Enumeration of random labeled graphs

Generated on Mon Sep 18 2023 for Enumeration of random labeled graphs by Doxygen 1.9.7

Mon Sep 18 2023 17:24:55

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 Graph_Params Struct Reference

`#include <rand_graph_lib.h>`

Data Fields

- unsigned short n
- unsigned short m_min
- unsigned short m_max
- unsigned short m_crit
- unsigned long long n_graphs
- unsigned long long n_trees
- double p_edge
- unsigned short * E
- unsigned short ** G

3.1.1 Detailed Description

Graph parameters structure, declared as a new type.

3.1.2 Field Documentation

3.1.2.1 n

`unsigned short n`

Number of labeled vertices.

3.1.2.2 m_min

`unsigned short m_min`

Minimal number of edges in a connected graph (tree):

$m_{min} = n - 1$ (tree).

3.1.2.3 m_max

`unsigned short m_max`

Maximal possible number of edges in the complete graph with n labeled vertices:

$m_{max} = \binom{n}{2} = \frac{n(n-1)}{2}$ (complete graph).

3.1.2.4 m_crit

`unsigned short m_crit`

Number of edges as connectedness threshold:

$m_{crit} = \binom{n-1}{2} = \frac{(n-1)(n-2)}{2}$.

3.1.2.5 n_graphs

`unsigned long long n_graphs`

Maximal possible number of graphs with n labeled vertices:

$n_{graphs} = 2^{m_{max}}$.

3.1.2.6 n_trees

`unsigned long long n_trees`

Maximal possible number of trees with n labeled vertices, p.292, Erdős [4] :

$n_{trees} = n^{n-2}$.

3.1.2.7 p_edge

`double p_edge`

Edge probability $0 \leq p \leq 1$

3.1.2.8 E

`unsigned short * E`

Pointer to memory allocation with an array of m_{max} vertex pair combinations, stored as $256u + v$:

Example for a graph on 3 vertices ($m_{max} = 3$): `[256 * 0 + 1 = 1, 256 * 0 + 2 = 2, 256 * 1 + 2 = 258]`.

3.1.2.9 G

`unsigned short ** G`

Pointer to memory allocation with 2d array $(n \times n)$ – graph in the form of adjacencies arrays.

Chapter 4

File Documentation

4.1 C:/Dropbox_full_copy/HSE/MDS/Study/Year_↩ 1/cython/proj/include/rand_graph_lib.h File Reference

```
#include <stdbool.h>
```

Data Structures

- struct Graph_Params

Functions

- Graph_Params set_rand_gnp_params (unsigned short n, double p_edge)
- void show_vertex_pairs (Graph_Params gp)
- void show_rand_gnp (Graph_Params gp)
- void construct_rand_gnp (Graph_Params gp)
- void destroy_rand_gnp (Graph_Params gp)
- void dfs (Graph_Params gp, unsigned short v, bool ∗visited)
- unsigned short count_connected_components (Graph_Params gp)
- bool is_connected (Graph_Params gp)
- unsigned long long A006125_total (unsigned short n)
- unsigned long long A001187_conn (unsigned short n)
- unsigned long long A054592_disconn (unsigned short n)
- double prob_conn (unsigned short n, double p_edge)

4.1.1 Detailed Description

Random graph library declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2023-08-30

Copyright

4.1.2 Function Documentation

4.1.2.1 set_rand_gnp_params()

```
Graph_Params set_rand_gnp_params (
            unsigned short n,
            double p_edge )
```

Populate and return graph parameters as a structure.

Parameters

n	Graph_Params::n
p_edge	Graph_Params::p_edge

Returns

Graph parameters as per the structure Graph_Params

Implementation notes:

1. Integer overflow unsafe, since the rhs's size is not checked before assignment.
2. Null pointer unsafe, since `calloc()` might fail, but this is not checked to reduce running time.
3. Time complexity: $\max[\mathcal{O}(\texttt{binom()}), \mathcal{O}(\texttt{pow()})]$.

4.1.2.2 show_vertex_pairs()

```
void show_vertex_pairs (
            Graph_Params gp )
```

Print out all possible vertex pairs.

Parameters

gp	Graph_Params

Implementation notes:

1. Vertex pair E_i is stored in one unsigned short allocation, thus extracting high part for u and low part for v.
2. Time complexity: $\mathcal{O}(m_{max})$.

4.1.2.3 show_rand_gnp()

```
void show_rand_gnp (
            Graph_Params gp )
```

Print out graph adjacencies for each vertex (row by row).

