Домашная работа по дискретной математике.

TeX Task

Задача о независимом множестве на дереве. Задача о построении матрицы кратчайших расстояний. Метод Флойда-Уоршелла.

Author: Gagin Artur, group 172, SE SPbU '25

Date: Second semester, 2022

1. Задача о независимом множестве на дереве.

Задание:

Найти максимальное независимое множество вершин на дереве (с наибольшим количеством элементов в нем).

Решение:

Перед решением быстренько вспомним определение дерева в теории графов, а также смысл независимого множества вершин (НМВ далее).

Итак, **дерево** — это связный ациклический граф. Связность в данном случае означает наличие маршрута между любой парой вершин. Ацикличность означает отсутствие циклов. **Множество вершин графа** называется независимым тогда и только тогда, когда никакие две вершины множества не соединены ребром. Теперь, вспомнив эти определения, задача стала совсем ясна.

Попробуем разработать алгоритм, получающий на вход дерево T (с корнем r), который возвращает мощность наибольшего МНВ.

Будем решать задачу, двигаясь по дереву снизу вверх. Сначала будем рассматривать множества, состоящие из одних только листьев, постепенно поднимаясь вверх, увеличивая мощности рассматриваемых множеств.

Заметим, что нам недостаточно иметь значение максимального независимого множества вершин в каком-либо поддереве, решая задачу расширением (добавлением выше находящихся узлов), поскольку когда мы добавим новый узел (стоящий выше), нам важно знать, можем ли мы его включить, либо же не можем. Для этого можно рассматривать два

независимых множества: одно будет включать вершину рассматриваего нами поддерева, а другое не будет. Таким образом, когда мы будем добавлять вершину, рассматривая новую совокупность вершин, более большую по количеству, мы однозначно сможем составить независимые множества вершин уже из большего по количеству вершин дерева.

Взглянем на пример:

а) б)

Рис. 1: Рисунок 1 и 2.

Будем обозначать за Q_I^+ мощность максимального НМВ поддерева, вершина (корень) которого – узел I, а узлы – все те, которые находятся глубже вершины I, при этом вершина I обязательно включена в это максимальное НМВ. За Q_I^- будем обозначать практически тоже самое, но вершина I обязательно не включена в это максимальное НМВ. Теперь, когда мы захотим рассмотреть новое дерево, вершина которого будет являться родителем вершины I, мы однозначно сможем найти мощность МНВ (максимальную).

Попытаемся написать рекурсивную формулу, за основу взяв ситуации, когда вершина является листом.

Очевидно, что если наше множество состоит из одной вершины, то $Q_I^+ = 1$ (сама вершина) и $Q_I^- = 0$ (здесь стоит отметить, что пустое множество тоже является НМВ). Тогда формулы приобретают следующий вид:

$$Q_I^+ = \begin{cases} 1 & , I \text{ является листом} \\ ... & , I \text{ не является листом} \end{cases}$$

$$Q_I^- = \begin{cases} 0 & , I \text{ является листом} \\ ... & , I \text{ не является листом} \end{cases}$$

Хорошо, но что насчет ситуаций, когда I не является листом? Попробуем разобраться на примере.

Рис. 2: Рисунок 3 и 4.

Q_I^+: это максимальное НМВ включает саму вершину I, поэтому мы ее считаем. Далее, нам надо взять максимальное МНВ слева (от вершины A). Так как вершины не должны быть связаны, то это будет Q_A^-. По такой же логике берем Q_B^- и Q_C^-. Сложив их и прибавив к единице, получим искомую максимальную мощность МНВ поддерева, вершина которого – I.

По похожей логике вычисляется Q_I^-: здесь мы единичку прибавлять не будем, ведь мы не включаем саму вершину I. Также, так как вершина не включена, мы имеем возможность взять как, например, Q_A^+, так и, например, Q_A^-. Поскольку мы ищем максимальное МНВ, то будем брать максимум между этими двумя мощностями.

Итак, завершая данную логику, приходим к выводу, что наши формулы будут выглядеть следующим образом:

$$Q_I^+ = \begin{cases} 1 & , I \text{ является листом} \\ 1 + \sum_{i \in child(I)} Q_i^- & , I \text{ не является листом} \end{cases}$$

$$Q_I^- = \begin{cases} 0 & , I \text{ является листом} \\ \sum_{i \in child(I)} max\{Q_i^+, Q_i^-\} & , I \text{ не является листом} \end{cases}$$

Дойдя до корня, у нас будет пара чисел: Q_R^+ и Q_R^-. Максимальное из них и будет являться ответом на поставленную задачу.

