Пребарување

Пребарување на податочни структури

Пребарување се нарекува процесот на барање на даден елемент од дадена структура на податоци. До бараниот елемент може да се пристапи на различни начини, кои се помалку или повеќе ефикасни. Понатаму ќе разгледаме пребарување на некои познати податочни структури како низи, листи, дрва итн.

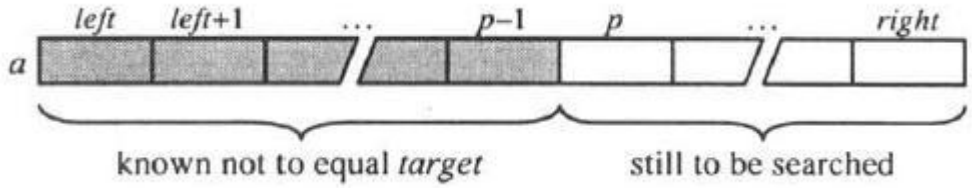
Низи

Ќе разгледаме пребарување во несортирани и сортирани низи.

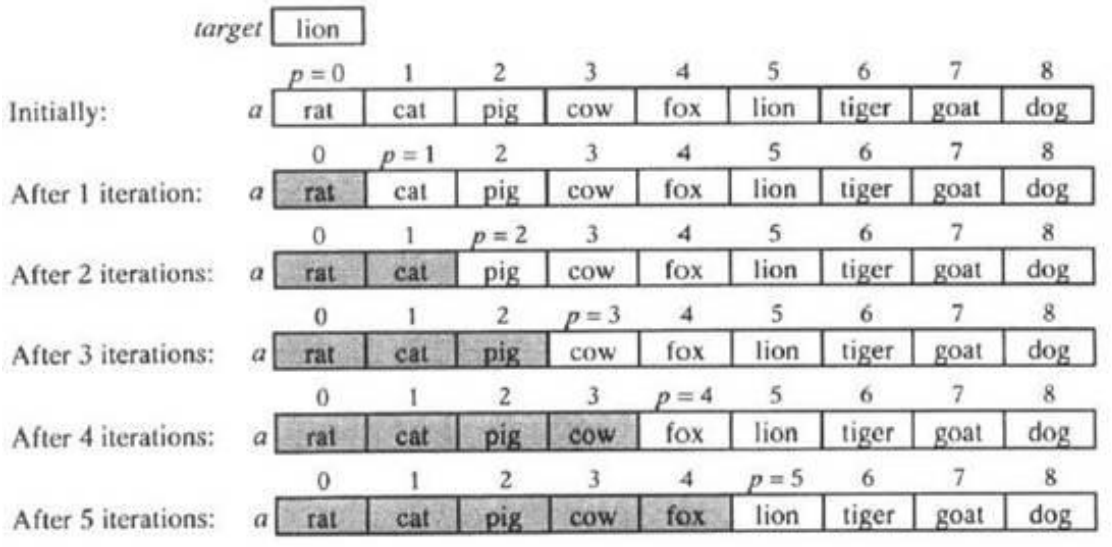
Несортирани низи

Линеарно пребарување

Со линеарното пребарување бараниот елемент се споредува со секој елемент на низата. Кога ќе се пронајде бараниот елемент, алгоритмот завршува и го враќа индексот на тој елемент, а доколку нема таков елемент, алгоритмот враќа соодветна порака. Карактеристична операција на овој начин на пебарување е споредувањето. Овој алгоритам има линеарна временска сложеност, O(n). Просечниот број на споредби е (n+1)/2 а најлошиот случај е n споредби.




```
function found(a:niza;x,n:integer):boolean;
var i:integer;
begin
    found:=false;
    for i:=1 to n do
    begin
        if a[i]=x
        then
        begin
            found:=true;
            exit;
        end;
    end;
end;
```



Сортирани низи

Линеарно пребарување

Линеарното пребарување може да се примени и на сортирана низа. Придобивка овде е тоа што доколку бараниот елемент го нема во низата, не мора да бараме до крај на низата, туку до првиот елемент поголем од бараниот (освен ако бараниот елемент не е поголем од сите елементи од низта). Истотака ако бараниот елемент е помал од првиот тогаш него го нема во низата.

Бинарно пребарување

Поефикасно пребарување на сортираните низи е бинарното пребарување. Ние бинарно пребаруваме, на пример кога бараме збор во речник. На почеток го отвораме речникот на средина и ако ја погодиме страната застануваме. Но ако не ја погодиме страната, во зависност од тоа дали почетната буква на бараниот збор е пред или по буквата на која сме наиделе, повторно отвораме во првата или втората половина на речникот итн. се до наоѓањето на зборот.

