**Постановка задачі**

Маючи список поїздок в місті та кортеж самоходних автомобілей, задачею є призначати поїздкт автомобілям таким чином, щоб пасажири добратись до місця призначення вчасно. За кожну поїздку, яка закінчилась вчассно (або раніше), нараховуються бали, які є пропорційні до відстані тієї поїздки плюс додаткові бонусні бали, якщо поїздка почалась точно вчасно.

**Опис проблеми**

Мапа

Місто представлене прямокутною сіткою вулиць з R горизонтальних вулиць (рядки) та C вертикальних вулиць (колонки). Перетини вулиць позначаються координатами перетину r горизонтальної та c вертикальної вулиці.

Автомобілі

У кортежі знаходиться F автомобілей. На початку симуляції, всі автомобілі знаходяться в перетині [0, 0]. Ліміту на кількість автомобілей в одному перетині немає.

Час та відстань

Симуляція проходить в Т кроків, з 0 до Т-1.

Івідстань між двлма перетинами визначається як мінімальна загальна кітькість слоків, які автомобіль проїжджає, щоб дістатись від одного перетину до іншого. Тобто, відставнь між перетином [a,b]та [x,y] дорівнює |a-x|+|b-y|.

Кількість кроків, необхідна для того, щоб здійснити поїздку між двома перетинами дорівнює відстані між ними.

Поїздки

Є N заброньованих поїздок.   
Кожна поїздка характеризується наступним чином:

* Початок перетину – щоб почати поїздку, автомобіль повинен бути в цьому перетині
* Кінець перетину – щоб закінчити поїздку, автомобіль має бути в цьому перетині. Кінцевий перетин завжди відрізняється від початкового.
* Найраніший початок – найраніший часовий крок, в якому автомобіль може почати поїздку. Він також може почати запізно.
* Найпізніший кінець – останній часовий крок, до якого поїздка повинна закінчитись, щоб отримати за неї бали.

Симуляція

Кожен автомобіль проїжджає призначену поїздку в визначеному алгоритмом порядку:

* Спочатку, автомобіль їде від своєї поточної позиції до початкового перетину наступної поїздки.
* Тоді, якщо поточний часовий крок є менший ніж найраніший початок, автомобіль чекає початку.
* Тоді автомобіль приїжджає в місце призначення.
* Тоді процес повторяється для наступної призначеної поїздки доки всі заброньовані поїздки не є виконані, або симуляціє досягає часу T.
* Всі поїздки, що не були призначені ігноруються.

Вхідні дані

Перший рядок вхідного файлу містить наступні цілі числа розділені пробілами:

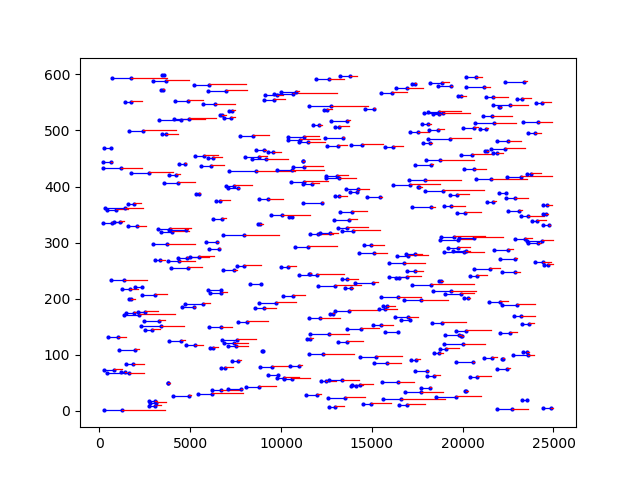
* R – кількість рядків в сітці ( 1 <= R <= 10000)
* C – кількість колонок в сітці ( 1 <= C <= 10000)
* F – кількість автомобілей в кортежі (1 <= F <= 1000)
* N – кількість поїздок ( 1 <= N <= 10000)
* B – бонусні бали за вчасний початок (1 <= B <= 10000)
* T – кількість часових кроків в симуляції (1<=T <=10^9)

N наступних рядків містять інформацію про індивідуальні поїздки, індесовані з 0 до N-1.

