Бинарен хип

(Binary heap)

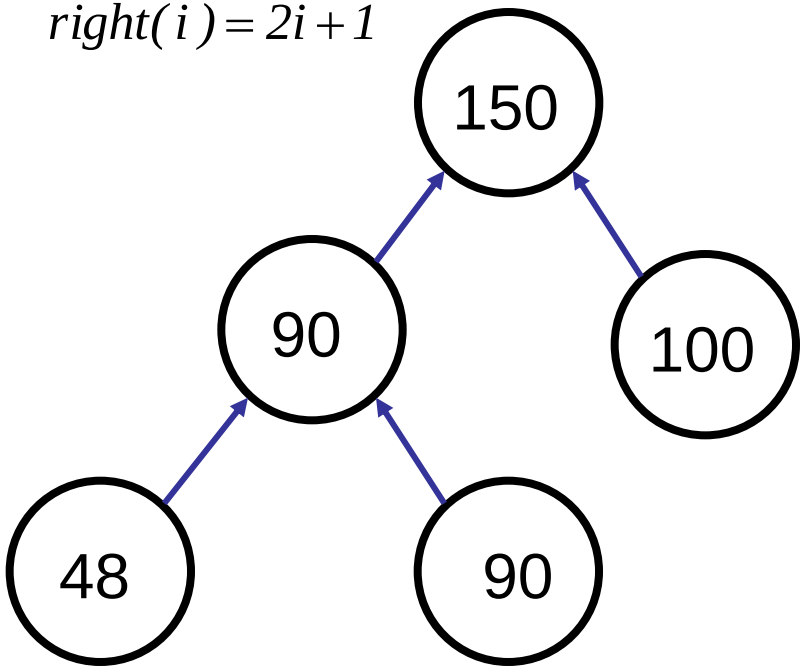
Што е бинарен хип? (1)

Бинарно дрво за кое важат одредени правила на игра.

