SOAD – LAB 1

1. El objetivo del programa

Para la primera sesión del laboratorio, hemos elegido la primera práctica de la asignatura de Inteligencia Artificial. En dicha práctica creamos una “IA” que genera las rutas de furgonetas reponedoras para el servicio de Bicing, en el que dado un tamaño de problema, en número de furgonetas, número de bicicletas y paradas, el sistema tenía que planificar la ruta más óptima para cumplir con la demanda de todas las paradas en la medida de lo posible.

Dados los costes computacionales de los algoritmos y heurística usados, en los que generamos todos los posibles estados en cada paso hasta llegar a una solución óptima y por tanto computamos todos los hijos de cada estado para saber si es esa la mejor configuración. Los costes principales que hay que medir són el consumo de CPU y Memoria, pues un mal uso de estas pueden ralentizar mucho nuestro programa.

Este programa no tiene ningún input pues viene definido en el problema en sí y hay que modificar esos parámetros dentro del propio código que ejecutaremos, el sistema nos dará una salida en formato gráfico, donde se podrán visualizar las diferentes paradas de bicing y las rutas que siguen las furgonetas.

2. ¿Qué información queremos obtener?

Sería interesante medir el uso que hace el programa de la CPU para varios tamaños del problema, por ejemplo qué pasa si doblo todos los parámetros del problema? Consumo el doble de CPU? Otra medición interesante es el espacio en memoria que ocupa la ejecución, ¿siempre es el mismo? Si hay más objetos usamos un tamaño mayor de memoria? ¿Cuánto tiempo tarda el programa en ejecutarse? depende de si el tamaño es más grande? ¿Cuánto tiempo tarda el programa en pintar la salida gráfica? si aumenta el tamaño del problema tarda más? sigue la ejecución mientras está abierta la ventana gráfica del programa?

Entonces necesitamos saber el consumo de la CPU que hace nuestro programa, la memoria que ocupa en ejecución, el tiempo que tarda en ejecutarse y cuando termina realmente el programa.

3. Las herramientas que usaremos

Toda esta información la podemos obtener usando diferentes herramientas que nos ofrece el propio sistema operativo.

Una de las herramientas que vamos a utilizar es vmstat que nos permite ver el estado del sistema desde los puntos de vista que queremos analizar, CPU y memoria.

También usaremos directrices en el código para saber el tiempo de uso total del programa y el tiempo de computación (en ms). Hemos creado 4 variables long que reciben el valor de System.currentTimeMillis que nos da el tiempo actual del sistema en milisegundos, de esta manera si sabemos el tiempo de inicio y el de final podemos saber cuánto ha tardado el programa.

4. Resultados

La primera prueba será con los valores base del problema 10 estaciones, 25 bicis y 5 furgonetas, seguidamente haremos lo mismo para 20 estaciones 50 bicis y 10 furgonetas y para terminar haremos la misma prueba con 100 estaciones 250 bicis y 50 furgonetas.

Dado que no hay más tiempo para usar otras métricas y analizar todo lo que queríamos sólo usaremos vmstat y las directrices dentro del propio código fuente del programa.

Aunque el uso de vmstat nos devuelve la información del sistema podemos ver como afecta nuestro programa a este si ejecutamos el programa sin ningúna otra aplicación en uso o en segundo plano. Para las ejecuciones únicamente hemos usado visual studio code, para facilitar la ejecución del proyecto que de otra manera se tendría que ir recopilando y rehaciendo todo el rato.

Cómo información adicional, hemos ejecutado el programa únicamente en java.

