Princípios de Desenvolvimento de Algoritmos MAC122

Prof. Dr. Paulo Miranda
IME-USP

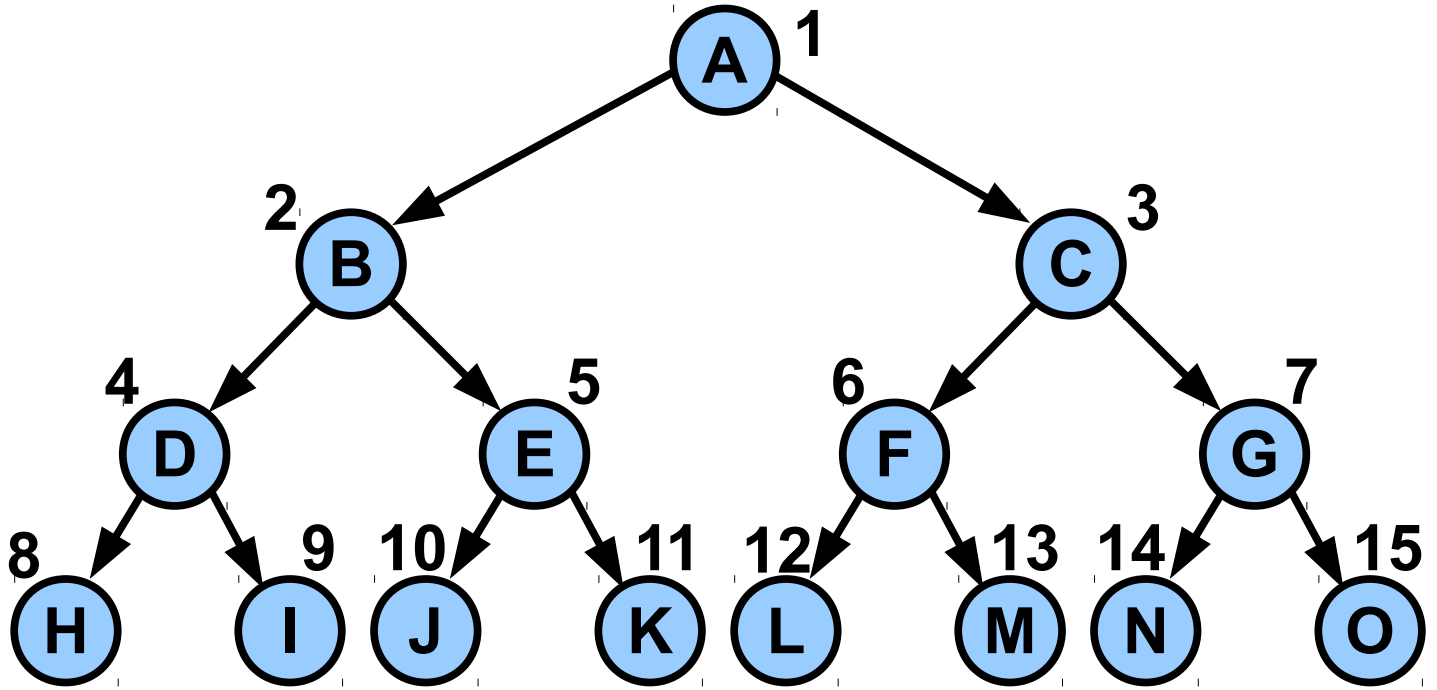
Filas de prioridade

Filas de prioridade

- Em *filas de prioridade,* os elementos com maior prioridade possuem preferência (saem primeiro da fila).
- Árvores binárias podem ser utilizadas na implementação eficiente de filas de prioridade, com suporte a operações de seleção ou remoção do maior (ou menor) elemento de uma coleção.
  - Para isso a árvore binária deve ter as seguintes propriedades:
    - A árvore é *completa* ou *quase completa*;
    - Para cada nó, temos que, o seu valor é maior do que o dos seus filhos.

Árvore binária completa

- Dizemos que uma árvore binária de altura h é completa se ela contiver o número máximo de nós ($2^h - 1$).