$$parent(i) = \lfloor i/2 \rfloor$$

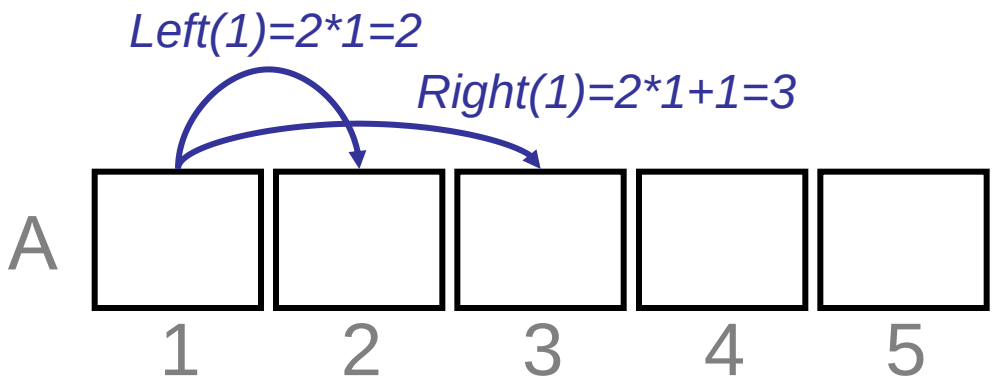
$$left(i) = 2i$$

$$right(i) = 2i + 1$$



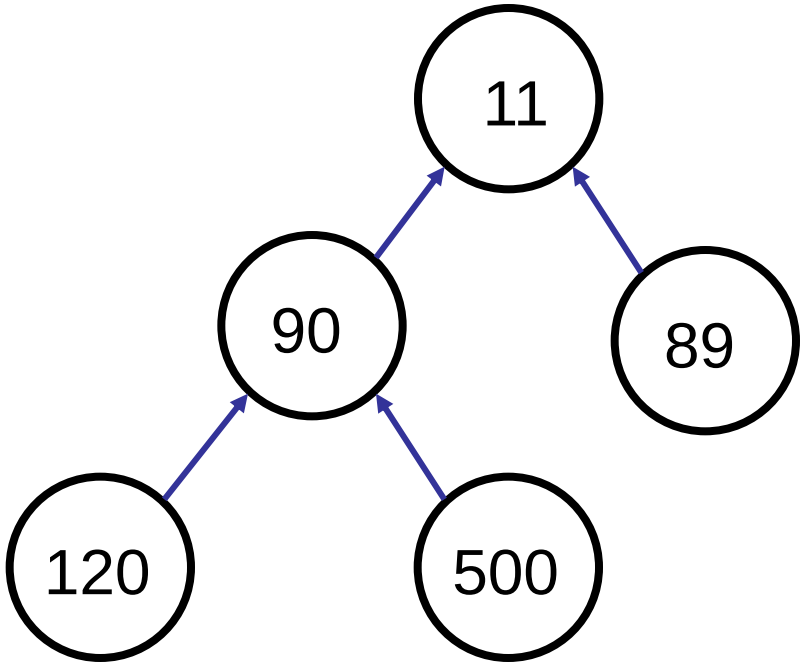
Макс-хип правило

Јазолот-родител има поголема или еднаква вредност од неговите јазли-деца.



$$A[\,parent(i)\,] \geq A[\,i\,]$$

Што е бинарен хип? (2)



Мин-хип правило

Јазолот-родител има помала или еднаква вредност од неговите јазли-деца.

$$A[\, parent(i)\,] \leq A[\, i\,]$$

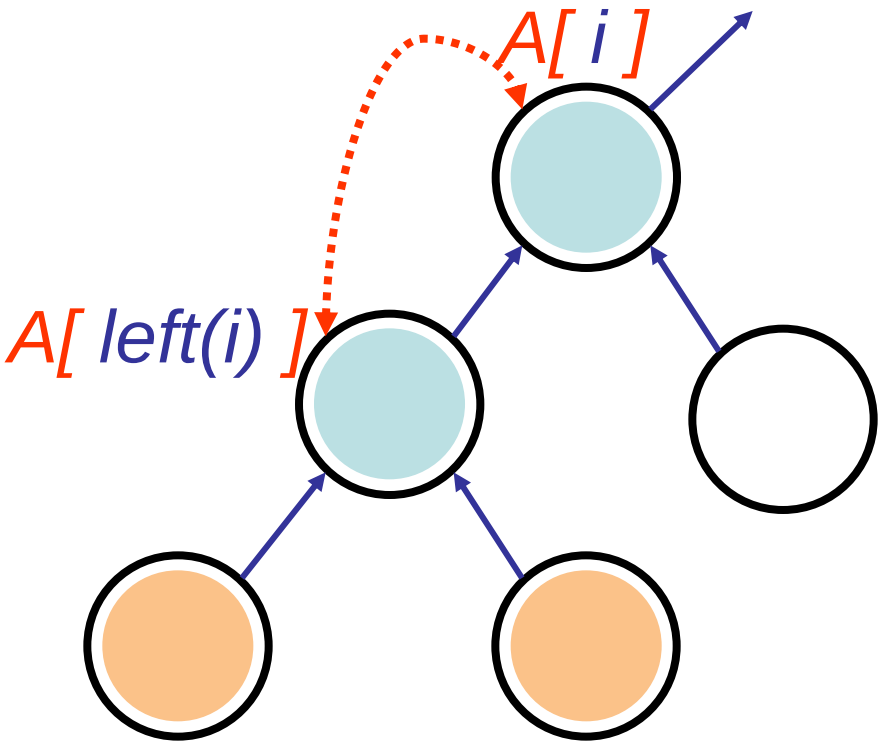
Што е бинарен хип? (3)

Големината на низата во којашто е складиран хипот не е иста со големината на хипот.

Ниеден елемент A[i] за којшто важи *heap_size(A) < i <= length(A)* не е дел од хипот !

Одржување на макс-хип правилото (1)

$A[\,parent(i)\,] \geq A[\,i\,]$



Макс-хип правилото мора да биде задоволено за секој јазол

Која постапка ќе ја спроведеме за i-тиот јазол за да се осигураме дека ваквото правило е задоволено ?

