ЗВІТ

Підготували: Штогрин Олег та Валігурський Антон

**Опис задачі**: написати алгоритми: Прима та Краскла- для знаходження найменшого каркасу; Флойда- Ворашала для знаходження найкоротших відстаней між усіма вершинами та Белмана- Форда для знаходження найкоротшої відстані від заданої точки.

**Опис експерименту**: дослідити ефективність написаних нами алгоритмів відносно вбудованих.

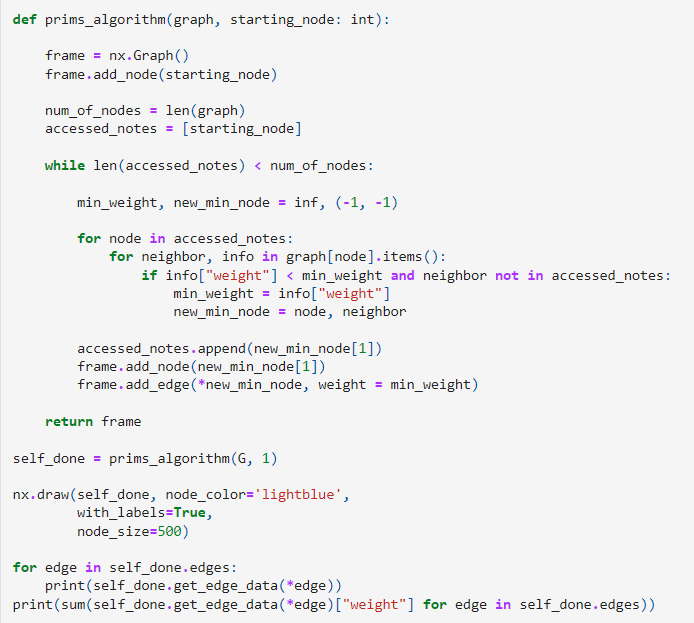
**За яким принципом проводиться експеримент:** ми будемо порівнювати наші та вбудовані алгоритми на вершинах: 5, 10, 20, 50, 100, 250, 500. Було використано бібліотеку time для занотування результатів ефективності алгоритмів. Графік роботи буде представлено за допомою бібліотеки matplotlib.pylot.

**Хід роботи**:

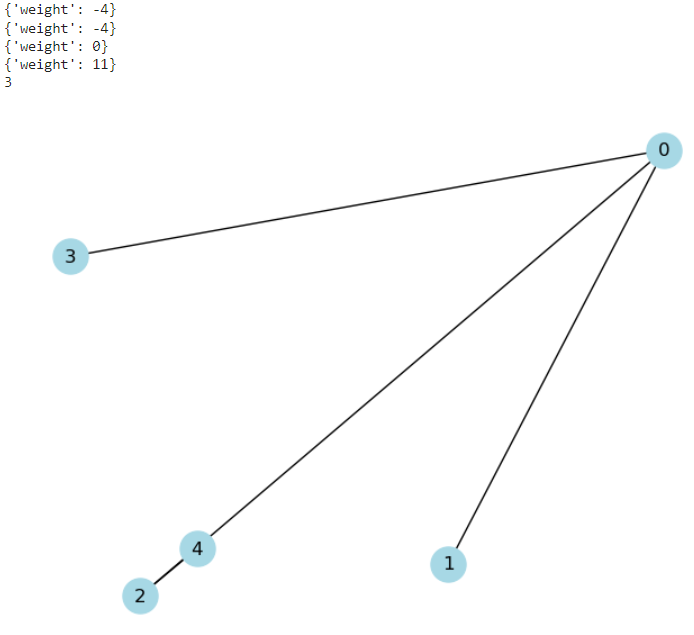
***Експеримент 1***:

Алгоритм Прима:

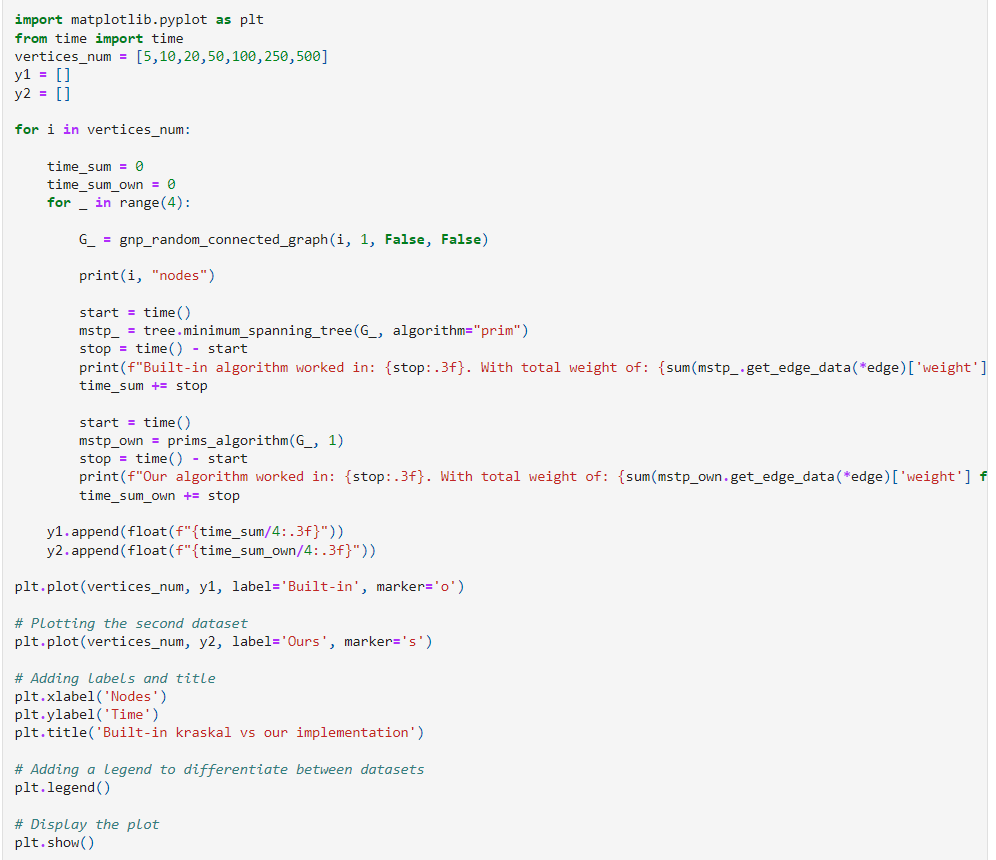
Код:



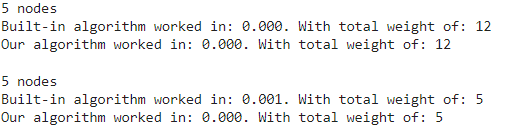
Вивід:

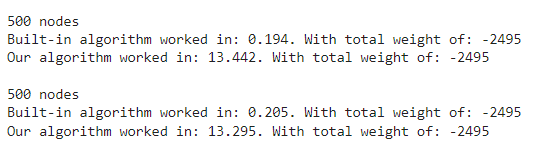


Код проведення експерименту:

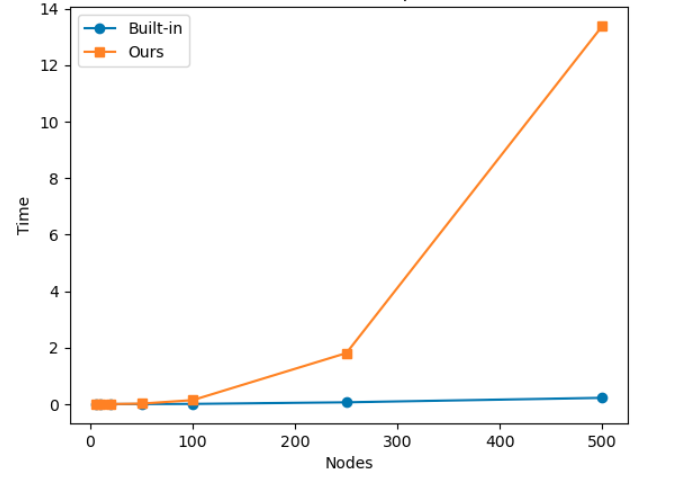


Приклад виводу:





Графік роботи вбудованого та нашого алгоритму:



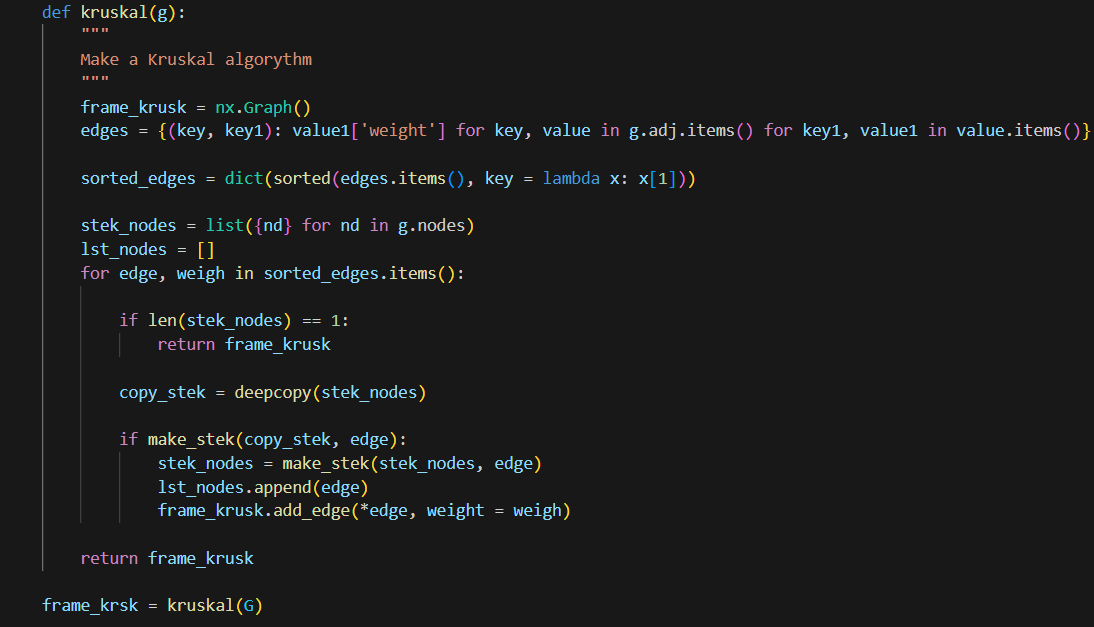
Підсумок: вбудований алгоритм працює краще нашого, хоча на невеликій кількості вершин це не є помітним. Наш алгоритм працює гірше через те, що у вбудованому є додаткові умови(if, else), які не дають робити ‘зайві’ кроки алгоритму, тобто пропускаючи непотрібні ітерації.