Поточно постапката се одвива на следниот начин:

1. **Постави dolna=1 и gorna=n(каде n е бројот на елементи на низата)**
2. **Додека dolna<=gorn повторувај**

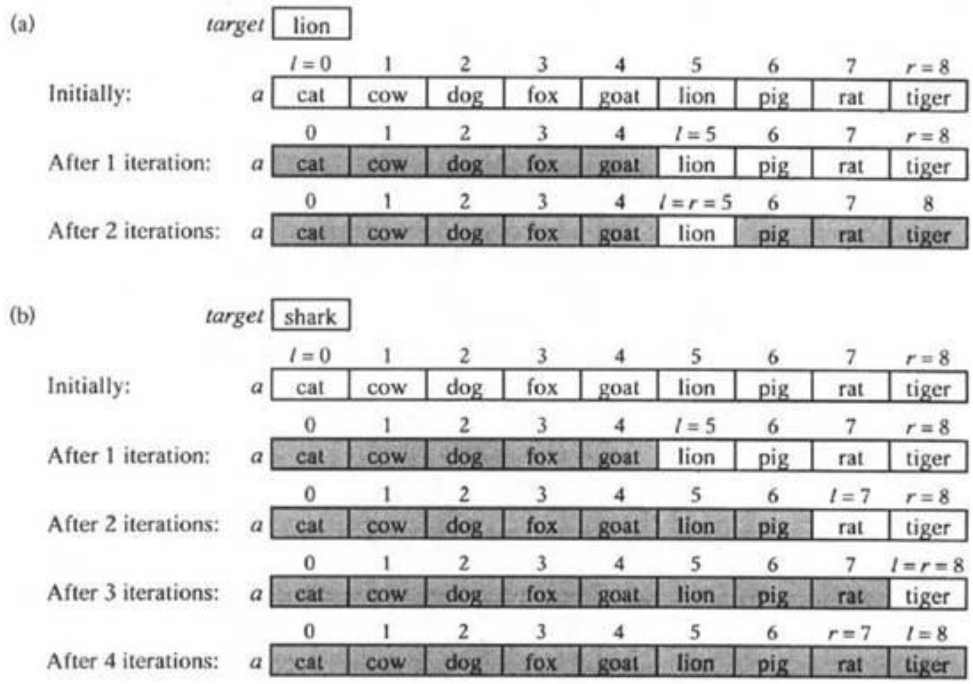
   2.1 **Постави sredina=(dolna + gorna) / 2**

   2.2 **Ако бараниот елемент е a[sredina] заврши со излез sredina**

   2.3 **Ако бараниот елемент е помал од a[sredina] тогаш gorna=sredina-1**

   2.4 **Ако бараниот елемент е поголем од a[sredina] тогаш dolna=sredina-1**

3**. Заврши со излез 0**



Бројот на итерации на овој алгоритам во најлош случај е еднаков на бројот на делење на должината на низата со два се додека должината не стане нула т.е Log_2n па и временската сложеност е $O(Log_2n)$.

```
    type pokazuvac=^element;
  element=record
          broj:integer;
          sleden:pokazuvac;
      end;
function najdi(baran:integer):boolean;
begin
  tekoven:=prv;
  while (tekoven^.broj<>baran) and (tekoven^.sleden<>nil)do
     tekoven:=tekoven^.sleden;
  end;
  if tekoven^.broj=baran
     then najdi:=true
     else najdi:=false;
end;
```

Поврзани листи

Линеарно пребарување

Линеарното пребарување на поврзаните листи е идентично како кај низите и временската сложеност е исто O(n).

Бинарно пребарување

Бинарното пребарување е ефикасно кај низи зашто директно може да се пристапи до средниот елемент на низата. Кај поврзаните листи, за да се стигне до средниот елемент треба да се мине половина листа. Затоа временската сложеност на бинарното пребарување кај поврзаните листи е O(n), каде n е должината на листата. (НАПОМЕНА: Листата мора да биде сортирана за кристење на овој метод.)

