Sortări

1 Obiective

În lucrarea de față vom prezenta mai multe metode de sortare implementate în Prolog.

2 Considerații teoretice

2.1 Sortarea prin permutări

Aceasta metodă generează toate permutările posibile ale listei de intrarea până când se ajunge la o permutare care are toate elementele ordonate și atunci se oprește.

```
perm_sort(L,R):-perm(L, R), is_ordered(R), !.
```

Numărul de permutări ale unei liste cu *n* elemente este egal cu *n!*. Pentru a genera permutările, vom alege un element H aleatoriu din lista de intrare și îl vom pune pe prima poziție în lista rezultat. Alegerea lui H se va realiza folosind nedeterminismul predicatului *append.* Lista de intrare poate fi scrisă ca o concatenare de 2 sub-liste. Elementul H îl alegem să fie primul element din a doua sub-listă.

```
perm(L, [H|R]):-append(A, [H|T], L), append(A, T, L1), perm(L1, R).
perm([], []).
```

Mai avem de scris predicatul care verifică dacă lista rezultat este ordonată. Alegem ca lista rezultat să fie ordonată crescător.

```
is_ordered([H1, H2|T]):-H1 =< H2, is_ordered([H2|T]).
is_ordered([_]). % daca ii doar un element ii deja ordonata
```

Urmărește execuția la:

```
?- append(A, [H|T], [1, 2, 3]), append(A, T, R).
?- perm([1, 2, 3], L).
?- is_ordered([1, 2, 4, 4, 5]).
?- perm_sort([1, 4, 2, 3, 5], R).
```

2.2 Sortarea prin selecție

Această sortare alege la fiecare pas cel mai mic (cel mai mare) element din lista nesortată și îl mută la capătul liste deja sortate. Cel mai mic element din lista de intrarea reprezintă primul element din lista rezultat.

```
sel_sort(L, [M|R]):- min(L, M), delete(M, L, L1), sel_sort(L1, R).
sel_sort([], []).
```

Predicatele *min* și *delete* sunt de la lucrările precedente.

Urmărește execuția la:

```
?- sel_sort([3, 2, 4, 1], R).
```

2.3 Sortarea prin inserție

Această sortare extrage la fiecare pas un element din lista nesortată (de obicei primul element) și îl inserează în poziția corectă în lista sortată (folosind căutare liniară sau binară).

```
ins_sort([H|T], R):- ins_sort(T, R1), insert_ord(H, R1, R).
ins_sort([], []).

insert_ord(X, [H|T], [H|R]):-X>H, !, insert_ord(X, T, R).
insert_ord(X, T, [X|T]).
```

Urmărește execuția la:

```
?- insert_ord(3, [], R).
?- insert_ord(3, [1, 2, 4, 5], R).
?- insert_ord(3, [1, 3, 3, 4], R).
?- ins_sort([3, 2, 4, 1], R).
```

2.4 Sortarea bulelor

Sortarea bulelor realizează mai multe treceri peste lista de intrare. La fiecare trecere verifică două câte două elemente consecutive și le interschimbă dacă nu respectă condiția de ordine. Dacă la o trecere nu s-a făcut nici o interschimbare atunci sortarea s-a terminat. Pentru a verifica dacă s-a făcut o interschimbare ne vom folosi de un *flag* (dacă variabila este inițializată înseamnă că nu am terminat).

```
bubble_sort(L,R):-one_pass(L,R1,F), nonvar(F), !, bubble_sort(R1,R).
bubble_sort(L,L).

one_pass([H1,H2|T], [H2|R], F):-H1>H2, !, F=1, one_pass([H1|T],R,F).
one_pass([H1|T], [H1|R], F):-one_pass(T, R, F).
one_pass([], [] ,_).
```

Urmărește execuția la:

```
?- one_pass([1, 2, 3, 4], R, F).
?- one_pass([2, 3, 1, 4], R, F).
?- bubble_sort([1, 2, 3, 4], R).
?- bubble_sort([2, 3, 1, 4], R).
```

2.5 Sortare rapidă

Sortarea rapidă se folosește de un pivot care împarte lista de intrare în două sub-liste (tehnică *divide et impera*). O sub-listă cu elementele mai mici decât pivotul și o a doua sub-listă cu elementele mai mari decât pivotul. Repetă acest proces până când sub-listele devin vide.

```
quick_sort([H|T], R):- % alegem pivot primul element
     partition(H, T, Sm, Lg),
     quick_sort(Sm, SmS), % sortam sublista cu elementele mai mici
decât pivotul
     quick_sort(Lg, LgS), % sortam sublista cu elementele mai mari
decât pivotul
     append(SmS, [H|LgS], R).
quick_sort([], []).

partition(H, [X|T], [X|Sm], Lg):-X<H, !, partition(H, T, Sm, Lg).
partition(H, [X|T], Sm, [X|Lg]):-partition(H, T, Sm, Lg).
partition(_, [], [], []).
```

Urmărește execuția la:

```
?- partition(3, [4, 2, 6, 1, 3], Sm, Lg).
?- quick_sort([3, 2, 5, 1, 4, 3], R).
?- quick_sort([1, 2, 3, 4], R).
```

2.6 Sortare prin interclasare

Sortarea prin interclasare împarte lista de intrare în doua sub-liste de lungimi egale (tehnica *divide et impera*). Apoi sortează cele două sub-liste și în final interclasează sub-listele deja sortate.

```
merge_sort(L, R):-
     split(L, L1, L2), % împarte L în doua subliste de lungimi egale
     merge_sort(L1, R1),
     merge_sort(L2, R2),
     merge(R1, R2, R). % interclasează sublistele ordonate
merge_sort([H], [H]). % split returnează fail dacă lista ii vidă sau
are doar un singur element
merge_sort([], []).

split(L, L1, L2):-
     length(L, Len),
     Len>1,
     K is Len/2,
     splitK(L, K, L1, L2).

splitK([H|T], K, [H|L1], L2):-K>0,!,K1 is K-1,splitK(T, K1, L1, L2).
splitK(T, _, [], T).
```

```
merge([H1|T1], [H2|T2], [H1|R]):-H1<H2, !, merge(T1, [H2|T2], R).
merge([H1|T1], [H2|T2], [H2|R]):-merge([H1|T1], T2, R).
merge([], L, L).
merge(L, [], L).
```

Urmărește execuția la:

```
?- split([2, 5, 1, 6, 8, 3], L1, L2).
?- split([2], L1, L2).
?- merge([1, 5, 7], [3, 6, 9], R).
?- merge([1, 1, 2], [1], R).
?- merge([], [3], R).
?- merge_sort([4, 2, 6, 1, 5], R).
```

3 Exerciții

1. Rescrieți predicatul *perm* fără a apela predicatul *append*. Extragerea și ștergerea unui element trebuie realizate altfel.
2. Rescrieți predicatul *sel_sort* astfel încât să sorteze descrescător.
3. Scrieți un predicat care să sorteze o listă de caractere ASCII. *Sugestie: folosiți char_code*

```
?- sort_chars([e, t, a, v, f], L).
L = [a, e, f, t, v] ;
false
```

4. Scrieți un predicat care să sorteze o lista de sub-liste în funcție de lungimea sub-listelor.

```
?- sort_len([[a, b, c], [f], [2, 3, 1, 2], [], [4, 4]], R).
R = [[], [f], [4, 4], [a, b, c], [2, 3, 1, 2]] ;
false
```