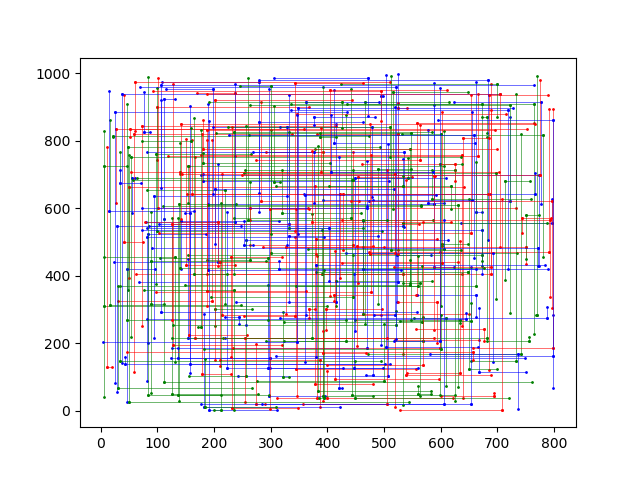
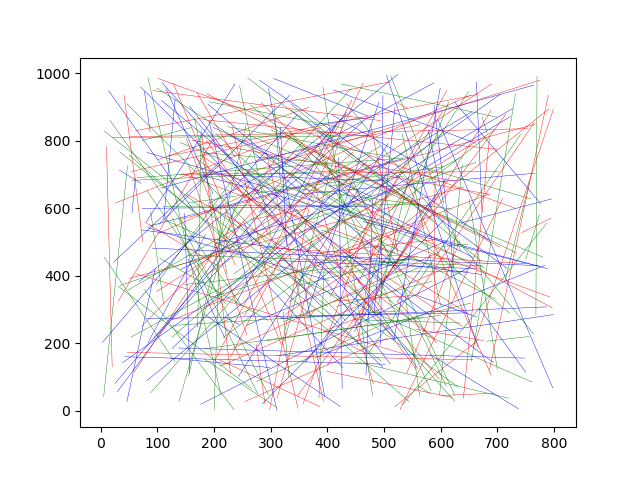
* a – рядок початку
* b – колонка початку
* x – рядок кінця
* y – колонка кінця
* s – найраніший початок
* f – найпізніший кінець

**Данні**

1. b\_should\_be\_easy.in



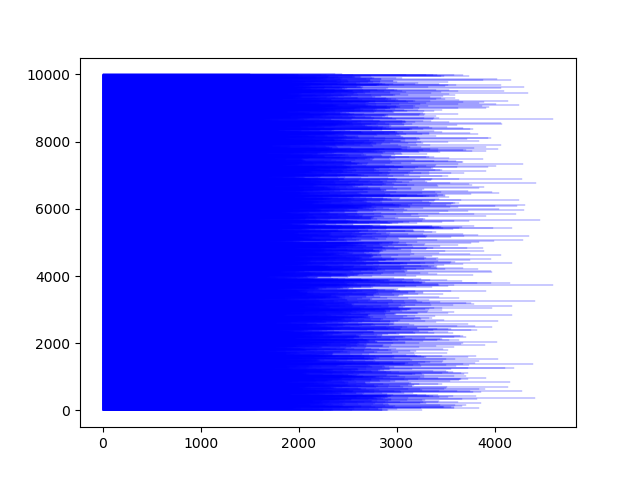
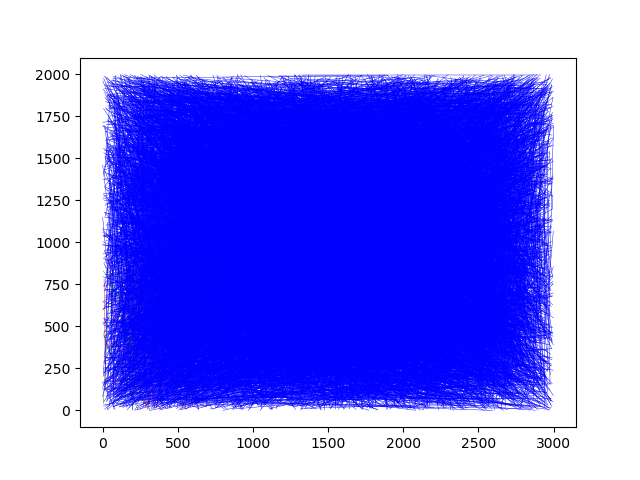
* Х – час від 0 до 25000
* Синя лінія це довжина поїздки припускаючи, що вона почалась вчасно
* Червона лінія частина часу, який залишився до найпізнішого кінця поїздки



