discreture

1

Generated by Doxygen 1.8.5

Mon Mar 28 2016 13:56:15

Contents

Chapter 1

Discreture

This is a modern C++ 11 (and 14) library designed to facilitate combinatorial research by providing fast and easy iterators to a few combinatorial objects, such as combinations, permutations, partitions, and others. The idea is to have them resemble the STL containers as much as possible, without actually storing the whole set of objects in memory.

Discreture is designed to follow the STL containers as closely as possible, by providing the standard ways of iterating. In addition, many of the algorithms described in the standard <algorithm> library work as-is in these containers, as if the containers were constant.

Quick preview:

“‘c++ #include <iostream> #include "discreture.hpp" using namespace std; using namespace dscr; int main() { combinations X(5,3); for (const auto& x : X) cout << x << endl; return 0; } “‘ The above code would produce the following output:

```
[ 0 1 2 ]
[ 0 1 3 ]
[ 0 2 3 ]
[ 1 2 3 ]
[ 0 1 4 ]
[ 0 2 4 ]
[ 1 2 4 ]
[ 0 3 4 ]
[ 1 3 4 ]
[ 2 3 4 ]
```

Of course, you need to link with the discreture library: g++ -O3 -ldiscreture main.cpp

Some tests show discreture is usually faster when compiled with clang++ instead of g++. Full benchmarks at the end of the readme.

Installation

To download and install on linux, run the following commands:

“‘sh git checkout https://github.com/mraggi/discreture.git cd discreture sh install_linux.sh “‘

This will compile the library and copy the necessary files to /usr/lib and /usr/include. It will ask for your root password. If you just wish to compile and then link manually, do the following: “‘sh git checkout https://github.com/mraggi/discreture.git cd discreture mkdir build cd build cmake .. make “Furthermore, it is recommended to compile using the clang compiler instead of gcc. One can do this by runningcmake .. -DUSE_CLANG‘, or editing the CMakeLists.txt and switch the "OFF" option of USE_CLANG to "ON".

You can run the tests by running the executable: `./testdiscreture`

How to start using the library

To use the library, after compiling, just add `#include <discreture/discreture.hpp>` to your project and link to `libdiscreture.so`. With the GCC compiler or CLANG, this can be done by compiling like this: `g++ -ldiscreture myfile.cpp`

Combinatorial Objects

Within this library, one can construct a few combinatorial objects, such as:

- Combinations
- Permutations
- Subsets
- Multisets
- Partitions
- Dyck Paths
- Motzkin Paths
- Range
- Set Partitions

All follow the same design principle: The templated class is calles basic_SOMETHING<class T>, and the most reasonable type for T is instantiated as SOMETHING. For example, `combinations` is a typedef of `basic_combinations<int>`, and `partitions` is a typedef of `basic_partitions<int>`.

Advanced use

Although the full reference is in the doxygen documentation, here is a quick preview. Remember to always `#include "discreture.hpp"` (or `#include <discreture/discreture.hpp>` and use `using namespace dscr;` or add `dscr::` to everything.):

“‘c++ combinations X(30,10); for (const auto& x : X) { // x is of type const vector<int>&, so anything that works with vectors works on x } “‘

You can iterate in reverse too, in the same way you would reverse-iterate an STL container. “‘c++ combinations X(30,10); for (auto it = X.rbegin(); it != X.rend(); ++it) { const auto& x = *it; // x is of type const vector<int>&, so anything that works with vectors works on x } “‘

Combinations, subsets and permutations are a random-access container (although they are slower as such than forward or reverse iteration), so something like this works too: “‘c++ combinations X(30,10); for (size_t i = 0; i < X.size(); ++i) { auto x = X[i]; } “‘

This is much slower if one plans to actually iterate over all of them, but iterator arithmetic is implemented, so one could even do the following: “‘c++ #include <algorithm> // ... combinations X(30,10); std::partition_point(X.begin(), X.end(), predicate); “`where`predicate`is a unary predicate that takes a`const vector<int>&‘ as an argument and returns true or false, in a way that for all the first combinations it returns true and the last ones return false. This would do binary search.

Benchmarks.

On a i7-5820K CPU @ 3.30GHz, on Linux, compiling with -Ofast yields the following results:

Task	**Time taken CLANG 3.7.0**	**Time taken GCC 5.2.0**
Time taken to see all (32 choose 16) = 601080390 combinations	**2.29281s**	3.36332s
Time taken to see all (32 choose 16) = 601080390 combinations in reverse order	**1.67853s**	3.98176s
Time taken to see all 12! = 479001600 permutations	1.70865s	**1.33693s**
Time taken to see all 2^29 = 536870912 subsets	2.54663s	**2.14877s**
Time taken to see all 2^29 = 536870912 subsets (fast mode)	2.10764s	**1.84649s**
Time taken to see all 56634173 partitions of size 90	**1.41834s**	1.48321s
Time taken to see all 559872000 multisets	**1.84566s**	1.90435s
Time taken to see all 477638700 dyck paths of size 18	**2.16288s**	2.74891s
Time taken to see all 50852019 motzkin paths of size 20	**1.30359s**	1.46971s
Time taken to see all 27644437 set partitions of size 13	0.960195s	**0.79946s**
Time taken to see all 42355950 set partitions a set of 15 elements with 4 parts	1.20166s	**1.01687s**
Total Time	**19.7s**	22.1s

Acknowledgements

I would like to thank Manuel Alejandro Romo de Vivar (manolo) for his work on dyck and motzkin paths.

Chapter 2

Todo List

Member dscr::basic_combinations< IntType >::find_all (PartialPredicate pred)

Perhaps one should be able to iterate over all such permutations without constructing a vector of them!

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

Namespace Documentation

6.1 dscr Namespace Reference

Namespace under which all the discreture library resides.

Classes

- class basic_combinations

  class of all n choose k combinations of size k of the set {0,1,...,n-1}.
- class basic_dyck_paths

  Class for iterating through all dyck (dyck) paths.
- class basic_motzkin_paths

  Class for iterating through all motzkin paths.
- class basic_multisets
- class basic_partitions

  class of partitions of the number n.
- class basic_permutations

  class of all n! permutation of size n of the set {0,1,...,n-1}.
- class range

  Similar to python range(n) or range(n,m) or range(n,m,step).
- class basic_set_partitions

  class of set_partitions of the number n.
- class basic_subsets

  class of all 2^n subsets of the set {0,1,...,n-1}, expressed as incidence vectors
- class RClock

Typedefs

- typedef short int **sint**
- typedef long int **lint**
- typedef long long int **llint**
- typedef unsigned char **uchar**
- typedef short unsigned int **suint**
- typedef unsigned int **nuint**
- typedef long unsigned int **luint**
- typedef long long unsigned int **lluint**
- using **combinations** = basic_combinations< int >

- using **dyck_paths** = basic_dyck_paths< int >
- using **motzkin_paths** = basic_motzkin_paths< int >
- typedef basic_multisets< int > **multisets**
- using **partitions** = basic_partitions< int >
- typedef basic_permutations< int > **permutations**
- using **set_partitions** = basic_set_partitions< int >
- typedef basic_subsets< bool > **subsets**
- typedef basic_subsets
  < uint_fast8_t > **subsets_fast**
- typedef
  std::chrono::time_point
  < std::chrono::high_resolution_clock > **clockt**
- typedef vector< bool > **VB**
- typedef vector< char > **VC**
- typedef vector< sint > **VSI**
- typedef vector< int > **VI**
- typedef vector< lint > **VLI**
- typedef vector< nuint > **VUI**
- typedef vector< suint > **VSUI**
- typedef vector< size_t > **VLUI**
- typedef vector< uchar > **VUC**
- typedef vector< double > **VR**

Functions

- luint **factorial** (luint n)
- luint **binomial** (nuint n, nuint r)
- double linear_convert (double x, double a, double b, double u, double v)

  This function of x is just the linear function from [a,b] to [u,v].
- long abs (long a)

  For those who hate typing fabs, labs, llabs instead of abs.
- long long **abs** (long long a)
- template<class NumType >
  NumType **abs** (NumType a)
- template<class IntType >
  IntType modulo (IntType a, IntType b)

  This is what operator % should be but isn't (!).
- size_t twoD_to_oneD (nuint x, nuint y, nuint width, nuint height)