Parameters

gp	Graph_Params

Implementation notes:

1. Time complexity: $\mathcal{O}(n^2)$.

4.1.2.4 construct_rand_gnp()

```
void construct_rand_gnp (
            Graph_Params gp )
```

Construct a random $\mathcal{G}(n, p)$ graph with n vertices and pm_{max} edges (on average).

Parameters

gp	Graph_Params

Implementation notes:

1. Vertex pair E_i is stored in one unsigned short allocation, thus extracting high part for u and low part for v.
2. Time complexity: $\mathcal{O}(m_{max}$ `bernoulli_rv()` $)$.

Usage notes:

1. `destroy_rand_gnp()` must be called immediately after the created graph is no longer used to clean graphs's adjacencies arrays.

4.1.2.5 destroy_rand_gnp()

```
void destroy_rand_gnp (
            Graph_Params gp )
```

Destroy a random graph – clean graph's adjacencies arrays.

Parameters

gp	Graph_Params

Implementation notes:

1. Time complexity: $\mathcal{O}(n$ `memset()` $)$.

Usage notes:

1. `destroy_rand_gnp()` must be called immediately after the created graph is no longer used to clean graph's adjacencies arrays.

4.1.2.6 dfs()

```
void dfs (
            Graph_Params gp,
            unsigned short v,
            bool * visited )
```

Visit every vertex once.

Parameters

gp	Graph_Params
v	Current vertex
visited	Array of visited vertices

Implementation notes:

1. Recursive Depth-first search (DFS) algorithm.
2. Time complexity: $\mathcal{O}(|V| + |E|)$.

4.1.2.7 count_connected_components()

```
unsigned short count_connected_components (
            Graph_Params gp )
```

Count the number of connected components in a graph.

Parameters

gp	Graph_Params

Returns

Number of connected components

Implementation notes:

1. The algorithm counts the number of connected components by employing `dfs()`.
2. Time complexity: $\mathcal{O}(|V| + |E|)$.

4.1.2.8 is_connected()

```
bool is_connected (
            Graph_Params gp )
```

Check if a graph is connected by counting the number of connected components in it.

Parameters

gp	Graph_Params

Returns

`True (1)` if G has exactly one connected component, i.e. it is connected, `false (0)` otherwise

Implementation notes:

1. The algorithm first counts the number of connected components by employing `count_connected_components()` and then returns `true (1)`, if there is only one connected component in the graph, `false (0)` otherwise.
2. Time complexity: $\mathcal{O}(|V| + |E|)$.

4.1.2.9 A006125_total()

```
unsigned long long A006125_total (
            unsigned short n )
```

Find the total number of labeled graphs (connected and disconnected) with n nodes – sequence $A006125$ [5] .

Parameters

n	Graph order – Graph_Params::n

Returns

Number of labeled graphs

Implementation notes:

1. The number of labeled graphs (connected and disconnected) with n nodes is Graph_Params::n_graphs.
2. Integer overflow unsafe, since the result's size is not checked before return.
3. Time complexity: $\max[\mathcal{O}(\texttt{ binom() }), \mathcal{O}(\texttt{ pow() })]$.

Results for $n \in [0, 11]$:

n	A006125(n)
0	1
1	1
2	2
3	8
4	64
5	1 024
6	32 768
7	2 097 152
8	268 435 456
9	68 719 476 736
10	35 184 372 088 832
11	36 028 797 018 963 968

4.1.2.10 A001187_conn()

```
unsigned long long A001187_conn (
            unsigned short n )
```

Find the number of connected labeled graphs with n nodes constructed with the Erdős–Rényi model $\mathcal{G}(n, M)$ – sequence $A001187$ [6] .

Parameters

n	Graph order – Graph_Params::n

Returns

Number of connected labeled graphs

Implementation notes:

1. In this model each graph is chosen randomly with equal probability of $1/n_{graphs}$, where n_{graphs} is Graph_Params::n_graphs.
2. The number C_n of labeled connected graphs of order n is given by the recursive formula (1.2.1), p. 7, Harary [2] ; p was substituted with n to avoid confusion with notation of probability.

$$C_n = 2^{\binom{n}{2}} - \frac{1}{n}\sum_{k=1}^{n-1} k \binom{n}{k} 2^{\binom{n-k}{2}} C_k.$$

3. Integer overflow unsafe, since the result's size is not checked during calculation and before return.
4. Time complexity: $\mathcal{O}(2^{\ (n \max[\mathcal{O}(\texttt{ binom() }), \mathcal{O}(\texttt{ pow() })] - 1)\ })$.