* Х – місто columns, У – місто rows
* Лінія – поїздка від початкового до кінцевої точки Графік1 не враховуючи догори в місті Графік2 враховуючи дороги
* Колір лінії: голубий – найраніший час поїздки в межах від 0 до 1/3 всього часу симуляції , червоний – від 1/3 до 2/3 часу симуляції, зелений – 2/3 до кінця симуляції

**Висновок:** поїздки рівномірно розприділені по території міста, поїздки за довжиною досить короткі.

1. c\_no\_hurry.in

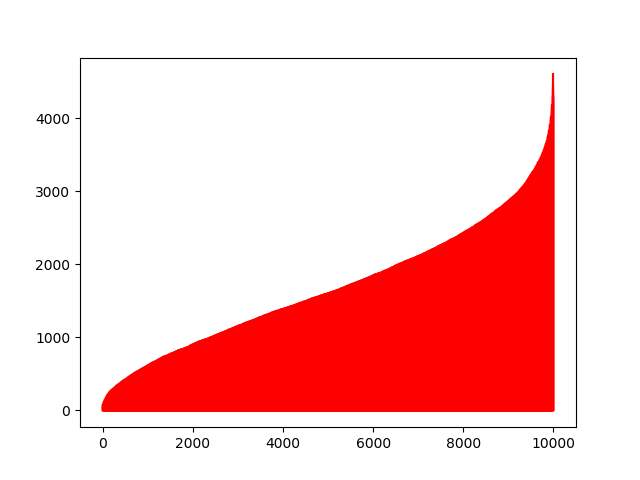
Графік 1

* Х – час від 0 до 25000
* Синя лінія це довжина поїздки припускаючи,

що вона почалась вчасно

Графік2

* Х – місто columns, У – місто rows
* Лінія – поїздка від початкового до кінцевої точки не враховуючи догори в місті
* Колір лінії: голубий – найраніший час поїздки в межах від 0 до 1/3 всього часу симуляції , червоний – від 1/3 до 2/3 часу симуляції, зелений – 2/3 до кінця симуляції