Оценим сложность работы алгоритма: легко заметить, что мы проходим по всем вершинам, при этом ребро обрабатывается один или два раза.

Таким образом: $O(|V| + |E|)$ (отбросив const).

Попробуем решить данную задачу на искусственном примере (Рисунок 5) для того, чтобы еще точнее понять, как работает алгоритм, а также для того, чтобы убедиться, что логика, построенная нами, верна.

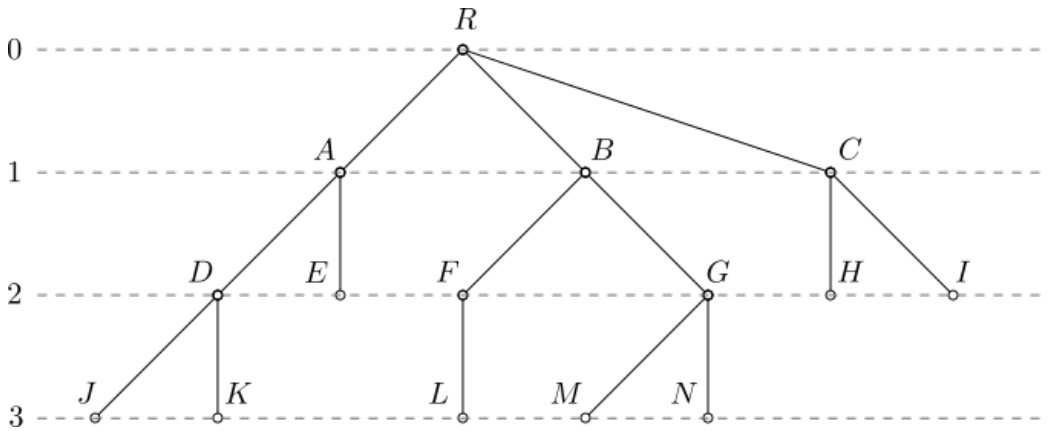

Рис. 3: Рисунок 5.

Использовав наивный алгоритм (перебрав все возможные множества), легко понять, что мощность максимального МНВ равна 9.

Решим пример с помощью наших формул. Начнем с глубины 3:

$Q_J^- = 0$, $Q_J^+ = 1$;

$Q_K^- = 0$, $Q_K^+ = 1$;

$Q_L^- = 0$, $Q_L^+ = 1$;

$Q_M^- = 0$, $Q_M^+ = 1$;

$Q_N^- = 0$, $Q_N^+ = 1$;

Глубина 2:

$Q_D^- = max\{Q_J^- = 0,\ Q_J^+ = 1\} + max\{Q_K^- = 0,\ Q_K^+ = 1\} = 2$, $Q_D^+ = 1 + Q_J^- + Q_K^- = 1 + 0 + 0 = 1$;

$Q_E^- = 0$, $Q_E^+ = 1$;

$Q_F^- = 1$, $Q_F^+ = 1 + 0 = 1$;

$Q_G^- = 2$, $Q_G^+ = 1$ (аналогично вершине D).

$Q_H^- = 0$, $Q_H^+ = 1$;

$Q_I^- = 0$, $Q_I^+ = 1$;

Глубина 1:

$Q_A^- = max\{Q_D^- = 2, Q_D^+ = 1\} + max\{Q_E^- = 0, Q_E^+ = 1\} = 2 + 1 = 3$, $Q_A^+ = 1 + Q_D^- + Q_E^- = 1 + 2 + 0 = 3$;

$Q_B^- = max\{Q_F^- = 1, Q_F^+ = 1\} + max\{Q_G^- = 2, Q_G^+ = 1\} = 1 + 2 = 3$, $Q_B^+ = 1 + Q_F^- + Q_G^- = 1 + 1 + 2 = 4$;

$Q_C^- = 2$, $Q_C^+ = 1$;

Глубина 0:

$Q_R^- = max\{Q_A^- = 3, Q_A^+ = 3\} + max\{Q_B^- = 3, Q_B^+ = 4\} + max\{Q_C^- = 2, Q_C^+ = 1\} = 3 + 4 + 2 = 9$, $Q_R^+ = 1 + Q_A^- + Q_B^- + Q_C^- = 1 + 3 + 3 + 2 = 9$.

