Paradigmes i Llenguatges de Programació
Pràctica Prolog
Curs 2024-25

Capítol 1

Enunciat

1.1 Introducció

En aquesta pràctica implementarem diferents mètriques per calcular el grau de desordre d'una llista. El programa desenvolupat es podria aplicar sobre una baralla de cartes, una llista de participants en un esdeveniment, etc. L'únic que necessitem és definir la noció d'ordre entre dos elements de la llista. Per facilitar el procés de desenvolupament, ens abstraurem de quina informació conté la llista en particular, i treballarem sobre llistes de llargada N que contenen una permutació dels nombres entre 0 i $N-1$. **Podem assumir que N és múltiple de 2**. Es considerarà que una llista està ordenada si està ordenada de petit a gran, és a dir, si és la llista $[0, 1, \ldots, N-1]$.

1.2 Mètriques de desordre d'una llista

Considerarem les mètriques següents per calcular el grau de desordre. Per cada mètrica, el valor 0 significa que la llista està ordenada, i valors més grans representen major grau de desordre. El valor més gran possible de desordre és diferent per cada mètrica.

1.2.1 Nombre d'elements desubicats

La mètrica *des* correspon al nombre d'elements que no estan a la posició que els hi pertoca de la llista ordenada.

Exercici 1. *Implementa:*

```
%nombre_desubicats(+L,?Des)
%El nombre d'elemens que no es troben a la posicio correcta de la llista L es Des
```

```
| ?- nombre_desubicats([0,1,2,3],Des).
Des = 0
yes
| ?- nombre_desubicats([0,3,1,2],Des).
Des = 3
yes
| ?- nombre_desubicats([2,7,3,4,5,8,6,9,1,0],Des).
Des = 9
yes
```

1.2.2 Suma de desplaçaments

La mètrica *sum* correspon a la suma de diferències (en valor absolut) entre la posició que ocupa un nombre i la posició que hauria d'ocupar.

Exercici 2. *Implementa:*

```
%suma_desplacaments(+L,Sum)
%Sum es la suma de diferencies (en valor absolut) entre la posicio que ocupa un nombre a L i la
    posicio que hauria d'ocupar.
```

```
| ?- suma_desplacaments([0,1,2,3],Sum).
Sum = 0
yes
| ?- suma_desplacaments([1,0,2,3],Sum).
Sum = 2
yes
| ?- suma_desplacaments([2,7,3,4,5,8,6,9,1,0],Sum).
Sum = 32
yes
```

1.2.3 Nombre mínim de passos per a ordenar

Les dues mètriques anteriors tenen mancances a l'hora d'identificar el que el sentit comú ens portaria a identificar com a llistes poc desordenades. Considereu per exemple les tres llistes següents, que tenen valor alt tant de *des* com de *sum*, però que per exemple, en una baralla de cartes, representarien baralles massa poc barrejades:

- $[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]$: $Des = 10$, $Sum = 50$.
- $[1, 3, 5, 7, 9, 0, 2, 4, 6, 8]$: $Des = 10$, $Sum = 30$.
- $[5, 6, 7, 8, 9, 0, 1, 2, 3, 4]$: $Des = 10$, $Sum = 50$.

Definirem la mètrica *pas* com **el nombre mínim de passos necessaris per ordenar la llista**. Calcular la mètrica *pas* correspon a un problema de *planning*[1]. Aquesta mètrica és paramètrica a quines accions es poden realitzar en cada pas. Considerarem tres accions: *inserir*, *capgirar* i *intercalar*.

Inserir

Aquesta acció s'inspira en l'algorisme d'ordenació per inserció, però en lloc d'inserir els elements d'un en un, es permet inserir una subllista ordenada. Dividirem els elements d'una llista en quatre subllistes consecutives: `Prefix1`, `Prefix2`, `Fragment` i `Sufix`. Definirem `Prefix` com la concatenació de `Prefix1` i `Prefix2`. Sabem que tota llista sempre té un `Prefix` ordenat de com a mínim un element. Aquesta acció consisteix en agafar una subllista ordenada (`Fragment`) que segueix immediatament el `Prefix`, i inserir-la dins de la posició correcta del `Prefix`, és a dir, entre `Prefix1` i `Prefix2`. Identificarem cada aplicació d'aquesta acció amb la tupla:

```
pas_inserir(Prefix1,Prefix2,Fragment,Sufix)
```

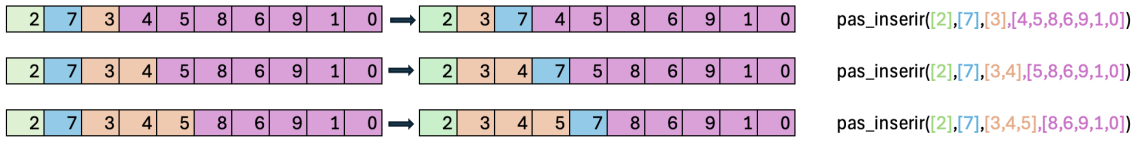
Per poder-la aplicar, cal complir les condicions següents:

- `Prefix2` no és buida, però `Prefix1` pot ser-ho.