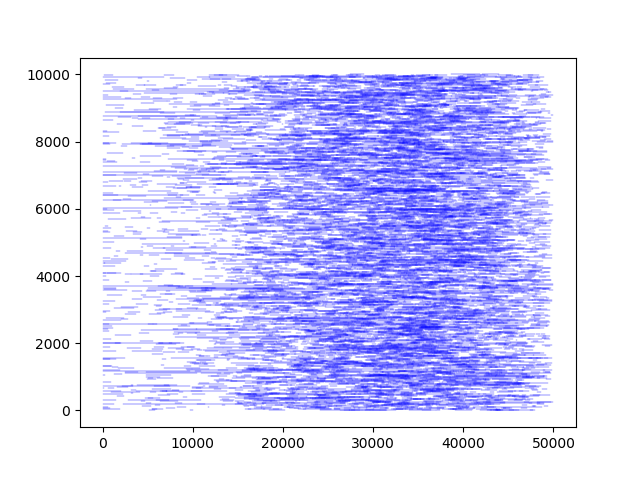


Графік 3

* Х – номер поїздки, у – довжина поїздки

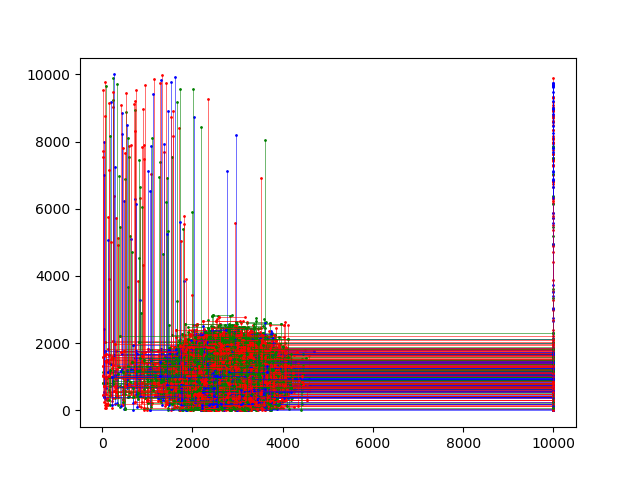
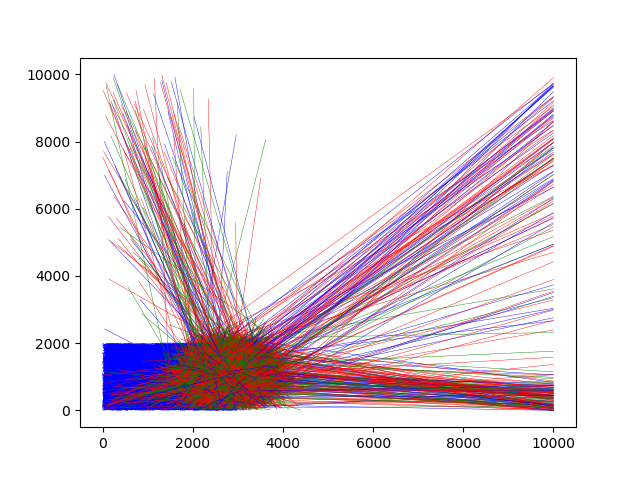
**Висновок:** для цих даних як можна побачити на Графіку 1 найраніший час поїздки 0 і бонуси за вчасний початок поїздки дуже малі (1 бонус), тому в нашому алгоритмі не потрібно концентруватись на вчасному початку поїздки. Для цих даних потрібно реалізувати алгоритм, який вибирає найдовші поїздки в першу чергу, для отримання великих бонусів відповідних до їх довжини. На Графіку 3 можна помітити, що поїздок довжиною від 2000 до 4500 близько 4000, тобто їх досить багато, а на Графіку 2 можна побачити, що ці поїздки доволі рівномірно розприділені по всій території міста.

1. d\_metropolis.in



Графік 1

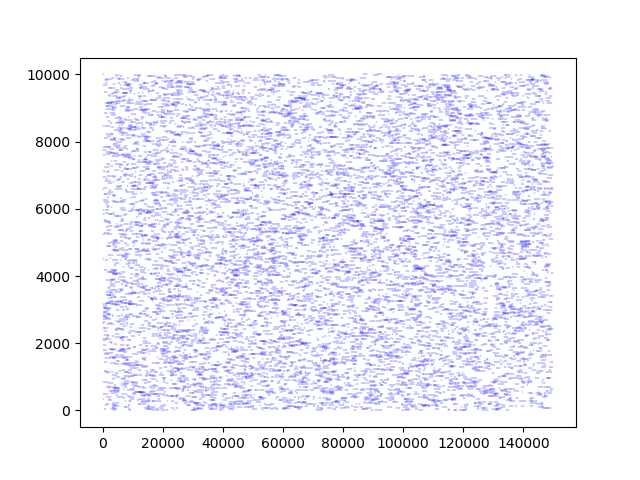
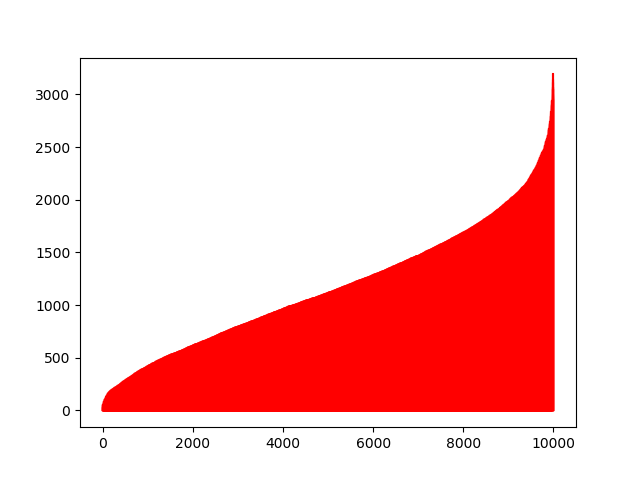
* Х – час від 0 до 50000
* Синя лінія це довжина поїздки припускаючи, що вона почалась вчасно