Results for $n \in [0, 11]$:

n	A001187(n)
0	1
1	1
2	1
3	4
4	38
5	728
6	26 704
7	1 866 256
8	251 548 592
9	66 296 291 072
10	34 496 488 594 816
11	35 641 657 548 953 344

4.1.2.11 A054592_disconn()

```
unsigned long long A054592_disconn (
            unsigned short n )
```

Find the number of disconnected labeled graphs with n nodes – sequence $A054592$ [7] .

Parameters

n	Graph order – Graph_Params::n

Returns

Number of disconnected labeled graphs

Implementation notes:

1. Number of labeled graphs (connected and disconnected) with n nodes is sequence $A006125$ [5] . Number of connected labeled graphs with n nodes is sequence $A001187$ [6] . Thus, the number of disconnected labeled graphs with n nodes is simply $A054592(n) = A006125(n) - A001187(n)$.
2. Integer overflow unsafe, since the result's size is not checked during calculation and before return.
3. Time complexity: $\max(\mathcal{O}($ `A006125_total()` $), \mathcal{O}($ `A001187_conn()` $))$.

Results for $n \in [0, 11]$:

n	A054592(n)
0	0
1	0
2	1
3	4
4	26
5	296
6	6 064
7	230 896
8	16 886 864
9	2 423 185 664
10	687 883 494 016
11	387 139 470 010 624

4.1.2.12 prob_conn()

```
double prob_conn (
            unsigned short n,
            double p_edge )
```

Find the probability of connectedness of a random labeled graph constructed with the Gilbert model $\mathcal{G}(n,p)$.

Parameters

n	Graph order – Graph_Params::n
p_edge	Edge probability Graph_Params::p_edge

Returns

$P_n = P(\mathcal{G}(n,p)$ is connected$)$

Implementation notes:

1. In this model every possible edge occurs independently with probability p. The probability of obtaining any one particular random graph with m edges is $p^m(1-p)^{N-m}$, where N is Graph_Params::m_max.
2. The probability of connectedness of a random labeled graph is given by the recursive formula (3), p. 2, Gilbert [3] ; q was substituted by $1-p$ to avoid introducing unnecessary new variable; N was substituted by n to avoid confusion with N above.
$$P_n = 1 - \sum_{k=1}^{n-1} \binom{n-1}{k-1} (1-p)^{k(n-k)} P_k.$$
3. Integer overflow unsafe, since the result's size is not checked during calculation and before return.
4. Time complexity: $\mathcal{O}(2^{\ (n \max[\mathcal{O}(\texttt{binom()}), \mathcal{O}(\texttt{pow()})] - 1)\ })$.

Results for P_n for $n \in [2,11]$ and $p \in [0.1, 0.9]$:

n / p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
2	0.10000	0.20000	0.30000	0.40000	0.50000	0.60000	0.70000	0.80000	0.90000
3	0.02800	0.10400	0.21600	0.35200	0.50000	0.64800	0.78400	0.89600	0.97200
4	0.01293	0.08250	0.21865	0.40038	0.59375	0.76550	0.89249	0.96666	0.99581
5	0.00810	0.08195	0.25626	0.48965	0.71094	0.87026	0.95751	0.99166	0.99949
6	0.00621 †	0.09230	0.31690	0.59555	0.81494 ‡	0.93652	0.98497	0.99805	0.99994
7	0.00551	0.11127	0.39385	0.69878	0.88990	0.97072	0.99484	0.99955	0.99999
8	0.00541	0.13851	0.47987	0.78627	0.93709	0.98677	0.99824	0.99990	1.00000
9	0.00574	0.17396	0.56714	0.85325	0.96474	0.99408	0.99941	0.99998	1.00000
10	0.00644	0.21723	0.64897	0.90128	0.98045	0.99737	0.99980	0.99999	1.00000
11	0.00752	0.26729	0.72107	0.93445	0.98925	0.99885	0.99994	1.00000	1.00000

† Table 1, p. 2, Gilbert [3] : 0.00624.

‡ Table 1, p. 2, Gilbert [3] : 0.81569.

4.2 C:/Dropbox_full_copy/HSE/MDS/Study/Year_↩ 1/cython/proj/include/stats_lib.h File Reference

```
#include <stdbool.h>
```

Functions

- unsigned long long binom (unsigned short n, unsigned short k)
- bool bernoulli_rv (double p)
- double mean (double ∗array, unsigned short length)
- double max (double ∗array, unsigned short length)
- double min (double ∗array, unsigned short length)
- double quantile (double ∗array, unsigned short length, double q)

4.2.1 Detailed Description

Mathematical statistics library declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2023-08-30

Copyright

Copyright (c) 2023

4.2.2 Function Documentation

4.2.2.1 binom()

```
unsigned long long binom (
            unsigned short n,
            unsigned short k )
```

Calculate binomial coefficient $C(n, k)$.