[1] `https://en.wikipedia.org/wiki/Automated_planning_and_scheduling`

- L'últim element de `Prefix1` (si no és buit) és més petit que el primer element de `Fragment`.
- L'últim element de `Fragment` és més petit que el primer element de `Prefix2`.
- `Prefix` és la concatenació de `Prefix1` i `Prefix2`, i `Prefix` està ordenat.
- `Fragment` està ordenat.

Les possibles aplicacions de l'acció `inserir` sobre la llista $[2, 7, 3, 4, 5, 8, 6, 9, 1, 0]$ són exactament les següents:



Exercici 3. *Implementa:*

```
%a_inserir(+L,?L2,?Pas)
%L2 es el resultat d'aplicar l'accio inserir a L, i Pas conte la tupla pas_inserir(Prefix1,Prefix2,
    Fragment,Sufix)
```

```
| ?- a_inserir([2,7,3,4,5,8,6,9,1,0],L2,Pas).
L2 = [2,3,7,4,5,8,6,9,1,0]
Pas = pas_inserir([2],[7],[3],[4,5,8,6,9,1,0]) ? ;
L2 = [2,3,4,7,5,8,6,9,1,0]
Pas = pas_inserir([2],[7],[3,4],[5,8,6,9,1,0]) ? ;
L2 = [2,3,4,5,7,8,6,9,1,0]
Pas = pas_inserir([2],[7],[3,4,5],[8,6,9,1,0]) ? ;
no
```

Capgirar

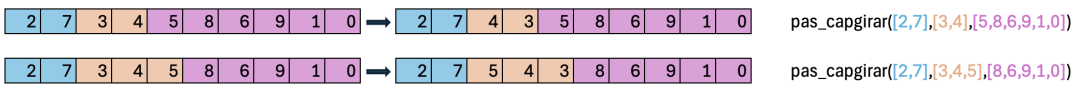
Aquesta acció consisteix en agafar un **Fragment** ordenat (pot ser creixent però **també decreixent**) de nombres consecutius i capgirar-lo. **El fragment com a mínim ha de ser de dos elements**. Identificarem cada aplicació d'aquesta acció amb la tupla:

`pas_capgirar(Prefix,Fragment,Sufix)`

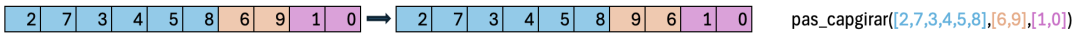
Distingirem 5 subtipus d'accions diferents, dissenyades per tal que el resultat d'aplicar una acció sigui més proper a una llista ordenada que la llista original. A continuació s'especifiquen les restriccions que ha de complir una llista per poder-hi aplicar cadascuna de les subaccions:

Capgirar tot: Es capgira tota la llista (és a dir, el prefix i el sufix són buits). A la llista original, el fragment ha d'estar ordenat **decreixent**.

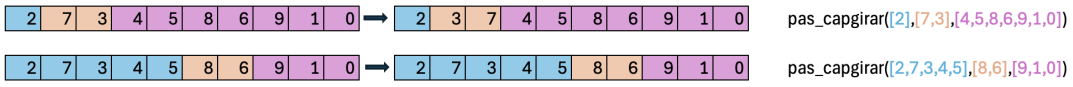
Capgirar creix esquerra: Es capgira un fragment ordenat creixent. El prefix no és buit, i l'últim element del prefix és més gran que l'últim element del fragment.



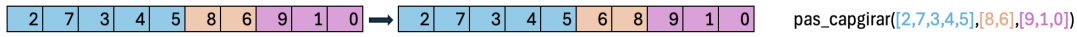
Capgirar creix dreta: Es capgira un fragment ordenat creixent. El sufix no és buit, i el primer element del sufix és més petit que el primer element del fragment.



Capgirar decreix esquerra: Es capgira un fragment ordenat decreixent. El prefix no és buit, i l'últim element del prefix és més petit que l'últim element del fragment.