Ответом будет служить максимальное из чисел, то есть 9. Что мы и получили, сославшись на наивное решение.

Таким образом можем заключить, что проблема решена. Мы составили рекурсивную формулу, которая решает поставленную нами задачу.

2. Задача о построении матрицы кратчайших расстояний. Метод Флойда-Уоршелла.

Задание:

Найти кратчайшие пути для любой пары вершин во взвешенном ориентированном графе (тем самым построив матрицу), воспользовавшись для данного задания методом Флойда-Уоршелла.

Решение:

Перед решением предлагаем вспомнить, что такое взвешенный граф. **Взвешенным графом** называется граф, каждому ребру которого поставлено в соответствие некое значение (вес ребра). **Кратчайшее расстояние между узлами в графе** – такое расстояние, при котором сумма весов составляющих его ребер минимальна. **Алгоритм Флойда — Уоршелла** – это алгоритм поиска кратчайших путей во взвешенном (ориентированном в том числе) графе с положительным или отрицательным весом ребер (но без отрицательных циклов).

Рассмотрим идею данного алгоритма: пусть у нас есть две вершины i и k, а также расстояние из одной вершины в другую (существует возможность попасть из первой вершины в последнюю). Рассмотрим также и вспомогательную вершину j, для которой выполняется следующее условие: из вершины i можно попасть в вершину j, из вершины j можно попасть в вершину k. При этом расстояния из вершины i в вершину j и из вершины j в вершину k не должны равняться бесконечности. Обозначим эти расстояния d_{ij} и d_{jk}. Рассмотрим сумму этих расстояний. Если $d_{ij} + djk < d_{ik}$ (наше первоначальное расстояние), то мы заменяем это первоначальное расстояние в нашей матрице на полученную сумму.

Попробуем реализовать данный алгоритм на языке программирования C#. Данную идею алгоритма можно реализовать следующим образом:

$$\text{d[i, j] = Math.Min(d[i, j], d[i, k] + d[k, j]);}$$

Стоит отметить, что если в графе не существует расстояния из одной вершины в другую, то такое расстояние мы будем отмечать бесконечностью. Расстояние из вершины в себя отмечаем нулем.

Рассмотрим идею на следующем примере:

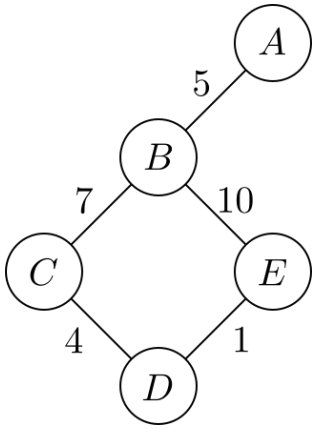

Рисунок 6.

На первом шаге мы просто получаем наш граф на вход. Матрица смежности будет выглядеть следующим образом (вершины в графе пронумерованы):

$$\begin{bmatrix} & (1) & (2) & (3) & (4) & (5) \\ (1) & 0 & 5 & \infty & \infty & \infty \\ (2) & 5 & 0 & 7 & \infty & 10 \\ (3) & \infty & 7 & 0 & 4 & \infty \\ (4) & \infty & \infty & 4 & 0 & 1 \\ (5) & \infty & 10 & \infty & 1 & 0 \end{bmatrix}$$

То есть, на ней отображены расстояния от вершин к близлежащим им вершинам. Стоит отметить, что в данном примере мы рассматриваем

неориентированный граф, поэтому матрица, подаваемая на вход, получилась симметричная. Очевидно, что все следующие действия, проделываемые нами в этом примере, будут работать и для ориентированного графа, просто он перестанет быть симметричным.

Далее, мы разобьем наш алгоритм на шаги, в каждом из которых мы будем рассматривать по вершине. То есть, всего шагов будет столько, сколько вершин в нашем графе. Нам необходимо рассмотреть функцию, которая возвращает кратчайший возможный путь от i до j с использованием вершин только из множества $\{1, 2, \ldots, k\}$ в качестве промежуточных точек на этом пути, где k – номер нашего шага. Перед k-тым шагом мы будем считать, d_{ij} – длина кратчайшего пути из вершины i в вершину j.

