# Algorísmia i Programació III

## – PROJECTE 2025 –

Una coneguda marca automobilística ens demana ajuda per tal de millorar l'eficiència en la producció del seu model de més èxit. Quan compren un cotxe, els clients poden optar per instal·lar-hi una gran quantitat de millores (sostre solar, GPS, aparcador automàtic, sensors de distància, etc.), definint així el que anomenen una classe: un subconjunt de millores.

Tot els cotxes passen per la mateixa línia de producció, on per cada millora existeix una única estació encarregada de la seva instal·lació als cotxes que la requereixen. Cada estació $e$ té associats dos enters $c_e$ i $n_e$ indicant que, per qualsevol conjunt de $n_e$ cotxes consecutius, com a molt $c_e$ d'ells poden requerir la instal·lació de la millora. No obstant, aquesta restricció és tova, és a dir, permetem finestres de $n_e$ cotxes consecutius amb més de $c_e$ cotxes on s'hi hagi d'instal·lar la millora, però aquestes finestres tindran una penalització. La penalització total s'obté sumant les penalitzacions associades a cadascuna de les estacions. L'objectiu d'aquest projecte és, donat el nombre de cotxes de cada classe que s'han de produir i la definició de cada classe, ordenar-los de manera que es minimitzi el nombre total de penalitzacions.

De forma més precisa, si una estació $e$ ha de processar una finestra de $n_e$ cotxes consecutius, amb $k$ d'ells requerint la millora, aquesta finestra incorrerà en una penalització de $\max\{k - c_e, 0\}$ euros. Per exemple, si una estació pot processar com a molt 1 cotxe de cada 3, la seqüència

| ○ | ○ | × | ○ | × | ○ | ○ | × | × | × | ○ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

on × indica que el cotxe requereix la millora i ○ que no la requereix, incorre en una penalització de 5 euros: les finestres $[3 \dots 5], [7 \dots 9]$ i $[9 \dots 11]$ incorren en una penalització de 1 euro cadascuna, mentre que la finestra $[8 \dots 10]$ contribueix amb una penalització de 2 euros. Un altre detall a considerar és que les finestres incompletes a l'inici o al final de la seqüència també es tenen en compte. És a dir, la seqüència

| × | × | ○ | ○ | × | ○ | × | × |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

incorre en una penalització de 5: les finestres completes $[1 \dots 3], [5 \dots 7], [6 \dots 8]$ i les incompletes $[1 \dots 2], [7 \dots 8]$. La penalització total és la suma de les penalitzacions associades a cada estació.

**Format entrada:**

Una entrada comença amb 3 naturals estrictament majors que zero: $C$, $M$, $K$ que representen el nombre de cotxes, millores i classes. A continuació trobarem $M$ enters estrictament majors que zero, que corresponen a la $c_e$ de cada millora, seguits de $M$ enters estrictament majors que zero indicant la corresponent $n_e$, amb $c_e \leq n_e$.

Finalment, trobarem $K$ línies que especifiquen les classes: cada línia comença amb un natural que identifica la classe. Aquest nombre estarà entre 0 i $K - 1$ i podeu assumir que les línies

estaran ordenades per aquest nombre. A continuació, cada línia conté un natural estrictament major que zero que indica el nombre de cotxes d'aquesta classe que cal produir. Finalment, segueixen $M$ Booleans (1/0), que indiquen si la classe requereix la $i$-èsima millora o no.

**Format de sortida:**

Començarà amb dos nombres: un enter que correspon a la penalització de la solució obtinguda i un double (amb com a molt 1 decimal), que indicarà els segons que han estat necessaris per a trobar aquesta solució. A continuació hi haurà $C$ enters, tots ells entre 0 i $K-1$ que indiquen quin és l'ordre de producció dels cotxes. És a dir, l'enter $i$-èsim indica la classe del cotxe que es produeix en la posició $i$-èsima. Un exemple d'entrada/sortida el podeu veure a continuació:

*Entrada:*

```
10 5 3
1 1 1 2 1
2 2 2 3 2
0 4 1 1 0 0 1
1 3 0 1 0 1 0
2 3 0 0 1 0 0
```

*Sortida:*

```
3 2.3
0 1 0 1 2 0 2 0 2 1
```

**Tasques a realitzar:**

El projecte es realitzarà en parelles. Heu d'entregar tres arxius (i només tres!) escrits en Python i que compilin en Codon, amb **exactament** el noms *exh.cc, greedy.cc* i *mh.cc*. No acceptarem arxius comprimits. Els tres programes han de llegir un arxiu d'entrada pel canal estàndard d'entrada i han d'escriure la sortida en un arxiu que es donarà com a primer argument de l'executable a la línia de comandes. És a dir, si l'executable té nom *exh*, aleshores

```
./exh sort.txt < ent.txt
```

ha de llegir un arxiu d'entrada amb nom *ent.txt* i escriure el resultat en un arxiu de sortida *sort.txt*. Evidentment, l'anterior comanda ha de funcionar per a qualsevol nom d'arxiu d'entrada i sortida. No tenir aquest comportament implicarà **sense cap excepció** una penalització a la nota. Adjuntem un arxiu `in-out.py` que mostra com escriure en un arxiu de sortida. L'entrega es farà a través del Racó i la data límit és **dimecres 7 de gener**.

- *exh.cc*: ha d'implementar una cerca exhaustiva. Cada vegada que obtingui una solució millor, l'ha de sobreescriure a l'arxiu de sortida. És a dir, volem que si avortem el programa, dins l'arxiu de sortida hi hagi la millor solució trobada fins al moment.
- *greedy.cc*: ha d'implementar un algorisme golafre. Degut a la dificultat computacional del problema, no s'espera que obtingui una solució òptima. El que sí demanarem és que sigui ràpid: hauria de trigar menys de mig segon en les instàncies més grans que us proporcionem.
- *mh.cc*: ha d'implementar una metaheurística de les explicades a classe (excepte Basic Local Search). Com amb la cerca exhaustiva, volem que cada vegada que obtingui una solució millor, la sobreescrigui a l'arxiu de sortida.

**Criteris d'avaluació:**

Dels 2 punts del projecte, 0.5 punts tindran en compte la llegibilitat del codi i la seva correctesa, i la resta la qualitat de les solucions obtingudes i el temps necessari per a calcular-les. Us pot ser útil saber que no deixarem executar cap programa més d'un minut. Avaluarem el vostre codi sobre un joc de proves privat on els exemples són de mida similar al joc de proves públic que us facilitem. Per tal de comprovar que les vostres solucions són correctes podeu utilitzar el checker que us proporcionem. Compte, aquest checker no comprova que la vostra solució sigui òptima, sinó que comprova que el format de sortida sigui correcte i que la solució tingui el cost que dieu que té. Per a tenir certes garanties d'optimalitat, l'arxiu `resultsEasy.txt` conté els costos òptims d'un subconjunt del joc de proves.

El checker és un programa en C++ que cal compilar:

```
g++ checker.cc -o checker
```

Un cop compilat, podeu executar

```
./checker entrada.txt sortida.txt
```

comprovarà que `sortida.txt` és una solució correcta (potser no òptima) al problema `entrada.txt`.