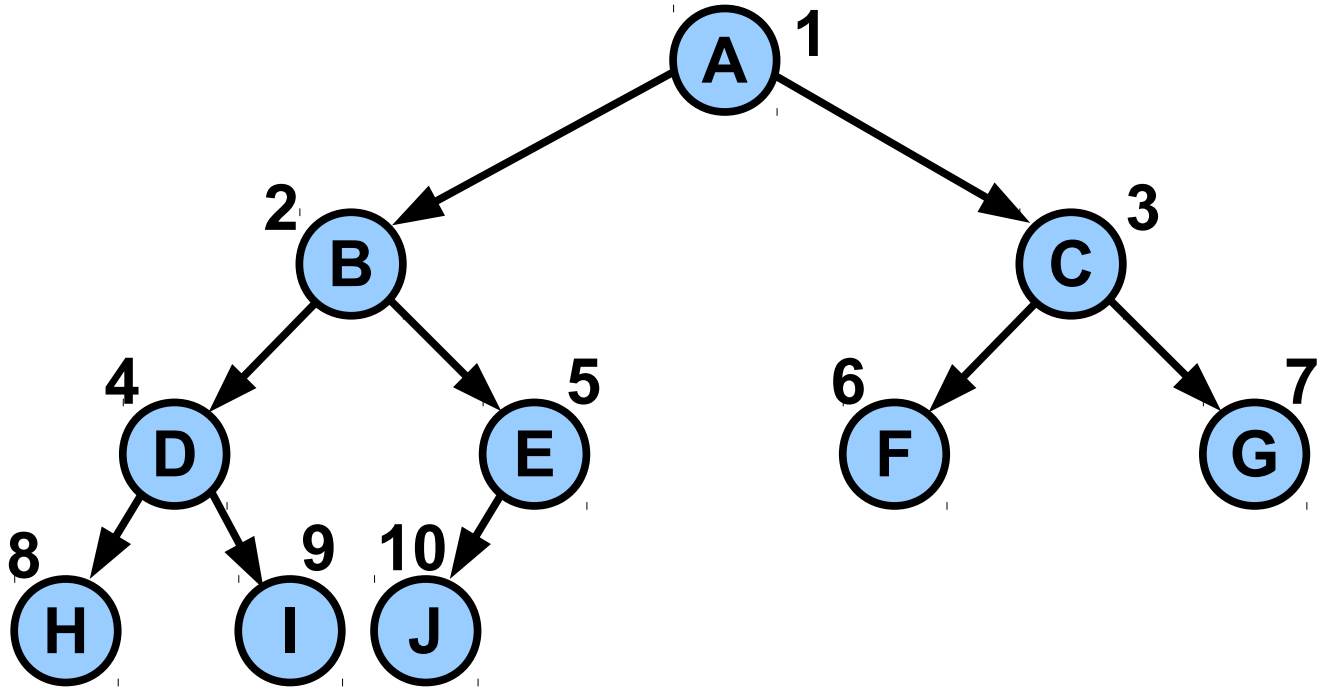


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Pai(i) = $\lfloor i/2 \rfloor$, FilhoEsq(i) = 2i, FilhoDir(i) = 2i + 1

Árvore quase completa

- Semelhantes as completas, porém falta uma seqüência de elementos no final (direita) do último nível.

