Benjamin Kataliko Viranga
8842942
CSI2520

Partie III - Projet Intégrateur - Prolog

→ **Fichiers :** dprogramming.pl , p1.txt
→ **Output :**

```
10 ?- solveKnapsack('p1.txt', Value, L_items_list).

Reading : p1.txt

> Collected Items : [item(A,1,1),item(B,6,2),item(C,10,3),item(D,15,5)]
> Items weights : [1,2,3,5]
> Items values : [1,6,10,15]

[0,0,0,0,0,0,0,0]
[0,1,1,1,1,1,1,1]
[0,1,6,7,7,7,7,7]
[0,1,6,10,11,16,17,17]
[0,1,6,10,11,16,17,21]

Value = 21,
L_items_list = [item("B", 6, 2), item("D", 15, 5)] .
```

Note : Le fichier .sol est généré dans le même dossier avec le contenu de la solution.

Ce document contient les prédicats utilisés pour développer cette solution en programmation dynamique pour le knapsack problem.
Également, les références sont incluses en commentaire au-dessus des prédicats correspondants dans le fichier du projet **dprogramming.pl.**

Les prédicats principaux

- **read_file(Stream, Line)** : pour la lecture du fichier d'entrée. Le fichier est parcouru ligne après ligne et ses données sont retournées dans une liste.

```
% read a file in prolog
% at_end_of_stream succeeds when the last line is read.
read_file(Stream, Lines) :-
    read_line_to_codes(Stream, Codes),  % Attempt a read Line from the stream
    atom_chars(Line, Codes),
    (  at_end_of_stream(Stream)     % If we're at the end of the stream then...
    -> Lines = [Line]  % at_end_of_stream succeeds when the last line is read.
    ;  Lines = [Line|NewLines],       % Otherwise, Lines is Line followed by
       read_file(Stream, NewLines)    %   a read of the rest of the file
    ).
```

- **get_data(Filename, Capacity, L_items_weight, L_items_value, All_items)**: Parcours le fichier **Filename** et en extrait les données du problème knapsack en initialisant la capacité maximale du knapsack (**Capacity**) , la liste des poids des items à ajouter dans le knapsack (**L_items_weight**), la liste des valeurs des items à ajouter dans le knapsack (**L_items_value**), et la liste des tous les items (**All_items)** obtenue du fichier sous le format **item(Name,Value, Weight).** Ce prédicat fait appel au prédicat ***process_data()*** et ***read_file()*** **.**

```
% Ce predicat retourne les donnees du fichier
get_data(Filename, Capacity, L_items_weight, L_items_value, All_items):-
    open(Filename, read, Str),nl,
    write('Reading : '), writeln(Filename),
    read_file(Str, Data),
    close(Str),
    %write(Data), nl,
   process_data(Data,Capacity,    L_items_weight,L_items_value,All_items).
%process data
```

- **process_data(Data, Capacity, L_items_weight, L_items_value, All_items)**: prédicat utilisé pour initialiser **Capacity**, **L_items_weight**, **L_items_value** et **All_items** en fonction des données contenues dans la liste **Data**. Ce prédicat utilise le prédicat **process_items()** et utilise les prédicats utilitaires **remove_first(), no_space_str_to_int()** et **trim().** Le premier élément de la liste **Data** correspond au nombre d'éléments à ajouter dans le knapsack et le dernier élément de la liste **Data** correspond à la capacité maximale du knapsack lue du fichier.

```
process_data(Data, Capacity, L_items_weight, L_items_value, All_items):-
   remove_first(Data, F, LL)% LL is Data without the first element N_items
    no_space_str_to_int(F,N_items),
    last(LL, Capacity_str),
    no_space_str_to_int(Capacity_str, Capacity),
    trim(LL,N_items, LL_new),      % process items
    process_items(LL_new, L_items_weight, L_items_value, All_items).
```

