Instrucciones de uso

El presente documento es un manual para el uso de los scripts adjuntos. Para los análisis de complejidad, experimentos, resultados y conclusiones del miniproyecto, refiérase a `informe.pdf`.

El archivo comprimido enviado contiene:

- El script `proy.py`, que soluciona el problema planteado.
- El script `gen.py`, que genera entradas del problema que `proy.py` entiende.
- Archivos de entrada `in1`, `in2`, `in3`, `in4` que `proy.py` entiende. Estos son ejemplos de problemas.
- La hoja de Excel `data.xlsx` que contiene los resultados en bruto de las pruebas realizadas.
- El documento `informe.pdf`, que contiene un análisis detalladado de la complejidad de ejecución de la solución, resultados de pruebas y conclusiones en base a los resultados.
- El presente documento.

Tanto `proy.py` como `gen.py` fueron escritos y probados en Python 3.7.7.

Para resolver el problema con datos de entrada registrados en el archivo `in1` ordenando con CountingSort, por ejemplo, se debe ejecutar el comando `python3 proy.py count in1`.

proy.py

El script `proy.py` tiene a su disposición tres algoritmos de ordenamiento. El primer argumento debe indicarle qué algoritmo debe usar, con las banderas `count` para CountingSort, `merge` para MergeSort, o `insert` para InsertionSort.

```
proy.py <algortimo>
proy.py <algoritmo> <archivo_entrada>
    Donde <algorimo> := count | merge | insert
```

El script `proy.py` entradas de dos posibles formas, leyendo de la consola o leyendo de un archivo.

Digitar la entrada por la consola es el comportamiento predeterminado, solo se debe especificar el algoritmo, por ejemplo `python3 proy.py merge`.

Si la entrada del problema está en un archivo, el archivo se debe especificar como última bandera, por ejemplo `python3 proy.py insert in2`.

De cualquiera de las dos formas, `proy.py` requiere que la entrada esté formateada de una manera específica.

- La primera línea corresponde a n, el número de animales.
- La segunda línea corresponde a m, el número de partes.
- La tercera línea corresponde a k, el número de escenas por parte.
- La cuarta línea corresponde a los nombres de los animales, separados por una coma y un espacio.
- La quinta línea corresponde a las grandezas de los animales, separados por una coma y un espacio. Deben ser números parseables por la función `int()` de Python.
- La sexta linea corresponde a la apertura, donde las escenas son separadas por una coma y un espacio.
- Cada una de las siguientes $m - 1$ líneas corresponde a una parte, donde las escenas son separadas por una coma.

Las escenas son delimitadas por llaves, y los animales adentro de ellos son son separados por un espacio y una coma. Ej. {`tapir, nutria, perro`} es una escena válida.

Por ejemplo, para que `proy.py` etienda el problema propuesto dentro del documento del enunciado, este se debe formatear así:

```
6
3
2
gato, libelula, ciempies, nutria, perro, tapir
3, 2, 1, 6, 4, 5
{tapir, nutria, perro},{tapir, perro, gato},{ciempies, tapir,gato},{gato, ciempies, libelula}
{tapir, nutria, perro},{ciempies, tapir, gato}
{gato, ciempies, libelula}, {tapir, perro, gato}
```

Este último se puede encontrar en `in1`.

gen.py

El script `gen.py` genera entradas al problema aleatorias pero consistentes. Si no se le pasan argumentos, elige aleatoriamente $n, m, k \in N$ tal que $3 \leq n < 100$, $2 < m \leq 60$ y $1 \leq k \leq n$.

```
gen.py
gen.py <n> <m> <k>
```

Estos parámetros tambien se pueden especificar; debe hacerse en ese orden. Por ejemplo, para $n = 6, m = 3, k = 2$ se debe ejecutar `python3 gen.py 6 3 2`

Este último comando genera:

```
6
3
2
vo, jo, ha, ho, ve, yo
5, 6, 3, 2, 1, 4
{vo, ho, yo}, {ho, vo, yo}, {ha, ve, jo}, {vo, jo, ha}
{ha, ve, jo}, {ho, vo, yo}
{vo, ho, yo}, {vo, jo, ha}
```

`python3 gen.py 27 3 2` genera:

```
27
3
2
zoso, goba, tose, neoe, eeqi, reni, leoa, madu, iari, wexu, woxa, boui, ieea, cuvo, zuro,
fefa, bido, ooze, sexa, neqa, eelo, auho, faha, muni, yauo, biyi, ciha
4, 25, 24, 6, 5, 12, 3, 14, 9, 1, 10, 19, 11, 7, 18, 8, 13, 21, 15, 2, 16, 22, 17, 20, 26,
27, 23
{reni, goba, biyi}, {sexa, zuro, ciha}, {yauo, madu, reni}, {ieea, ooze, madu}
{sexa, zuro, ciha}, {reni, goba, biyi}
{ieea, ooze, madu}, {yauo, madu, reni}
```

Nos aseguramos que la complejidad de la generación fuera $O(n + mk)$, por lo que no debería tener mayor impacto en los resultados de las pruebas. Los nombres de los animales se generan proceduralmente (si tu nombre no no tiene dos consonantes seguidas, también puede salir generado! Solo son necesarios suficientes animales).

Los dos scripts se pueden unir en cualquier interprete de comandos POSIX con `|`. Por ejemplo, para solucionar un problema similar al anterior con CountingSort, ejecútese

`python3 gen.py 27 3 2 | python3 proy.py count`