Рассмотрим $k = 1$. Идейно, мы рассматриваем вершину с этим номером как дополнительную вершину. Берем каждую пару вершин i и j, а затем смотрим, может ли эта вершина k повлиять на расстояние между этими вершинами, можно ли, пройдя только через вершины $\{1, 2, \ldots, k\}$, как дополнительные, попасть из одной в другую за меньшее расстояние. Из рисунка очевидно, что вершина $A(1)$ никаким образом не способна повлиять на расстояния между какими-либо парами вершин. Таким образом, матрица после первого шага сохраняется (остается прежней).

Рассмотрим $k = 2$. Теперь смотрим, можем ли мы с помощью второй вершины найти новые кратчайшие пути для пар вершин. Очевидно, что это можно сделать. Например, из вершины $A(1)$ в вершину $C(3)$ можно попасть с помощью второй вершины: $(1) \to (3) = (1) \to (2) + (2) \to (3)$. Для пары 1 и 4 это невозможно сделать, поскольку, хоть и первой вершины во вторую у нас существует путь, из второй вершины в четвертую пути нет (бесконечность). Таким же образом рассматриваем оставшиеся пары. То есть, пути из i в k, из k в j. Если их сумма меньше существующего расстояния из i в j, то записываем в матрицу значение суммы. Результат после второго шага выглядит следующим образом:

$$\begin{bmatrix} & (1) & (2) & (3) & (4) & (5) \\ (1) & 0 & 5 & 12 & \infty & 15 \\ (2) & 5 & 0 & 7 & \infty & 10 \\ (3) & 12 & 7 & 0 & 4 & 17 \\ (4) & \infty & \infty & 4 & 0 & 1 \\ (5) & 15 & 10 & 17 & 1 & 0 \end{bmatrix}$$

Рассмотрим $k = 3$. Результат после третьего шага выглядит следую-

щим образом:

$$\begin{bmatrix} & (1) & (2) & (3) & (4) & (5) \\ (1) & 0 & 5 & 12 & 16 & 15 \\ (2) & 5 & 0 & 7 & 11 & 10 \\ (3) & 12 & 7 & 0 & 4 & 17 \\ (4) & 16 & 11 & 4 & 0 & 1 \\ (5) & 15 & 10 & 17 & 1 & 0 \end{bmatrix}$$

Рассмотрим $k = 4$. Результат после четвертого шага выглядит следующим образом:

$$\begin{bmatrix} & (1) & (2) & (3) & (4) & (5) \\ (1) & 0 & 5 & 12 & 16 & 15 \\ (2) & 5 & 0 & 7 & 11 & 10 \\ (3) & 12 & 7 & 0 & 4 & 5 \\ (4) & 16 & 11 & 4 & 0 & 1 \\ (5) & 15 & 10 & 5 & 1 & 0 \end{bmatrix}$$

Рассмотрим $k = 5$. Результат после этого шага выглядит аналогично матрице выше.

Так что насчет реализации данного алгоритма? Все достаточно просто:

```
using System;
namespace FloydAlgorithm
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Введите количество вершин в графе:");
            var n = Int32.Parse(Console.ReadLine());
            var matrix = new double[n, n];
            for (var i = 0; i < n; i++)
            {
                for (var j = 0; j < n; j++)
                {
                    Console.WriteLine(
                        $"Введите расстояние между вершинами {i + 1} и {j + 1}.
```

```
                    var answer = Console.ReadLine();
                    if (double.TryParse(answer, out var doubleAnswer))
                    {
                        matrix[i, j] = doubleAnswer;
                    }
                    else
                    {
                        matrix[i, j] = double.PositiveInfinity;
                    }
                }
            }
            Console.WriteLine("Ответ:");
            for (var k = 0; k < n; k++)
            {
                for (var i = 0; i < n; i++)
                {
                    for (var j = 0; j < n; j++)
                    {
                        matrix[i, j] = Math.Min(matrix[i, j], matrix[i, k] + mat
                    }
                }
            }

            for (var i = 0; i < n; i++)
            {
                for (var j = 0; j < n; j++)
                {
                    Console.Write($"{matrix[i, j]} ");
                }
                Console.WriteLine();
            }
        }
    }
}
```

Таким образом, сложность работы данного алгоритма составляет $O(N^3)$, где N – количество вершин в графе. Наша задача решена.