* Х – місто columns, У – місто rows
* Лінія – поїздка від початкового до кінцевої точки Графік1 не враховуючи догори в місті Графік2 враховуючи дороги
* Колір лінії: голубий – найраніший час поїздки в межах від 0 до 1/3 всього часу симуляції , червоний – від 1/3 до 2/3 часу симуляції, зелений – 2/3 до кінця симуляції

**Висновок:** на Графіку 1 помітно, що найбільша кількість поїздок припадає на час від 20000 до 45000, також з Графіку 3 бачимо, що поїдки сконцентровані в певній частині міста

1. e\_high\_bonus.in

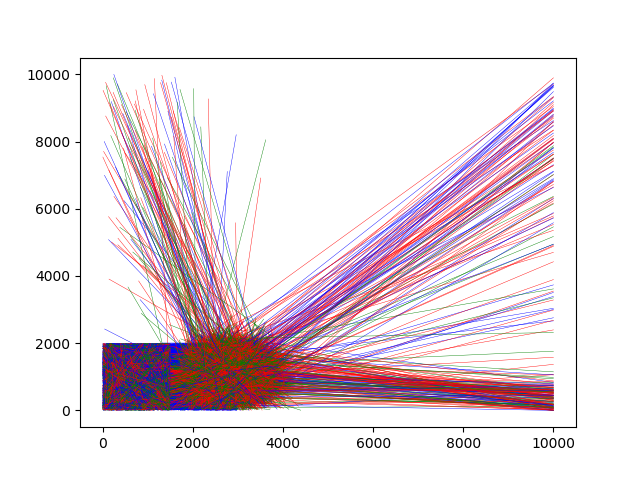
 

Графік 1

* Х – час від 0 до 150000
* Синя лінія це довжина поїздки припускаючи, що вона почалась вчасно

Графік2

* Х – номер поїздки, у – довжина поїздки



Графік 3

* Х – місто columns, У – місто rows
* Лінія – поїздка від початкового до кінцевої точки не враховуючи догори в місті
* Колір лінії: голубий – найраніший час поїздки в межах від 0 до 1/3 всього часу симуляції , червоний – від 1/3 до 2/3 часу симуляції, зелений – 2/3 до кінця симуляції

**Висновок:** на Графіку 3 помітно, що поїдки сконцентровані в певній частині міста, на Графіку 1 видно, що всі поїздки рівномірно розприділені у часі, на Графіку 2 бачимо, що дуже довгих (> 2000) поїздок досить мало приблизно 600, а оскільки в цьому місті високі бонуси, то при написанні алгоритму варто концентруватись на вчасному початку, а не на довжині поїздки.

**SimpleSortAlgorithm**

Універсальний найпростіший алгоритм.

При створенні списку поїздок сортуємо його за найранішим часом поїздки

На кожен момент часу для кожної вільної машини шукає поїздку наступним способом:

1. З списку поїздок беремо першу
2. Перевіряємо чи встигне машина доїхати до початкової точки і здійснити поїдку до найпізнішого часу закінчення поїздки.
3. Якщо поїздка підходить, то присвоюємо машині цю поїздку і видаляємо поїздку зі списку поїздок
4. Якщо поїздка n не підходить переходимо до пункту 1 і вибираємо n+1 поїздку
5. Якщо поїздка так і не знайшлась, то на цей момент часу не присвоюємо цій машині поїздку і переходимо до наступної машини
6. Коли всім машинам визначені поїздки переходимо на наступний момент часу.

**LongestRidesAlgorithm**

При створенні списку поїздок сортуємо їх за довжиною від найдовшої до найкоротшої.