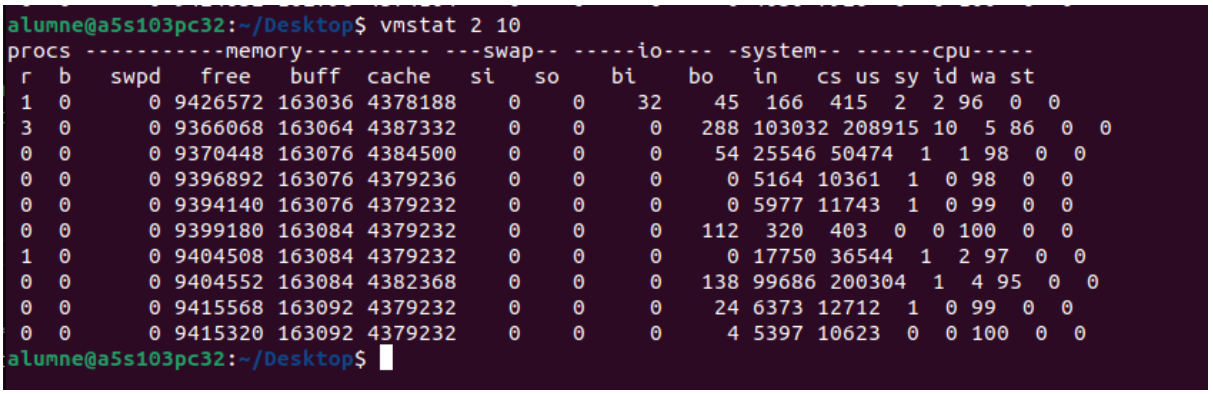
La llamada a vmstat con los parámetros 2 y 10 nos permite hacer la llamada 10 veces, una cada 2 segundos, y de esta manera asegurar que vmstat se llama con nuestro programa en ejecución.

Primer experimento

```
alumne@a5s103pc32:~/Desktop$ vmstat 2 10
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 9477280 162740 4373156    0    0    32    45  132  347  2  2 96  0  0
 0  0      0 9396108 162780 4382420    0    0     0   216 77716 158298  7  4 89  0  0
 1  0      0 9397508 162780 4377300    0    0     0     0 100277 203070  2  4 94  0  0
 0  0      0 9407456 162780 4374180    0    0     0     0 79395 159425  1  3 96  0  0
 1  0      0 9407152 162784 4374164    0    0     0    34 28432 56866  1  1 98  0  0
 0  0      0 9416524 162784 4374164    0    0     0     0 15083 30292  1  1 99  0  0
 1  0      0 9416524 162784 4374164    0    0     0     0 5893 11618  0  0 100  0  0
 0  0      0 9424628 162796 4374156    0    0     0    42 25444 51403  2  1 97  0  0
 0  0      0 9424632 162796 4374164    0    0     0     0 5041 9899  0  0 100  0  0
 0  0      0 9424632 162796 4374164    0    0     0     0 4056 7926  0  0 100  0  0
alumne@a5s103pc32:~/Desktop$
```

En este primer experimento el tiempo de computación (tiempo hasta encontrar la solución) ha sido de 296 ms y el tiempo total de 2376 ms.
El tiempo de ejecución de código fuera del kernel es muy poco dado que el tiempo total de ejecución del programa ha sido de 2376 ms, 2 segundos aproximadamente. Se ve claramente que la segunda llamada a vmstat ha sido en la ejecución de nuestro programa ya que tenemos el mayor tiempo de ejecución de código no-kernel y código kernel 7 y 4 respectivamente. Esto se debe a que hacemos una llamada a System.currentTimeMillis(), que accede a la información del Kernel para servir los datos del tiempo del sistema.

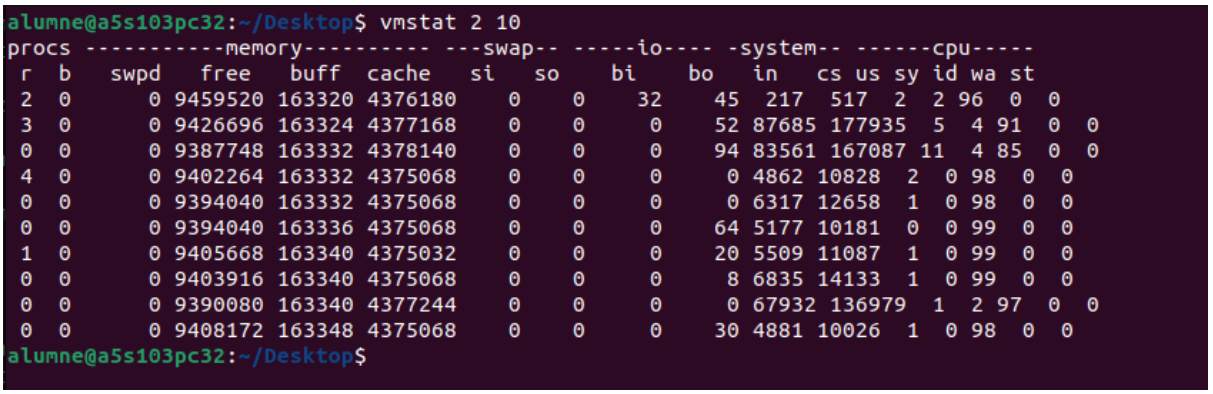
Segundo experimento



```
alumne@a5s103pc32:~/Desktop$ vmstat 2 10
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 9426572 163036 4378188    0    0    32    45  166  415  2  2 96  0  0
 3  0      0 9366068 163064 4387332    0    0     0   288 103032 208915 10  5 86  0  0
 0  0      0 9370448 163076 4384500    0    0     0    54 25546 50474  1  1 98  0  0
 0  0      0 9396892 163076 4379236    0    0     0     0 5164 10361  1  0 98  0  0
 0  0      0 9394140 163076 4379232    0    0     0     0 5977 11743  1  0 99  0  0
 0  0      0 9399180 163084 4379232    0    0     0   112  320  403  0  0 100  0  0
 1  0      0 9404508 163084 4379232    0    0     0     0 17750 36544  1  2 97  0  0
 0  0      0 9404552 163084 4382368    0    0     0   138 99686 200304  1  4 95  0  0
 0  0      0 9415568 163092 4379232    0    0     0    24 6373 12712  1  0 99  0  0
 0  0      0 9415320 163092 4379232    0    0     0     4 5397 10623  0  0 100  0  0
alumne@a5s103pc32:~/Desktop$
```