  Helper function to linearize tables.
- template<class T >
  T Clamped (T x, T a, T b)

  Clamps x to be in the interval [a,b].
- template<typename T >
  int signof (T val)

  Equivalent to x/$|x|$ when x != 0, and 0 when x = 0.
- std::default_random_engine & **random_engine** ()
- bool **probability_of_true** (double p)
- void **randomize** ()
- double **random_real** (double from, double upto)
- void **set_seed_with_time** ()
- template<class IntType >
  IntType **random_int** (IntType from, IntType thru)
- double **random_real** ()

- template<class IntType >
  range< IntType >::iterator **operator+** (typename range< IntType >::iterator it, long int n)
- template<class IntType >
  range< IntType >::iterator **operator-** (typename range< IntType >::iterator it, long int n)
- template<class T >
  void **overwrite** (vector< T > &lhs, range< T > rhs)
- lluint binomial (lluint n, lluint r)

  The number of subsets of size r chosen from a set of size n.
- lluint catalan (lluint n)

  The n-th catalan number.
- lluint motzkin (lluint n)

  The n-th motzkin number.
- lluint partition_number (lluint n)

  The n-th partition number.
- lluint partition_number (lluint n, lluint k)

  The number of partitions of n with k parts.
- lluint stirling1 (lluint n, lluint k)

  The number of permutations of n which have exactly k cycles.
- llint stirling2 (lluint n, lluint k)

  The number of partitions of a set of n elements with k parts.
- constexpr lluint factorial (lluint n)

  n!
- double **diffclock** (clock_t a, clock_t b)
- double **diffclockt** (clockt a, clockt b)
- double **TimeFromStart** ()
- double **Chronometer** ()
- double **ChronometerPeek** ()
- VB operator& (const VB &A, const VB &B)

  Bitwise and for vector<bool>
- VB operator| (const VB &A, const VB &B)

  Bitwise or for vector<bool>
- std::ostream & **operator<<** (std::ostream &os, const VUI &rhs)
- std::ostream & operator<< (std::ostream &os, const VUC &rhs)

  Specialization for vector printouts for vector<unsigned char>
- std::ostream & **operator<<** (std::ostream &os, const VSUI &rhs)
- std::ostream & operator<< (std::ostream &os, const VB &rhs)

  Specialization for vector printouts for vector<bool> so that it doesn't print out spaces.
- std::ostream & **operator<<** (std::ostream &os, const vector< VB > &rhs)
- template<class T , class U >
  vector< T > Convert (const vector< U > &G)

  Converts a vector<U> into a vector<T>, provided U can be converted to T.
- template<class numType >
  double Sum (const vector< numType > &vi)

  Finds the sum of all elements of vector. Returns a double because it's easier.
- template<class T >
  std::ostream & operator<< (std::ostream &os, const vector< T > &rhs)

  prints out a space separated vector.
- template<class T >
  std::ostream & operator<< (std::ostream &os, const std::list< T > &rhs)

  prints out a space separated list.
- template<class T >
  T min (const vector< T > &v)

Find the minimum value of a vector.

- template<class T >
  T max (const vector< T > &v)

  Find the max value of a vector.
- template<class T >
  size_t argmin (const vector< T > &v)

  Find the minimum index of a vector.
- template<class T >
  size_t argmax (const vector< T > &v)

  Find the maximum index of a vector.
- template<class T >
  vector< T > operator+ (const vector< T > &U, const vector< T > &V)

  vector coordinate-wise addition.
- template<class T >
  void operator+= (vector< T > &U, const vector< T > &V)

  inplace vector coordinate-wise addition.
- template<class T , class NumType >
  void operator/= (vector< T > &U, NumType t)

  inplace vector coordinate-wise division by a number.
- template<class T , class NumType >
  void operator*= (vector< T > &U, NumType t)

  inplace vector coordinate-wise multiplication by a number.
- template<class T , class NumType >
  vector< T > operator* (vector< T > U, NumType t)

  coordinate-wise multiplication by a number.
- template<class T , class NumType >
  vector< T > operator/ (vector< T > U, NumType t)

  coordinate-wise division by a number.
- template<class T >
  vector< T > mincac (const vector< T > &U, const vector< T > &V)

  returns a vector W such that for each coordinate i, W[i] = min(V[i],U[i])
- template<class T >
  vector< T > maxcac (const vector< T > &U, const vector< T > &V)

  returns a vector W such that for each coordinate i, W[i] = max(V[i],U[i])
- template<class T >
  bool operator<= (const vector< T > &A, const vector< T > &B)

  Lexicographic compare vector A and B.
- template<class T >
  bool operator== (const vector< T > &A, const vector< T > &B)

  Equality comparison of vectors.
- template<class T >
  VB CombinationToSubset (const vector< T > &C, size_t size)

  Given a subset S, written in combination form (1,2,4), returns the same subset written in subset form (01101)
- template<class vecT , class UIntType >
  vecT compose (const vecT &f, const vector< UIntType > &g)

  Function composition.
- template<class T >
  bool **AreTheyAllDifferent** (const vector< T > &G)

Variables

- constexpr double **pi** = 3.14159265358979323846264338327950
- constexpr double **e** = 2.718281828459045
- constexpr double **phi** = 1.618033988749895

6.1.1 Detailed Description

Namespace under which all the discreture library resides.

6.1.2 Function Documentation

6.1.2.1 lluint dscr::binomial (lluint *n*, lluint *r*)

The number of subsets of size r chosen from a set of size n.

Parameters

n	is a (small) nonnegative integer
r	is a small integer between 0 and n (inclusive)

Returns

n!/(r!∗(n-r)!)

6.1.2.2 lluint dscr::catalan (lluint *n*)

The n-th catalan number.

Parameters

n	is a (small) nonnegative integer

Returns

binomial(2n,n)/(n+1)

6.1.2.3 template<class vecT , class UIntType > vecT dscr::compose (const vecT & *f*, const vector< UIntType > & *g*)

Function composition.

Returns

f o g

6.1.2.4 constexpr lluint dscr::factorial (lluint *n*)

n!

Parameters

n	is a (small) nonnegative integer.

Returns

n!

6.1.2.5 double dscr::linear_convert (double *x*, double *a*, double *b*, double *u*, double *v*) [inline]

This function of x is just the linear function from [a,b] to [u,v].

Returns

f(x), where f:[a,b]->[u,v] is the only linear, monotone, biyective function.

6.1.2.6 template<class IntType > IntType dscr::modulo (IntType *a*, IntType *b*) [inline]

This is what operator % should be but isn't (!).

C++ modulo operator % is dumb for negative integers: (-7)%3 returns -1, instead of 2. This fixes it.

Returns

an integer in [0,b)

6.1.2.7 lluint dscr::motzkin (lluint *n*)

The n-th motzkin number.

Parameters

n	is a (small) nonnegative integer

Returns

M_n

6.1.2.8 template<class T > bool dscr::operator<= (const vector< T > & *A*, const vector< T > & *B*)

Lexicographic compare vector A and B.

Returns

A <= B in lexicographic order.

6.1.2.9 template<class T > bool dscr::operator== (const vector< T > & *A*, const vector< T > & *B*)

Equality comparison of vectors.

Returns

A <= B in lexicographic order.

6.1.2.10 lluint dscr::partition_number (lluint *n*)

The n-th partition number.

Parameters

n	is a (small) nonnegative integer

Returns

P_n

6.1.2.11 lluint dscr::partition_number (lluint *n*, lluint *k*)

The number of partitions of n with k parts.

Parameters

n	is a (small) nonnegative integer
k	<= n is a (small) nonnegative integer

Returns

P_{n,k}

6.1.2.12 template< typename T > int dscr::signof (T *val*)

Equivalent to x/|x| when x != 0, and 0 when x = 0.

Returns

1 if val is positive, -1 if it's negative, and 0 if it's 0

6.1.2.13 lluint dscr::stirling1 (lluint *n*, lluint *k*)

The number of permutations of n which have exactly k cycles.

Parameters

n	is a (small) nonnegative integer
k	<= n is a (small) nonnegative integer

Returns

The stirling number of the first kind S(n,k)

6.1.2.14 llint dscr::stirling2 (lluint *n*, lluint *k*)

The number of partitions of a set of n elements with k parts.

