**Генератор модели графа Эрдеша–Реньи**

horizontal line

# Описание

Задается количество вершин графа V, далее по схеме Бернулли с заданной вероятностью p каждое ребро независимо от других включается в граф. Граф хранится в виде списка ребер, src\_ids[] и dst\_ids[] - массивы, содержащие вершины начала и конца ребра в соответствующих индексах. Далее в конструкторе класса Graph есть двойной цикл, который имитирует обход половины матрицы смежности. Если срабатывает случайный генератор, то ребро включается в список.

## Свойства

Ссылаясь на [статью МФТИ](https://mipt.ru/upload/30d/Pages_130-140_from_Trud-8-14-arphcxl1tgs.pdf), можно проверить некоторые свойства графа. Для того, чтобы граф на выходе получил конкретные свойства, нужно задать функцию вероятности от количества вершин p(V). Примеры графов с заданными функциями вероятности прилагаются внизу.

1. P = c \* ln(V) / V. Если c > 1, то почти всегда случайный граф связен. Если c < 1, то почти всегда случайный граф не является связным.
2. P = c / V . Если c < 1, то все связные компоненты графа, скорее всего, крошечные — имеющие логарифмический от общего числа вершин размер. Если же c > 1, то, скорее всего, найдется компонента с числом вершин порядка n. Такая компонента называется гигантской.

## 

## 

## Интерфейс функции void graph\_generator\_reny (int \*src\_ids, int \*dst\_ids, int V, long long \*E, double prob)

Массивы src\_ids и dst\_ids используются для хранения списка ребер. V- количество вершин. Так как функция должна быть void, то количество ребер возвращается через указатель E. Prob - заданная вероятность. Функция вызывает конструктор Graph, возвращает количество рёбер по ссылке и печатает время генерации списка ребер и их количество.

## Параллелизм

В конструкторе Graph при помощи #pragma omp **for** ускоряется работа двойного цикла: каждая нить проходит строку в матрице смежности. Использовались 128 нитей  
**Время генерации графа (generation time в печати) при параллельной обработке и последовательной.**

|  |  |  |
| --- | --- | --- |
| Vertex and Probability | Parallel | Sequential |
| 250 0.01 | 0.021443 | 0.002873 |
| 1000 0.01 | 0.050880 | 0.089670 |
| 10000 0.01 | 2.560025 | 5.203905 |
| 30000 0.01 | 23.523339 | 52.367126 |
| 40000 0.01 | 40.265438 | 85.351777 |

## При небольшом количестве вершин параллелизм уступает последовательной обработке. Я предполагаю, что это связано с накладными расходами по запуску нитей и смене контекстов. Чем больше вершин у графа, тем эффективнее выглядит параллельная обработка. Примеры графов

Параметры компиляции: g++ -Wall Reny.cpp -o Reny -lcgraph -lgvc -fopenmp  
1.png - ./Reny 250 0.01 (первый рисунок внизу)  
2.png - ./Reny 50 0.1  
3.png - граф с probability = 0.5 \* log(V) / V; Вызов ./Reny 50 0.01; Последний параметр не используется, может быть любым. Граф получился не связным. (второй рисунок внизу)  
4.png - граф с probability = 1.2 \* log(V) / V; Вызов ./Reny 50 0.01; Граф получился связным. (третий рисунок снизу)  
5.png - граф с probability = 0.3 / V; Вызов ./Reny 200 0.01; Все компоненты графа достаточно маленькие.  
6.png - граф с probability = 1.1 / V; Вызов ./Reny 200 0.01; Присутствуют большие  
компоненты связности.





