filas de prioridade

fila
& FIFO
first in, first out

pilha
↳ LIFO → last in, first out

- → API de fila de prioridades crescente ←
- → API de fila de prioridades decrescente ←

→ filas de prioridade são ADT que manipulam conjuntos de "~~coisas~~ Itens" comparáveis que chamaremos de "ITENS"

→ Um item k é máximo se nenhum item é estritamente maior que k e mínimo se nenhum item é estritamente menor que k. Podemos ter mais de um item máximo e mais de um item mínimo.

→ Uma fila de prioridade Decrescente ou PQ máximo é um ADT que manipula um conjunto de itens por meio de duas operações fundamentais:

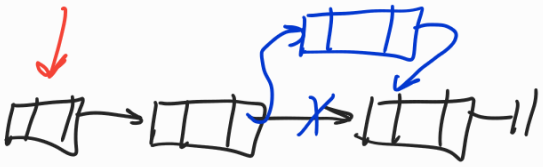
- Inserção
- Remoção

- Uma fila de prioridade crescente, ou ~~PQ de mínimo~~ é definida de maneira análoga.

Implementações de fila de prioridades

↳ listas encadeadas

Caro ↳ Se a inserção for ordenada, o custo é proporcional a $O(N)$.



Barato ↳ A remoção é rápida, custa $O(1)$

↳ Se a inserção não for ordenada

↳ Inserção é barato, $O(1)$

↳ Remoção $O(N)$

↳ Vetores?

↳ O mesmo caso que listas encadeadas

	Inserir	Remover MAX	Remover	Find MAX	Mudar Prioridade
Vet. Ordenado	N	1	N	1	N
Lista. Ordenada	N	1	1	1	N
Vet. desordenado	1	N	1	N	1
Lista desordenada	1	N	1	N	1
HEAP	$\lg(N)$	$\lg(N)$	$\lg(N)$	1	$\lg(N)$

HEAP

↳ Um tipo de árvore binária para implementar fila de prioridade

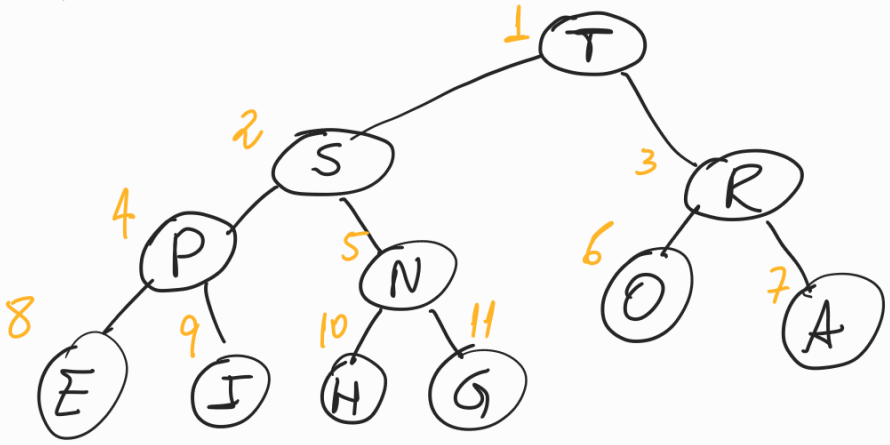
eficientemente.

↳ Dois sabores de Heaps: **Decrescente** (o máximo fica no topo) e **Crescente** (o mínimo fica no topo)

↳ Propriedade de **Heap decrescente**: o item de qualquer nó é menor ou igual que o item do pai.

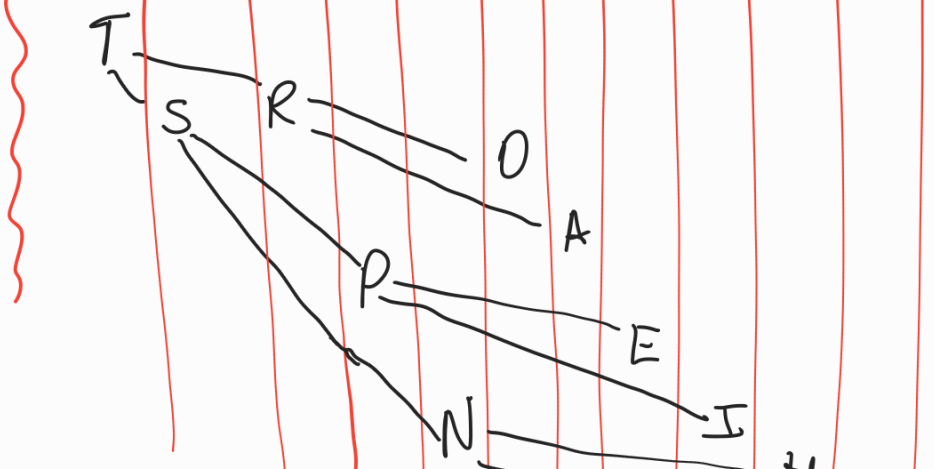
↳ Propriedade de **Heap crescente** o item de qualquer nó é maior ou igual que o item do pai.