Ако A[i]<A[left(i)] ?

Одржување на макс-хип правилото (1)

```
MAX-HEAPIFY(A,i)

     L←Left(i)
     R←Right(i)

     If (L<=heap_size(A))and(A[L]>A[R])
          then largest←L
          else largest←i

     If (R<=heap_size(A))and(A[R]>A[largest])
          then largest←R

     If(largest<>i)
          then exchange A[i]<->A[largest]
               MAX-HEAPIFY(A,largest)
```

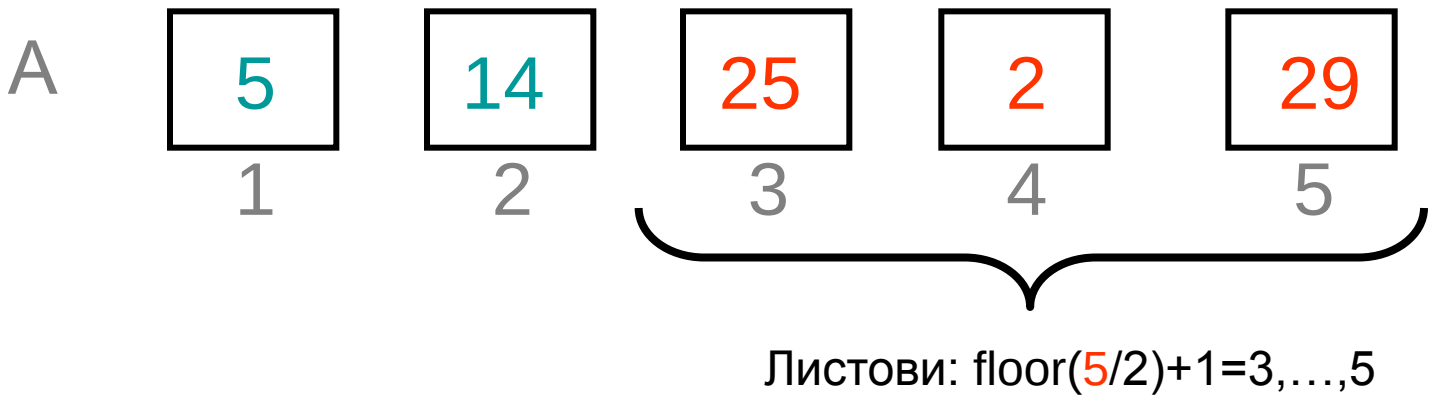
Од произволна низа во хип ? (1)

Имаме низа со n елементи, сакаме да ја подредиме така што ќе добиеме хип структура, односно ќе биде задоволено макс-хип правилото.