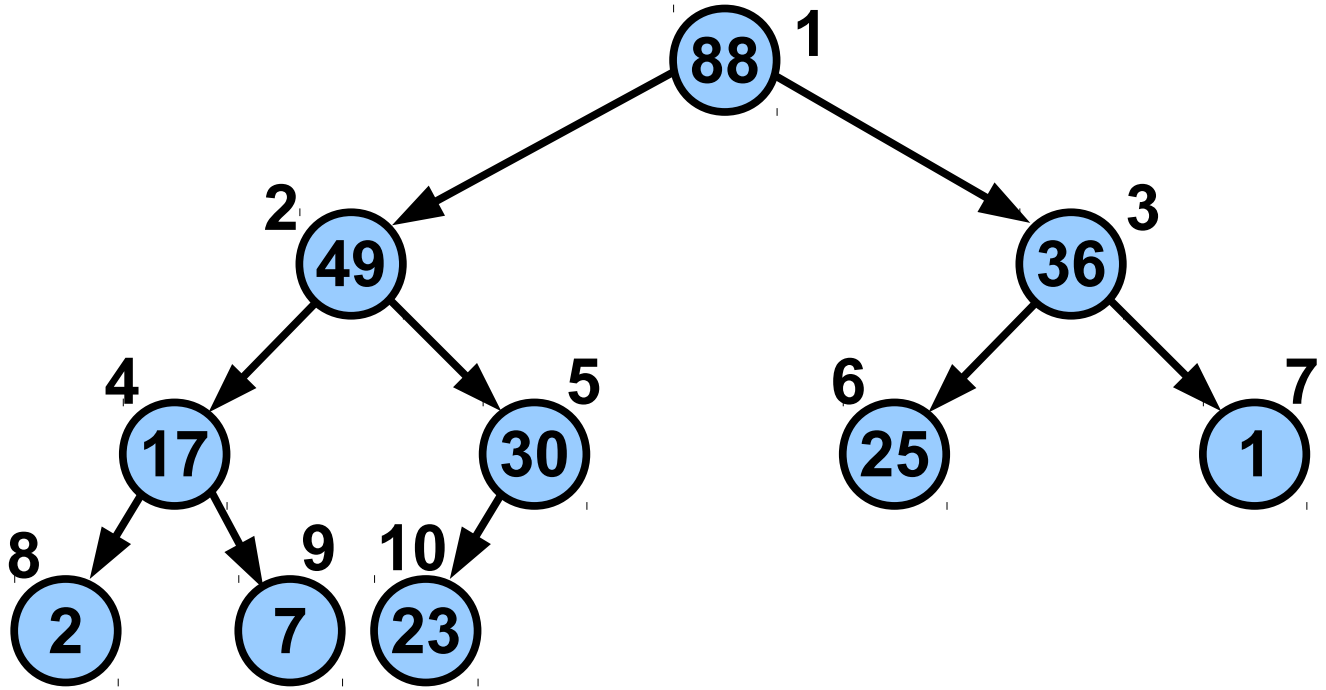


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D	E	F	G	H	I	J					

Pai(i) = $\lfloor i/2 \rfloor$, FilhoEsq(i) = 2i, FilhoDir(i) = 2i + 1

Heap - binário

- Para cada nó, temos que, o seu valor é maior do que o dos seus filhos.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
88	49	36	17	30	25	1	2	7	23					

Pai(i) = $\lfloor i/2 \rfloor$, FilhoEsq(i) = 2i, FilhoDir(i) = 2i + 1

Heap - binário

- Definição típica da estrutura utilizada.

```
#define HEAPMAX 1000

typedef int TipoChave;
typedef int TipoInfo;

typedef struct _Registro{
  TipoChave chave;
  TipoInfo  info;
} Registro;


typedef struct _Heap{
  int n; /* Comprimento atual. */
  Registro V[HEAPMAX];
} Heap;
```

Heap - binário

- Funções para calcular os índices: Pai, filhos esq. e dir.

```
int HeapPai(int i){
  return ((i - 1) / 2);
}


int HeapFilhoEsq(int i){
  return (2 * i + 1);
}


int HeapFilhoDir(int i){
  return (2 * i + 2);
}
```

Heap - binário

- Operação básica de *subida* no heap.

```
void Sobe(Heap *p, int i){
  Registro reg;
  TipoChave chv;
  int j;

  reg = p->V[i];
  chv = reg.chave;
  j = HeapPai(i);

  while(i>0 && p->V[j].chave<chv){
    p->V[i] = p->V[j];
    i = j;
    j = HeapPai(j);
  }
  p->V[i] = reg;
}
```

Heap - binário

- Operação básica de *descida* no heap.

```
void Desce(Heap *p, int i){
  int k;
  Registro reg;
  TipoChave chv;
  reg = p->V[i];
  chv = reg.chave;
  k = HeapFilhoEsq(i);
  while(k<p->n){
    if(k+1<p->n){ /* Pega o maior filho. */
      if(p->V[k].chave < p->V[k+1].chave)
        k = k+1;
    }
    if(p->V[k].chave > chv){
      p->V[i] = p->V[k];
      i = k;
      k = HeapFilhoEsq(k);
    }
    else break;
  }
  p->V[i] = reg;
}
```

Heap - binário

- Construção do heap.

```
/* O(n log n) */
void ConstroiHeap(Heap *p){
  int i;

  for(i=1; i<p->n; i++)
    Sobe(p, i);
}
```

```
/* O(n)  */
void OutroConstroiHeap(Heap *p){
  int i,u;

  /* último nó não folha. */
  u = HeapPai(p->n - 1);

  for(i=u; i>=0; i--)
    Desce(p, i);
}
```

Heap - binário

- Inserção e remoção.

```
void InsereHeap(Heap *p,
                Registro r){
  p->V[p->n] = r;
  p->n++;
  Sobe(p, p->n-1);
}
```

```
Registro RemoveHeap(Heap *p){
  Registro r;

  if(p->n==0) exit(-1);

  r = p->V[0];
  p->V[0]= p->V[p->n-1];
  p->n--;
  Desce(p, 0);
  return r;
}
```