УДК 519.1 https://doi.org/10.33619/2414-2948/67/01

АВТОМАТИЧЕСКАЯ ГЕНЕРАЦИЯ ГРАФОВ ДЛЯ ЭЛЕКТРОННЫХ ОБУЧАЮЩИХ СИСТЕМ

©Зайнуллина Р. Ф., *ORCID: 0000-0002-8027-3767, Казанский национальный исследовательский технический университет им. А.Н. Туполева - КАИ, г. Казань, Россия, ananasfeer@gmail.com*

AUTOMATIC GRAPH GENERATION FOR E-LEARNING SYSTEMS

©Zainullina R., *ORCID: 0000-0002-8027-3767, Kazan National Research Technical University named after A. N. Tupolev - KAI, Kazan, Russia, ananasfeer@gmail.com*

Аннотация. Предметом исследования является один из способов актуализации современных обучающих систем решения задач теории графов, а именно, автоматическая генерация графов. Такой подход позволит уменьшить нагрузку на базу данных обучающей системы, и без обновления банка задач в реальном времени генерировать для пользователя задачи. В ходе работы были выявлены преимущества и недостатки такого подхода. Выбран наиболее подходящий для реализации исследования способ представления графов в электронных вычислительных машинах. Выявлены и обоснованы требования к генерируемым графам и возможные способы реализации этих требований. А именно: в реализуемой программе будут генерироваться простые связные неориентированные графы. Рассмотрели важную деталь в работе с графами — обход графа при помощи алгоритма «Поиск в глубину (ширину)», в данной задаче используемый для проверки графа на связность. Приведен результат работы — программная реализация алгоритма генерации графа на языке программирования C#. В ней графы представляются списком смежности, генерируются случайно и проверяются на связность при помощи функции DFS (Depth First Search). Функция DFS является программной реализацией алгоритма «Поиск в глубину».

Abstract. The subject of the research is one of the ways of updating modern training systems for solving problems of graph theory, namely, automatic generation of graphs. This approach will reduce the load on the training system database and generate tasks for the user in real-time without updating the bank of tasks. In the course of the work, the advantages and disadvantages of this approach were identified. The most suitable method for the implementation of the research was chosen to represent graphs in electronic computers. The requirements for generated graphs and possible ways of implementing these requirements are identified and substantiated. Namely: in the implemented program, simple connected undirected graphs will be generated. We considered an important detail in working with graphs — graph traversal using the “Depth (width) search” algorithm, which in this task is used to check the graph for connectivity. The result of the work is presented — a software implementation of the graph generation algorithm in the C# programming language. In it, graphs are represented by an adjacency list, generated randomly, and checked for connectivity using the DFS (Depth First Search) function. DFS is a software implementation of the Depth First Search algorithm.

Т. 7. №6. 2021
https://doi.org/10.33619/2414-2948/67

Ключевые слова: теория графов, обучающая система, поиск в глубину, связность графов, списки смежности.

Keywords: graph theory, training system, depth-first search, graph connectivity, adjacency lists.

В связи с пандемией коронавируса возросла потребность в автоматизированных системах обучения, так как они позволяют расширить возможности дистанционного образования. Одними из их достоинств являются: осуществление удаленного доступа, самостоятельное изучения материала студентами, а также упрощение процесса проверки знаний преподавателем. Однако информационная сфера требует постоянного развития, поэтому обучающие системы необходимо совершенствовать, добавляя в них новые возможности и повышая уровень комфорта использования, стараясь при этом использовать ресурсы как можно более оптимально. В этой работе рассматривается один из важных аспектов разрабатываемой обучающей системы решения некоторых задач теории графов, а именно: генерация графов. Автоматизированные системы обучения, нацеленные на решение задач, связанных с графами, не являются новшеством в современном мире. Нередко в таких системах задания выбираются из некоторого ограниченного набора, хранящегося в банке задач. Данный подход имеет ряд недостатков [1–2].

1. Большое количество задач нагружает базу данных системы, что приводит к долгому выполнению приложения.