Parameters

n	is a (small) nonnegative integer
k	<= n is a (small) nonnegative integer

Returns

P_{n,k}

Chapter 7

Class Documentation

7.1 dscr::basic_combinations< IntType > Class Template Reference

class of all n choose k combinations of size k of the set {0,1,...,n-1}.

```
#include <Combinations.hpp>
```

Classes

- class iterator

  Random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.
- class reverse_iterator

  Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

Public Types

- typedef long long int **difference_type**
- typedef unsigned long long int **size_type**
- typedef vector< IntType > **value_type**
- typedef vector< IntType > **combination**

Public Member Functions

- basic_combinations (IntType n, IntType k)

  Constructor.
- size_type size () const

  The total number of combinations.
- size_type get_index (const combination &comb) const

  Returns the ID of the iterator whose value is comb. That is, the index of combination comb in the lexicographic order.
- IntType **get_n** () const
- IntType **get_k** () const
- iterator **get_iterator** (const combination &comb)
- const iterator & **begin** () const
- const iterator & **end** () const
- const reverse_iterator & **rbegin** () const
- const reverse_iterator & **rend** () const
- combination operator[] (size_type m) const

  Access to the m-th combination (slow for iteration)

- template<class PartialPredicate >
  iterator find_if (PartialPredicate pred)

  This is an efficient way to construct a combination of size k which fully satisfies a predicate.
- template<class PartialPredicate >
  vector< combination > find_all (PartialPredicate pred)

  This is an efficient way to construct all combination of size k which fully satisfy a predicate.

Static Public Member Functions

- static IntType **next_combination** (combination &data, IntType hint=0)
- static void **prev_combination** (combination &data)
- static void **construct_combination** (combination &data, size_type m)
- static bool compare (const combination &lhs, const combination &rhs)

  Combination comparison "less than" operator. Assumes lhs and rhs have the same size.

7.1.1 Detailed Description

template<class IntType>class dscr::basic_combinations< IntType >

class of all n choose k combinations of size k of the set {0,1,...,n-1}.

Parameters

IntType	should be an integral type with enough space to store n and k. It can be signed or unsigned.
n	the size of the set
k	the size of the combination (subset). Should be an integer such that n choose k is not bigger than the largest unsigned long int there is. For example, typically 50 choose 25 is already larger than the largest long unsigned int. **Example:**

```
combinations X(6,3);
for (const auto& x : X)
    cout << x << " ";
```

Prints out:

```
[ 0 1 2 ] [ 0 1 3 ] [ 0 2 3 ] [ 1 2 3 ] [ 0 1 4 ] [ 0 2 4 ] [ 1 2 4 ] [ 0 3 4 ] [ 1 3 4 ] [ 2 3 4 ] [ 0 1 5 ]
```

Example 2:

```
basic_combinations<short int> X(5,1);
for (const auto& x : X)
    cout << x << " ";
Prints out:
    [0] [1] [2] [3] [4]
```

Example 3:

```
string A = "helloworld";
combinations X(A.size(),2);
for (const auto& x : X)
{
    auto b = compose(A,x);
    cout << b << "-";
}
```

```
Prints out:
    he-hl-el-hl-el-ll-ho-eo-lo-lo-hw-ew-lw-lw-ow-ho-eo-lo-lo-oo-wo-hr-er-lr-lr-or-wr-or-hl-el-ll-ll-ol-wl-ol-r
```

7.1.2 Constructor & Destructor Documentation

7.1.2.1 template<class IntType > **dscr::basic_combinations**< IntType >::**basic_combinations** (IntType *n*, IntType *k*) [inline]

Constructor.

Parameters

n	is an integer >= 0
k	is an integer with 0 <= k <= n

7.1.3 Member Function Documentation

7.1.3.1 template<class IntType > static bool **dscr::basic_combinations**< IntType >::compare (const combination & *lhs*, const combination & *rhs*) [inline], [static]

Combination comparison "less than" operator. Assumes lhs and rhs have the same size.

Returns

true if lhs would appear before rhs in the normal iteration order, false otherwise

7.1.3.2 template<class IntType > template<class PartialPredicate > vector<combination> **dscr::basic_combinations**< IntType >::find_all (PartialPredicate *pred*) [inline]

This is an efficient way to construct all combination of size k which fully satisfy a predicate.

This function is similar to find_if, but it returns a vector with all combinations which satisfy pred,

Example:

```
combinations X(10,5);
auto vall = X.find_all([](const vector<int>& comb) -> bool
{
    for (int i = 0; i < comb.size()-1; ++i)
    {
        if (comb[i]+1 == comb[i+1])
            return false;
    }
    return true;
});
for (const auto& v : vall)
    cout << v << endl;
```

Prints out: [0 2 4 6 8] [0 2 4 6 9] [0 2 4 7 9] [0 2 5 7 9] [0 3 5 7 9] [1 3 5 7 9] which are all combinations which don’t contain two consecutive elements

Parameters

Pred	should be what we call a *partial predicate*: It takes a combination as a parameter and returns either true or false.

Returns

An vector<combination> filled will all permutations which fully satisfy the predicate.

Todo Perhaps one should be able to iterate over all such permutations without constructing a vector of them!

7.1.3.3 template<class IntType > template<class PartialPredicate > iterator dscr::basic_combinations< IntType >::find_if (PartialPredicate *pred*) [inline]

This is an efficient way to construct a combination of size k which fully satisfies a predicate.

This function is conceptually equivalent to std::find_if(begin(), end(), Pred), but much faster if the predicate can be evaluated on a partial combination (so as to prune the search tree)

Example:

```
combinations X(40,6);
auto it = X.find_if([](const vector<int>& comb) -> bool
{
    for (int i = 0; i < comb.size()-1; ++i)
    {
        if (2*comb[i] + 1 > comb[i+1])
            return false;
    }
    return true;
});
cout << *it << endl;
```

Prints out: [0 1 3 7 15 31]

Parameters

Pred	should be what we call a *partial predicate*: It takes a combination as a parameter and returns either true or false.

Returns

An interator to a combination which fully satisfies the predicate.

7.1.3.4 template<class IntType > size_type dscr::basic_combinations< IntType >::get_index (const combination & *comb*) const [inline]

Returns the ID of the iterator whose value is comb. That is, the index of combination comb in the lexicographic order.

Inverse of operator[]. If combination x is the m-th combination, then get_index(x) is m. If one has a combinations::iterator, then the member function ID() should return the same value.

Returns

the index of combination comb, as if basic_combinations was a proper data structure

Note

This constructs the proper index from scratch. If an iterator is already known, calling ID on the iterator is much more efficient.

7.1.3.5 template<class IntType > combination dscr::basic_combinations< IntType >::operator[] (size_type *m*) const [inline]

Access to the m-th combination (slow for iteration)

This is equivalent to calling ∗(begin()+m)

Parameters

m	should be an integer between 0 and size(). Undefined behavior otherwise.

Returns

The m-th combination, as defined in the order of iteration (lexicographic)

7.1.3.6 template<class IntType > size_type dscr::basic_combinations< IntType >::size ( ) const [inline]

The total number of combinations.

Returns

binomial(n,r)

The documentation for this class was generated from the following file:

- Combinations.hpp

7.2 dscr::basic_dyck_paths< IntType > Class Template Reference

Class for iterating through all dyck (dyck) paths.

```
#include <DyckPaths.hpp>
```

Classes

- class iterator

  Forward iterator class.
- class reverse_iterator

  Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

Public Types

- typedef long long int **difference_type**
- typedef unsigned long long int **size_type**
- typedef vector< IntType > **value_type**
- typedef vector< IntType > **dyck_path**

Public Member Functions

- basic_dyck_paths (IntType n)

  Constructor.
- size_type size () const

  The total number of dyck_paths.
- IntType **get_n** () const
- const iterator & **begin** () const
- const iterator & **end** () const
- const reverse_iterator & **rbegin** () const
- const reverse_iterator & **rend** () const

Static Public Member Functions

- static void **next_dyck_path** (dyck_path &data)
- static void **prev_dyck_path** (dyck_path &data, IntType n)
- static std::string **to_string** (const dyck_path &data, const string &delim="()")

7.2.1 Detailed Description

template<class IntType>class dscr::basic_dyck_paths< IntType >

Class for iterating through all dyck (dyck) paths.

Parameters

IntType	must be a SIGNED integer type.

