Heapsort

O algoritmo **heapsort** é um algoritmo de ordenação generalista, e faz parte da família de algoritmos de ordenação por seleção. Foi desenvolvido em 1964 por **Robert W. Floyd** e **J.W.J. Williams**.

Definição

Tem um desempenho em tempo de execução muito bom em conjuntos ordenados aleatoriamente, tem um uso de memória bem comportado e o seu desempenho em pior cenário é praticamente igual ao desempenho em cenário médio. Alguns algoritmos de ordenação rápidos têm desempenhos espectacularmente ruins no pior cenário, quer em tempo de execução, quer no uso da memória. O Heapsort trabalha no lugar e o tempo de execução em pior cenário para ordenar *n* elementos é de O (*n* lg *n*). Lê-se logaritmo (ou log) de "n" na base 2. Para valores de *n*, razoavelmente grande, o termo lg *n* é quase constante, de modo que o tempo de ordenação é quase linear com o número de itens a ordenar.

Características

- Comparações no pior caso: $2n \log_2 n + O(n)$ é o mesmo que $2n \lg n + O(n)$
- Trocas no pior caso: $n \log_2 n + O(n)$ é o mesmo que $n \lg n + O(n)$
- Melhor e pior caso: $O(n \log_2 n)$ é o mesmo que $O(n \lg n)$

Funcionamento

O heapsort utiliza uma estrutura de dados chamada heap para ordenar os elementos a medida que os insere na estrutura. Assim, ao final das inserções, os elementos podem ser sucessivamente removidos da raiz da heap, na ordem desejada, lembrando-se sempre de manter a propriedade de max-heap.

A heap pode ser representada como uma árvore ou como um vetor. Para uma ordenação crescente, deve ser construído um heap máximo (o maior elemento fica na raiz). Para uma ordenação decrescente, deve ser construído um heap mínimo (o menor elemento fica na raiz).

