Guide to Prolog Programming *© Roman Barták, 1998*

Sorting

[naive] [insert] [bubble] [merge] [quick]

This lecture covers sort algorithms. Notice the natural and short representation of sort algorithms in Prolog.

naive sort

Naive sort is not very efficient algorithm. It generates all permutations and then it tests if the permutation is a sorted list.

```
naive_sort(List,Sorted):-perm(List,Sorted),is_sorted(Sorted).

is_sorted([]).
is_sorted)[_]).
is_sorted([X,Y|T]):-X=<Y,is_sorted([Y|T]).
```

Naive sort uses the **generate and test** approach to solving problems which is usually utilized in case when everything else failed. However, sort is not such case.

insert sort

Insert sort is a traditional sort algorithm. Prolog implementation of insert sort is based on idea of accumulator.

```
insert_sort(List,Sorted):-i_sort(List,[],Sorted).
i_sort([],Acc,Acc).
i_sort([H|T],Acc,Sorted):-insert(H,Acc,NAcc),i_sort(T,NAcc,Sorted).

insert(X,[Y|T],[Y|NT]):-X>Y,insert(X,T,NT).
insert(X,[Y|T],[X,Y|T]):-X=<Y.
insert(X,[],[X]).
```

bubble sort

Bubble sort is another traditional sort algorithm which is not very effective. Again, we use accumulator to implement bubble sort.

```
bubble_sort(List,Sorted):-b_sort(List,[],Sorted).
b_sort([],Acc,Acc).
b_sort([H|T],Acc,Sorted):-bubble(H,T,NT,Max),b_sort(NT,[Max|Acc],Sorted).

bubble(X,[],[],X).
bubble(X,[Y|T],[Y|NT],Max):-X>Y,bubble(X,T,NT,Max).
bubble(X,[Y|T],[X|NT],Max):-X=<Y,bubble(Y,T,NT,Max).
```

merge sort

Merge sort is usually used to sort large files but its idea can be utilized to every list. If properly implemented it could be a very efficient algorithm.

```
merge_sort([],[]).     % empty list is already sorted
merge_sort([X],[X]).   % single element list is already sorted
merge_sort(List,Sorted):-
    List=[_,_|_],divide(List,L1,L2),     % list with at least two elements is divided into two parts
        merge_sort(L1,Sorted1),merge_sort(L2,Sorted2),  % then each part is sorted
        merge(Sorted1,Sorted2,Sorted).                  % and sorted parts are merged
merge([],L,L).
merge(L,[],L):-L\=[].
merge([X|T1],[Y|T2],[X|T]):-X=<Y,merge(T1,[Y|T2],T).
merge([X|T1],[Y|T2],[Y|T]):-X>Y,merge([X|T1],T2,T).
```

We can use distribution into even and odd elements of list

```
divide(L,L1,L2):-even_odd(L,L1,L2).
```

or traditional distribution into first and second half (other distibutions are also possible)

```
divide(L,L1,L2):-halve(L,L1,L2).
```

quick sort

Quick sort is one of the fastest sort algorithms. However, its power is often overvalued. The efficiency of quick sort is sensitive to choice of pivot which is used to distribute list into two "halfs".

```
quick_sort([],[]).
quick_sort([H|T],Sorted):-
        pivoting(H,T,L1,L2),quick_sort(L1,Sorted1),quick_sort(L2,Sorted2),
        append(Sorted1,[H|Sorted2]).

pivoting(H,[],[],[]).
pivoting(H,[X|T],[X|L],G):-X=<H,pivoting(H,T,L,G).
pivoting(H,[X|T],L,[X|G]):-X>H,pivoting(H,T,L,G).
```

Similarly to merge sort, quick sort exploits the **divide and conquer** method of solving problems.

The above implementation of quick sort using append is not very effective. We can write better program using accumulator.

```
quick_sort2(List,Sorted):-q_sort(List,[],Sorted).
q_sort([],Acc,Acc).
q_sort([H|T],Acc,Sorted):-
        pivoting(H,T,L1,L2),
        q_sort(L1,Acc,Sorted1),q_sort(L2,[H|Sorted1],Sorted)
```

Some type of sort can be found in almost all current programs.

[naive] [insert] [bubble] [merge] [quick]

Designed and maintained by Roman Barták

Previous | Contents | Next