Parameters

n	Integer number n
k	Integer number k

Returns

Binomial coefficient

Implementation notes:

1. Optimized algorithm without explicit calculation of factorial.
2. Integer overflow unsafe, since the result size is not checked during calculation and before return.
3. Time complexity: $\mathcal{O}(r)$, where $r = \min(k, n-k)$.

4.2.2.2 bernoulli_rv()

```
bool bernoulli_rv (
            double p )
```

Return Bernoulli distributed random variable.

Parameters

p	Probability of a single outcome

Returns

`True (1)` or `false (0)`

Implementation notes:

1. Simulates Bernoulli process and returns either 1 or 0. If $p = 0.5$, returned values will simulate tosses of a fair coin (H = 1, T = 0).
2. First uniformly distributed random variable with `p = 1 / RAND_MAX` is obtained with `rand()` which is then transformed into Bernoulli distributed random variable.
3. Assumes that `RAND_MAX` might be at least 16 bit, thus the result of `rand()` is stored in the unsigned int variable.
4. Time complexity: $\mathcal{O}($ `rand()` $)$.

Usage notes:

1. Init random seed by calling `srand(time(NULL))`, if needed, before calling `bernoulli_rv()`.

4.2.2.3 mean()

```
double mean (
            double * array,
            unsigned short length )
```

Find mean of an array.

Parameters

array	Array of real numbers
length	Length of the array

Returns

Arithmetic mean of the array's values

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Time complexity: $\mathcal{O}(length)$.

4.2.2.4 max()

```
double max (
            double * array,
            unsigned short length )
```

Find maximal value in an array.

Parameters

array	Array of real numbers
length	Length of the array

Returns

Arithmetic maximum of the array's values

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Time complexity: $\mathcal{O}(length)$.

4.2.2.5 min()

```
double min (
            double * array,
            unsigned short length )
```

Find minimal value in an array.

Parameters

array	Array of real numbers
length	Length of the array

Returns

Arithmetic minimum of the array's values

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Time complexity: $\mathcal{O}(length)$.

4.2.2.6 quantile()

```
double quantile (
            double * array,
            unsigned short length,
            double q )
```

Find q-th quantile of an array.

Parameters

array	Array of real numbers
length	Length of the array
q	Quantile $\in [0, 1]$

Returns

Array's value corresponding to q

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Input array is first copied to a new array, so that the original array is not corrupted, and then the copy is sorted in ascending order in place by employing the `sort_array()` function.
3. If array's length is even, the arithmetic mean (average) of two neighbor elements is returned. Example: median $(q = 0.5)$ for the array `{1, 2, 3, 4}` is $(2 + 3)/2 = 2.5$.
4. Time complexity: $\max(\mathcal{O}(length), \mathcal{O}($ `sort_array()` $))$.

4.3 C:/Dropbox_full_copy/HSE/MDS/Study/Year_↩ 1/cython/proj/include/utils_lib.h File Reference

```
#include <stdbool.h>
```

Functions

- void show_array (double ∗array, unsigned short length, char sep, bool wide)
- int comp_func_asc (const void ∗a, const void ∗b)
- int comp_func_desc (const void ∗a, const void ∗b)
- void sort_array (double ∗array, unsigned short length, bool ascending)

4.3.1 Detailed Description

Utils library declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2023-08-30

Copyright

Copyright (c) 2023

4.3.2 Function Documentation

4.3.2.1 show_array()

```
void show_array (
            double * array,
            unsigned short length,
            char sep,
            bool wide )
```

Print out an array.

Parameters

array	Array of real numbers
length	Length of the array
sep	Character to separate array's elements
wide	Display style: `true` – shape $(1 \times length)$ (wide), `false` – shape $(length \times 1)$ (tall).

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Display precision cannot be set – as per `printf()`, by default 6 decimal places are printed.
3. Time complexity: $\mathcal{O}(length)$.

4.3.2.2 comp_func_asc()

```
int comp_func_asc (
            const void * a,
            const void * b )
```

Compare function for ascending `qsort()` used in `sort_array()`.

Parameters

a	The first value to compare
b	The second value to compare

Returns

Sort flag: `0 (a == b), 1 (a > b), -1 (a < b)`

4.3.2.3 comp_func_desc()

```
int comp_func_desc (
            const void * a,
            const void * b )
```

Compare function for descending `qsort()` used in `sort_array()`.

Parameters

a	The first value to compare
b	The second value to compare

Returns

Sort flag: `0 (a == b), 1 (a < b), -1 (a > b)`

4.3.2.4 sort_array()

```
void sort_array (
            double * array,
            unsigned short length,
            bool ascending )
```

Sort array's elements either in ascending or descending order.