Implementações

Assembly x86-gas-Linux

```
/* void heap_sort_as(int *x, int n);*/
.globl heap_sort_as
heap_sort_as:
      pushl %ebp
      movl %esp, %ebp
      /* 8(%ebp) -> arranjo*/
      /* 12(%ebp) -> num de elementos *4 */
      movl 12(%ebp), %eax
      movl $4, %ebx
      mul %ebx
      pushl %eax
      pushl 8(%ebp)
      call montar_heap_as
      addl $4, %esp
```

```
      popl %eax
      subl $4, %eax          /* eax eh (n*4)-4*/
      /* faz a troca */
      movl 8(%ebp), %ebx     /* tmp = *x */
      pushl %eax
      movl %ebx, %ecx
      addl %eax, %ecx
      movl (%ebx), %eax
      movl (%ecx), %edx
      movl %eax, (%ecx)
      movl %edx, (%ebx)
      popl %eax              /* eax representa (n*4)-4 */
      pushl $0
      pushl %eax
      pushl 8(%ebp)
heap_sort_as_loop:
      cmp $1, %eax
      jle heap_sort_as_fim
      call man_heap_as
      movl 4(%esp), %eax
      subl $4, %eax
      /* faz a troca */
      movl 8(%ebp), %ebx     /* tmp = *x */
      pushl %eax
      movl %ebx, %ecx
      addl %eax, %ecx
      movl (%ebx), %eax
      movl (%ecx), %edx
      movl %eax, (%ecx)
      movl %edx, (%ebx)
      popl %eax              /* eax representa (n*4)-4 */
      movl %eax, 4(%esp)
      jmp heap_sort_as_loop
heap_sort_as_fim:
      leave
      ret
montar_heap_as:
      pushl %ebp
      movl %esp, %ebp
      /* 8(%ebp) -> arranjo*/
      /* 12(%ebp) -> num de elementos *4 */
      movl 12(%ebp), %eax
      movl $2, %ecx
      cltd
      idivl %ecx
      movl $4, %ecx
      cltd
```

```
      idivl %ecx
      mul %ecx
      subl $4, %eax /* eax eh h*/
      pushl %eax
      pushl 12(%ebp)
      pushl 8(%ebp)
montar_heap_as_whmz:
      cmp $0, %eax
      jl montar_heap_as_fim
      call man_heap_as
      movl 8(%esp), %eax
      subl $4, %eax
      movl %eax, 8(%esp)
      jmp montar_heap_as_whmz
montar_heap_as_fim:
      leave
      ret
man_heap_as:
      pushl %ebp
      movl %esp, %ebp
      /* 8(%ebp) -> arranjo */
      /* 12(%ebp) -> num de elementos *4 */
      /* 16(%ebp) -> pai *4 */
      movl 16(%ebp), %eax
      pushl %eax
      movl $2, %ebx
      mul %ebx
      addl $4, %eax             /* agora eax eh f*/
man_heap_as_wfmn:
      cmp 12(%ebp), %eax
      jge man_heap_as_fim
      movl %eax, %ebx
      addl $4, %ebx             /* ebx eh f2 */
      movl 8(%ebp),%edx
      addl %eax, %edx           /* edx aponta p/ x[f] */
      movl (%edx), %edx
      cmp 12(%ebp), %ebx
      jge man_heap_as_testetroca
      movl 8(%ebp), %ecx
      addl %ebx, %ecx          /* ecx aponta p/ x[f2] */
      movl (%ecx), %ecx
      cmp %edx, %ecx
      jle man_heap_as_testetroca
      movl %ebx, %eax          /* f=f2 ou seja maior filho eh f2 */
      movl %ecx, %edx          /* movimentacao apenas p/ testar */
man_heap_as_testetroca:
      popl %ebx           /* ebx eh p */
```

```
      movl 8(%ebp), %ecx
      addl %ebx, %ecx     /* ecx eh x[p]*/
      movl (%ecx), %ecx
      cmp %ecx, %edx
      jle man_heap_as_fim
      /*fazer a troca */
      pushl %eax          /* salva f na pilha*/
      addl 8(%ebp), %eax
      movl %ecx, (%eax)   /* x[f] = x[p] */
      addl 8(%ebp), %ebx
      movl %edx, (%ebx)   /* x[p] = x[f] */
      movl (%esp), %eax
      movl $2, %ebx
      mul %ebx
      addl $4, %eax
      jmp man_heap_as_wfmn
man_heap_as_fim:
      leave
      ret
```

Código em C

```
void heapsort(tipo* a, int n)
{
   int i = n/2, pai, filho;
   tipo t;

   for (;;)
   {
      if (i > 0)
      {
          i--;
          t = a[i];
      }
      else
      {
          n--;
          if (n == 0)
             return;
          t = a[n];
          a[n] = a[0];
      }

      pai = i;
      filho = i*2 + 1;

      while (filho < n)
      {
```

```
            if ((filho + 1 < n)  &&  (a[filho + 1] > a[filho]))
                filho++;
            if (a[filho] > t)
            {
               a[pai] = a[filho];
               pai = filho;
               filho = pai*2 + 1;
            }
            else
               break;
         }
         a[pai] = t;
      }
}
```

Código em C++

```
template<class T>
void heap_sort( std::vector<T> &lista )
{
    int tam = static_cast<int>( lista.size() ), i;

    for( i = tam/2 - 1; i >= 0; --i )
    {
       maxHeapify(lista, i , tam );
    }

    std::vector<T>::reverse_iterator elem;

    for( elem = lista.rbegin(); elem != lista.rend(); elem++ )
    {
       std::iter_swap( elem, lista.begin() );
       maxHeapify( lista, 0, --tam );
    }
}

template<class T>
void maxHeapify( std::vector<T> &lista, const int pos, const int n )
{
    int max = 2 * pos + 1;

    if( max < n )
    {
       if( (max+1) < n && lista.at(max) < lista.at(max+1) )
       {
          ++max;
       }
       if( lista.at(max) > lista.at(pos) )
```

```
        {
           std::swap( lista[max], lista[pos] );
           maxHeapify( lista, max, n );
        }
    }
}
```

