 && diff_k <= diff_i && diff_k <= diff_j) {</pre>
00639
                for ( int k = k1; k \le kp; ++k)
00640
                     if(saddleback_ij(first, i1, im, j1, jn, k, value))
00641
00642
                         return true;
00643
                     return false;
00644
            }
00645
            /\!\star \text{If dimension i is larger, apply the algorithm to it.}\!\star/
00646
            if(diff_i >= diff_j && diff_i >= diff_k){
  int mid_j = (j1 + jn) » 1; /* floor of N/2 */
00647
00648
00649
                int mid_k = (k1 + kp) \gg 1; /* floor of P/2 */
00650
                int index_i = binary_search_i(first, i1, im, mid_j, mid_k, value);
if( index_i >= 0 && first[index_i][mid_j][mid_k] == value)
00651
00652
00653
                     return true;
00654
00655
                return MAHL_e(first, index_i+1, j1, k1, im, mid_j, kp, value) ||
                MAHL_e(first, i1, j1, mid_k, index_i, jn, kp, value) ||
MAHL_e(first, i1, mid_j+1, k1, im, jn, mid_k-1, value);
00656
00657
00658
           /*If dimension j is larger, apply the algorithm to it.*/
else if(diff_j >= diff_i && diff_j >= diff_k){
   int mid_i = (i1 + im) » 1; /* floor of M/2 */
00659
00660
00661
00662
                int mid_k = (k1 + kp) \gg 1; /* floor of P/2 */
00663
                int index_j = binary_search_j(first, mid_i, j1, jn, mid_k, value);
if(index_j >= 0 && first[mid_i][index_j][mid_k] == value)
00664
00665
00666
                     return true;
00667
                 return MAHL_e(first, mid_i, j1, k1, im, index_j, kp, value) ||
                MAHL_e(first, i1, j1, mid_k, mid_i-1, jn, kp, value) || MAHL_e(first, i1, index_j+1, k1, im, jn, mid_k-1, value);
00668
00669
00670
00671
            /*If dimension k is larger, apply the algorithm to it.*/
00672
            else{
00673
                int mid_i = (i1 + im) \gg 1; /* floor of M/2 */
00674
                int mid_j = (j1 + jn) \gg 1; /* foor of N/2 */
00675
00676
                int index_k = binary_search_k(first, mid_i, mid_j, k1, kp, value);
00677
                if(index_k >= 0 && first[mid_i][mid_j][index_k] == value)
00678
                    return true;
                return MAHL_e(first, mid_i, j1, k1, im, mid_j, kp, value) ||
MAHL_e(first, i1, j1, index_k+1, mid_i-1, jn, kp, value) ||
MAHL_e(first, i1, mid_j+1, k1, im, jn, index_k, value);
00679
00680
00681
00682
            }
00683 }
00684
```

```
00697 template<class ForwardIt, class T>
00698 bool MAHL_e(ForwardIt first, ForwardIt last, const T& value) {
00699     int im, jn, kp;
00700     im = (last - first) - 1;
00701     jn = first[0].size() - 1;
00702     kp = first[0][0].size() - 1;
00703     return MAHL_e(first, 0, 0, 0, im, jn, kp, value);
00704 }
00705
00706 #endif
```

Index

/home/ultron/TCC-W/sources/CPUTimer.cpp, 1	LinearIncreasingDistribution_3D
/home/ultron/TCC-W/sources/GeneratorInstance.hpp,	GeneratorInstance.hpp, 10
4, 15	LinearNormalDistribution
/home/ultron/TCC-W/sources/Main.cpp, 18, 28	GeneratorInstance.hpp, 11
/home/ultron/TCC-W/sources/SearchAlgorithms.hpp,	LinearNormalDistribution_2D
32, 56	GeneratorInstance.hpp, 12
	LinearNormalDistribution_3D
binary_search	GeneratorInstance.hpp, 13
SearchAlgorithms.hpp, 34	linialsaks_search
binary_search_i	SearchAlgorithms.hpp, 43, 44
SearchAlgorithms.hpp, 35	
binary_search_j	MAHL_e
SearchAlgorithms.hpp, 36	SearchAlgorithms.hpp, 46, 47
binary_search_k	main
SearchAlgorithms.hpp, 37	Main.cpp, 19
	Main.cpp
exponential_search	main, 19
SearchAlgorithms.hpp, 38	test_D1, 21
(learness)	test_D2, 24
fibonaccian_search	test_D3, 26
SearchAlgorithms.hpp, 39	
GeneratorInstance.hpp	saddleback_ij
LinearDecreasingDistribution, 5	SearchAlgorithms.hpp, 50
LinearDecreasingDistribution, 3 LinearDecreasingDistribution_2D, 6	saddleback_ik
LinearDecreasingDistribution_2D, 6 LinearDecreasingDistribution_3D, 7	SearchAlgorithms.hpp, 51
- —	saddleback_jk
LinearIncreasingDistribution, 8	SearchAlgorithms.hpp, 52
LinearIncreasingDistribution_2D, 9	saddleback_search
LinearIncreasingDistribution_3D, 10	SearchAlgorithms.hpp, 53
LinearNormalDistribution, 11	SearchAlgorithms.hpp
LinearNormalDistribution_2D, 12	binary_search, 34
LinearNormalDistribution_3D, 13	binary_search_i, 35
interpolation_search	binary_search_j, 36
SearchAlgorithms.hpp, 40	binary_search_k, 37
Geatch Agontims. hpp, 40	exponential_search, 38
jump_search	fibonaccian search, 39
SearchAlgorithms.hpp, 41	interpolation_search, 40
General agents and play to	jump_search, 41
linear_search	linear_search, 42
SearchAlgorithms.hpp, 42	linialsaks search, 43, 44
LinearDecreasingDistribution	MAHL_e, 46, 47
GeneratorInstance.hpp, 5	saddleback_ij, 50
LinearDecreasingDistribution_2D	saddleback_ik, 51
GeneratorInstance.hpp, 6	saddleback_jk, 52
LinearDecreasingDistribution_3D	saddleback_search, 53
GeneratorInstance.hpp, 7	shen search, 54, 55
LinearIncreasingDistribution	shen_search
GeneratorInstance.hpp, 8	SearchAlgorithms.hpp, 54, 55
LinearIncreasingDistribution_2D	Coardin agontimonipp, 04, 00
GeneratorInstance.hpp. 9	test D1

66 INDEX

Main.cpp, 21 test_D2 Main.cpp, 24 test_D3 Main.cpp, 26