Дрва

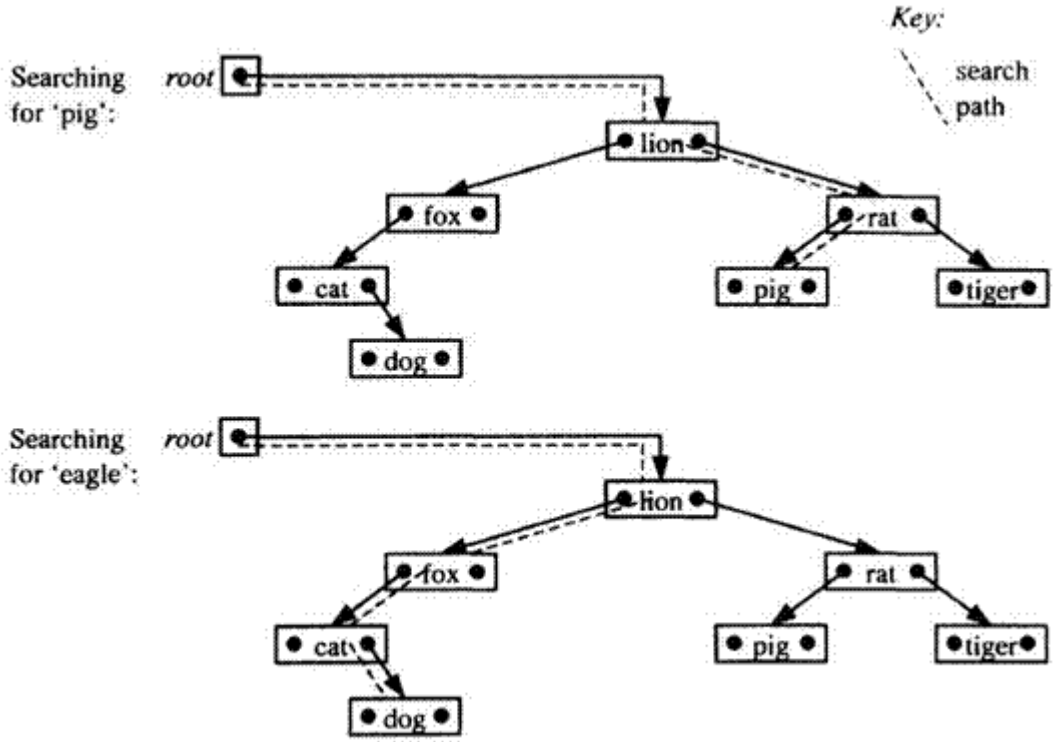
Бинарни пребарувачки дрва

Бинарно пребарувачко дрво е:

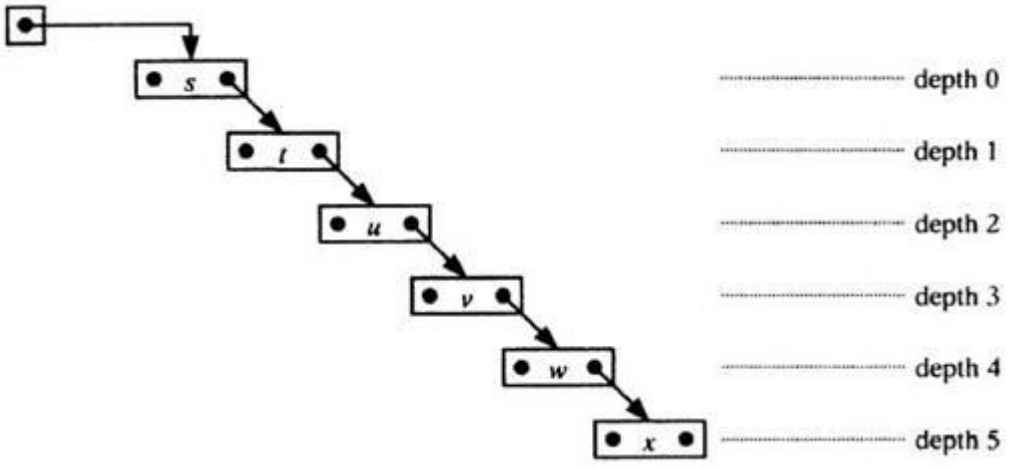
- празно дрво или
- непразно, во кој случај коренот содржи елемент, врска кон лево бинарно пребарувачко дрво со елементи помали од елементот на коренот и врска кон десно бинарно пребарувачко поддрво со елементи поголеми од елементот на коренот.