***Експеримент 2***:

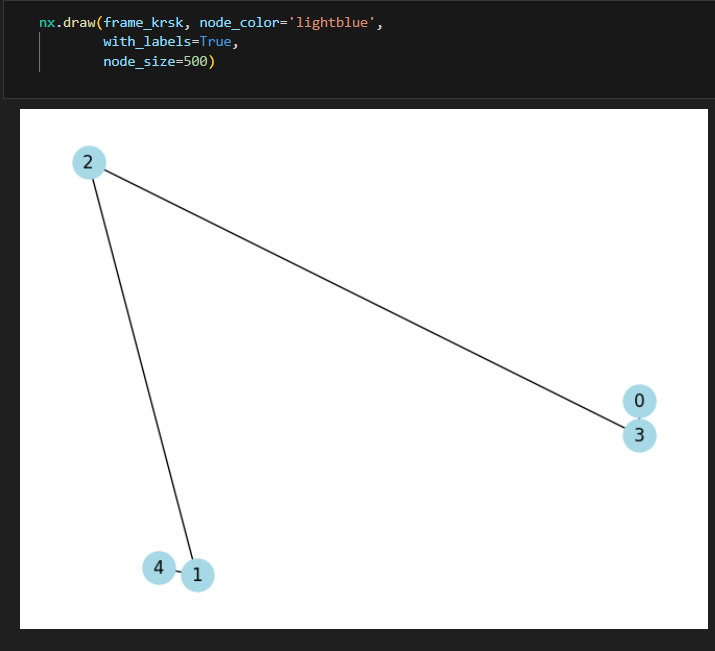
Алгоритм Краскла:

Код:

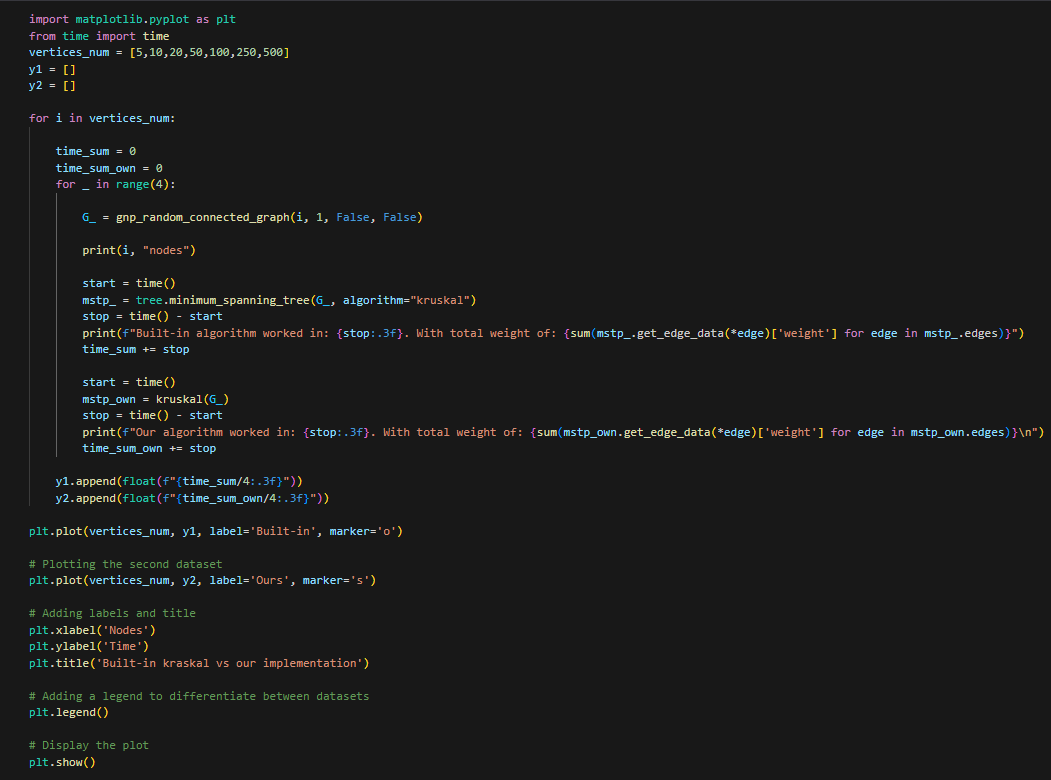




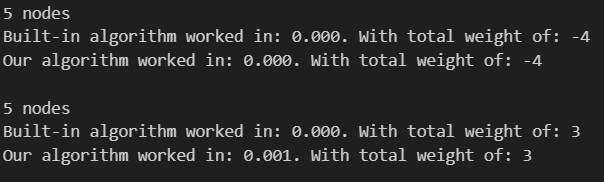
Візуалізація:

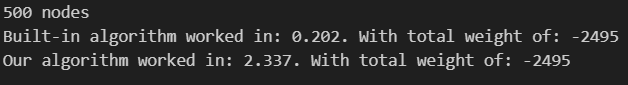


Код проведення експерименту:

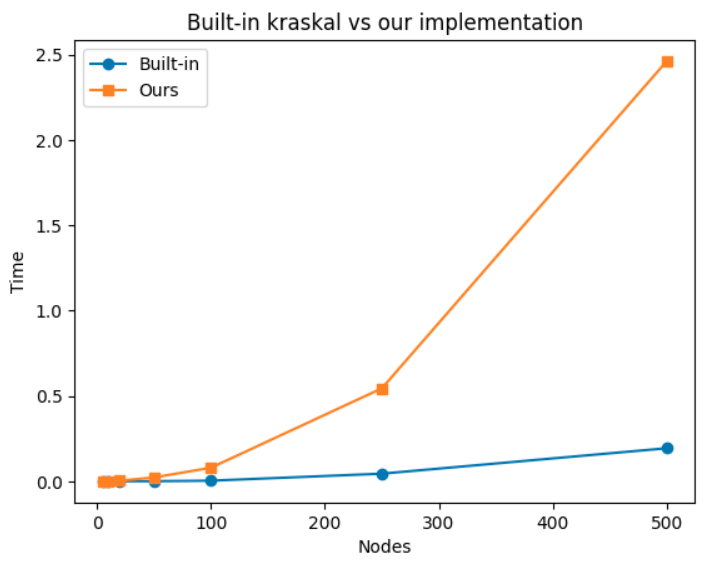


Приклад виводу:





Графік роботи вбудованого та нашого алгоритму:

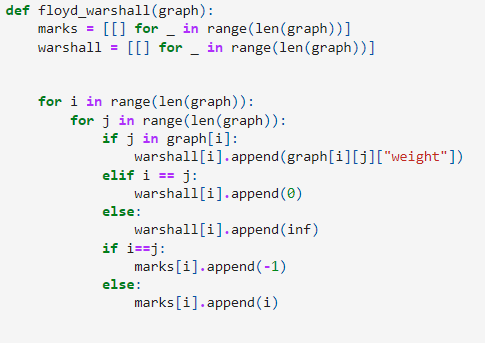


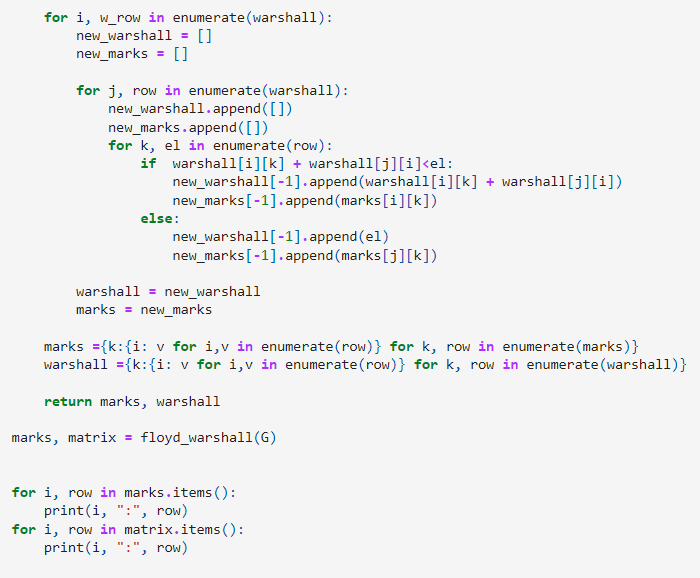
Підсумок: як і з алгоритмом Прима: вбудований алгоритм працює краще нашого, хоча на невеликій кількості вершин це не є помітним. Це пов’язано з тим, що функція make\_stek повторюється 2 рази , перед тим, як додати ребро, що ускладнює алгоритм, враховуючи, що в самій функції, виконуються операції copy та update, що дещо більше заповнює пам’ять та ускладнює алгоритм.

***Експеримент 3***:

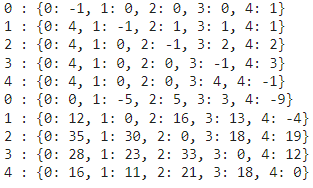
Алгоритм Флойда- Воршала:

Код:

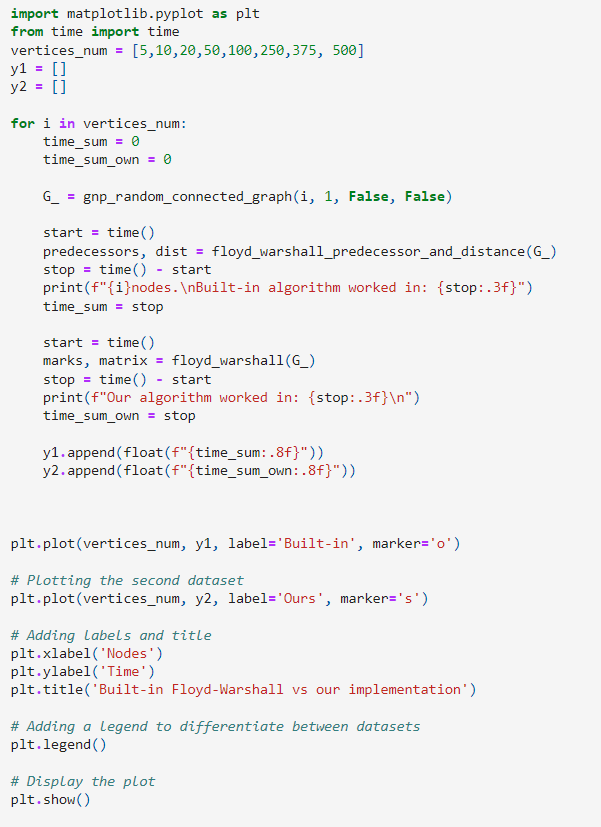




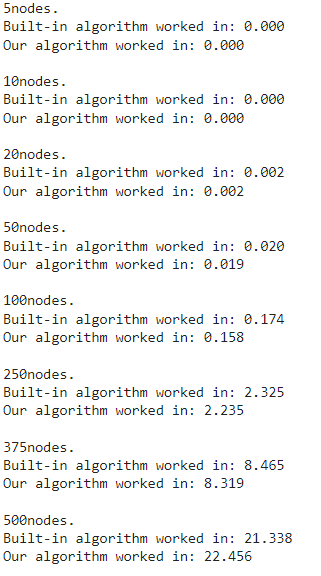
Приклад виводу:



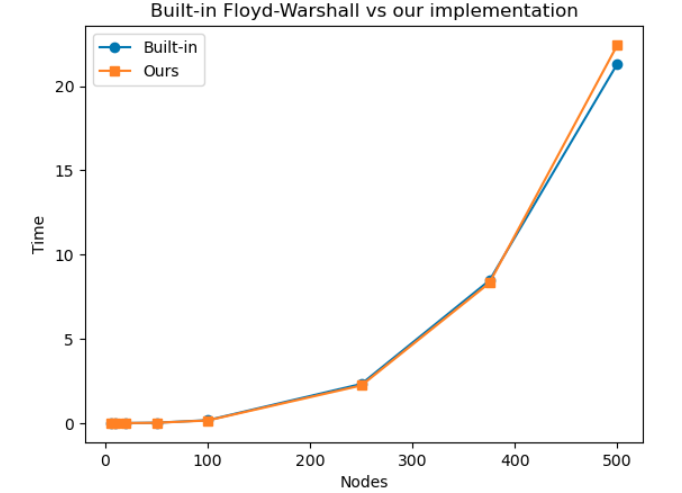
Код проведення експерименту:



Приклад виводу:



Графік роботи вбудованого та нашого алгоритму:

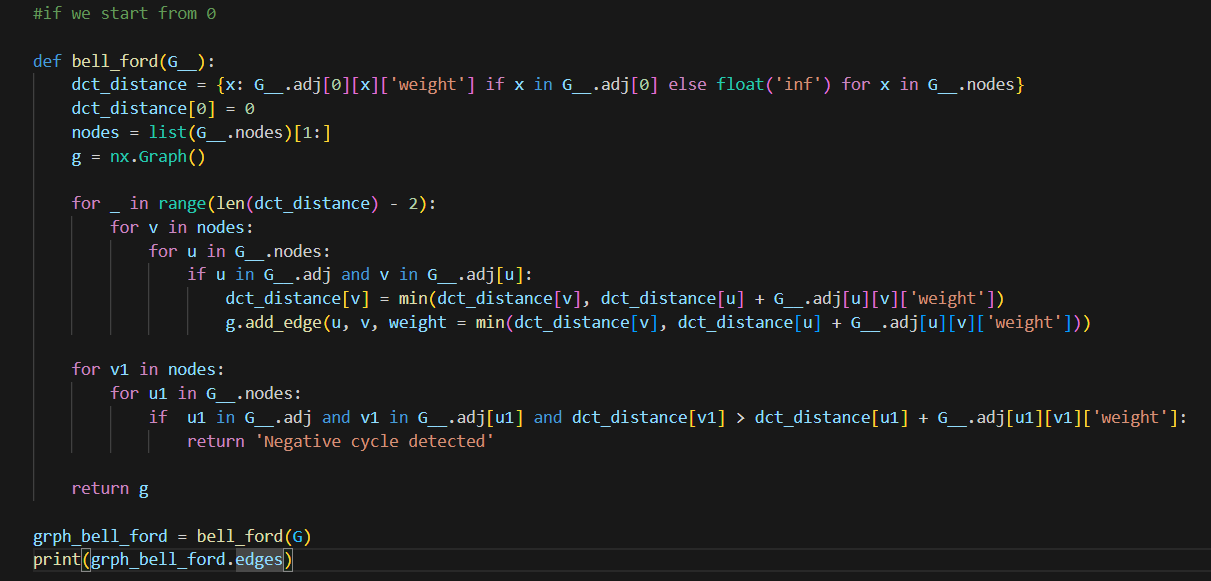


Підсумок: Наш алгоритм працює практично з такою самою ефективністю, як і вбудований. На малій кількості вершим він навіть дещо ефективніше. Невелика перевага, яку ми бачимо на 500 вершинах пов’язана з тим, що вбудований алгоритм має меншу складність: О(n^3).