Parameters

array	Array of real numbers
length	Length of the array
ascending	If `true`, sort in ascending order, otherwise in descending order

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Employs `qsort()` with `comp_func_asc()` and `comp_func_desc()`.
3. Time complexity: might be $\mathcal{O}(length \log(length))$, but in fact is not guaranteed [1].

4.4 C:/Dropbox_full_copy/HSE/MDS/Study/Year_1/cython/proj/src/rand_graph_lib.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include ¨stats_lib.h¨
```

Data Structures

- struct Graph_Params

Functions

- void show_vertex_pairs (Graph_Params gp)
- Graph_Params set_rand_gnp_params (unsigned short n, double p_edge)
- void show_rand_gnp (Graph_Params gp)
- void construct_rand_gnp (Graph_Params gp)
- void destroy_rand_gnp (Graph_Params gp)
- void dfs (Graph_Params gp, unsigned short v, bool *visited)
- unsigned short count_connected_components (Graph_Params gp)
- bool is_connected (Graph_Params gp)
- unsigned long long A006125_total (unsigned short n)
- unsigned long long A001187_conn (unsigned short n)
- unsigned long long A054592_disconn (unsigned short n)
- double prob_conn (unsigned short n, double p_edge)

4.4.1 Detailed Description

Library for enumerating and calculating the probability of connectedness of random labeled undirected graphs constructed with the Erdős-Rényi and Gilbert models.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2023-08-28

Copyright

Copyright (c) 2023

4.4.2 Function Documentation

4.4.2.1 show_vertex_pairs()

```
void show_vertex_pairs (
            Graph_Params gp )
```

Print out all possible vertex pairs.

Parameters

gp	Graph_Params

Implementation notes:

1. Vertex pair E_i is stored in one unsigned short allocation, thus extracting high part for u and low part for v.
2. Time complexity: $\mathcal{O}(m_{max})$.

4.4.2.2 set_rand_gnp_params()

```
Graph_Params set_rand_gnp_params (
            unsigned short n,
            double p_edge )
```

Populate and return graph parameters as a structure.

Parameters

n	Graph_Params::n
p_edge	Graph_Params::p_edge

Returns

Graph parameters as per the structure Graph_Params

Implementation notes:

1. Integer overflow unsafe, since the rhs's size is not checked before assignment.
2. Null pointer unsafe, since `calloc()` might fail, but this is not checked to reduce running time.
3. Time complexity: $\max[\mathcal{O}(\texttt{binom()}), \mathcal{O}(\texttt{pow()})]$.

4.4.2.3 show_rand_gnp()

```
void show_rand_gnp (
            Graph_Params gp )
```

Print out graph adjacencies for each vertex (row by row).

Parameters

gp	Graph_Params

Implementation notes:

1. Time complexity: $\mathcal{O}(n^2)$.

4.4.2.4 construct_rand_gnp()

```
void construct_rand_gnp (
            Graph_Params gp )
```

Construct a random $\mathcal{G}(n, p)$ graph with n vertices and pm_{max} edges (on average).

Parameters

gp	Graph_Params

Implementation notes:

1. Vertex pair E_i is stored in one unsigned short allocation, thus extracting high part for u and low part for v.

2. Time complexity: $\mathcal{O}(m_{max}$ `bernoulli_rv()`).

Usage notes:

1. `destroy_rand_gnp()` must be called immediately after the created graph is no longer used to clean graphs's adjacencies arrays.

4.4.2.5 destroy_rand_gnp()

```
void destroy_rand_gnp (
            Graph_Params gp )
```

Destroy a random graph – clean graph's adjacencies arrays.

Parameters

gp	Graph_Params

Implementation notes:

1. Time complexity: $\mathcal{O}(n$ `memset()`).

Usage notes:

1. `destroy_rand_gnp()` must be called immediately after the created graph is no longer used to clean graph's adjacencies arrays.

4.4.2.6 dfs()

```
void dfs (
            Graph_Params gp,
            unsigned short v,
            bool * visited )
```

Visit every vertex once.

Parameters

gp	Graph_Params
v	Current vertex
visited	Array of visited vertices

Implementation notes:

1. Recursive Depth-first search (DFS) algorithm.
2. Time complexity: $\mathcal{O}(|V| + |E|)$.

4.4.2.7 count_connected_components()

```
unsigned short count_connected_components (
            Graph_Params gp )
```

Count the number of connected components in a graph.

Parameters

gp	Graph_Params

Returns

Number of connected components

Implementation notes:

1. The algorithm counts the number of connected components by employing `dfs()`.
2. Time complexity: $\mathcal{O}(|V| + |E|)$.

4.4.2.8 is_connected()

