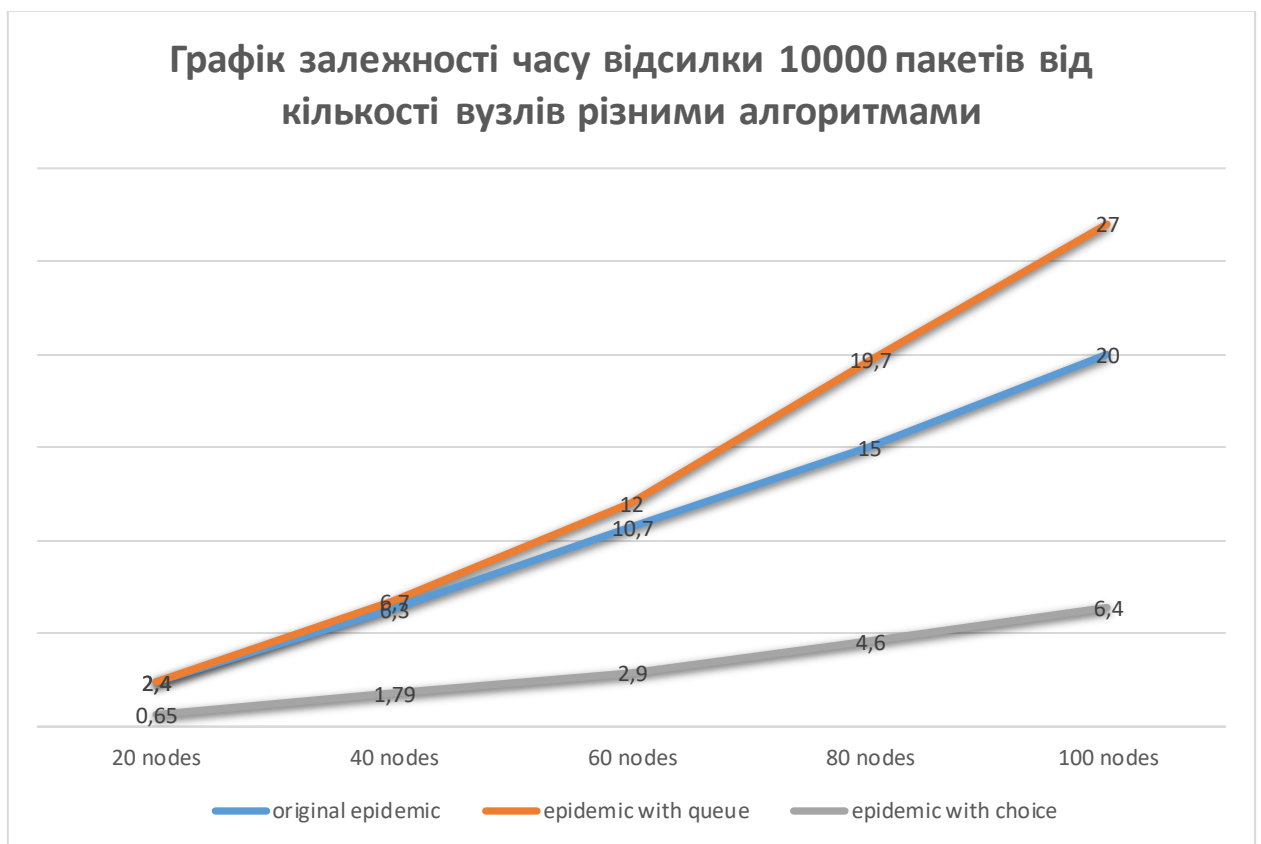


**Epidemic with queue** – це модифікація, де в пакет добавляється кожен відвіданий ним вузол, в результаті чого сервер не шле пакет в вузли де той раніше бував.

**Epidemic with choice** – це модифікація, в ньому 4(або інша кількість) вузли вибираються не випадково\*, а за алгоритмом. Для цього написана функція, що приймає індекс вузла в якому знаходиться пакет, та вибирає 4 найближчі вузли через одного + 2 на кожній ітерації. Тут ми вважаємо що кожен вузол бачить інші вузли відсортованими по індексам. В результаті чого всі вузли отримують пакет зі 100% ймовірністю.

**Після проведеного моделювання отримано наступні результати:**



Модифікація, де відвідані вузли зберігаються в тілі пакету показує приблизно на 10-30% кращу статистику розсилки пакетів, що зростає при збільшенні кількості вузлів та на 5-13% меншу кількість ітерацій. Через те що розмір пакету теж зростає ми втрачаємо час на кожній ітерації, звідси загальний час виконання маємо гірший ніж у звичайного epidemic, що видно графіку вище.

Порівнюючи Epidemic with queue з тестовим epidemic маємо:

При посиланні пакету на 100 вузлів: +32% cases, +28% time

При посиланні пакету на 50 вузлів: +18% cases, +32% time

При посиланні пакету на 20 вузлів: +10% cases, +1% time

Модифікація, де замість випадкового вибору вузлів використовується власний алгоритм працює краще за всі інші варіації. Час виконання зростає лінійно від кількості вузлів, кількість ітерацій завжди однакова для однакової кількості вузлів. Також ми завжди маємо 100% cases. Час одної ітерації однаковий порівнюючи із випадковим вибором вузлів, а кількість ітерацій менша приблизно в 2 рази. Загальний час виконання менший в 3.5 рази при однакових сценаріях, ніж у звичайного epidemic, що гарно видно на графіку вище.