2. Для поддержания актуальности банка заданий преподавателю необходимо постоянно его обновлять.

3. При малом объеме банка задач велика вероятность повторения заданий, что приводит к неточным результатам тестирования знаний студентов.

По сравнению с методом банка задач метод генерации графов имеет следующие преимущества:

1. Большее количество вариантов задач.

2. Снижение нагрузки с преподавателей, так как генерацией графов занимается сама система.

3. Снижения нагрузки на базу данных, так как задания генерируются непосредственно перед решением, а не хранятся в системе заранее.

Тем не менее существует ряд особенностей, которые необходимо учитывать при генерации графов. В рамках разрабатываемой обучающей системы первоначально реализуется отработка решения задач раскраски графов. Чаще всего при этом используются неориентированные связные графы без петель и кратных ребер, иными словами — простые связные графы. Поэтому рассмотрим генерацию графов именно такого вида. При составлении и решении задач на вычислительных машинах графы представляют дискретным способом. Таких способов существует большое количество и от выбора способа сильно зависит эффективность алгоритма генерации графов. Среди дискретных представлений графов наиболее популярными и используемыми являются матричные. В виде матрицы смежности, либо матрицы инцидентности. Кроме того, существует не матричные способы представления, но не менее удобные в некоторых случаях. Одним из таких, являются списки смежности. В этом способе граф представляется с помощью списочной структуры, которая определяет смежность вершин и состоит из массива указателей на списки смежных вершин. Такой способ представления в общем случае отличается более оптимальным использованием

Т. 7. №6. 2021
https://doi.org/10.33619/2414-2948/67

памяти. Для неориентированных графов, содержащих p вершин и q ребер, объем используемой памяти составляет O(p+2q) [1, с. 244].

Также стоит вопрос: как можно распределить генерируемые задачи по сложности? Наиболее простой способ — распределение по количеству вершин. Такой способ, возможно, не даст высокий уровень точности, но и не нагрузит систему. Будем генерировать графы трех уровней сложности:

1. простые — от 5 до 7 вершин;
2. средние — от 8 до 10;
3. сложные — от 11 до 13.

Необходимо учитывать, что генерируемый граф должен быть связным. Для обеспечения этого условия будем придерживаться следующего алгоритма.

1. Генерируются случайные непустые списки смежности графа с заданным числом вершин.

2. Выполняется проверка на связность графа. В том случае, если граф не связен, осуществляется переход к пункту 1.

Осталось определиться со способом проверки графа на связность. Существует теорема, что если граф связен и конечен, то поиск в ширину и поиск в глубину обойдут все вершины по одному разу [1, с. 246].

Также одно из следствий этой теоремы утверждает, что поиск в ширину в среднем вдвое медленнее, чем поиск в глубину [1, с. 247].

Таким образом, для проверки связности графа будем использовать метод поиска в глубину. В процессе обхода графа будем помечать вершины, которые уже были рассмотрены. Если по завершении обхода не останется ни одной не помеченной вершины, граф является связным. Удобным представлением графов при реализации алгоритмов обхода графов являются списки смежности. С учетом требований к генерируемым графам, отмеченным выше, был написан программный код на языке C# для решения поставленной задачи.

Для представления графа был создан класс Graph (Рисунок 1). В переменной Vertex будем хранить вершину графа, а в VertexList — список смежных ей вершин.

```
public class Graph
{
    public int Vertex { get; set; }
    public List<int> VertexList { get; set; }
}
```

Рисунок 1. Класс Graph

Следующая часть кода отвечает за генерацию случайного графа (Рисунок 2). Список смежности графа будем хранить в списке graph. Для каждой из вершин (количество вершин в данном примере задано фиксированно для удобства тестирования и демонстрации) генерируем случайный непустой список смежных вершин. Здесь же проверяем, чтобы в списке не было вершины, для которой он составляется, т. е. избегаем петель. После того, как граф сгенерирован, остается проверить его на связность. Реализация алгоритма обхода в глубину продемонстрирована на Рисунках 3–4.