Dyck paths, also called Catalan Paths, are paths that go from $(0,0)$ to $(0,2n)$, which never go below the $y=0$ line, in which each step is from (x,y) to either $(x+1,y+1)$ or $(x+1,y-1)$ #Example Usage:

```
dyck_paths X(3)
for (const auto& x : X)
    cout << x << endl;
```

Prints out: [1 1 1 -1 -1 -1] [1 1 -1 1 -1 -1] [1 -1 1 1 -1 -1] [1 1 -1 -1 1 -1] [1 -1 1 -1 1 -1]

Example: Parenthesis

```
dyck_paths X(3)
for (const auto& x : X)
    cout << dyck_paths::to_string(x, "()") << endl;
```

Prints out: ((())) (()()) ()(()) (())() ()()()

7.2.2 Constructor & Destructor Documentation

7.2.2.1 template< class IntType > dscr::basic_dyck_paths< IntType >::basic_dyck_paths (IntType *n*) [inline]

Constructor.

Parameters

n	is an integer >= 0

7.2.3 Member Function Documentation

7.2.3.1 template< class IntType > size_type dscr::basic_dyck_paths< IntType >::size ( ) const [inline]

The total number of dyck_paths.

Returns

binomial(2n,n)/(n+1)

The documentation for this class was generated from the following file:

- DyckPaths.hpp

7.3 dscr::basic_motzkin_paths< IntType > Class Template Reference

Class for iterating through all motzkin paths.

`#include <Motzkin.hpp>`

Classes

- class iterator

  Forward iterator class.

Public Types

- typedef long long int **difference_type**
- typedef unsigned long long int **size_type**
- typedef vector< IntType > **value_type**
- typedef vector< IntType > **motzkin_path**
- typedef basic_combinations < IntType >::iterator **comb_i**
- typedef basic_dyck_paths < IntType >::iterator **dyck_i**

Public Member Functions

- basic_motzkin_paths (IntType n)

  Constructor.
- size_type size () const

  The total number of motzkin_paths.
- IntType **get_n** () const
- const iterator & **begin** () const
- const iterator & **end** () const

Static Public Member Functions

- static std::string **to_string** (const motzkin_path &data, const string &delim="(-)")

7.3.1 Detailed Description

template<class IntType>class dscr::basic_motzkin_paths< IntType >

Class for iterating through all motzkin paths.

Parameters

IntType	must be a SIGNED integer type.

Motzkin paths are paths that go from $(0,0)$ to $(0,2n)$, which never go below the $y=0$ line, in which each step is from (x,y) to either $(x+1,y+1)$ or $(x+1,y-1)$ or $(x+1,y)$ #Example Usage:

```
motzkin_paths X(4)
for (const auto& x : X)
    cout << x << endl;
```

Prints out: [0 0 0 0] [1 -1 0 0] [1 0 -1 0] [0 1 -1 0] [1 0 0 -1] [0 1 0 -1] [0 0 1 -1] [1 1 -1 -1] [1 -1 1 -1]

Example: Parenthesis

```
motzkin_paths X(4)
for (const auto& x : X)
    cout << motzkin_paths::to_string(x, "(-)") << endl;
```

Prints out:

()– (-)- -()- (–) -(-) –() (()) ()()

7.3.2 Constructor & Destructor Documentation

7.3.2.1 template<class IntType > **dscr::basic_motzkin_paths**< IntType >::**basic_motzkin_paths** (IntType *n*) [inline]

Constructor.

Parameters

n	is an integer >= 0

7.3.3 Member Function Documentation

7.3.3.1 template<class IntType > size_type **dscr::basic_motzkin_paths**< IntType >::size ( ) const [inline]

The total number of motzkin_paths.

Returns

M_n

The documentation for this class was generated from the following file:

- Motzkin.hpp

7.4 dscr::basic_multisets< IntType > Class Template Reference

Classes

- class iterator

Public Types

- typedef long long int **difference_type**
- typedef unsigned long long int **size_type**
- typedef vector< IntType > **value_type**
- typedef vector< IntType > **multiset**

Public Member Functions

- basic_multisets (const vector< IntType > &set)

  class of all submultiset of a given set, expressed as incidence vectors with multiplicities
- **basic_multisets** (IntType size, IntType n=1)

- size_type **size** () const
- const iterator & **begin** () const
- const iterator & **end** () const

7.4.1 Constructor & Destructor Documentation

7.4.1.1 template<class IntType > dscr::basic_multisets< IntType >::basic_multisets (const vector< IntType > & *set*) [inline]

class of all submultiset of a given set, expressed as incidence vectors with multiplicities

Parameters

IntType	can be an int, uint, etc. It can be signed or unsigned (the negatives are not used) **Example:**

```
multisets X({1,0,3,1});
for (const auto& x : X)
    cout << x << " ";
```

Prints out:

```
[ 0 0 0 0 ]
[ 1 0 0 0 ]
[ 0 0 1 0 ]
[ 1 0 1 0 ]
[ 0 0 2 0 ]
[ 1 0 2 0 ]
[ 0 0 3 0 ]
[ 1 0 3 0 ]
[ 0 0 0 1 ]
[ 1 0 0 1 ]
[ 0 0 1 1 ]
[ 1 0 1 1 ]
[ 0 0 2 1 ]
[ 1 0 2 1 ]
[ 0 0 3 1 ]
[ 1 0 3 1 ]
```

TODO: Make it a random-access class and more like the others. It's not hard.

The documentation for this class was generated from the following file:

- Multisets.hpp

7.5 dscr::basic_partitions< IntType > Class Template Reference

class of partitions of the number n.

`#include <Partitions.hpp>`

Classes

- class iterator

  Forward iterator class.

Public Types

- typedef long long int **difference_type**
- typedef unsigned long long int **size_type**
- typedef vector< IntType > **value_type**
- typedef vector< IntType > **partition**

Public Member Functions

- basic_partitions (IntType n)

  Constructor.
- basic_partitions (IntType n, IntType numparts)

  Constructor.
- basic_partitions (IntType n, IntType minnumparts, IntType maxnumparts)

  Constructor.
- size_type size () const

  The total number of partitions.
- IntType **get_n** () const
- const iterator & **begin** () const
- const iterator & **end** () const

Static Public Member Functions

- static void **next_partition** (partition &data, IntType n)
- static void **first_with_given_number_of_parts** (partition &data, IntType n, IntType k)
- static partition **conjugate** (const partition &P)

7.5.1 Detailed Description

template< class IntType > class dscr::basic_partitions< IntType >

class of partitions of the number n.

Parameters

IntType	should be an integral type with enough space to store n and k. It can be signed or unsigned. **Example:**

```
partitions X(6);
for (auto& x : X)
    cout << x << " ";
```

Prints out:

```
[ 1 1 1 1 1 1 ] [ 2 1 1 1 1 ] [ 3 1 1 1 ] [ 2 2 1 1 ] [ 4 1 1 ] [ 3 2 1 ] [ 2 2 2 ] [ 5 1 ] [ 4 2 ] [ 3 3 ] [
```

7.5.2 Constructor & Destructor Documentation

7.5.2.1 template< class IntType > dscr::basic_partitions< IntType >::basic_partitions (IntType *n*) [inline]

Constructor.

Parameters

n	is an integer >= 0

7.5.2.2 template< class IntType > dscr::basic_partitions< IntType >::basic_partitions (IntType *n*, IntType *numparts*) [inline]

Constructor.

Parameters

n	is an integer >= 0
numparts	is an integer >= 1 and <= n

7.5.2.3 template< class IntType > dscr::basic_partitions< IntType >::basic_partitions (IntType *n*, IntType *minnumparts*, IntType *maxnumparts*) [inline]

Constructor.

Parameters

n	is an integer >= 0
minnumparts	is an integer >= 1 and <= n
maxnumparts	is an integer >= minnumparts and <= n

7.5.3 Member Function Documentation

7.5.3.1 template< class IntType > size_type dscr::basic_partitions< IntType >::size ( ) const [inline]

The total number of partitions.