***Експеримент 4***:

Алгоритм Белмана- Форда:

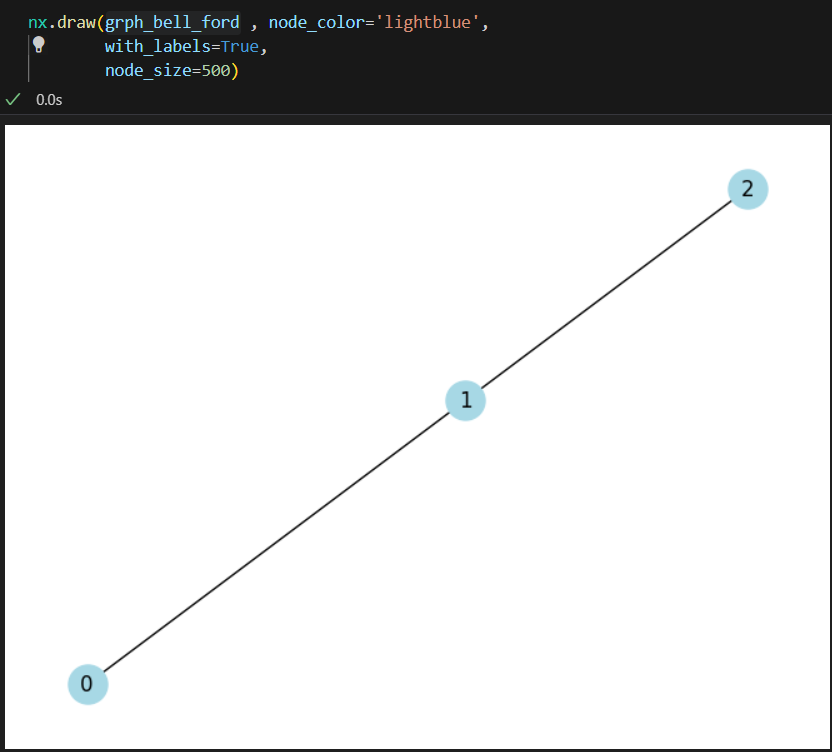
Код:



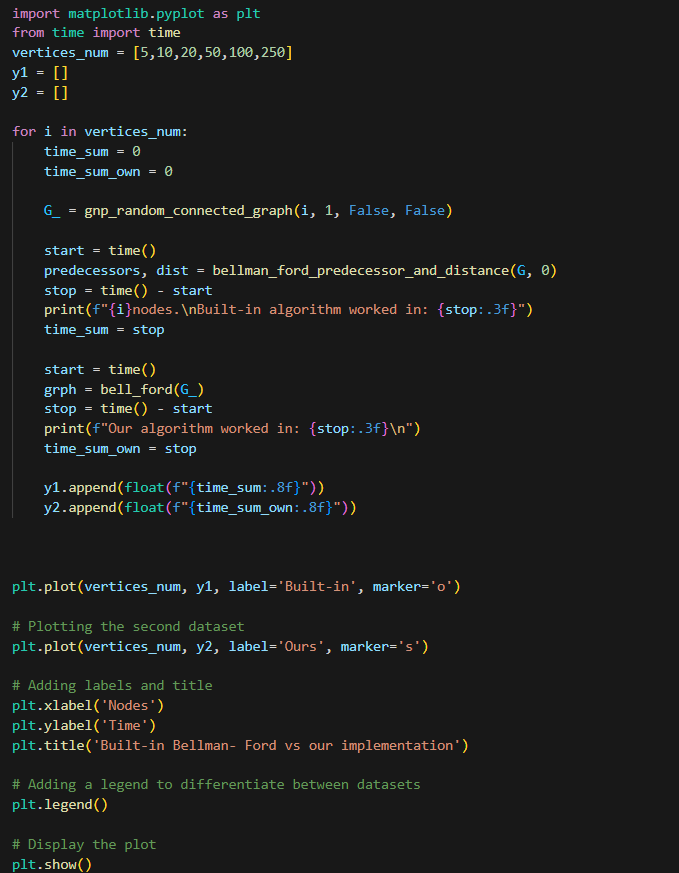
Приклад виводу:



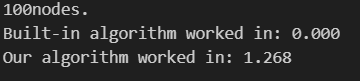
Приклад візуалізації:



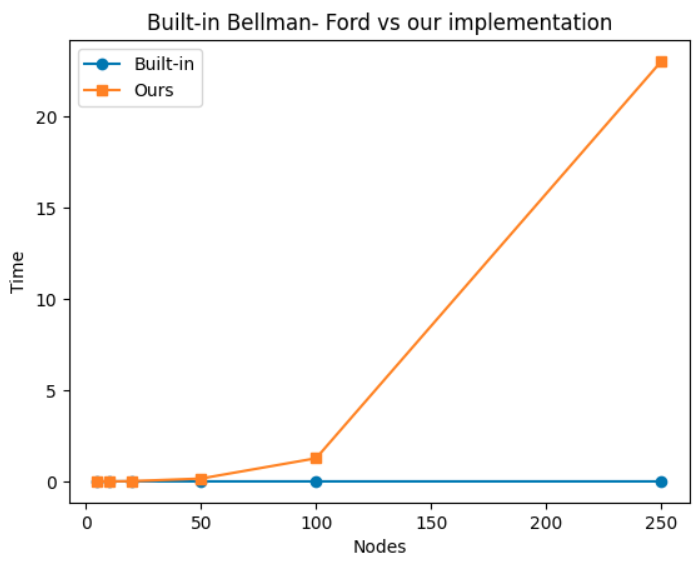
Код проведення експерименту:



Приклад виводу:



Графік роботи вбудованого та нашого алгоритму:



Підсумок: Алгоритм працює гірше вбудованого. На малих вершинах алгоритм працює непогано. Це може бути пов’язано з способом перевірки на негативні цикли вбудованого алгоритму та нашого: можливо, вбудований алгоритм одразу перевіряє наявність негативних простих циклів та видає про це повідомлення, коли ж наш алгоритм робить це опісля всіх дій.

**Загальний підсумок**:

Алгоритми Краскла та Прима найкраще працюють на графах з малою кількістю вершин, очевидно, пов’язано з тим, що алгоритм має проходитись по значно більшій кількості ребер, що, практично, експоненційно збільшує його час.

Алгоритм Флойда- Ворашала, в порівнянні з вбудованим, працює просто прекрасно на будь- якій кількості вершин( як мінімум до 500). Якщо дивитись не в порівняні, то час так само збільшується експоненційно, через більшу кількість операцій.

Наш алгоритм Белмана- Форда може зрівнятись з вбудованим до приблизно 50 вершин. Пізніше він видає гірші результати.

Нюанси: можлива похибка в графіках, яка пов’язана з особливістю роботи процесора, адже він постійно пробує ‘вгадати’ наступні операції, через це дані експерименту можуть різнитись на тисячні, а то й соті секунди, бо в одному випадку процесор ‘вгадав’ наступну операцію, а в іншому ні.

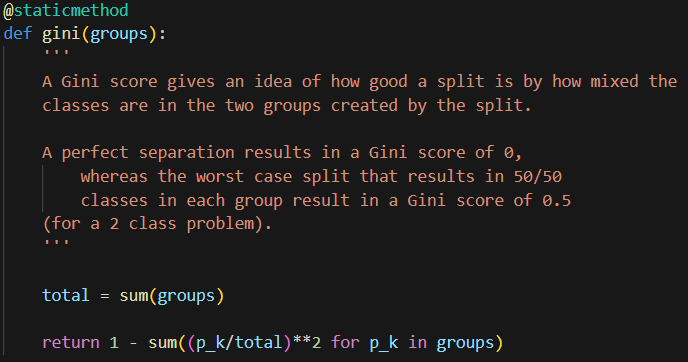
*Невелике порівняння наших алгоритмів:*

Краскалу на 250вершин вдалось знайти каркас за 0.542 секунди, коли ж Приму знадобилось 1.171 секунди.

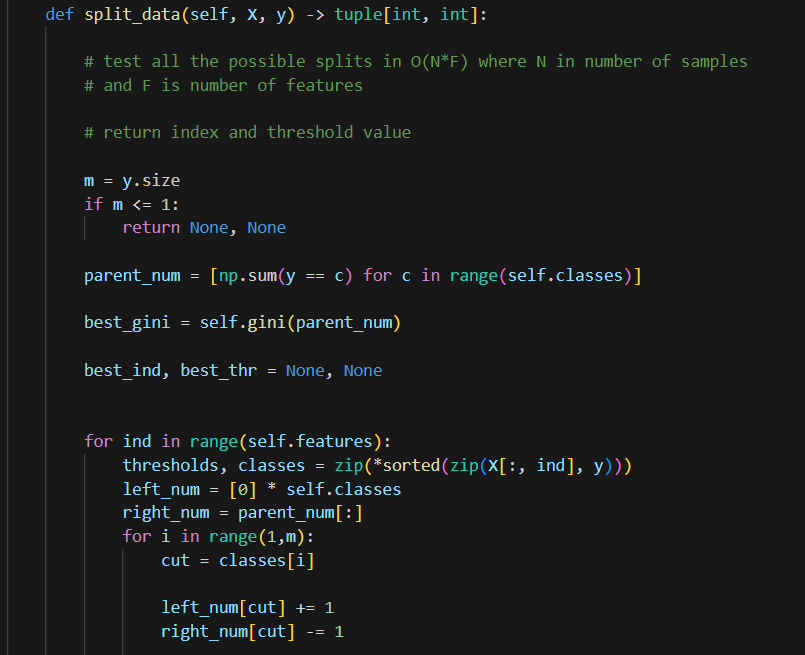
Белману- Форду на подолання 250 вершин знадобилось 23 секунди, коли ж Флойду- Воршалу тільки 2.235 секунди.

