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:

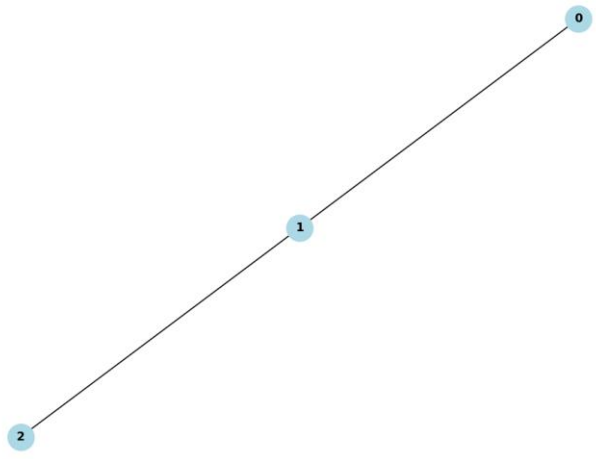
new_T ← T \ {v}

stack.push((S ∪ {v}, new_T))

Вернуть X

Пример работы алгоритма

Для примера рассмотрим граф.



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

Инициализация

- X — пустое множество (будет хранить максимальное независимое множество).
- max_size— 0 (размер максимального независимого множества).
- Стек stack инициализируется с начальным состоянием: stack=[({},{0,1,2})]
- Где:
    - S — текущее независимое множество (начально пустое),
    - T — вершины, которые ещё не добавлены в S (начально все вершины графа).

2. Первая итерация

- Извлекаем состояние из стека:

  (S,T)=({},{0,1,2})

  Это означает, что в данный момент S пусто, а T={0,1,2}

- Обновляем максимальное независимое множество:
    - Поскольку |S|=0, это больше, чем текущий max_size=0, поэтому обновляем: X={},max_size=0
- Перебираем вершины из T={0,1,2}
    - Вершина 0: Добавляем её в S, так как она не соединена с любыми вершинами в S (в S нет вершин).
        - Новое состояние: stack.push(({0},{1,2}))
        - Вершина 0 добавляется в независимое множество, а оставшиеся вершины для проверки — {1,2}
    - Вершина 1: Мы проверяем, можем ли добавить её в S. Вершина 1 соединена с вершиной 0 (которая уже в S), значит, мы не можем добавить вершину 1 в S в этой итерации.
    - Вершина 2: Добавляем её в S, так как она не соединена с вершинами в S (в S только вершина 0, и она не соединена с вершиной 2).
        - Новое состояние: stack.push(({2},{0,1}))
    - Стек на данный момент:

      stack=[({0},{1,2}),({2},{0,1})]

3. Вторая итерация

Извлекаем из стека первое состояние: ({0},{1,2})

- Обновление максимального независимого множества:
    - Поскольку |S|=1, это больше текущего max_size=0, обновляем: X={0}, max_size=1
- Перебираем вершины из T={1,2}
    - Вершина 1: Она не соединена с вершиной 0 (которая в S), поэтому добавляем её в S.
        - Новое состояние: stack.push(({0,1},{2}))
    - Вершина 2: Она соединена с вершиной 1, которая уже в T, так что её нельзя добавить в S на этом шаге.
    - Стек на данный момент:

      stack=[({2},{0,1}),({0,1},{2})]

4. Третья итерация

Извлекаем из стека состояние: ({2},{0,1})

- Обновление максимального независимого множества:
    - |S|=1, но max_size уже равно 1, так что мы не обновляем X.
- Перебираем вершины из T={0,1}
    - Вершина 0: Она не соединена с вершиной 2 (которая в S), поэтому добавляем её в S.
        - Новое состояние: stack.push(({2,0},{1}))
    - Вершина 1: Она соединена с вершиной 2, которая уже в S, поэтому её нельзя добавить.
    - Стек на данный момент:

      stack=[({0,1},{2}),({2,0},{1})]

5. Четвертая итерация

Извлекаем из стека состояние ({0,1},{2}), но |S|=2— это максимальное независимое множество для этого графа.

Для определения наименьшего вершинного покрытия воспользуемся свойством графа количество вершин графа равно его минимальному номеру покрытия вершин плюс размеру максимального независимого множества. Следовательно наименьше вершинное покрытие графа будет множество {2}

Сложность алгоритма

Алгоритм имеет экспоненциальную сложность $O(2^V \cdot V)$. Общее число таких подмножеств — 2^V В худшем случае алгоритм выполняет O(2^V) итераций, и на каждой итерации выполняется проверка независимости за 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.