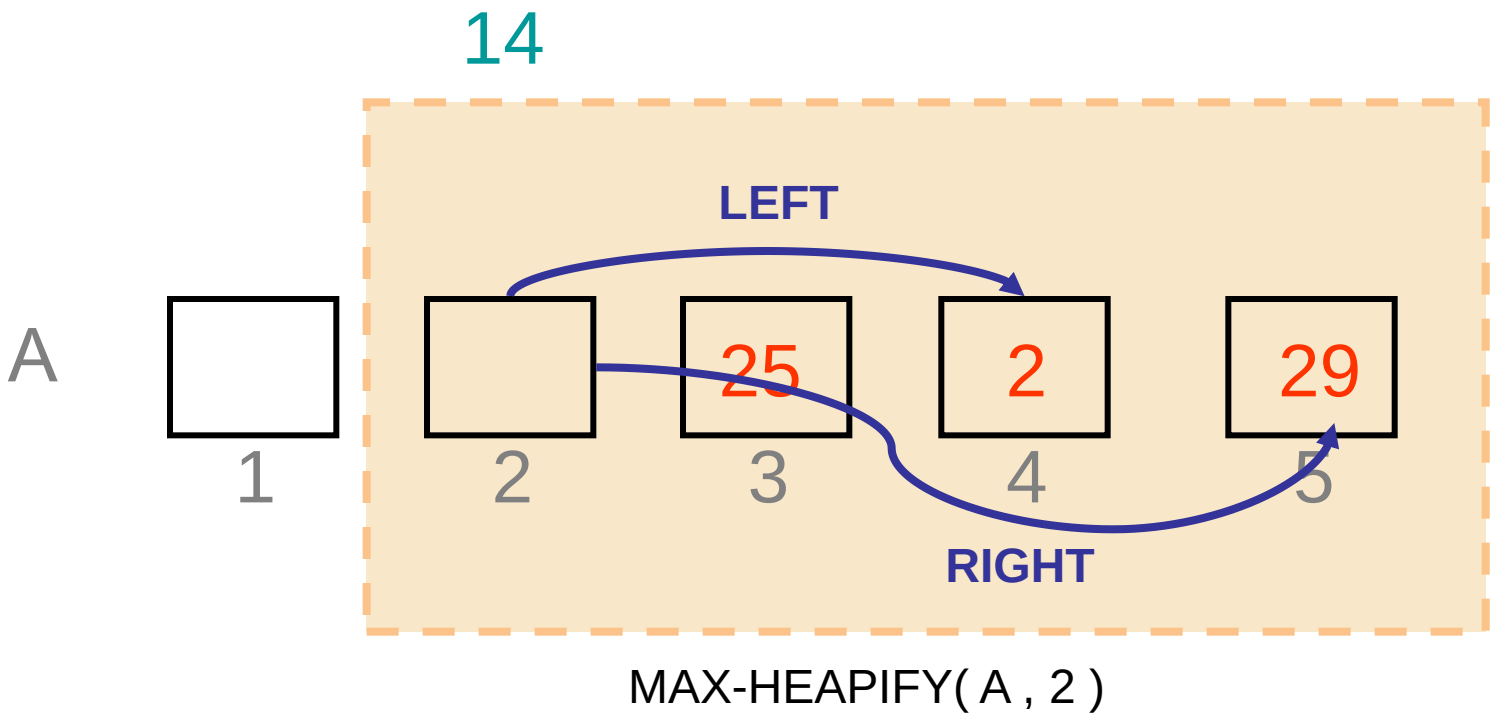
За ваквиот хип ќе важи дека

heap_size(A)=length(A)

Пола од елементите во хипот A[floor(n/2)+1],…,A[n] се листови на хип-дрвото. Зошто ?*