Returns

p_n

The documentation for this class was generated from the following file:

- Partitions.hpp

7.6 dscr::basic_permutations< IntType > Class Template Reference

class of all n! permutation of size n of the set {0,1,...,n-1}.

```
#include <Permutations.hpp>
```

Classes

- class iterator

  Random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.
- class reverse_iterator

  Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

Public Types

- typedef long long int **difference_type**
- typedef unsigned long long int **size_type**
- typedef vector< IntType > **value_type**
- typedef vector< IntType > **permutation**

Public Member Functions

- basic_permutations (IntType n)

  Constructor.
- size_type size () const

  The total number of permutations.
- permutation identity () const

  Returns the identity permutation: [1, 2, 3, ... , (n-1)].
- permutation random () const

  Constructs a random permutation of {0,1,2,...,n-1}.
- size_type get_index (const permutation &perm, size_t start=0)

  Returns the ID of the iterator whose value is perm. That is, the index of permutation perm in the lexicographic order.
- const iterator & **begin** () const
- const iterator & **end** () const
- const reverse_iterator & **rbegin** () const
- const reverse_iterator & **rend** () const
- permutation operator[] (size_type m) const

  Access to the m-th permutation (slow for iteration)

Static Public Member Functions

- static void **construct_permutation** (permutation &data, size_type m)

7.6.1 Detailed Description

template<class IntType>class dscr::basic_permutations< IntType >

class of all n! permutation of size n of the set {0,1,...,n-1}.

Parameters

IntType	should be an integral type with enough space to store n and k. It can be signed or unsigned.
n	should be an integer <= 20, since 20! already exceeds the numeric limits of a long unsigned int C++ **Example:**

```
permutations X(3);
for (const auto& x : X)
    cout << x << " ";

Prints out:
    [ 0 1 2 ] [ 0 2 1 ] [ 1 0 2 ] [ 1 2 0 ] [ 2 0 1 ] [ 2 1 0 ]
```

Example 3:

```
string A = "abc";
permutations X(A.size());
for (const auto& x : X)
{
    auto b = compose(A,x);
    cout << b << "-";
}
```

Prints out: abc-acb-bac-bca-cab-cba-

7.6.2 Constructor & Destructor Documentation

7.6.2.1 template<class IntType > **dscr::basic_permutations**< IntType >::basic_permutations (IntType *n*) [inline]

Constructor.

Parameters

n	is an integer >= 0

7.6.3 Member Function Documentation

7.6.3.1 template<class IntType > size_type **dscr::basic_permutations**< IntType >::get_index (const permutation & *perm*, size_t *start* = 0) [inline]

Returns the ID of the iterator whose value is perm. That is, the index of permutation perm in the lexicographic order.

Inverse of operator[]. If permutation x is the m-th permutation, then get_index(x) is m. If one has a permutations::iterator, then the member function ID() should return the same value.

Returns

the index of permutation comb, as if basic_permutations was a proper data structure

Note

This constructs the proper index from scratch. If an iterator is already known, calling ID() on the iterator is much more efficient.

7.6.3.2 template<class IntType > permutation **dscr::basic_permutations**< IntType >::identity ( ) const [inline]

Returns the identity permutation: [1, 2, 3, ... , (n-1)].

Parameters

n	is an integer >= 0

7.6.3.3 template<class IntType > permutation **dscr::basic_permutations**< IntType >::operator[] (size_type *m*) const [inline]

Access to the m-th permutation (slow for iteration)

This is equivalent to calling ∗(begin()+m)

Parameters

m	should be an integer between 0 and size(). Undefined behavior otherwise.

Returns

The m-th permutation, as defined in the order of iteration (lexicographic)

7.6.3.4 template<class IntType > size_type dscr::basic_permutations< IntType >::size ( ) const [inline]

The total number of permutations.

Returns

n!

The documentation for this class was generated from the following file:

- Permutations.hpp

7.7 dscr::basic_set_partitions< IntType > Class Template Reference

class of set_partitions of the number n.

```
#include <SetPartitions.hpp>
```

Classes

- class iterator

  Forward iterator class.

Public Types

- typedef long long int **difference_type**
- typedef unsigned long long int **size_type**
- typedef vector< vector< IntType > > **value_type**
- typedef vector< vector< IntType > > **set_partition**
- typedef vector< IntType > **number_partition**

Public Member Functions

- basic_set_partitions (IntType n)

  Constructor.
- basic_set_partitions (IntType n, IntType numparts)

  Constructor.
- basic_set_partitions (IntType n, IntType minnumparts, IntType maxnumparts)

  Constructor.
- size_type size () const

  The total number of set_partitions.
- IntType **get_n** () const
- const iterator & **begin** () const
- const iterator & **end** () const

Static Public Member Functions

- static bool **next_set_partition** (set_partition &data, const number_partition &part)
- static void **fill_first_set_partition** (set_partition &data, const number_partition &part)

7.7.1 Detailed Description

template<class IntType>class dscr::basic_set_partitions< IntType >

class of set_partitions of the number n.

Parameters

IntType	should be an integral type with enough space to store n and k. It can be signed or unsigned. **Example:**

```
set_partitions X(3);
for (auto& x : X)
    cout << x << endl;
```

Prints out all set partitions of {0,1,2}:

```
[ [ 0 ] [ 1 ] [ 2 ] ]
[ [ 0 1 ] [ 2 ] ]
[ [ 0 2 ] [ 1 ] ]
[ [ 1 2 ] [ 0 ] ]
[ [ 0 1 2 ] ]
```

Example 2:

One can specify the number of parts:

```
set_partitions X(4,2);
for (auto& x : X)
    cout << x << endl;
```

Prints out all set partitions of {0,1,2,3,4} with exactly 2 parts:

```
[ [ 0 1 2 ] [ 3 ] ]
[ [ 0 1 3 ] [ 2 ] ]
[ [ 0 2 3 ] [ 1 ] ]
[ [ 1 2 3 ] [ 0 ] ]
[ [ 0 1 ] [ 2 3 ] ]
[ [ 0 2 ] [ 1 3 ] ]
[ [ 0 3 ] [ 1 2 ] ]
```

7.7.2 Constructor & Destructor Documentation

7.7.2.1 template< class IntType > dscr::basic_set_partitions< IntType >::basic_set_partitions (IntType *n*) [inline]

Constructor.

Parameters

n	is an integer >= 0

7.7.2.2 template<class IntType > dscr::basic_set_partitions< IntType >::basic_set_partitions (IntType *n*, IntType *numparts*) `[inline]`

Constructor.

Parameters

n	is an integer >= 0
numparts	is an integer >= 1 and <= n

7.7.2.3 template<class IntType > dscr::basic_set_partitions< IntType >::basic_set_partitions (IntType *n*, IntType *minnumparts*, IntType *maxnumparts*) `[inline]`

Constructor.

Parameters

n	is an integer >= 0
minnumparts	is an integer >= 1 and <= n
maxnumparts	is an integer >= minnumparts and <= n

7.7.3 Member Function Documentation

7.7.3.1 template<class IntType > size_type dscr::basic_set_partitions< IntType >::size ( ) const `[inline]`

The total number of set_partitions.

Returns

If the number of parts was not specified, then the bell number B_n. If it was, then the sum of the appropiate stirling numbers of the second kind.

The documentation for this class was generated from the following file:

- SetPartitions.hpp

7.8 dscr::basic_subsets< BoolType > Class Template Reference

class of all 2^n subsets of the set {0,1,...,n-1}, expressed as incidence vectors

`#include <Subsets.hpp>`

Classes

- class iterator

  Random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.
- class reverse_iterator

  Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

Public Types

- typedef long long int **difference_type**
- typedef unsigned long long int **size_type**
- typedef vector< BoolType > **value_type**
- typedef vector< BoolType > **subset**

Public Member Functions

- basic_subsets (size_t n)

  Constructor.
- size_type size () const

  The total number of subsets.
- size_type get_index (const subset &set) const

  Returns the ID of the iterator whose value is set. That is, the index of subset sub in the lexicographic order.
- const iterator & **begin** () const
- const iterator & **end** () const
- const reverse_iterator & **rbegin** () const
- const reverse_iterator & **rend** () const
- subset operator[] (size_type m) const

  Access to the m-th subset (slow for iteration)

Static Public Member Functions

- static void **next_subset** (subset &data)
- static void **prev_subset** (subset &data)
- static void **construct_subset** (subset &data, size_type m)

7.8.1 Detailed Description

template<class BoolType>class dscr::basic_subsets< BoolType >

class of all 2^n subsets of the set {0,1,...,n-1}, expressed as incidence vectors

Parameters

BoolType	is at least a bool, but it can be an int, uint, etc. It can be signed or unsigned. **Example:**

```
subsets X(4);
for (const auto& x : X)
    cout << x << " ";
```

Prints out:

```
[0000] [1000] [0100] [1100] [0010] [1010] [0110] [1110] [0001] [1001] [0101] [1101] [0011] [1011] [0111] [1111
```

7.8.2 Constructor & Destructor Documentation

7.8.2.1 template<class BoolType > dscr::basic_subsets< BoolType >::basic_subsets (size_t *n*) [inline]

Constructor.