Совршено бинарно дрво со висина h е бинарно дрво со следните особини

- Ако h=0 тогаш левото и десното поддрво се празни
-Ако h>0 тогаш и левото и десното поддрво се совршени бинарни дрва со висина h-1



Временската сложеност во најлош случај е O(log n). Доколку дрвото не е совршено, временската сложеност ќе биде поголема. Дрвото во примерот се однесува како едностран лист.



Алгоритам за пребарување на бинарно пребарарувачко дрво

1. Поставете го curr на вредноста на коренот
2. Повторувај:

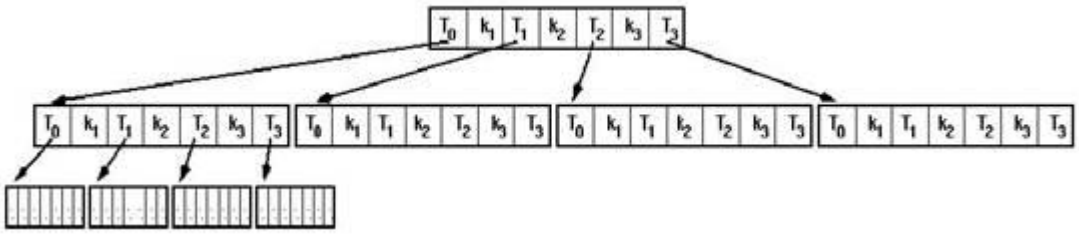
   2.1. Ако curr има вредност nil, заврши
   2.2. Инаку, ako target има иста вредност со curr, заврши со излез curr
   2.3. Инаку, ako target има помала вредност од curr, дајте му ја на curr вредноста на неговото лево дете
   2.4. Инаку, ako target има поголема вредност од curr, дајте му ја на curr вредноста на неговото десно дете

N-арни пребарувачки дрва

N-арно дрво е:

Празно дрво
Еден јазол има n-1 клучеви и n деца. Секој јазол има покажувачи на поддрва и клучеви: поддрво | клуч | поддрво | клуч | поддрво | клуч | поддрво

- Сите клучеви што се лево од некој клуч се помали од него
- Сите клучеви што се десно од некој клуч се поголеми од него



B-дрво е балансирано N-арно дрво кај кое:

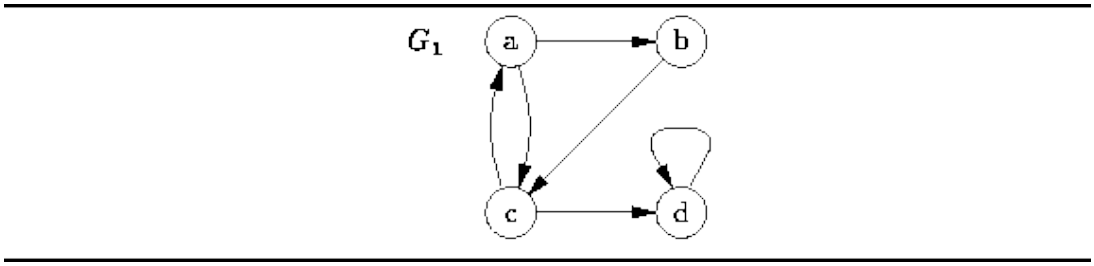
1. Сите листови се на исто ниво
2. Сите јазли освен коренот и листовите имаат најмалку n/2 а најмногу n деца. Коренот има најмалку 2 а најмногу n деца.

```
btree = ^node;
   node  = record
      d : 0..2*M;
      k : array [1..2*M] of typekey;
      p : array [0..2*M] of btree
      end;

procedure search( key : typekey; t : btree );

   var i : integer;
   begin
   if t=nil then {*** Not Found ***}
      notfound( key )
   else with t^ do begin
      i := 1;
      while ( i<d ) and ( key>k[i] ) do i := i+1;
      if key = k[i] then   {*** Found ***}
         found( t^, i )
      else if key < k[i] then search( key, p[i-1] )
         else            search( key, p[i] )
      end
   end;
```

Графови

Повеќе за графови – во предавањето за Графови.
Директен или насочен граф е подреден пар G=(V,E) со следните својства:

- V е конечно, непразно множество. Елементите на V се наречени темиња.
- E е конечно множество од подредени парови на темиња. Елементите на ова множество се нарекуваат ребра.
- Реброто (a, b) е различно од реброто (b, a)