Capgirar decreix dreta: Es capgira un fragment ordenat decreixent. El sufix no és buit, i el primer element del sufix és més gran que el primer element del fragment.



Exercici 4. *Implementa:*

```
%a_capgirar(+L,?L2,Pas)
%L2 es el resultat d'aplicar alguna de les subaccions de capgirar a L, i Pas conte la tupla
    pas_capgirar(Prefix,Fragment,Sufix)
```

```
| ?- a_capgirar([2,7,3,4,5,8,6,9,1,0],L2,Pas).
L2 = [2,3,7,4,5,8,6,9,1,0]
Pas = pas_capgirar([2],[7,3],[4,5,8,6,9,1,0]) ? ;
L2 = [2,7,4,3,5,8,6,9,1,0]
Pas = pas_capgirar([2,7],[3,4],[5,8,6,9,1,0]) ? ;
L2 = [2,7,5,4,3,8,6,9,1,0]
Pas = pas_capgirar([2,7],[3,4,5],[8,6,9,1,0]) ? ;
L2 = [2,7,3,4,5,6,8,9,1,0]
Pas = pas_capgirar([2,7,3,4,5],[8,6],[9,1,0]) ? ;
L2 = [2,7,3,4,5,6,8,9,1,0]
Pas = pas_capgirar([2,7,3,4,5],[8,6],[9,1,0]) ? ;
L2 = [2,7,3,4,5,8,9,6,1,0]
Pas = pas_capgirar([2,7,3,4,5,8],[6,9],[1,0]) ? ;
no

| ?- a_capgirar([9,8,7,6,5,4,3,2,1,0],L2,Pas).
L2 = [0,1,2,3,4,5,6,7,8,9]
Pas = pas_capgirar([],[9,8,7,6,5,4,3,2,1,0],[]) ? ;
no
```

Nota: *Fixeu-vos que es pot donar una mateixa resposta més d'una vegada, perquè es pot obtenir aplicant dues subaccions diferents.*

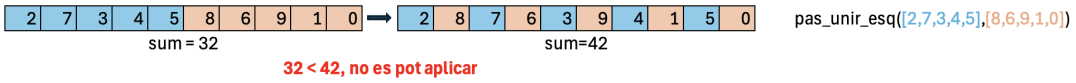
Intercalar

Aquesta acció consisteix en agafar les dues meitats de la llista i intercalar-les, o al revés, agafar els elements parells i els element senars, i separar-los en dues meitats. Per tal d'evitar cicles infinits, **només permetrem completar l'acció quan la mètrica** *sum* **de la llista resultant sigui estrictament més petita que la mètrica** *sum* **de la llista original**.

Distingirem 4 subtipus d'accions diferents, il·lustrades a continuació. Cada subacció tindrà la seva pròpia representació del pas.

Unir esquerra: Es divideix la llista original en dues meitats. Després d'aplicar l'acció, la primera meitat ocuparà les posicions parells (la primera posició és 0, parell) i la segona meitat ocuparà les senars. El pas es representa amb:

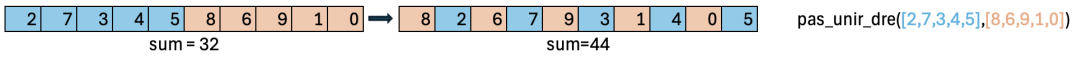
`pas_unir_esq(Esquerra,Dreta)`



32 < 42, no es pot aplicar

Unir dreta: Es divideix la llista original en dues meitats. Després d'aplicar l'acció, la primera meitat ocuparà les posicions senars i la segona meitat ocuparà les parells. El pas es representa amb:

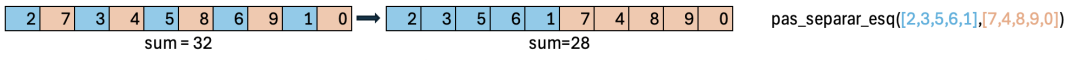
`pas_unir_dre(Esquerra,Dreta)`



32 < 44, no es pot aplicar

Separar esquerra: Es divideix la llista original en posicions parells i posicions senars. Després d'aplicar l'acció, les posicions parells ocuparan la primera meitat del resultat i les senars la segona meitat. El pas es representa amb:

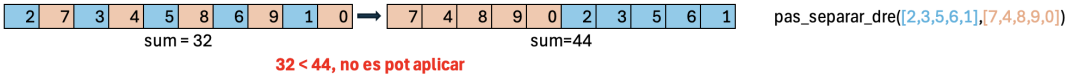
`pas_separar_esq(Parells,Senars)`



32 > 28, es pot aplicar

Separar dreta: Es divideix la llista original en posicions parells i posicions senars. Després d'aplicar l'acció, les posicions senars ocuparan la primera meitat del resultat, i les parells la segona meitat. El pas es representa amb:

`pas_separar_dre(Parells,Senars)`



32 < 44, no es pot aplicar

Exercici 5. *Implementa:*

```
%a_intercalar(+L,?L2,Pas)
%L2 es el resultat d'aplicar alguna de les subaccions d'intercalar a L, i Pas conte la tupla que
    representa l'accio aplicada
```

```
?- a_intercalar([2,7,3,4,5,8,6,9,1,0],L2,Pas).
L2 = [2,3,5,6,1,7,4,8,9,0]
Pas = pas_separar_esq([2,3,5,6,1],[7,4,8,9,0]) ? ;
no
```

```
| ?- a_intercalar([0,2,4,6,8,1,3,5,7,9],L2,Pas).
L2 = [0,1,2,3,4,5,6,7,8,9]
Pas = pas_unir_esq([0,2,4,6,8],[1,3,5,7,9]) ? ;
L2 = [1,0,3,2,5,4,7,6,9,8]
Pas = pas_unir_dre([0,2,4,6,8],[1,3,5,7,9]) ? ;
no

a_intercalar([5,0,6,1,7,2,8,3,9,4],L2,Pas).
L2 = [0,1,2,3,4,5,6,7,8,9]
Pas = pas_separar_dre([5,6,7,8,9],[0,1,2,3,4])
yes
```

Nota: *Tot i que aquest exemple no ho mostra, igual com amb les accions de capgirar, es pot obtenir una mateixa resposta aplicant dues subaccions diferents, quan la llista és de llargada 2 o 4.*

Càlcul de la mètrica *pas*

Una vegada definides les tres accions permeses, cal implementar un predicat per calcular la mètrica *pas*. Les accions `a_inserir` i `a_capgirar` són suficients per elles soles per poder ordenar la llista, però l'acció `a_intercalar` no ho és, caldrà complementar-la amb les altres accions.

Exercici 6. *Implementa:*

```
%ordenacio_minima(+L,+Accions,?L2,?Pas,-LlistaPassos)
%L2 es la llista L ordenada.
%S'ha ordenat amb una sequencia d'aplicacions de les accions dins de la llista Accions, que pot
    ser qualsevol subconjunt de {a_inserir, a_capgirar, a_intercalar}. El nombre de passos de la
    sequencia es el minim possible.
%Pas es el nombre passos aplicats (metrica pas)
%LlistaPassos conte la llista de passos aplicats, per ordre
%El predicat ordenacio_minima es demostra una sola vegada
```

Fixeu-vos que el predicat es demostra una sola vegada, així que ***caldrà aturar la cerca quan haguem trobat la primera solució****. Diferents implementacions poden donar valors de* `LlistaPassos` *diferents, però el valor de* `Pas` *serà el mateix en qualsevol implementació correcta.*

```
%Exemples d'execucio al fitxer exercicis.pl
```

1.3 Escriptura dels passos

Implementarem predicats que ens permetin escriure per pantalla amb un format visual i entenedor la seqüència mínima de passos trobada durant el càlcul d'una mètrica *pas*.

Exercici 7. *Implementa:*

```
%escriure_passos(+L) ==> escriu tots els passos de la llista L

%escriure_pas(+Pas) ==> escriu el pas Pas, que es algun dels vistos en les accions inserir,
    capgirar i intercalar
```

Cal que el format de sortida sigui exactament el que es mostra en els exemples, on s'identifiquen entre parèntesis els elements que s'han vist afectats pel pas.

```
%Exemples d'execucio al fitxer exercicis.pl
```

1.4 Programa principal

Integrarem les diferents funcionalitats implementades dins d'un programa principal, que carrega una llista desordenada i demana diferents opcions a efectuar fins que decidim sortir del programa. Més concretament, el programa ha de fer el següent:

---

1. Demana si es vol crear una llista aleatòria o si es vol entrar una llista manualment.
   - Si es tria aleatòria, llegeix la mida de la llista i llavor del random i es crea la llista. **Cal utilitzar el predicat `llista_aleatoria/3` ja proporcionat**. Assumirem que la mida és parell.
   - Si es tria manual, llegeix la llista. Assumirem que la llista entrada és una permutació de `[0,...,Mida-1]`.
2. Repeteix:
   - Mostra el menú
   - Llegeix una opció
   - Executa l'opció (o diu que l'opció és incorrecta).

   Fins que l'opció és `sor`.

