

Множества. Деревья





Множества

- ❖ Создание множества
- ❖ Операции со множествами (объединение, пересечение, разность, проверка включения, симметрическая разность, дополнение)



Создание множества

`list_set([],[]).` /* пустой список является множеством */

`list_set ([H|T],[H|T1]) :-`

`delete_all(H,T,T2),`

`/* T2 — результат удаления`

`вхождений первого элемента`

`исходного списка H из хвоста T */`

`list_set (T2,T1).`

`/* T1 — результат удаления`

`повторных вхождений элементов`

`из списка T2 */`



Объединение множеств

`union([],S2,S2).`

`union([H|T],S2,S):-`

`member3(H,S2),`

`!,`

`union(T,S2,S).`

`union([H |T],S2,[H|S]):-`

`union(T,S2,S).`

Пересечение множеств

```
intersection([],_,[]).
intersection([H|T1],S2,[H|T]):-
    member3(H,S2),
    !,
    intersection(T1,S2,T).
intersection([_|T],S2,S):-
    intersection(T,S2,S).
```

Разность множеств

`minus([],_,[]).`

`minus([H|T],S2,S):-`

`member3(H,S2),`

`!,`

`minus(T,S2,S).`

`minus([H|T],S2,[H|S]):-`

`minus(T,S2,S).`



Проверка включения

`subset([],_).`

`subset([H|T],S):-
 member3(H,S),
 subset(T,S).`

`subsetU(A,B):-
 union(A,B,B).`

`subsetI(A,B):-
 intersection(A,B,A).`



Проверка включения

$\text{equal}(A,B):-$

$\text{subset}(A,B),$

$\text{subset}(B,A).$

$\text{Prop_subset}(A,B):-$

$\text{subset}(A,B),$

$\text{not}(\text{equal}(A,B)).$



Симметрическая разность

Sim_minus(A,B,SM):–
minus(A,B,A_B),
minus(B,A,B_A),
union(A_B,B_A,SM).



Дополнение множества

$\text{supp}(A, D):-$

$U = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],$

$\text{minus}(U, A, D).$



Объединение и пересечение через дополнение

$\text{unionI}(A, B, AB):-$

$\text{supp}(A, A_),$

$\text{supp}(B, B_),$

$\text{intersection}(A_ , B_ , A_ B),$

$\text{supp}(A_ B, AB).$

$\text{intersectionU}(A, B, AB):-$

$\text{supp}(A, A_),$

$\text{supp}(B, B_),$

$\text{union}(A_ , B_ , A_ B),$

$\text{supp}(A_ B, AB).$



Деревья

- ❖ Принадлежность значения дереву
- ❖ Замена в дереве всех вхождений одного значения на другое
- ❖ Подсчет общего количества вершин дерева
- ❖ Подсчет количества листьев дерева
- ❖ Сумма чисел, расположенных в вершинах дерева
- ❖ Вычисление высоты дерева
- ❖ Проверка принадлежности значения двоичному справочнику
- ❖ Добавление в двоичный справочник нового значения
- ❖ Алгоритм, генерирующий дерево, которое является двоичным справочником и состоит из заданного количества вершин, в которых будут размещены случайные целые числа
- ❖ Удаление заданного значения из двоичного справочника
- ❖ Преобразование произвольного списка в двоичный справочник
- ❖ «Сворачивание» двоичного справочника в список с сохранением порядка элементов



Принадлежность значения дереву

DOMAINS

tree=empty;tr(i,tree,tree)

CLAUSES

tree_member(X,tr(X,_,_)):-!.
tree_member(X,tr(_,L,_)):-
tree_member(X,L),!.
tree_member(X,tr(_,_,R)):-
tree_member(X,R).