Т. 7. №6. 2021
https://doi.org/10.33619/2414-2948/67

```
Random rnd = new Random();

int p;
p = 10;
bool connected = false;

List<Graph> graph = new List<Graph> (p);
while (!connected)
{
    for (int i = 1; i <= p; i++)
    {
        int count = rnd.Next(1, p - 3);
        List<int> vertexList = new List<int>();
        for (int k = 0; k < count;)
        {
            int value = rnd.Next(1, p);
            if (value != 0 && value != i && !vertexList.Contains(value))
            {
                vertexList.Add(value);
                k++;
            }
        }
        graph.Add(new Graph() { Vertex = i, VertexList = vertexList });
    }

    foreach (var g in graph)
    {
        foreach (var vert in g.VertexList)
        {
            if (!graph[vert - 1].VertexList.Contains(g.Vertex))
            {
                graph[vert - 1].VertexList.Add(g.Vertex);
            }
        }
    }
```

Рисунок 2. Генерация графа

```
bool[] check = new bool[p+1];
check[0] = true;

DFS(graph, 1, check);

if (!check.Contains(false)) connected = true;
else graph.Clear();
```

Рисунок 3. Проверка графа на связность

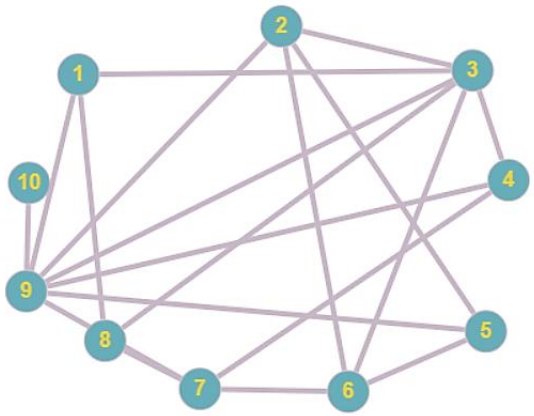
```
static public void DFS (List<Graph> graph, int start, bool[] check)
{
    check[start] = true;

    foreach (var u in graph[start-1].VertexList)
    {
        if (!check[u])
        {
            DFS(graph, u, check);
        }
    }
}
```

Рисунок 4. Функция DFS – Обход графа в глубину

Функция DFS — рекурсивная функция, реализующая алгоритм обхода графа в глубину. В случае, если граф не связный, программа генерирует новый граф и проверяет на связность уже его. Результат работы программы продемонстрирован на Рисунке 5а — это составленный список смежности графа, удовлетворяющего поставленным требованиям. Для наглядности приведена также диаграмма этого графа на Рисунке 5б.

```
1 :  8 3 9
2 :  6 3 9
3 :  8 6 2 1 4 9
4 :  3 7 9
5 :  9 6
6 :  5 7 2 3
7 :  8 9 4 6
8 :  7 1 3
9 :  2 3 7 1 4 5 10
10 :  9
```



а) список смежности б) диаграмма

Рисунок 5. Сгенерированный граф

Список литературы:

1. Новиков Ф. А. Дискретная математика для программистов. СПб.: Питер, 2009. 384 с.
2. Иванов Б. Н. Дискретная математика: Алгоритмы и программы. М.: Лаб. базовых знаний, 2001. 288 с.

References:

1. Novikov, F. A. (2009). Diskretnaya matematika dlya programmistov. St. Petersburg. (in Russian).
2. Ivanov, B. N. (2001). Diskretnaya matematika: Algoritmy i programmy. Moscow. (in Russian).

Работа поступила
в редакцию 12.05.2021 г.

Принята к публикации
18.05.2021 г.

Ссылка для цитирования:
Зайнуллина Р. Ф. Автоматическая генерация графов для электронных обучающих систем // Бюллетень науки и практики. 2021. Т. 7. №6. С. 12-16. https://doi.org/10.33619/2414-2948/67/01

Cite as (APA):
Zainullina, R. (2021). Automatic Graph Generation for E-learning Systems. *Bulletin of Science and Practice, 7*(6), 12-16. (in Russian). https://doi.org/10.33619/2414-2948/67/01