- **process_items(L_items_str, L_items_weight, L_items_value, All_items)**: Initialise ses paramètres à travers le prédicat **get_items()** et imprime les listes collectées sur le terminal.

```
%retourne les items en termes composés dans la list All_items
%L_items_weight : liste des poids
%L_items_value  : liste des valeurs
%All_items  : Liste des items en termes composés
process_items(L_items_str, L_items_weight, L_items_value, All_items):-
```

```
    % get the corresponding items
    %writeln(L_items_str),
    get_items(L_items_str,L_items_weight,L_items_value,All_items),nl,
    write('> Collected Items : '), writeln(All_items),
    write('> Items weights : '),  writeln(L_items_weight),
    write('> Items values : '),  writeln(L_items_value),nl.
```

- **get_items([I|L],L_items_weight,L_items_value, All_items)**: Initialise les listes des poids, des valeurs ainsi que de tous les items :

```
% get the items and initialize the corresponding lists
get_items([],[],[],[]).
get_items([I|L],L_items_weight,L_items_value, All_items):-
       get_items(L, W, V, All),
       % knowing that I is a string
       split_string(I, " ", " ", I_list),
       % index 0 is the name of the item
       % index 1 is the value of the item
       % index 2 is the weight of the item
       length(I_list, 3), % ensure the size of the list is 3.
       nth0(0, I_list, Item_name),  % Item name in string
       nth0(1, I_list, Item_value_str),
       nth0(2, I_list, Item_weight_str),
       % get item and weigth value in integer
       no_space_str_to_int(Item_value_str, Item_value),
       no_space_str_to_int(Item_weight_str, Item_weight),
       % initialize the item
       Item = item(Item_name,Item_value,Item_weight),
       % ajouter les termes composés pour les items dans la liste
       append([Item],All,All_items),
       % collecter les poids des items
       append([Item_weight], W,L_items_weight),
       % collecter les valeurs des items
       append([Item_value], V, L_items_value).
```

- Les prédicats **row_gen()** : Ces prédicats permettent d'initialiser les rangées du tableau pour la programmation dynamique tel que l'indique l'image ci-dessous montrant le résultat attendu avec la programmation dynamique.

	0	1	2	3	4	5	6	7
No item	0	0	0	0	0	0	0	0
Item A	0	1	1	1	1	1	1	1
Items A,B	0	1	6	7	7	7	7	7
Items A,B,C	0	1	6	10	11	16	17	17
Items A,B,C,D	0	1	6	10	11	16	17	**21**

```
% Générateur de liste pour l'item ajouté
% B_cap : Capacité du sac (Bag capacity)
% Capacité : représentant la valeur maximale du sac
% retourne une liste vide pour la rangee correspondante
row_gen(Max_cap,_,_,_,B_cap,[]):- B_cap > Max_cap, !.

% initial row - with zeros
row_gen(Max_cap,_, 0,0,B_cap,[0|RR]):-
    Next_cap is B_cap + 1,
    row_gen(Max_cap,[], 0, 0,Next_cap,RR).

% initialisation de la ligne suivante
row_gen(Max_cap,Previous_row,Item_value,Item_weight,0,[0|RR]):-
    row_gen(Max_cap,Previous_row, Item_value, Item_weight,1,RR).

% if the item weight is above the maximum allowed capacity
row_gen(Max_cap, Previous_Row, Item_value, Item_weight, B_cap,[VV|RR]):-
    B_cap < Item_weight,  % the item can not fit inside the knapsack
    nth0(B_cap, Previous_Row, Previous_Value),
    VV is Previous_Value, % assign the previous value to the current value
for the row
    Next_cap is B_cap + 1,
    row_gen(Max_cap, Previous_Row,Item_value, Item_weight, Next_cap, RR).


% if the item weight can fit the current knapsack capacity
row_gen(Max_cap, Previous_Row, Item_value, Item_weight, B_cap,[VV|RR]):-
    B_cap >= Item_weight,   % the item can fit inside the knapsack
    Diff_weight is B_cap - Item_weight,
```

```
    nth0(Diff_weight, Previous_Row, Previous_value),
     % evaluate the corresponding value VV_temp based on the difference
between the weights
    % B_cap - Item_weight
    VV_temp is  Previous_value + Item_value,
    % get current Bcap of Previous row
    nth0(B_cap, Previous_Row, B_cap_prev_value),
    (B_cap_prev_value >  VV_temp     % compare the maximum between the two
rows at index B_cap
        -> VV is B_cap_prev_value    % if the previous value is higher VV
is the previous value
       ; VV is VV_temp              % if not VV is VV_temp
    ),
    % writeln(VV),nl,
    Next_cap is B_cap + 1,
    row_gen(Max_cap, Previous_Row, Item_value, Item_weight, Next_cap,RR).
```