Parameters

n	is an integer >= 0

7.8.3 Member Function Documentation

7.8.3.1 template<class BoolType > size_type dscr::basic_subsets< BoolType >::get_index (const subset & *set*) const [inline]

Returns the ID of the iterator whose value is set. That is, the index of subset sub in the lexicographic order.

Inverse of operator[]. If subset x is the m-th subset, then get_index(x) is m. If one has a subsets::iterator, then the member function ID() should return the same value.

Returns

the index of subset sub, as if basic_subsets was a proper data structure

Note

This constructs the proper index from scratch. If an iterator is already known, calling ID on the iterator is much more efficient.

7.8.3.2 template<class BoolType > subset dscr::basic_subsets< BoolType >::operator[] (size_type *m*) const [inline]

Access to the m-th subset (slow for iteration)

This is equivalent to calling ∗(begin()+m)

Parameters

m	should be an integer between 0 and size(). Undefined behavior otherwise.

Returns

The m-th subset, as defined in the order of iteration (lexicographic)

7.8.3.3 template<class BoolType > size_type dscr::basic_subsets< BoolType >::size ( ) const [inline]

The total number of subsets.

Returns

2^n

The documentation for this class was generated from the following file:

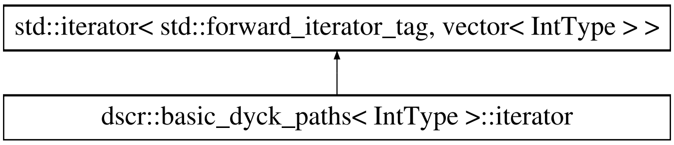
- Subsets.hpp

7.9 dscr::basic_dyck_paths< IntType >::iterator Class Reference

Forward iterator class.

```
#include <DyckPaths.hpp>
```

Inheritance diagram for dscr::basic_dyck_paths< IntType >::iterator:


std::iterator< std::forward_iterator_tag, vector< IntType > >
dscr::basic_dyck_paths< IntType >::iterator

Public Member Functions

- **iterator** (IntType n)
- iterator & **operator++** ()
- iterator & **operator--** ()
- const vector< IntType > & **operator∗** () const
- const dyck_path ∗ **operator->** () const
- size_type **ID** () const
- bool **operator==** (const iterator &it) const
- bool **operator!=** (const iterator &it) const
- bool **is_at_end** (IntType n) const
- void **reset** (IntType r)

Friends

- class **basic_dyck_paths**
- difference_type **operator-** (const iterator &lhs, const iterator &rhs)

7.9.1 Detailed Description

template<class IntType>class dscr::basic_dyck_paths< IntType >::iterator

Forward iterator class.

The documentation for this class was generated from the following file:

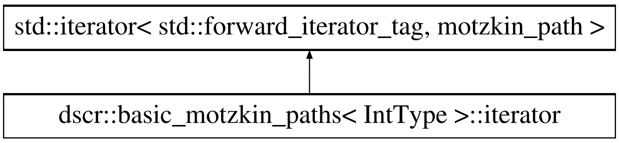
- DyckPaths.hpp

7.10 dscr::basic_motzkin_paths< IntType >::iterator Class Reference

Forward iterator class.

`#include <Motzkin.hpp>`

Inheritance diagram for dscr::basic_motzkin_paths< IntType >::iterator:


std::iterator< std::forward_iterator_tag, motzkin_path >
dscr::basic_motzkin_paths< IntType >::iterator

Public Member Functions

- **iterator** (IntType n)
- iterator & **operator++** ()
- iterator & **operator--** ()
- const vector< IntType > & **operator*** () const
- const motzkin_path * **operator->** () const
- size_type **ID** () const
- bool **operator==** (const iterator &it) const
- bool **operator!=** (const iterator &it) const
- void **reset** (IntType n)

Friends

- class **basic_motzkin_paths**
- difference_type **operator-** (const iterator &lhs, const iterator &rhs)

7.10.1 Detailed Description

template< class IntType > class dscr::basic_motzkin_paths< IntType >::iterator

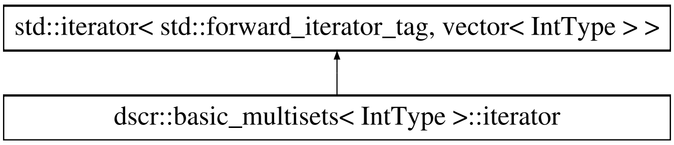
Forward iterator class.

The documentation for this class was generated from the following file:

- Motzkin.hpp

7.11 dscr::basic_multisets< IntType >::iterator Class Reference

Inheritance diagram for dscr::basic_multisets< IntType >::iterator:


std::iterator< std::forward_iterator_tag, vector< IntType > >
dscr::basic_multisets< IntType >::iterator

Public Member Functions

- **iterator** (const vector< IntType > &total)
- iterator & **operator++** ()
- const vector< IntType > & **operator*** () const
- vector< IntType > & **operator*** ()
- bool **operator==** (const iterator &it) const
- bool **operator!=** (const iterator &it) const

Friends

- class **basic_multisets**

The documentation for this class was generated from the following file:

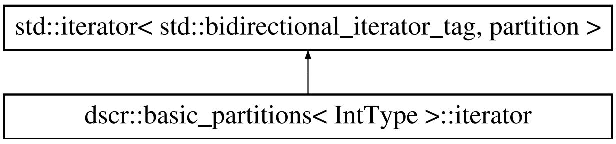
- Multisets.hpp

7.12 dscr::basic_partitions< IntType >::iterator Class Reference

Forward iterator class.

```
#include <Partitions.hpp>
```

Inheritance diagram for dscr::basic_partitions< IntType >::iterator:


std::iterator< std::bidirectional_iterator_tag, partition >
dscr::basic_partitions< IntType >::iterator

Public Member Functions

- **iterator** (IntType n, IntType numparts)
- iterator & **operator++** ()
- const vector< IntType > & **operator**∗ () const
- const partition ∗ **operator->** () const
- size_type **ID** () const
- bool **operator==** (const iterator &it) const
- bool **operator!=** (const iterator &it) const

Friends

- class **basic_partitions**
- difference_type **operator-** (const iterator &lhs, const iterator &rhs)

7.12.1 Detailed Description

template<class IntType>class dscr::basic_partitions< IntType >::iterator

Forward iterator class.

The documentation for this class was generated from the following file:

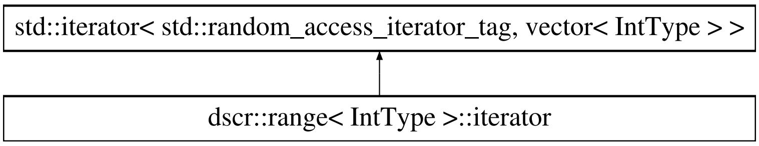
- Partitions.hpp

7.13 dscr::range< IntType >::iterator Class Reference

Random access iterator class.

```
#include <Range.hpp>
```

Inheritance diagram for dscr::range< IntType >::iterator:


std::iterator< std::random_access_iterator_tag, vector< IntType > >
dscr::range< IntType >::iterator

Public Member Functions

- **iterator** (size_type t_from)
- **iterator** (size_type t_from, size_type t_step)
- iterator & **operator++** ()
- iterator & **operator--** ()
- const IntType & **operator*** () const
- iterator & operator+= (long int n)

  Random access capabilities to the iterators.
- iterator & **operator-=** (long int n)
- bool **operator==** (const iterator &it)
- bool **operator!=** (const iterator &it)
- difference_type **operator-** (const iterator &it)
- size_type **step** () const

Friends

- class **range**

7.13.1 Detailed Description

template<class IntType>class dscr::range< IntType >::iterator

Random access iterator class.

7.13.2 Member Function Documentation

7.13.2.1 template<class IntType > iterator& dscr::range< IntType >::iterator::operator+= (long int *n*) [inline]

Random access capabilities to the iterators.