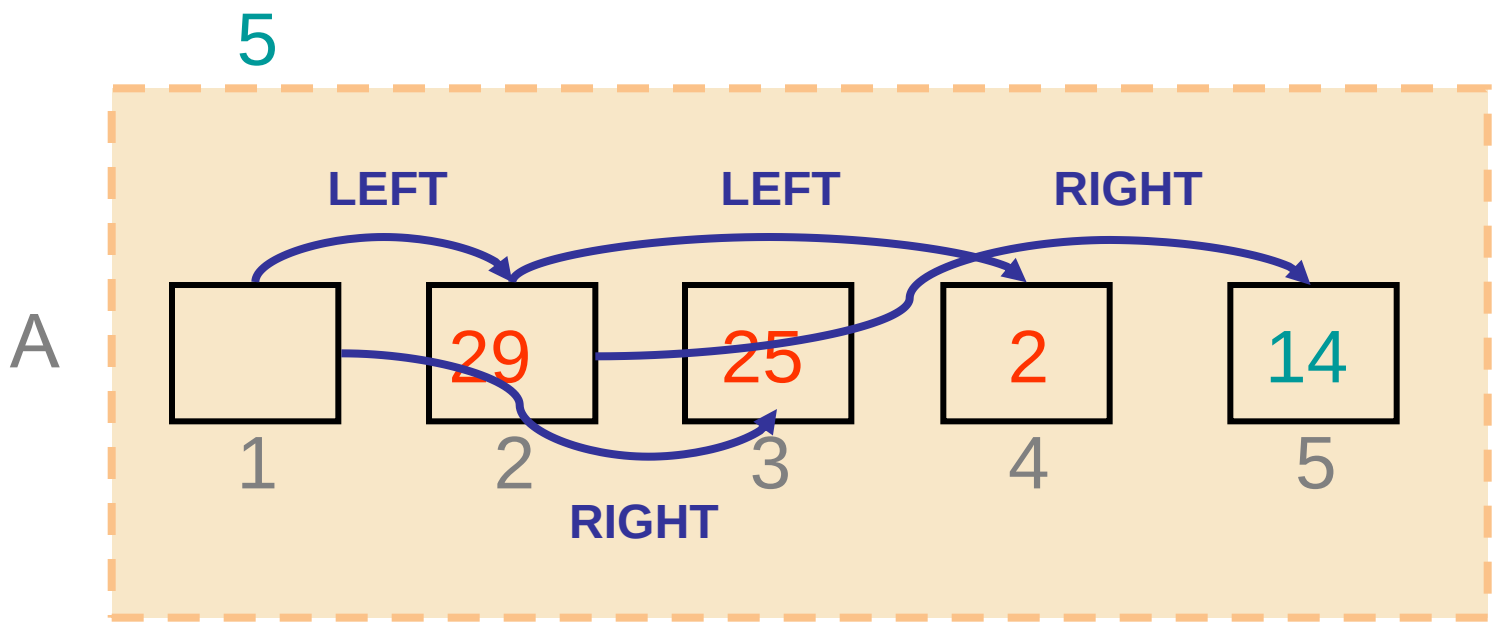


Од произволна низа во хип ? (2)



MAX-HEAPIFY(A , 2)

Од произволна низа во хип ? (3)



MAX-HEAPIFY(A , 1)

Од произволна низа во хип ? (4)

```
BUILD-MAX-HEAP(A)

     heap_size(A)←length(A)
     for i←floor(length(A)/2) downto 1 do
          MAX-HEAPIFY(A,i)
```

Приоритетни редови

ELEMENT -X | KEY

Врз приоритетниот ред A (што во суштина е бинарен хип) можат да се вршат следниве операции:

MAX-HEAP-INSERT(A,x) – додавање на елемент x во A.

HEAP-MAXIMUM(A) – го враќа елементот од A со најголем клуч

HEAP-EXTRACT-MAX(A) – го чита и го отсранува елементот од A со најголем клуч

HEAP-INCREASE-KEY(A,x,key) – ја менува вредноста на клучот на елементот x со поголемата вредност key

HEAP-MAXIMUM(A)

```
HEAP-MAXIMUM(A)
        return A[1]
```

HEAP-EXTRACT-MAX(A)

```
HEAP-EXTRACT-MAX(A)

        if(heap_size(A)<1)
                then ERROR-NO-ELEMENTS
                     exit
        max←A[1]
        A[1]←A[heap_size(A)]
        heap_size(A)←heap_size(A)-1
        MAX-HEAPIFY(A,1)
        return max
```

HEAP-INCREASE-KEY(A,i,key)

```
HEAP-INCREASE-KEY(A,i,key)

          if (key<A[i])
                    then ERROR-NEW-KEY-SMALLER-THAN-CURRENT-KEY
                         exit
          A[i]←key
          while (i>1) and (A[parent(i)]<A[i])
                    do exchange A[i]<->A[parent(i)]
                       i←parent(i)
```

MAX-HEAP-INSERT(A,key)

```
MAX-HEAP-INSERT(A,key)

        heap_size(A)←heap_size(A)+1
        A[heap_size[A]]←-INFINITY
        HEAP-INCREASE-KEY(A,heap_size(A),key)
```

Каде користат приоритетните редови ?

Секаде кадешто постои некакво подредување според некаков приоритет.

Убавината на приоритетните редови е тоа што секогаш во коренот се наоѓа максимумот или минимумот, и што е најважно многу лесно се одржува макс-хип правилото односно мин-хип правилото.

Задачи(процеси) коишто треба да се извршат во оперативен систем имаат различен приоритет. Приоритетот може да се измени од страна на корисникот.

Што тогаш треба да се направи за да се испочитуваат правилата ?

Референци

[1] *Introduction to algorithms* – Thomas H. Cormen et al., MIT Press

[2] TopCoder Algorithm Tutorials