- Le prédicat **knapsack_process(Capacity, [W|WW], [V|VV], Result, Value, []) :** Ce prédicat retourne la valeur optimale obtenue pour le knapsack avec W pour Weight et V pour Valeur.

```
% Knapsack process
% Ce predicat renvoie la valeur optimal pour la rangee
% cette valeur correspond au dernier élément de la liste
knapsack_process(Capacity,[],[],Result,0,[]):-
    row_gen(Capacity,_, 0,0,0,Result), writeln(Result). % initial row of
zeros
knapsack_process(Capacity, [W|WW], [V|VV], Result, Value, []):-
    knapsack_process(Capacity, WW, VV, Res,_, _),
    row_gen(Capacity, Res, V, W, 0, Result),
    writeln(Result),
    last(Result,Value).     % knapsack optimal value is the last value
```

- Le prédicat **solve_subset(MaxVal,Lin,Lout)** : établit la somme des subset des items d'une liste **Lin** qui correspondent à une valeur spécifique **MaxVal** et initialise la liste correspondante par **Lout**. Ce prédicat utilise le prédicat utilitaire **combs().**

```
solve_subset(MaxVal,Lin,Lout):-
    combs(Lin,Lout),    % generate all the possible combinations
     sumlist(Lout,Val),   % this sum of the subset correspond to the
specific value
    Val = MaxVal.       % MaxVal is the sum
```

- Le prédicat **knapsack(Capacity, L_items_weight, L_items_value, Value, L_items_list)** : collecte la valeur optimale ***(Value)*** ainsi que la liste des valeurs des items correspondant à la valeur optimale **(*L_items_list)*.** Ce prédicat fait appel au prédicat **knapsack_process ()** et au prédicat **solve_subset().**

```
% knapsack predicate
% En input :
% Capacity : La valeur maximale de la capacité du Knapsack
% L_items_weight :  la liste des poids des items à ajouter
% L_items_value : la liste des valeurs des items du Knapsack
% En output :
% L_items_list :  la liste des valeurs des items dans le knapsack optimal
knapsack(Capacity, L_items_weight, L_items_value, Value, L_items_list):-

    % reverse the list - because the recursivity is starting from the last
the item
    reverse(L_items_weight, R_weight),
    reverse(L_items_value, R_value),
    % debug purpose
    % writeln(R_weight),
    % writeln(R_value),
    % knapsack
    knapsack_process(Capacity, R_weight, R_value,_, Value,_),
    nl,
    % get list of optimal values
    solve_subset(Value,L_items_value, L_items_list).
```

- Le prédicat **write_data(Filename, Value, L_items_list) :** écrit la solution optimale dans le fichier .sol .

```
write_data(Filename, Value, L_items_list):-
   % split string
   split_string(Filename, '.', '.', File_no_ext_arr),
   % remove the filename from the array
   remove_first(File_no_ext_arr,File_no_ext,_),
   % add the .sol extension to the file
   atom_concat(File_no_ext,'.sol', New_filename),
   % open the stream for writing
   open(New_filename, write, Out),
   % write the content of the file
   writeln(Out, Value),
   % collect the names of the items from the L_items_list
   findall(N, (member(X,L_items_list), item(N,_,_) = X), Names),
```

```
   % concatenate the name with a space as separator
   atomic_list_concat(Names, " " , W),
   % new line
   % write W in the .sol file
   writeln(Out,W),
   close(Out).
```