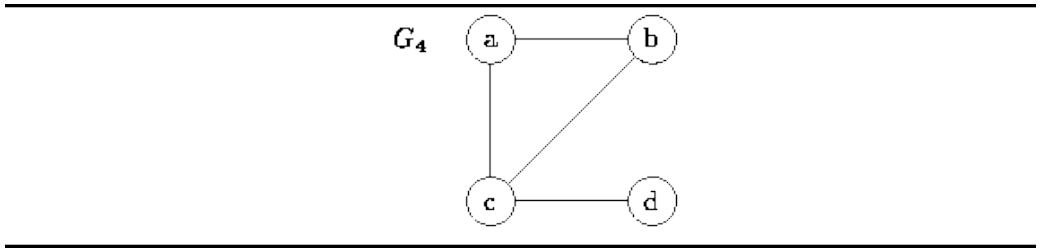
G1=(V1,E1)

V1={a, b, c, d}

E1={(a, b), (a, c), (b, c), (c, a), (c, d), (d, d)}

Индиректен или ненасочен граф со следните својства:

1. V е конечно, непразно множество. Елементите на V се наречени темиња.E е конечно множество од подредени парови на темиња.
2. Елементите на ова множество се нарекуваат ребра.
3. Реброто (a, b) е исто со реброто (b, a)



G2=(V2,E2)

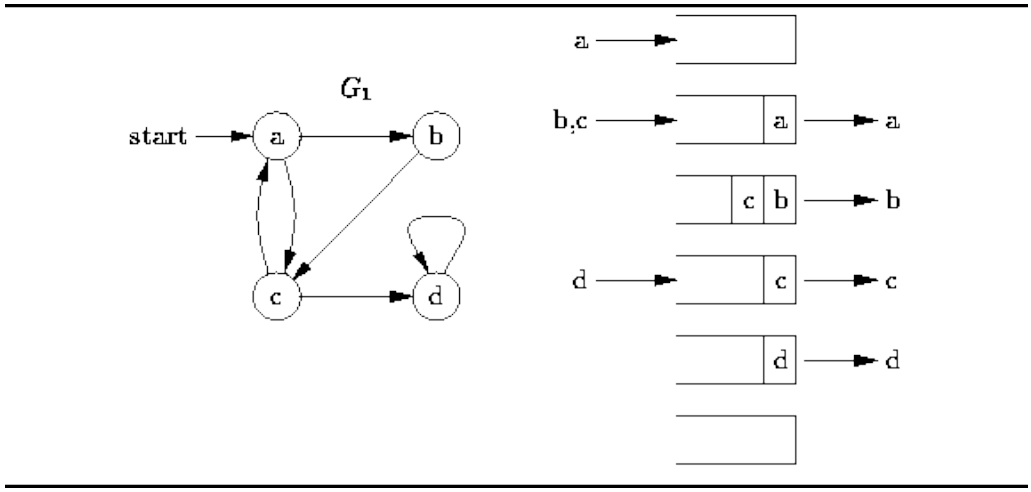
V2={a, b, c, d}

E2={(a, b), (a, c), (b, c), (c, d)}

Пребарување на графови по длабочина (Depth-First Search)

Овој метод може да се примени и на дрва, при што пребарувањето почнува од коренот. Кај графовите мора да го назначиме почетното теме. Со пребарувањето по длабочина се посетува теме а потоа рекурзивно се посетува секое теме поврзано со него. Попрецизно, откако ќе го одбереме почетното теме потоа се посетува теме поврзано со почетното и се продолжува со посетување на сите со него поврзани темиња.

Доколку почетното теме е поврзано со повеќе темиња, потоа се поситуваат и тие како и темињата поврзани со нив. Притоа графот може да содржи циклуси но секое теме мора барем еднаш да биде посетено. Затоа се памтат темињата што биле посетени. Во примерот подолу за почетно теме се зема C, потоа се посетува A, како теме поврзано со C па во длабочина B како теме поврзано со A. Потоа се навраќаме на темето D како второ теме поврзано со C.



```
DepthFirstSearch(G);
begin
  for each vertex x in V do num[x]:=0;
  TreeEdges:= 0;
  i:= 0;
  for each vertex x in V do
begin
        if num[x] = 0 then DepthFirstSearch(x);
  end;
  DepthFirstSearch(v);
  i:= i + 1;
  num[v]:= i;
  for each vertex w in Adj[v] do
  begin
        if num[w] = 0 then { w е ново теме}
        TreeEdges:= TreeEdges (v,w); {(v;w) е ново ребро}
        DepthFirstSearct(w);
  end;
end;
```