```
bool is_connected (
            Graph_Params gp )
```

Check if a graph is connected by counting the number of connected components in it.

Parameters

gp	Graph_Params

Returns

`True (1)` if G has exactly one connected component, i.e. it is connected, `false (0)` otherwise

Implementation notes:

1. The algorithm first counts the number of connected components by employing `count_connected_components()` and then returns `true (1)`, if there is only one connected component in the graph, `false (0)` otherwise.
2. Time complexity: $\mathcal{O}(|V| + |E|)$.

4.4.2.9 A006125_total()

```
unsigned long long A006125_total (
            unsigned short n )
```

Find the total number of labeled graphs (connected and disconnected) with n nodes – sequence $A006125$ [5] .

Parameters

n	Graph order – Graph_Params::n

Returns

Number of labeled graphs

Implementation notes:

1. The number of labeled graphs (connected and disconnected) with n nodes is Graph_Params::n_graphs.
2. Integer overflow unsafe, since the result's size is not checked before return.
3. Time complexity: $\max[\mathcal{O}(\texttt{ binom() }), \mathcal{O}(\texttt{ pow() })]$.

Results for $n \in [0, 11]$:

n	A006125(n)
0	1
1	1
2	2
3	8
4	64
5	1 024
6	32 768
7	2 097 152
8	268 435 456
9	68 719 476 736
10	35 184 372 088 832
11	36 028 797 018 963 968

4.4.2.10 A001187_conn()

```
unsigned long long A001187_conn (
            unsigned short n )
```

Find the number of connected labeled graphs with n nodes constructed with the Erdős–Rényi model $\mathcal{G}(n, M)$ – sequence $A001187$ [6] .

Parameters

n	Graph order – Graph_Params::n

Returns

Number of connected labeled graphs

Implementation notes:

1. In this model each graph is chosen randomly with equal probability of $1/n_{graphs}$, where n_{graphs} is Graph_Params::n_graphs.
2. The number C_n of labeled connected graphs of order n is given by the recursive formula (1.2.1), p. 7, Harary [2] ; p was substituted with n to avoid confusion with notation of probability.

$$C_n = 2^{\binom{n}{2}} - \frac{1}{n}\sum_{k=1}^{n-1} k\binom{n}{k} 2^{\binom{n-k}{2}} C_k.$$

3. Integer overflow unsafe, since the result's size is not checked during calculation and before return.
4. Time complexity: $\mathcal{O}(2^{\ (n \max[\mathcal{O}(\texttt{ binom() }), \mathcal{O}(\texttt{ pow() })] - 1)\ })$.

Results for $n \in [0, 11]$:

n	A001187(n)
0	1
1	1
2	1
3	4
4	38
5	728
6	26 704
7	1 866 256
8	251 548 592
9	66 296 291 072
10	34 496 488 594 816
11	35 641 657 548 953 344

4.4.2.11 A054592_disconn()

```
unsigned long long A054592_disconn (
            unsigned short n )
```

Find the number of disconnected labeled graphs with n nodes – sequence $A054592$ [7] .

Parameters

n	Graph order – Graph_Params::n

Returns

Number of disconnected labeled graphs

Implementation notes:

1. Number of labeled graphs (connected and disconnected) with n nodes is sequence $A006125$ [5] . Number of connected labeled graphs with n nodes is sequence $A001187$ [6] . Thus, the number of disconnected labeled graphs with n nodes is simply $A054592(n) = A006125(n) - A001187(n)$.
2. Integer overflow unsafe, since the result's size is not checked during calculation and before return.
3. Time complexity: $\max(\mathcal{O}(\texttt{ A006125\_total() }), \mathcal{O}(\texttt{ A001187\_conn() }))$.

Results for $n \in [0, 11]$:

n	A054592(n)
0	0
1	0
2	1
3	4
4	26
5	296
6	6 064
7	230 896
8	16 886 864
9	2 423 185 664
10	687 883 494 016
11	387 139 470 010 624

4.4.2.12 prob_conn()