↳ Exemplo de heap decrescente (Itens são strings de 1 caractere)

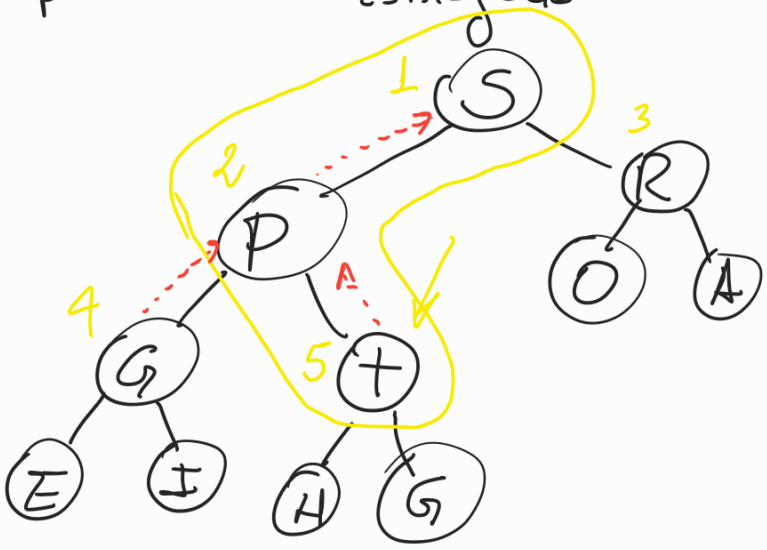


↳ Representação de Heap em um vetor $V[1..N]$ (o índice $\emptyset$ [zero] não é utilizado): os dois filhos de um nó no índice k são, respectivamente, $2*k$ e $2*k+1$. Reciprocamente, o pai de qualquer índice k é $\lfloor k/2 \rfloor$.

i	0	1	2	3	4	5	6	7	8	9	10	11	N 12	6
V[i]	–	T	S	R	P	N	O	A	E	I	H	G	K	

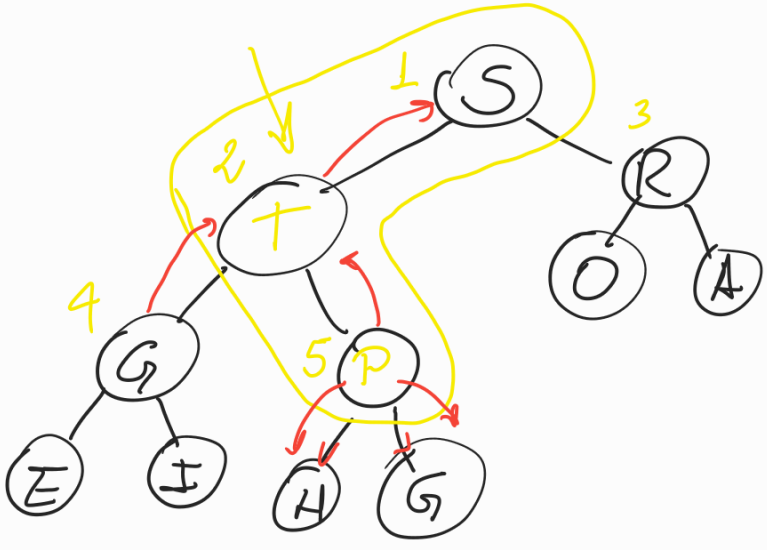


↳ Heap decrescente "estragada"