Código em Java

```
        public static void heapSort(int[] v)
        {
          buildMaxHeap(v);
          int n = v.length;

          for (int i = v.length - 1; i > 0; i--)
          {
             swap(v, i , 0);
             maxHeapify(v, 0, --n);
          }
        }
        private static void buildMaxHeap(int[] v)
        {
           for (int i = v.length/2 - 1; i >= 0; i--)
              maxHeapify(v, i , v. length );
        }
        private static void maxHeapify(int[] v, int pos, int n)
        {
           int max = 2 * pos + 1, right = max + 1;
           if (max < n)
           {
              if ( right < n && v[max] < v[right])
                 max = right;
              if (v[max] > v[pos])
              {
                 swap(v, max, pos);
                 maxHeapify(v, max, n);
              }
           }
        }

        public static void swap ( int[ ] v, int j, int aposJ )
        {
           int aux = v [ j ];
           v [ j ] = v [ aposJ ];
           v [ aposJ ] = aux;
        }
```

Código em Java (5.0)

```
    public static <T extends Comparable<? super T>> void heapSort(T[] v) {
        buildMaxHeap(v);
        int n = v.length;

        for (int i = v.length - 1; i > 0; i--) {
            swap(v, i, 0);
            maxHeapify(v, 0, --n);
        }
    }

    private static <T extends Comparable<? super T>> void buildMaxHeap(T v[]) {
        for (int i = v.length / 2 - 1; i >= 0; i--)
            maxHeapify(v, i, v.length);
    }

    private static <T extends Comparable<? super T>> void maxHeapify(T[] v, int pos,
            int n) {
        int max = 2 * pos + 1, right = max + 1;
        if (max < n) {
            if (right < n && v[max].compareTo(v[right]) < 0)
                max = right;
            if (v[max].compareTo(v[pos]) > 0) {
                swap(v, max, pos);
                maxHeapify(v, max, n);
            }
        }
    }

    public static void swap(Object[] v, int j, int aposJ) {
        Object aux = v[j];
        v[j] = v[aposJ];
        v[aposJ] = aux;
    }
```

Código em python

```
def heapsort(lst):
  ''' Heapsort. Note: this function sorts in-place (it mutates the
list). '''

  # in pseudo-code, heapify only called once, so inline it here
  for start in range((len(lst)-2)/2, -1, -1):
    siftdown(lst, start, len(lst)-1)

  for end in range(len(lst)-1, 0, -1):
    lst[end], lst[0] = lst[0], lst[end]
    siftdown(lst, 0, end - 1)
```

```
  return lst

def siftdown(lst, start, end):
  root = start
  while True:
    child = root * 2 + 1
    if child > end: break
    if child + 1 <= end and lst[child] < lst[child + 1]:
      child += 1
    if lst[root] < lst[child]:
      lst[root], lst[child] = lst[child], lst[root]
      root = child
    else:
      break
```

Ligações externas

- HeapSort [1]
- (http://www.ime.usp.br/~pf/algoritmos/aulas/hpsrt.html)
- *Animação do processo de ordenação pelo Heapsort* [2]
- Heapsort in C++ [3]
- Heapsort code [4]

Referências

[1] http://c2.com/cgi/wiki?HeapSort
[2] http://www.cs.ubc.ca/spider/harrison/Java/sorting-demo.html
[3] http://www.datastructures.info/what-is-heap-sort-and-how-does-it-work-heap-sort-algorithm/
[4] http://www.algorithm-code.com/wiki/Heapsort

Fontes e Editores da Página

Heapsort *Fonte*: http://pt.wikipedia.org/w/index.php?oldid=20397603 *Contribuidores*: Brunobraga, Ccuembej, EduM, Jorge, Leonardo.stabile, Luisfelipe1706, Mikue, Osias, Pedro Ivan de Albuquerque Lima, Rafael.afonso, Rafaeldhias, Santana-freitas, Thegoergen, Thiagoharry, 34 edições anónimas

Licença