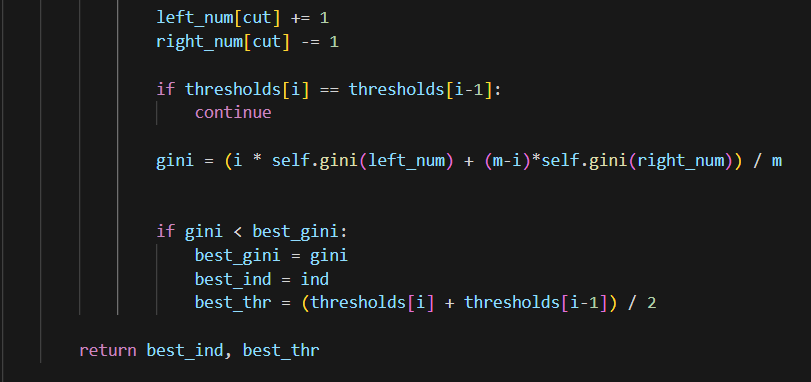
**Decision tree**

1) Визначає gini, на осонові якого визначається наскільки вдалим є розприділення.

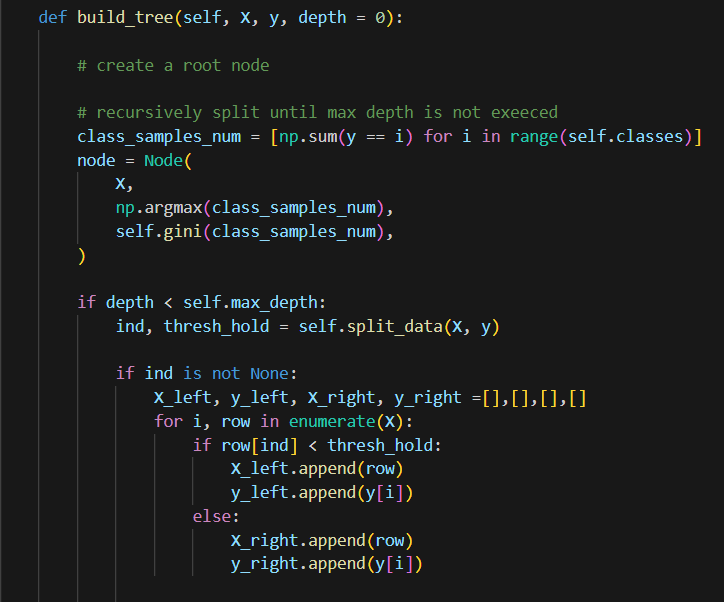


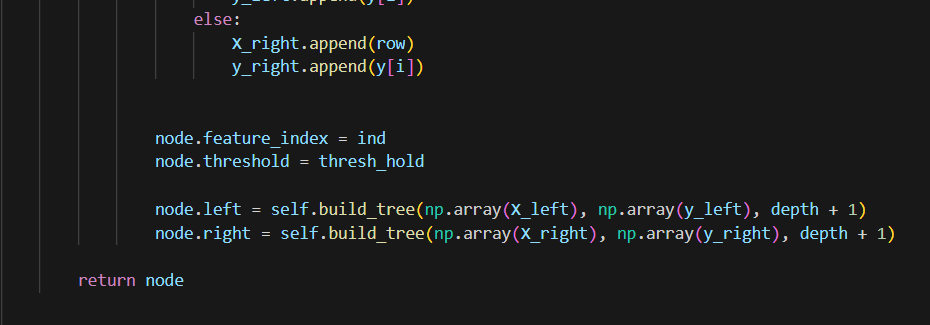
2) Перевіряє всі можливі розподіли в межах О(N\*F), N- кількість зразків, F- кількість ‘особливостей’.



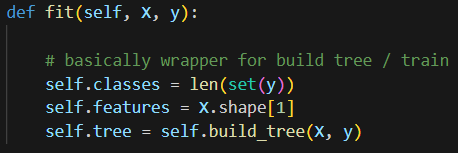


3) Створює кореневе дерево, рекурсивно розділяючи по ‘gini’ до того моменту, поки не буде досягнуто максимальної ‘глибини’.

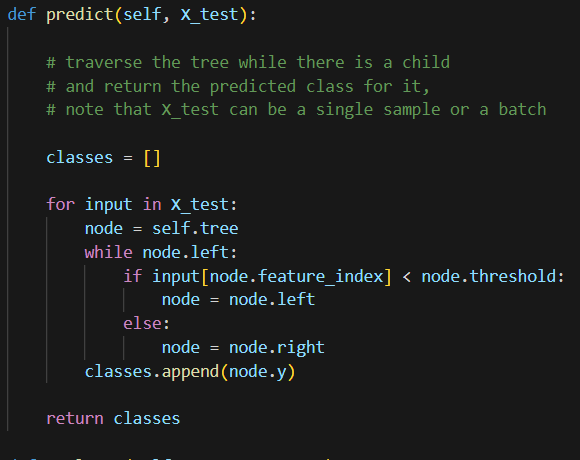




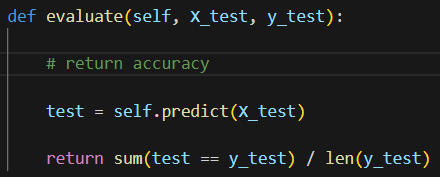
4) Визначає кількість ‘зразків’, ‘особливостей’ та генерує дерево



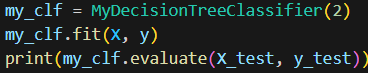
5) Обходить все дерево, поки в нього все ще є ‘діти’, вертаючи при цьому передбачений клас



6) Повертає точність наших передбачень



**В кінці ми просто викликаєм все:**



**Та отримуємо точність нашого передбачення щодо дерева:**