Les opcions del menú són:

esc S'escriu la llista desordenada `L` amb la que treballem.

des S'escriu la mètrica *des* de `L`.

sum S'escriu la mètrica *sum* de `L`.

pas S'escriu la mètrica *pas* de `L`. Primer es demana el conjunt (llista) d'accions que volem utilitzar d'entre `a_inserir,a_capgirar,a_intercalar`.

pase Fa el mateix que l'opció **pas**, però també s'escriu la seqüencia de passos que permet ordenar la llista.

sor S'acaba el programa.

Exercici 8. *Implementa:*

```
%main ==> s'executa el programa principal descrit anteriorment
```

```
%Exemples d'execucio als fitxers main[N]{inout|in|out}.txt
```

Capítol 2

Realització de la pràctica

2.1 Normativa

- Aquesta pràctica s'ha de fer per parelles. Tindreu disponible un fòrum a l'assignatura, anomenat *Busca de parelles*, on podreu comunicar-vos per buscar company els que no en tingueu.
- La pràctica ha de ser original vostra. No està permès l'ús d'eines tipus ChatGPT o Copilot.
- Puc demanar a qualsevol grup que vingui a fer una entrega presencial de la seva pràctica si tinc dubtes sobre la originalitat o si vull més informació per a la correcció. S'espera que els dos membres del grup siguin capaços d'explicar la pràctica entregada, i de tornar implementar parts del codi entregat.
- Qualsevol acció fraudulenta (còpies, pràctica no original, ...) suposarà treure un 0.
- Cal que el fitxer que m'entregueu **compili correctament amb GNU Prolog** o **amb SWI Prolog**. Altrament, la nota serà un 0.
- Cal respectar els noms dels predicats proporcionats als exercicis. Si un predicat té un nom incorrecte, es considerarà no implementat. Els jocs de prova proporcionats haurien de permetre identificar noms erronis.
- Podeu implementar tots els predicats auxiliars que vulgueu.
- Només podeu utilitzar els predicats integrats a GNU/SWI Prolog detallats a continuació. L'ús de qualsevol altre predicat integrat es penalitzarà amb un 0. En cas de dubte, pregunteu. Els permesos són:
  - append, member, between, length.
  - repeat, fail, !.
  - =, \=, is, operadors i comparadors (+,-,mod,<,...).
  - read, write, print, format, nl.
  - findall, call.

2.2 Avaluació

- La pràctica es puntua sobre 10. Completar tots els exercicis menys el 7, i les opcions del menú excepte *pase*: **fins a 8 punts**. Completar també *pase* (i l'exercici 7): **fins a 10 punts**.

- Es valorarà la netedat i documentació del codi. Especifiqueu amb les restriccions d'instanciació i amb una descripció cada predicat.
- Els warnings penalitzaran.
- Es valorarà l'eficiència. De totes maneres tingueu en compte que *pas* és un problema d'optimització, per llistes molt llargues pot no acabar en temps raonable.
- Es valorarà que no es repeteixin parts del codi i de l'execució innecessàriament.
- Cal utilitzar el predicat ja proporcionat `llista_aleatoria/3`, de manera que donades una mida i una llavor, la llista generada sigui exactament la que es veu en els diferents exemples.
- El valor de *pas* és únic donada una llista d'accions permeses, però la seqüència de passos (escrita a *pase*) pot ser diferent en cada implementació, un resultat diferent dels exemples no penalitza. Això també aplica als resultats de `a_inserir`, `a_capgirar` i `a_intercalar`: el conjunt de resultats ha de ser el mateix, però l'ordre pot variar.

2.3 Entrega

- S'obrirà una tasca al Moodle per fer l'entrega, aquesta inclourà el termini.
- Només ha de fer l'entrega un dels dos membres del grup.
- Heu d'entregar dos fitxers:
  - Fitxer anomenat **exactament** `desordre.pl`
  - Un breu informe, com més breu millor (no més d'una pàgina, pot ser un document .txt) que digui quines de les parts de la pràctica heu implementat i creieu que us funcionen, i amb quin Prolog (SWI/GNU).

2.4 Fitxers proporcionats

`desordre.pl` Fitxer amb el predicat `llista_aleatoria/3` i altres predicats auxiliars ja implementats. També us pot ser útil utilitzar `llista_enumerada/2`.

`exercicis.pl` Consultes i respostes dels exemples proporcionats en aquest enunciat.

`main[N]{inout|in|out}.txt` Exemples d'execució d'un main amb l'entrada i sortida tal com apareixen al terminal (inout), o nomes l'entrada (in) i només la sortida (out) corresponents.