Замена в дереве всех вхождений одного значения на другое

`tree_replace(_,_,empty,empty).`

`tree_replace(X,Y,tr(X,L,R),tr(Y,L1,R1)):-`

`!,`

`tree_replace(X,Y,L,L1),`

`tree_replace(X,Y,R,R1).`

`tree_replace(X,Y,tr(K,L,R),tr(K,L1,R1)):-`

`tree_replace(X,Y,L,L1),`

`tree_replace(X,Y,R,R1).`



Подсчет общего количества вершин дерева

`tree_length (empty,0).`

`tree_length(tr(_,L,R),N):-
 tree_length (L,N1),
 tree_length (R,N2),
 N=N1+N2+1.`



Подсчет количества листьев дерева

`tree_leaves(empty,0).`

`tree_leaves(tr(_,empty,empty),1):-!.`

`tree_leaves(tr(_,L,R),N):-`

`tree_leaves(L,N1),`

`tree_leaves(R,N2),`

`N=N1+N2.`



Сумма чисел, расположенных в вершинах дерева

$\text{tree_sum}(\text{empty}, 0).$

$\text{tree_sum}(\text{tr}(X, L, R), N) :-$
 $\text{tree_sum}(L, N1),$
 $\text{tree_sum}(R, N2),$
 $N = N1 + N2 + X.$



Вычисление высоты дерева

$\text{tree_height}(\text{empty}, 0).$

$\text{tree_height}(\text{tr}(_, L, R), D) :-$

$\text{tree_height}(L, D1),$

$\text{tree_height}(R, D2),$

$\text{max}(D1, D2, D_M),$

$D = D_M + 1.$



Проверка принадлежности значения двоичному справочнику

```
tree_member2(X,tr(X,_,_)):-!.  
tree_member2(X,tr(K,L,_)):-  
    X<K,!,  
    tree_member2(X,L).  
tree_member2(X,tr(K,_,R)):-  
    X>K,!,  
    tree_member2(X,R).
```



Добавление в двоичный справочник нового значения

$\text{tree_insert}(X, \text{empty}, \text{tr}(X, \text{empty}, \text{empty}))$.

$\text{tree_insert}(X, \text{tr}(X, L, R), \text{tr}(X, L, R))$:—!.

$\text{tree_insert}(X, \text{tr}(K, L, R), \text{tr}(K, L1, R))$:—

$X < K$,!,

$\text{tree_insert}(X, L, L1)$.

$\text{tree_insert}(X, \text{tr}(K, L, R), \text{tr}(K, L, R1))$:—

$\text{tree_insert}(X, R, R1)$.



Генерация двоичного справочника со случайными числами

```
tree_gen(0,empty):-!.  
tree_gen (N,T):-  
    random(100,X),  
    N1= N-1,  
    tree_gen (N1,T1),  
    tree_insert(X,T1,T).
```



Удаление заданного значения из двоичного справочника

```
tree_del_min(tr(X,empty,R), R, X).
tree_del_min(tr(K,L,R), tr(K,L1,R), X):-
    tree_del_min(L, L1, X).

tree_delete(X,tr(X,empty,R), R):-!.
tree_delete (X,tr(X,L,empty), L):-!.
tree_delete (X,tr(X,L,R), tr(Y,L,R1)):-
    tree_del_min(R,R1, Y).
tree_delete (X,tr(K,L,R), tr(K,L1,R)):-
    X<K,!,
    tree_delete (X,L,L1).
tree_delete (X,tr(K,L,R), tr(K,L,R1)):-
    tree_delete (X,R,R1).
```



Преобразование произвольного списка в двоичный справочник

`list_tree([],empty).`

`list_tree([H|T],Tr):-`

`list_tree(T,Tr1),`

`tree_insert(H,Tr1,Tr).`



«Сворачивание» двоичного справочника в список с сохранением порядка элементов

`tree_list(empty,[]).`

`tree_list(tr(K,L,R),S):-`

`tree_list(L,T_L),`

`tree_list(R,T_R),`

`conc(T_L,[K|T_R],S).`



Задачи для самостоятельного решения

- ❖ Разработать предикат, порождающий всевозможные перестановки исходного множества
- ❖ Разработать предикат, порождающий всевозможные подмножества исходного множества
- ❖ Разработать предикат, который создает справочник из количества вершин, большего максимального значения, задаваемого в `random`.
- ❖ Разработать предикат, который создает двоичный справочник, состоящий ровно из заданного количества вершин.
- ❖ Разработать предикат, выполняющий сортировку списка (с помощью двоичного справочника)