Порівнюючи Epidemic with choice з тестовим epidemic маємо:

При посиланні пакету на 80 вузлів: +257% cases, -3.5x time

При посиланні пакету на 60 вузлів: +170% cases, -3.5x time

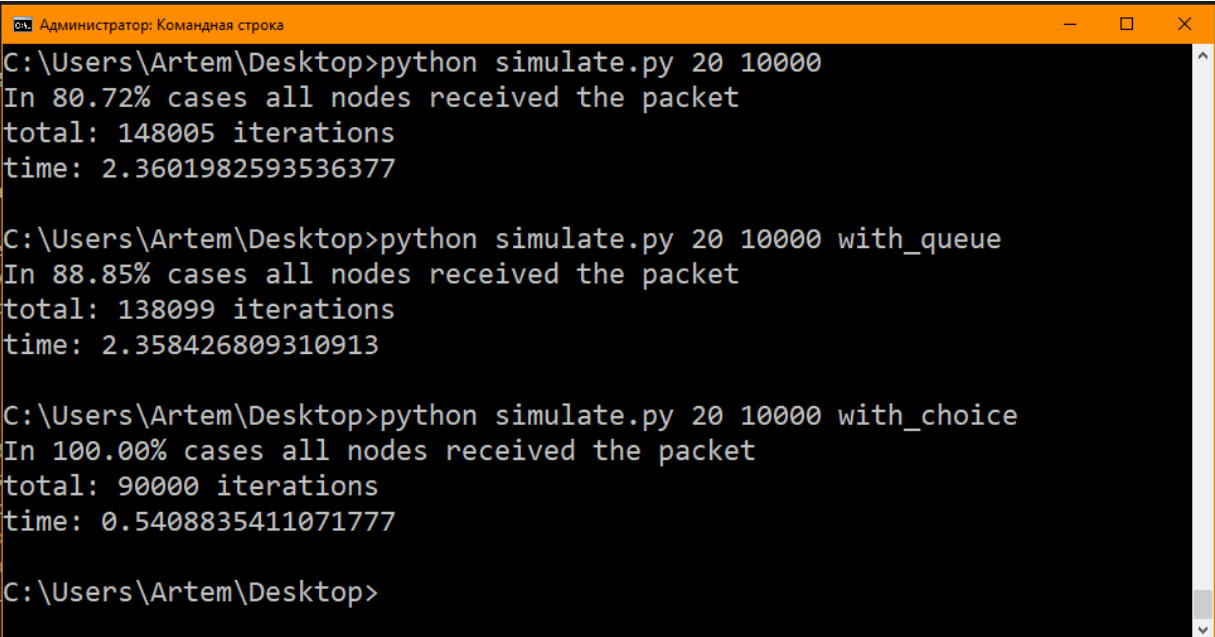
При посиланні пакету на 20 вузлів: +83% cases, -3.5x time

### Висновки:

Вважаю найбільш ефективним саме останній варіант модифікації з власним алгоритмом вибору вузлів: "Epidemic with choice", він:

- \* не програє в часі виконання одної ітерації;
- \* не програє в ресурсовитратності одної ітерації;
- \* гарантує 100% доставку пакету всім вузлам;
- \* має в 3-4 рази менший загальний час виконання.

Приклад емуляції роботи алгоритмів з консолі, параметри:  $x = 4$ ,  $n = 20$ ,  $i = 10000$ . Першим йде стандартний алгоритм, другим модифікація з чергою, третім модифікація з вибором. Також вивожу загальну кількість ітерацій\*\* та час виконання.



```
Администратор: Командная строка
C:\Users\Artem\Desktop>python simulate.py 20 10000
In 80.72% cases all nodes received the packet
total: 148005 iterations
time: 2.3601982593536377

C:\Users\Artem\Desktop>python simulate.py 20 10000 with_queue
In 88.85% cases all nodes received the packet
total: 138099 iterations
time: 2.358426809310913

C:\Users\Artem\Desktop>python simulate.py 20 10000 with_choice
In 100.00% cases all nodes received the packet
total: 90000 iterations
time: 0.5408835411071777

C:\Users\Artem\Desktop>
```

P.S.:

**\*\*** Важаю за одну ітерацію ситуацію коли нода шле x пакетів (наприклад 4)

**\*** При написанні емулятора роботи стандартного epidemic вважаю що нода не шле пакет сама в себе, та інші x(наприклад 4) вузли вибирає різними. Отже ситуації коли нода 0 шле пакети в ноди: 1, 2, 3 і знову в 1 не виникає! Також розсилка починається з ноди 0.

Нижче ілюструю як саме в мене все працює на прикладі коли  $x = 4$ ,  $n = 8$ ,  $i = 1$ :

0 --> 3 +

0 --> 2 +

0 --> 5 +

0 --> 7 +

[0, 3, 2, 5, 7]

# Розсилка почалась з вузла 0, він вибрав 4 вузли та додав в чергу

3 --> 5 -

3 --> 4 +

3 --> 6 +

3 --> 0 -

[0, 3, 2, 5, 7, 4, 6]

# Далі розсилка з вузла 3, він вибрав 4 вузли, але в чергу додав тільки 2,

# бо інші 2 вже є в черзі, а значить пакети отримали але ще не відправили

2 --> 0 -

2 --> 4 -

2 --> 7 -

2 --> 3 -

[0, 3, 2, 5, 7, 4, 6]

# Далі розсилка з вузла 2, він вибрав 4 вузли, але в чергу нічого не додав,

# бо всі вибрані вузли вже мають пакет

5 --> 3 -

5 --> 0 -

5 --> 1 +

5 --> 6 -

[0, 3, 2, 5, 7, 4, 6, 1]

# Остання четверта ітерація, розсилка з вузла 5, він вибрав 4 вузли, в чергу додав вузол 1,

# бо інші вибрані вузли вже мають пакет.

# Кінець роботи так як кількість елементів в [0, 3, 2, 5, 7, 4, 6, 1] рівна кількості всіх вузлів,

# що говорить про те що всі вузли отримали пакет