En este segundo experimento el tiempo de computación ha sido de 368 ms y el tiempo total de 4519 ms.
Del mismo modo que en el primer experimento la segunda llama a vmstat se hace en tiempo de ejecución de nuestro programa.

Tercer experimento



```
alumne@a5s103pc32:~/Desktop$ vmstat 2 10
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0      0 9459520 163320 4376180    0    0    32    45  217  517  2  2 96  0  0
 3  0      0 9426696 163324 4377168    0    0     0    52 87685 177935  5  4 91  0  0
 0  0      0 9387748 163332 4378140    0    0     0    94 83561 167087 11  4 85  0  0
 4  0      0 9402264 163332 4375068    0    0     0     0 4862 10828  2  0 98  0  0
 0  0      0 9394040 163332 4375068    0    0     0     0 6317 12658  1  0 98  0  0
 0  0      0 9394040 163336 4375068    0    0     0    64 5177 10181  0  0 99  0  0
 1  0      0 9405668 163340 4375032    0    0     0    20 5509 11087  1  0 99  0  0
 0  0      0 9403916 163340 4375068    0    0     0     8 6835 14133  1  0 99  0  0
 0  0      0 9390080 163340 4377244    0    0     0     0 67932 136979  1  2 97  0  0
 0  0      0 9408172 163348 4375068    0    0     0    30 4881 10026  1  0 98  0  0
alumne@a5s103pc32:~/Desktop$
```

En este último experimento el tiempo de computación ha sido de 749 ms y el tiempo total de 18416 ms.
Al igual que en los anteriores experimentos vemos como la tercera llamada de vmstat es cuando estamos ejecutando nuestro programa.

Conclusiones de los experimentos

Podemos ver por tanto como según vamos incrementando el tamaño del problema aumenta el tiempo de computación y pintado de solución y por tanto el tiempo de ejecución del programa. Hemos podido observar cómo se cumple una de nuestras hipótesis iniciales y es que a mayor tamaño tenga el problema más aumenta el tiempo de ejecución y más aumenta el tiempo que usa la CPU nuestro programa. También hemos comprobado que el programa no finaliza hasta acabar de pintar toda la solución de manera que consumimos más recursos si hay que pintar soluciones más grandes.

Dadas las limitaciones de tiempo no hemos podido analizar correctamente el apartado de memoria que usa nuestro programa pero podemos observar en las llamadas a vmstat que tras ejecutar el programa hay menos memoria libre.

5. Limitaciones

Durante la realización de la práctica la principal limitación ha sido la falta de información que podemos obtener de nuestro propio programa desde el sistema operativo, no tenemos instrucciones o comandas de terminal que nos permitan acceder solo a datos de ese programa, sino que debemos implementar métodos para hacer ese análisis en el propio código que ejecutamos, como por ejemplo el System.currentTimeMillis().

La comanda vmstat es útil para conocer el estado del sistema en un momento determinado y se puede programar para que salte durante nuestra ejecución, pero no es posible hacer la llamada solo para nuestro proceso en funcionamiento y hay que preparar el entorno para ello. Tenemos opciones para depurar la información y obtener lo que queremos en cada momento pero no es suficiente si solo analizamos el impacto de un programa en el sistema y no un conjunto de ellos.