- Le prédicat **solve_knapsack(Filename, Value, L_items_list)** : prédicat principal pour la partie III logique en Prolog. **Filename** est le fichier à parcourir, **Value** correspond à la valeur optimale et **L_items_list** correspond aux items dans le knapsack.

```
% solve Knapsack - initial predicate for the program
solveKnapsack(Filename, Value, L_items_list):-
   % collect data from the document
   get_data(Filename, Capacity, L_items_weight, L_items_value,All_items),
   % proceed with the optimal value
   knapsack(Capacity, L_items_weight, L_items_value, Value, L_opt_values),
   % get the items representation for L_items_list
   get_items_repr(All_items, L_items_value , L_opt_values, L_items_list),
   % write the result inside the .sol file
   write_data(Filename, Value, L_items_list).
```

Les prédicats utilitaires

Ces prédicats sont utilisés afin de compléter certaines tâches facilitant l'exécution des prédicats principaux.

- Le prédicat **Trim() :** Coupe une liste et retourne les N premiers éléments de la liste à couper.

```
% truncate list and get the first N elements
% L - ToTruncate
% N - Desired Length
% R - Result
trim(L,N,R) :-      % to trim N elements from a list
   length(R,N),     % - generate an unbound prefix list of the desired
length
   append(R,_,L). % - and use append/3 to get the desired suffix.
```

- Le prédicat **remove_first() :** retire le premier élément de la liste à traiter

```
%remove first element of list
%return the first element and the rest of the list
remove_first([F|L],F,LL):-
    append([],L,LL).
```

- Le prédicat **no_space_str_to_int() :** convertit un string en int tout en éliminant les espaces contenus dans le string initial.

```
% no_space_str_to_int
% ensuring there is no space as well
% Str as input and output the Int
% Le Str doit etre en representation string
no_space_str_to_int(Str, Int):-
    split_string(Str, " ", " ", Str_split),  % remove any space from the
string
    remove_first(Str_split, Str_new, _), % get the first elem without
space
    atom_number(Str_new, Int).           % get the integer
```

- Le prédicat **get_items_repr() :** permet d'extraire les items qui correspondent à la valeur optimale dans le knapsack sous la représentation **item(Name,Value,Weight)**.

```
% get items representation based on the list of values given as paramater
get_items_repr(All_items,L_items_value, L_opt_values, L_items_list):-
    % find all the index of the elements from L_opt_values inside
L_items_values
    findall(I, (member(X,L_opt_values),
member(X,L_items_value),nth0(I,L_items_value,X)), Bag),
    % get the items from the list of initial items
    findall(O,(member(X,Bag), nth0(X,All_items,O)), L_items_list).
```

Références

- **"**How to read a file in Prolog**":** https://stackoverflow.com/questions/37573618/how-to-read-a-file-in-prolog
- "Prolog - Unexpected end of ile " https://stackoverflow.com/questions/23411139/prolog-unexpected-end-of-file
- "at_end_of_stream/1" :https://www.swi-prolog.org/pldoc/man?predicate=at_end_of_stream/1
- "Truncate List " : https://stackoverflow.com/questions/27479915/how-to-trim-first-n-elements-from-in-list-in-prolog
- "Write a file in prolog" : https://stackoverflow.com/questions/22747147/swi-prolog-write-to-file
- "atom_list_concat": https://www.swi-prolog.org/pldoc/doc_for?object=atomic_list_concat/3

- “combinations of items in a list” : https://stackoverflow.com/questions/41662963/all-combinations-of-a-list-without-doubles-in-prolog
- “find all combinations of items of a list that sums up to a specific value”: https://stackoverflow.com/questions/49297842/find-all-combinations-of-elements-in-a-list-that-sum-up-to-a-value