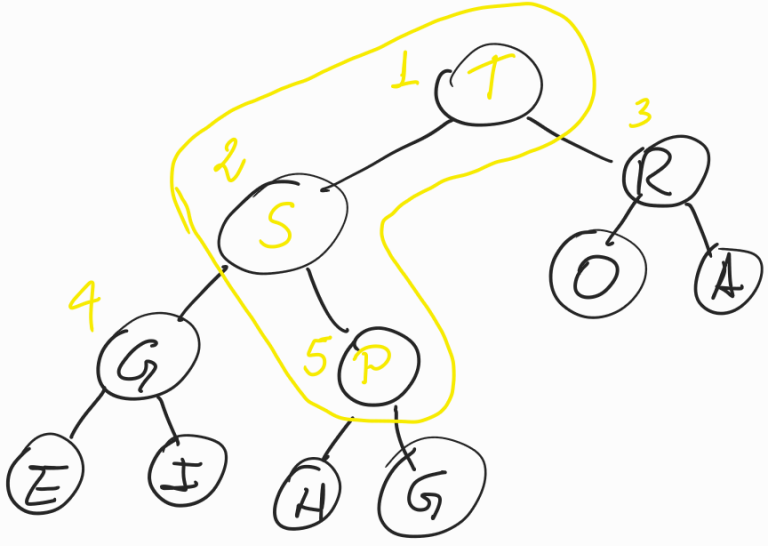


$K = 5 \therefore \lfloor \frac{5}{2} \rfloor = 2$

pai



$K = 2 \quad \therefore \lfloor \frac{2}{2} \rfloor = 1$



$K = 1$

→ "Borbulha para cima" (de-baixo-para-cima) (swim)

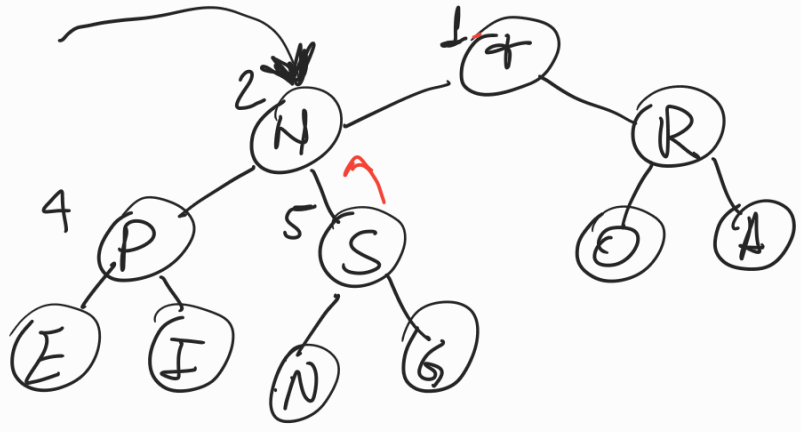
```
void fixUp(Item *v, int k)
```

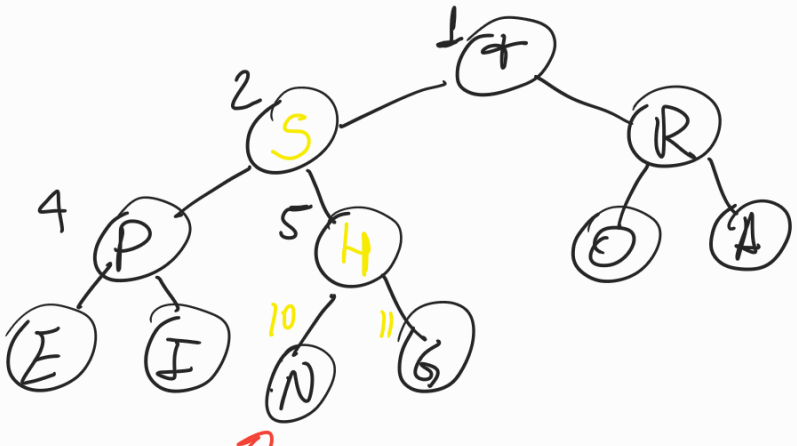
↑ fila de prioridade.

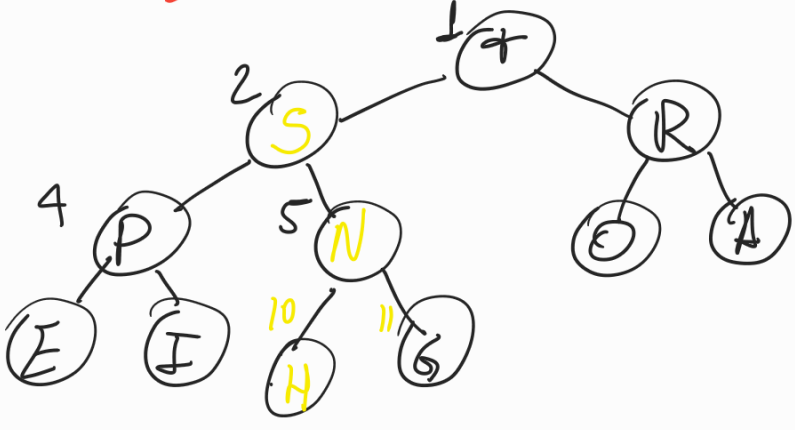
{

```
while (k > 1 && less(v[k/2], v[k]))
{
  exch(v[k], v[k/2]);
  k = k/2;
}
```

→ Heap decrescente "entrosada", começa-de-cima-para-baixo (sink)







tamanho da nossa heap em intervalo fechado

```
Void fixDown(Item *v, int k, int N)
{
  int j;
```