```
double prob_conn (
            unsigned short n,
            double p_edge )
```

Find the probability of connectedness of a random labeled graph constructed with the Gilbert model $\mathcal{G}(n,p)$.

Parameters

n	Graph order – Graph_Params::n
p_edge	Edge probability Graph_Params::p_edge

Returns

$P_n = P(\mathcal{G}(n,p)$ is connected$)$

Implementation notes:

1. In this model every possible edge occurs independently with probability p. The probability of obtaining any one particular random graph with m edges is $p^m(1-p)^{N-m}$, where N is Graph_Params::m_max.
2. The probability of connectedness of a random labeled graph is given by the recursive formula (3), p. 2, Gilbert [3] ; q was substituted by $1-p$ to avoid introducing unnecessary new variable; N was substituted by n to avoid confusion with N above.
$$P_n = 1 - \sum_{k=1}^{n-1} \binom{n-1}{k-1} (1-p)^{k(n-k)} P_k.$$
3. Integer overflow unsafe, since the result's size is not checked during calculation and before return.
4. Time complexity: $\mathcal{O}(2^{(n \max[\mathcal{O}(\texttt{binom()}), \mathcal{O}(\texttt{pow()})] - 1)})$.

Results for P_n for $n \in [2,11]$ and $p \in [0.1, 0.9]$:

n / p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
2	0.10000	0.20000	0.30000	0.40000	0.50000	0.60000	0.70000	0.80000	0.90000
3	0.02800	0.10400	0.21600	0.35200	0.50000	0.64800	0.78400	0.89600	0.97200

n / p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
4	0.01293	0.08250	0.21865	0.40038	0.59375	0.76550	0.89249	0.96666	0.99581
5	0.00810	0.08195	0.25626	0.48965	0.71094	0.87026	0.95751	0.99166	0.99949
6	0.00621 †	0.09230	0.31690	0.59555	0.81494 ‡	0.93652	0.98497	0.99805	0.99994
7	0.00551	0.11127	0.39385	0.69878	0.88990	0.97072	0.99484	0.99955	0.99999
8	0.00541	0.13851	0.47987	0.78627	0.93709	0.98677	0.99824	0.99990	1.00000
9	0.00574	0.17396	0.56714	0.85325	0.96474	0.99408	0.99941	0.99998	1.00000
10	0.00644	0.21723	0.64897	0.90128	0.98045	0.99737	0.99980	0.99999	1.00000
11	0.00752	0.26729	0.72107	0.93445	0.98925	0.99885	0.99994	1.00000	1.00000

† Table 1, p. 2, Gilbert [3] : 0.00624.

‡ Table 1, p. 2, Gilbert [3] : 0.81569.

4.5 C:/Dropbox_full_copy/HSE/MDS/Study/Year_1/cython/proj/src/stats_lib.c File Reference

```
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include ¨utils_lib.h¨
```

Macros

- #define NON_SUCCESS false
- #define SUCCESS true

Functions

- unsigned long long binom (unsigned short n, unsigned short k)
- bool bernoulli_rv (double p)
- double mean (double ∗array, unsigned short length)
- double max (double ∗array, unsigned short length)
- double min (double ∗array, unsigned short length)
- double quantile (double ∗array, unsigned short length, double q)

4.5.1 Detailed Description

Library for mathematical statistics functions.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2023-08-28

Copyright

Copyright (c) 2023

4.5.2 Macro Definition Documentation

4.5.2.1 NON_SUCCESS

```
#define NON_SUCCESS false
```

Macro for `bernoulli_rv()` non-success return value.

4.5.2.2 SUCCESS

```
#define SUCCESS true
```

Macro for `bernoulli_rv()` success return value.

4.5.3 Function Documentation

4.5.3.1 binom()

```
unsigned long long binom (
            unsigned short n,
            unsigned short k )
```

Calculate binomial coefficient $C(n, k)$.

Parameters

n	Integer number n
k	Integer number k

Returns

Binomial coefficient

Implementation notes:

1. Optimized algorithm without explicit calculation of factorial.
2. Integer overflow unsafe, since the result size is not checked during calculation and before return.
3. Time complexity: $\mathcal{O}(r)$, where $r = \min(k, n - k)$.

4.5.3.2 bernoulli_rv()

```
bool bernoulli_rv (
            double p )
```

Return Bernoulli distributed random variable.

Parameters

p	Probability of a single outcome

Returns

`True (1)` or `false (0)`

Implementation notes:

1. Simulates Bernoulli process and returns either 1 or 0. If $p = 0.5$, returned values will simulate tosses of a fair coin (H = 1, T = 0).
2. First uniformly distributed random variable with `p = 1 / RAND_MAX` is obtained with `rand()` which is then transformed into Bernoulli distributed random variable.
3. Assumes that `RAND_MAX` might be at least 16 bit, thus the result of `rand()` is stored in the unsigned int variable.
4. Time complexity: $\mathcal{O}($ `rand()` $)$.

Usage notes:

1. Init random seed by calling `srand(time(NULL))`, if needed, before calling `bernoulli_rv()`.

4.5.3.3 mean()

```
double mean (
            double * array,
            unsigned short length )
```

Find mean of an array.

Parameters

array	Array of real numbers
length	Length of the array

Returns

Arithmetic mean of the array's values

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Time complexity: $\mathcal{O}(length)$.

4.5.3.4 max()