Parameters

n	-> This assumes 0 <= n+ID <= size(n)

The documentation for this class was generated from the following file:

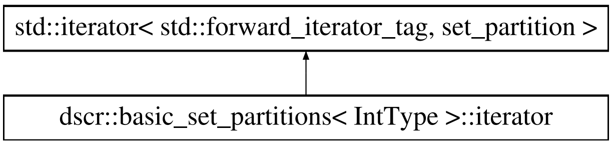
- Range.hpp

7.14 dscr::basic_set_partitions< IntType >::iterator Class Reference

Forward iterator class.

`#include <SetPartitions.hpp>`

Inheritance diagram for dscr::basic_set_partitions< IntType >::iterator:


std::iterator< std::forward_iterator_tag, set_partition >
dscr::basic_set_partitions< IntType >::iterator

Public Member Functions

- **iterator** (IntType n, IntType numparts)
- iterator & **operator++** ()
- const set_partition & **operator∗** () const
- const set_partition ∗ **operator->** () const
- size_type **ID** () const
- bool **operator==** (const iterator &it) const
- bool **operator!=** (const iterator &it) const

Friends

- class **basic_set_partitions**
- difference_type **operator-** (const iterator &lhs, const iterator &rhs)

7.14.1 Detailed Description

template< class IntType >class dscr::basic_set_partitions< IntType >::iterator

Forward iterator class.

The documentation for this class was generated from the following file:

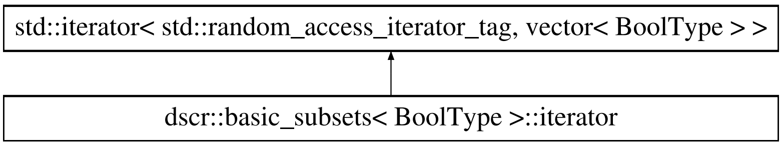
- SetPartitions.hpp

7.15 dscr::basic_subsets< BoolType >::iterator Class Reference

Random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

```
#include <Subsets.hpp>
```

Inheritance diagram for dscr::basic_subsets< BoolType >::iterator:


std::iterator< std::random_access_iterator_tag, vector< BoolType > >
dscr::basic_subsets< BoolType >::iterator

Public Member Functions

- **iterator** (size_t n)
- iterator & **operator++** ()
- iterator & **operator--** ()
- const vector< BoolType > & **operator∗** () const
- iterator & operator+= (difference_type n)

  Random access capabilities to the iterators.
- iterator & **operator-=** (difference_type n)
- size_type **ID** () const
- bool **operator==** (const iterator &it) const
- bool **operator!=** (const iterator &it) const
- bool **is_at_end** (size_t n) const
- void **reset** (size_t n)

Friends

- class **basic_subsets**
- iterator **operator+** (iterator lhs, difference_type n)
- iterator **operator-** (iterator lhs, difference_type n)
- difference_type **operator-** (const iterator &lhs, const iterator &rhs)

7.15.1 Detailed Description

template<class BoolType>class dscr::basic_subsets< BoolType >::iterator

Random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

7.15.2 Member Function Documentation

7.15.2.1 template<class BoolType > iterator& dscr::basic_subsets< BoolType >::iterator::operator+= (difference_type *n*) [inline]

Random access capabilities to the iterators.

Parameters

n	-> This assumes 0 <= n+ID <= size(n,k)

The documentation for this class was generated from the following file:

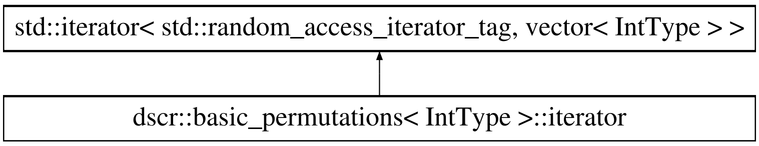
- Subsets.hpp

7.16 dscr::basic_permutations< IntType >::iterator Class Reference

Random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

```
#include <Permutations.hpp>
```

Inheritance diagram for dscr::basic_permutations< IntType >::iterator:


std::iterator< std::random_access_iterator_tag, vector< IntType > >
dscr::basic_permutations< IntType >::iterator

Public Member Functions

- **iterator** (IntType n)
- iterator & **operator++** ()
- iterator & **operator--** ()
- const vector< IntType > & **operator∗** () const
- iterator & operator+= (long int n)

  Random access capabilities to the iterators.
- iterator & **operator-=** (long int n)
- size_type **ID** () const
- bool **operator==** (const iterator &it) const
- bool **operator!=** (const iterator &it) const
- bool **is_at_end** (IntType n) const
- void **reset** (IntType r)

Friends

- class **basic_permutations**
- iterator **operator+** (iterator lhs, difference_type n)
- iterator **operator-** (iterator lhs, difference_type n)
- difference_type **operator-** (const iterator &lhs, const iterator &rhs)

7.16.1 Detailed Description

template<class IntType>class dscr::basic_permutations< IntType >::iterator

Random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

7.16.2 Member Function Documentation

7.16.2.1 template<class IntType > iterator& dscr::basic_permutations< IntType >::iterator::operator+= (long int *n*) [inline]

Random access capabilities to the iterators.

Parameters

n	-> This assumes 0 <= n+ID <= size(n,k)

The documentation for this class was generated from the following file:

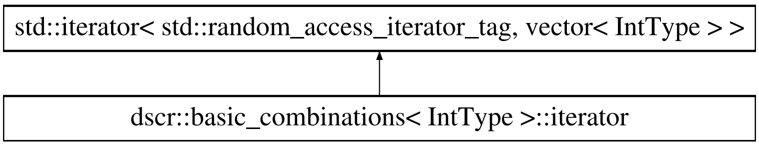
- Permutations.hpp

7.17 dscr::basic_combinations< IntType >::iterator Class Reference

Random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

```
#include <Combinations.hpp>
```

Inheritance diagram for dscr::basic_combinations< IntType >::iterator:


std::iterator< std::random_access_iterator_tag, vector< IntType > >
dscr::basic_combinations< IntType >::iterator

Public Member Functions

- **iterator** (IntType n, IntType r)
- iterator & **operator++** ()
- iterator & **operator--** ()
- const vector< IntType > & **operator*** () const
- const combination * **operator->** () const
- iterator & operator+= (difference_type n)

  Random access capabilities to the iterators.
- iterator & **operator-=** (difference_type n)
- size_type **ID** () const
- bool **operator==** (const iterator &it) const

- bool **operator!=** (const iterator &it) const
- bool **is_at_end** (IntType n) const
- void **reset** (IntType n, IntType r)

Friends

- class **basic_combinations**
- iterator **operator+** (iterator lhs, difference_type n)
- iterator **operator-** (iterator lhs, difference_type n)
- difference_type **operator-** (const iterator &lhs, const iterator &rhs)

7.17.1 Detailed Description

template<class IntType>class dscr::basic_combinations< IntType >::iterator

Random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

7.17.2 Member Function Documentation

7.17.2.1 template<class IntType > iterator& dscr::basic_combinations< IntType >::iterator::operator+= (difference_type *n*) [inline]

Random access capabilities to the iterators.

Parameters

n	-> This assumes 0 <= n+ID <= size(n,k)

The documentation for this class was generated from the following file:

- Combinations.hpp

7.18 dscr::range< IntType > Class Template Reference

Similar to python range(n) or range(n,m) or range(n,m,step).

```
#include <Range.hpp>
```

Classes

- class iterator

  Random access iterator class.

Public Types

- typedef long int **difference_type**
- typedef IntType **size_type**
- typedef IntType **value_type**

Public Member Functions

- range (IntType n)

  Constructor.
- **range** (IntType t_from, IntType t_to, IntType t_step=1)
- size_type **size** () const
- **operator vector**< **IntType** > () const
- const iterator & **begin** () const
- const iterator & **end** () const
- IntType **operator[]** (size_type m) const

7.18.1 Detailed Description

template<class IntType>class dscr::range< IntType >

Similar to python range(n) or range(n,m) or range(n,m,step).

Parameters

n	is an integer

Returns

an abstract random-access container whose elements are {n,n+1,n+2,...,m-1}

7.18.2 Constructor & Destructor Documentation

7.18.2.1 template<class IntType > dscr::range< IntType >::range (IntType *n*) [inline]

Constructor.

