Санкт-Петербургский Политехнический Университет

Высшая школа прикладной математики и вычислительной физики, ФизМех

01.03.02 Прикладная математика и информатика

Лабораторная работа №4

Дисциплина “Дискретная математика”

Тема “ Циклы и раскраска”

Вариант “ Наибольшее независимое множество вершин и наименьшее вершинное покрытие”

Выполнил студент гр. 5030102/20201 Мелко Тимофей Андреевич

Поставленная задача

Найти в графе и вывести любые наибольшее независимое множество вершин и наименьшее вершинное покрытие.

Используемый язык программирования

Python 3.12.6

Описание алгоритма

Нахождение наибольшего независимого множества вершин:

Инициализация:

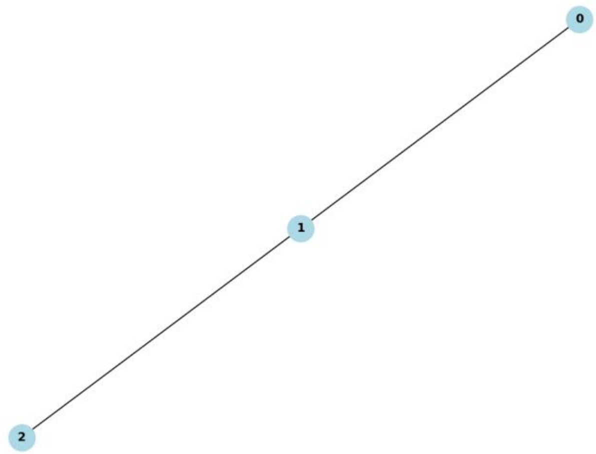
```
  X ← ∅
  max_size ← 0
  stack ← [(∅, {0, 1, 2, ..., V-1})]
 Пока stack не пуст:
  (S, T) ← stack.pop()
  Если |S| > max_size:
    X ← S
    max_size ← |S|
  Для каждой вершины v из T:
    Если ∀u ∈ S: adj_matrix[v][u] = 0:
      new_T ← T \ {v}
      stack.push((S ∪ {v}, new_T))
 Вернуть X
```

Нахождение наименьшего вершинного покрытия множества:

```
min_vertex_cover = [v for v in range(self.V) if v not in max_independent_set]
```

Пример работы алгоритма

Для примера рассмотрим граф.



	0	1	2
0	0	1	0
1	1	0	1
2	0	1	0

Поиск наибольшего независимого множества:

Инициализация:

X = ∅

max_size = 0

stack = [(∅, {0, 1, 2})]

Шаг 1:

Достаем из стека: (S, T) = (∅, {0, 1, 2})

max_size = 0, S = ∅ → Ничего не обновляем

Перебираем вершины в T:

v = 0:

Условие ∀u ∈ S: adj_matrix[0][u] = 0 выполняется

new_T = {1, 2}

Добавляем в стек: ({0}, {1, 2})

v = 1:

Условие ∀u ∈ S: adj_matrix[1][u] = 0 выполняется

new_T = {0, 2}

Добавляем в стек: ({1}, {0, 2})

v = 2:

Условие ∀u ∈ S: adj_matrix[2][u] = 0 выполняется

new_T = {0, 1}

Добавляем в стек: ({2}, {0, 1})

Шаг 2:

Достаем из стека: (S, T) = ({2}, {0, 1})

max_size = 0, |S| = 1 → Обновляем: X = {2}, max_size = 1

Перебираем вершины в T:

v = 0:

Условие ∀u ∈ S: adj_matrix[0][u] = 0 выполняется

new_T = {1}

Добавляем в стек: ({2, 0}, {1})

v = 1:

Условие ∀u ∈ S: adj_matrix[1][u] = 1 не выполняется

Шаг 3:

Достаем из стека: (S, T) = ({2, 0}, {1})

max_size = 1, |S| = 2 → Обновляем: X = {2, 0}, max_size = 2

Перебираем вершины в T:

v = 1:

Условие ∀u ∈ S: adj_matrix[1][u] = 1 не выполняется

Шаг 4:

Достаем из стека: (S, T) = ({1}, {0, 2})

max_size = 2, |S| = 1 → Ничего не обновляем

Перебираем вершины в T:

v = 0:

Условие ∀u ∈ S: adj_matrix[0][u] = 1 не выполняется v = 2:

Условие ∀u ∈ S: adj_matrix[2][u] = 1 не выполняется

Шаг 5:

Достаем из стека: (S, T) = ({0}, {1, 2})

max_size = 2, |S| = 1 → Ничего не обновляем

Перебираем вершины в T:

v = 1:

Условие ∀u ∈ S: adj_matrix[1][u] = 1 не выполняется

v = 2:

Условие ∀u ∈ S: adj_matrix[2][u] = 0 выполняется

new_T = {1}

Добавляем в стек: ({0, 2}, {1})

Шаг 6:

Достаем из стека: (S, T) = ({0, 2}, {1})

max_size = 2, |S| = 2 → Ничего не обновляем

Перебираем вершины в T:

v = 1:

Условие ∀u ∈ S: adj_matrix[1][u] = 1 не выполняется

Завершение:

Стек пуст.

Ответ: X = {0, 2}, max_size = 2

Поиск наименьшего вершинного покрытия:

- Вершины графа: V = {0, 1, 2}.
- Независимое множество: I = {0, 2}.
- Наименьшее вершинное покрытие:

min_vertex_cover = V \ I = {1}.

Сложность алгоритмов

Наибольшее независимое множество: Алгоритм перебирает все возможные подмножества вершин графа, а их количество равно 2^V. Для каждого состояния (S, T) выполняется проверка смежности между вершинами множества S, что занимает O(V) операций в худшем случае. Итоговая временная сложность: O(V *2^V)

Наименьшее вершинное покрытие: Генерация списка всех вершин range(self.V) занимает O(V). Проверка каждой вершины на принадлежность занимает O(1). Итоговая временная сложность: O(V)

Входные и выходные данные

Входные данные. Квадратная матрица n×n, где n — количество вершин в графе. Каждая строка матрицы представляет связи (ребра) для одной вершины.

Выходные данные. Записывается в файл строки вида:
Наибольшее независимое множество вершин: {наибольшее независимое множество вершин}

Наименьшее вершинное покрытие: {множество вершин которое является наименьшим вершинным покрытием}

Область применимости

Алгоритм для нахождения максимального независимого множества (MIS) и минимального вершинного покрытия (MVC) подходит для определённых типов задач и графов. MIS часто используется для разделения графов на независимые компоненты или кластеры. Пример — задачи кластеризации в сетях, когда нужно выделить группы объектов, которые не взаимодействуют напрямую. MVC используется для минимизации количества узлов, которые должны быть подключены для обеспечения полной связи между всеми элементами сети. В задачах настройки беспроводных и проводных сетей — это критично для сокращения затрат на оборудование и обеспечение эффективной работы сети.

Представление графов в программе

Для представления графа в программе я буду использовать матрицу смежности. Доступ к данным в матрице занимает O(1) что делает возможным прямую и быструю работу с каждой парой вершин. Кроме того, добавление или удаление ребра также выполняется за O(1), так как достаточно изменить один элемент матрицы.

Вывод

Алгоритмы для нахождения максимального независимого множества (MIS) и минимального вершинного покрытия (MVC) являются важными инструментами в теории графов и находят широкое применение в различных областях, таких как оптимизация сетевых структур, планирование, биоинформатика и логистика. Несмотря на их теоретическую значимость, выбранный алгоритм имеет экспоненциальную сложность.