На кожен момент часу для кожної вільної машини шукає поїздку наступним способом:

1. Якщо кількість поїздок, що залишилась на даний момент часу більша від третини всього числа поїздок в місті:
2. Із відсортованого списку поїздок беремо першу
3. Перевіряємо чи встигне машина доїхати до початкової точки і здійснити поїдку до найпізнішого часу закінчення поїздки.
4. Якщо поїздка підходить, то присвоюємо машині цю поїздку і видаляємо поїздку зі списку поїздок
5. Якщо поїздка n не підходить переходимо до пункту 1 і вибираємо n+1 поїздку
6. Якщо поїздка так і не знайшлась, то на цей момент часу не присвоюємо цій машині поїздку і переходимо до наступної машини
7. Якщо кількість поїздок, що залишилась на даний момент часу менша ніж третина всіх поїздок в місті:

1. Розділяємо відсортований список поїздок на три частини

2. Бермо першу частину і сортуємо її за відстанню до розташування нашої машини від найближчої до найдальшої

3. Беремо першу поїздку із цієї відсортованої частини

4. Якщо поїздка підходить, то присвоюємо машині цю поїздку і видаляємо поїздку зі списку поїздок

5. Якщо поїздка не підходить, то переходимо до наступної поїздки в цій частині

6. Якщо в першій частині правильна поїздка не знайдена переходимо до другої частини списку

7. Якщо поїздка не була знайдена в жодній з трьох частин, то не присвоюємо на цей момент часу поїздку даній машині.

**TimePeriodAlgorithm**

Для представлення поїздок створюємо словник з ключами 1-10 відповідно до часових періодів і додаємо поїздки в цей словник відповідно до найшвидшого часу їх початку.

Сортуємо значення словника за ключами від 1-8 за довжиною від найдовшого до найкоротшого, і значення за ключами від 8-10 сортуємо навпаки від найкоротшого до найдовшого. Робимо так, адже спочатку в нас буде достатньо часу, щоб брати довгі поїздки, а в останні 2 часові періоди часу на довгі поїздки буде мало, тому будемо брати спочатку найкоротші.

На кожен момент часу для кожної вільної машини шукає поїздку наступним способом:

1. Визначаємо для машини в даний час в якому часовому періоді вона зараз
2. За періодом доступаємось до списку поїздок в словнику, найшвидший початок яких також в даному часовому періоді
3. Цей відсортований список поїздок розділяємо на три частини
4. Бермо першу частину і сортуємо її за відстанню до розташування нашої машини від найближчої до найдальшої
5. Беремо першу поїздку із цієї відсортованої частини
6. Якщо поїздка підходить, то присвоюємо машині цю поїздку і видаляємо поїздку зі списку поїздок
7. Якщо поїздка не підходить, то переходимо до наступної поїздки в цій частині
8. Якщо в першій частині правильна поїздка не знайдена переходимо до другої частини списку
9. Якщо поїздка не була знайдена в жодній з трьох частин, то переходимо на наступний часовий період і доступаємось до наступного списку поїздок за цим ключем і повторюємо наступні кроки
10. Якщо поїздка не знайшлась, то не присвоюємо цій машині на цей момент часу поїздку

**CombinedAlgorithm**   
Алгоритм є скомбінованою версією усіх попередніх.   
Взалежності від типу вхідних даних, використовується алгоритм, який найкраще працює з цими даними.   
Для файлу should\_be\_easy – SimpleSortAlgorithm, no\_hurry – LongestRidesAlgorithm, metropolis – TimePeriodAlgorithm, high\_bonus – SimpleSortAlgorithm.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Файл | SimpleSortAlgorithm | LongestRidesAlgorithm | TimePeriodAlgorithm | CombinedAlgorithm |
| a\_example.in | 4 | 4 | 4 | 4 |
| b\_should\_be\_easy.in | 176877 | 171832 | 176877 | 176877 |
| c\_no\_hurry.in | 8069141 | 9549198 | 1696486 | 9549198 |
| d\_metropolis.in | 9720814 | 9618544 | 13481369 | 13481369 |
| e\_high\_bonus.in | 21465945 | 21187629 | 21438274 | 21465945 |
| sum | 39432781 | 40527207 | 36793006 | 44673389 |

**Висновок:**