Parameters

n	is an integer >= 0

The documentation for this class was generated from the following file:

- Range.hpp

7.19 dscr::RClock Class Reference

Static Public Member Functions

- static RClock & **Instance** ()

Public Attributes

- std::chrono::time_point
  < std::chrono::high_resolution_clock > **start_timer**
- std::chrono::time_point
  < std::chrono::high_resolution_clock > **running_timer**

The documentation for this class was generated from the following file:

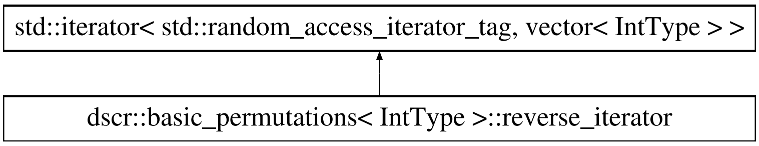
- TimeHelpers.hpp

7.20 dscr::basic_permutations< IntType >::reverse_iterator Class Reference

Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

```
#include <Permutations.hpp>
```

Inheritance diagram for dscr::basic_permutations< IntType >::reverse_iterator:


std::iterator< std::random_access_iterator_tag, vector< IntType > >
dscr::basic_permutations< IntType >::reverse_iterator

Public Member Functions

- **reverse_iterator** (IntType n)
- reverse_iterator & **operator++** ()
- reverse_iterator & **operator--** ()
- const permutation & **operator**∗ () const
- reverse_iterator & operator+= (long int m)

  Random access capabilities to the iterators.
- reverse_iterator & **operator-=** (long int n)
- size_type **ID** () const
- bool **operator==** (const reverse_iterator &it) const
- bool **operator!=** (const reverse_iterator &it) const
- void **reset** (IntType n)

Friends

- class **basic_permutations**
- reverse_iterator **operator+** (reverse_iterator lhs, difference_type n)
- reverse_iterator **operator-** (reverse_iterator lhs, difference_type n)
- difference_type **operator-** (const reverse_iterator &lhs, const reverse_iterator &rhs)

7.20.1 Detailed Description

template<class IntType>class dscr::basic_permutations< IntType >::reverse_iterator

Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

7.20.2 Member Function Documentation

7.20.2.1 template<class IntType > reverse_iterator& dscr::basic_permutations< IntType >::reverse_iterator::operator+= (long int *m*) [inline]

Random access capabilities to the iterators.

Parameters

n	-> This assumes 0 <= n+ID <= size(n,k)

The documentation for this class was generated from the following file:

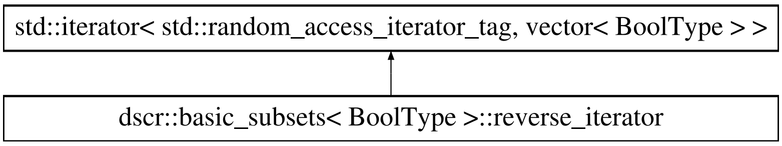
- Permutations.hpp

7.21 dscr::basic_subsets< BoolType >::reverse_iterator Class Reference

Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

```
#include <Subsets.hpp>
```

Inheritance diagram for dscr::basic_subsets< BoolType >::reverse_iterator:


std::iterator< std::random_access_iterator_tag, vector< BoolType > >
dscr::basic_subsets< BoolType >::reverse_iterator

Public Member Functions

- **reverse_iterator** (size_t n)
- reverse_iterator & **operator++** ()
- reverse_iterator & **operator--** ()
- const vector< BoolType > & **operator*** ()
- const vector< BoolType > & **operator*** () const
- reverse_iterator & operator+= (difference_type m)

  Random access capabilities to the iterators.
- reverse_iterator & **operator-=** (difference_type n)
- size_type **ID** () const
- bool **operator==** (const reverse_iterator &it)
- bool **operator!=** (const reverse_iterator &it)
- bool **is_at_end** () const
- void **reset** (BoolType n)

Friends

- class **basic_subsets**
- reverse_iterator **operator+** (reverse_iterator lhs, difference_type n)
- reverse_iterator **operator-** (reverse_iterator lhs, difference_type n)
- difference_type **operator-** (const reverse_iterator &lhs, const reverse_iterator &rhs)

7.21.1 Detailed Description

template<class BoolType>class dscr::basic_subsets< BoolType >::reverse_iterator

Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

7.21.2 Member Function Documentation

7.21.2.1 template<class BoolType > reverse_iterator& dscr::basic_subsets< BoolType >::reverse_iterator::operator+= (difference_type *m*) [inline]

Random access capabilities to the iterators.

Parameters

n	-> This assumes 0 <= n+ID <= size(n,k)

The documentation for this class was generated from the following file:

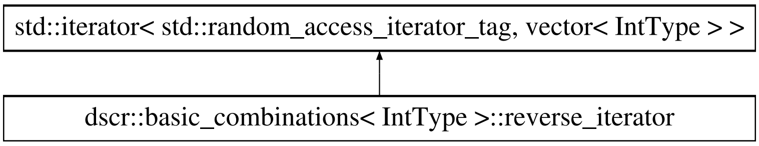
- Subsets.hpp

7.22 dscr::basic_combinations< IntType >::reverse_iterator Class Reference

Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

`#include <Combinations.hpp>`

Inheritance diagram for dscr::basic_combinations< IntType >::reverse_iterator:


std::iterator< std::random_access_iterator_tag, vector< IntType > >
dscr::basic_combinations< IntType >::reverse_iterator

Public Member Functions

- **reverse_iterator** (IntType n, IntType r)
- reverse_iterator & **operator++** ()
- reverse_iterator & **operator--** ()
- const combination & **operator*** ()
- const combination & **operator*** () const
- const combination * **operator->** () const
- reverse_iterator & operator+= (difference_type m)

  Random access capabilities to the iterators.
- reverse_iterator & **operator-=** (difference_type n)
- size_type **ID** () const
- bool **operator==** (const reverse_iterator &it)
- bool **operator!=** (const reverse_iterator &it)
- bool **is_at_end** () const
- void **reset** (IntType n, IntType r)

Friends

- class **basic_combinations**
- reverse_iterator **operator+** (reverse_iterator lhs, difference_type n)
- reverse_iterator **operator-** (reverse_iterator lhs, difference_type n)
- difference_type **operator-** (const reverse_iterator &lhs, const reverse_iterator &rhs)

7.22.1 Detailed Description

template<class IntType>class dscr::basic_combinations< IntType >::reverse_iterator

Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

7.22.2 Member Function Documentation

7.22.2.1 template<class IntType > reverse_iterator& dscr::basic_combinations< IntType >::reverse_iterator::operator+= (difference_type *m*) [inline]

Random access capabilities to the iterators.

Parameters

n	-> This assumes 0 <= n+ID <= size(n,k)

The documentation for this class was generated from the following file:

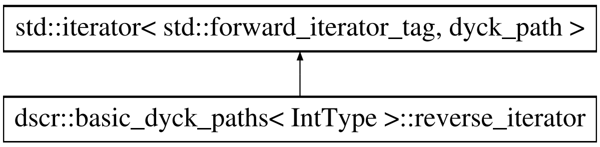
- Combinations.hpp

7.23 dscr::basic_dyck_paths< IntType >::reverse_iterator Class Reference

Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

```
#include <DyckPaths.hpp>
```

Inheritance diagram for dscr::basic_dyck_paths< IntType >::reverse_iterator:


std::iterator< std::forward_iterator_tag, dyck_path >
dscr::basic_dyck_paths< IntType >::reverse_iterator

Public Member Functions

- **reverse_iterator** (IntType n)
- reverse_iterator & **operator++** ()
- reverse_iterator & **operator--** ()
- const dyck_path & **operator*** ()
- const dyck_path & **operator*** () const
- const dyck_path * **operator->** () const
- size_type **ID** () const
- bool **operator==** (const reverse_iterator &it)
- bool **operator!=** (const reverse_iterator &it)

Friends

- class **basic_dyck_paths**
- difference_type **operator-** (const reverse_iterator &lhs, const reverse_iterator &rhs)

7.23.1 Detailed Description

template< class IntType > class dscr::basic_dyck_paths< IntType >::reverse_iterator

Reverse random access iterator class. It's much more efficient as a bidirectional iterator than purely random access.

The documentation for this class was generated from the following file:

- DyckPaths.hpp

Index