```
double max (
            double * array,
            unsigned short length )
```

Find maximal value in an array.

Parameters

array	Array of real numbers
length	Length of the array

Returns

Arithmetic maximum of the array's values

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Time complexity: $\mathcal{O}(length)$.

4.5.3.5 min()

```
double min (
            double * array,
            unsigned short length )
```

Find minimal value in an array.

Parameters

array	Array of real numbers
length	Length of the array

Returns

Arithmetic minimum of the array's values

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Time complexity: $\mathcal{O}(length)$.

4.5.3.6 quantile()

```
double quantile (
            double * array,
            unsigned short length,
            double q )
```

Find q-th quantile of an array.

Parameters

array	Array of real numbers
length	Length of the array
q	Quantile $\in [0, 1]$

Returns

Array's value corresponding to q

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Input array is first copied to a new array, so that the original array is not corrupted, and then the copy is sorted in ascending order in place by employing the `sort_array()` function.
3. If array's length is even, the arithmetic mean (average) of two neighbor elements is returned. Example: median $(q = 0.5)$ for the array `{1, 2, 3, 4}` is $(2 + 3)/2 = 2.5$.
4. Time complexity: $\max(\mathcal{O}(length), \mathcal{O}($ `sort_array()` $))$.

4.6 C:/Dropbox_full_copy/HSE/MDS/Study/Year_1/cython/proj/src/utils_lib.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
```

Functions

- void show_array (double *array, unsigned short length, char sep, bool wide)
- int comp_func_asc (const void *a, const void *b)
- int comp_func_desc (const void *a, const void *b)
- void sort_array (double *array, unsigned short length, bool ascending)

4.6.1 Detailed Description

Library for various handy functions.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2023-08-28

Copyright

Copyright (c) 2023

4.6.2 Function Documentation

4.6.2.1 show_array()

```
void show_array (
            double * array,
            unsigned short length,
            char sep,
            bool wide )
```

Print out an array.

Parameters

array	Array of real numbers
length	Length of the array
sep	Character to separate array's elements
wide	Display style: `true` – shape $(1 \times length)$ (wide), `false` – shape $(length \times 1)$ (tall).

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Display precision cannot be set – as per `printf()`, by default 6 decimal places are printed.
3. Time complexity: $\mathcal{O}(length)$.

4.6.2.2 comp_func_asc()

```
int comp_func_asc (
            const void * a,
            const void * b )
```

Compare function for ascending `qsort()` used in `sort_array()`.

Parameters

a	The first value to compare
b	The second value to compare

Returns

Sort flag: `0 (a == b), 1 (a > b), -1 (a < b)`

4.6.2.3 comp_func_desc()

```
int comp_func_desc (
            const void * a,
            const void * b )
```

Compare function for descending `qsort()` used in `sort_array()`.

Parameters

a	The first value to compare
b	The second value to compare

Returns

Sort flag: `0 (a == b), 1 (a < b), -1 (a > b)`

4.6.2.4 sort_array()

```
void sort_array (
            double * array,
            unsigned short length,
            bool ascending )
```

Sort array's elements either in ascending or descending order.

Parameters

array	Array of real numbers
length	Length of the array
ascending	If `true`, sort in ascending order, otherwise in descending order

Implementation notes:

1. Array's length is limited by `unsigned short` size, i.e. most likely 65536 elements max.
2. Employs `qsort()` with `comp_func_asc()` and `comp_func_desc()`.
3. Time complexity: might be $\mathcal{O}(length \log(length))$, but in fact is not guaranteed [1].

Bibliography

[1] cppreference.com. *qsort, qsort_s*. cppreference.com, 2022. 21, 36

[2] Frank Harary, Edgar M. Palmer. *Graphical Enumeration*. Academic Press, 1973. 12, 27

[3] E. N. Gilbert. *Random Graphs*. Bell Telephone Laboratories, Inc., 1959. 14, 28, 29

[4] P. Erdős, A. Rényi. *On Random Graphs*. Budapest, 1959. 6

[5] N. J. A. Sloane. *a(n) = 2 ^n*(n-1)/2) (Formerly M1897)*. The On-Line Encyclopedia of Integer Sequences, 1991. 11, 13, 25, 27

[6] N. J. A. Sloane. *Number of connected labeled graphs with n nodes*. The On-Line Encyclopedia of Integer Sequences, 1991. 12, 13, 26, 27

[7] N. J. A. Sloane. *Number of disconnected labeled graphs with n nodes*. The On-Line Encyclopedia of Integer Sequences, 2000. 13, 27

Index