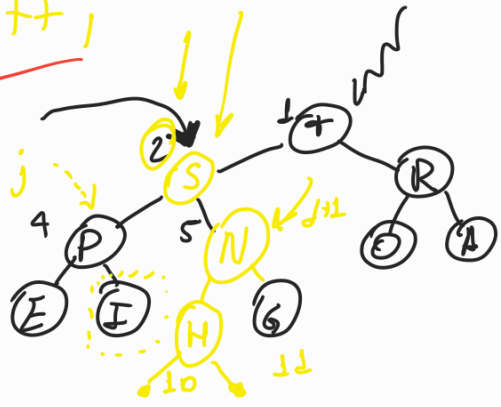
Tipo PQ elemento

```
int j;
while (2*k <= N)
{
  j = 2*k;
  if (j<N && less( v[j], v[j+1])) j++;
  if (!less(v[k], v[j])) break;
  exch (v[k], v[j]);
  k = j;
}
```

e ser anunado

k = 8 10

j = 10




```
static Item *pq;
static int N;
void PQinit(int maxN, struct pq-st *PQ)
{
  PQ->pq = malloc (sizeof(Item) * (maxN+1));
  PQ->N=0;
}
```

```
struct pq-st
{ Item *pq;
  int N;
};
```

```
int PQempty(struct pq-st *PQ)
{ return PQ->N == 0;
}
```

```
void PQinsert(struct pq-st *PQ, Item novo)
{
  PQ->pq[++PQ->N] = novo;
  fixUP(PQ->pq, PQ->N);
}
```

```
Item PQdelMax(struct pq-st *PQ)
{
  exch(PQ->pq[1], PQ->pq[PQ->N]);
  fixDown(PQ->pq, 1, --PQ->N);
```

return PQ->pq[

}

void PQchange(struct pq-st *PQ
↳ TODO!

fila de prioridade baseada em índice!

Item *data; ← genedo fone de fila de prioridade, e possui os elementos que estão sendo manipulados pelo programe.

Possui definições das funções "less" que sabem comparer os elementos do tipo Item.

↳ No lado de fila de prioridade, temos:

struct pq-ist
{
  int N;
  int *pq; // fila de prioridades, que armazena os índices de um conjunto de Item
  int *qp; // mantém a posição do heap para o elemento de índice k do conjunto de Item
}
↳ Vai funcionar como uma hash

void PQinit(struct pq-ist *PQ, int MAX)
{
  PQ->N = 0;
  PQ->pq = malloc(sizeof(int)*(MAX+1));
  PQ->qp = malloc(sizeof(int)*(MAX+1));
}

int PQempty(struct pq-ist *PQ)
{
  return !PQ->N;
}

void PQinsert(struct pq-ist *PQ, int k)
{
  PQ->N++;

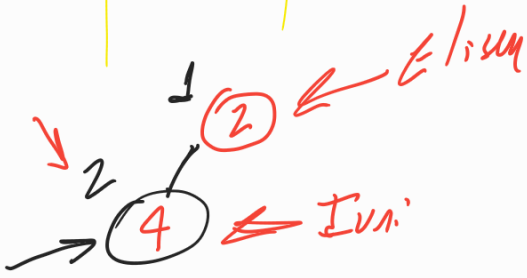
adicione k=2 N=2

k	qp[k]	pq[i]	Item Data

```
PQ->qp[k] = ...
PQ->pq[PQ->N] = k;
fixUP(PQ->pq, PQ->N);
```

exch deve ser modificado

```
}
void exch(int i, int j)
{
  int t;
  t = qp[i]; qp[i] = qp[j]; qp[j] = t;
  pq[qp[i]] = i; pq[qp[j]] = j;
}
```



exch(2, 4)

t = qp[2]; qp[2] = qp[4]; qp[4] = 2

pq[1] = 2

pq[2] = 4

```
void fixUp(Item *v, int k)
{
  while (k > 1 && less(v[k/2], v[k]))
  {
    exch(pq[k], pq[k/2]);
    k = k/2;
  }
}
```

```
void PQchangE(struct pq_ist *PQ, int k)
{
  fixUP(PQ->pq, qp[k]);
  fixDown(PQ->pq, qp[k], PQ->N);
}
```

→ Imprimir os 100 menores números de um conjunto de 10^6 (1kk) de números

```
int main(void)
{ int x;
  PQinit(1 to 2);
  for(int i=0; i<100; i++)
```

Qual o motivo de ser um pouco maior?

filhos de cada nó ficam nos índices (2*k) e (2*k+1)

```
    scanf("%d", &x);
    PQinsert(x);
}
while(scanf("%d", &x) == 1)
{
    PQinsert(x);   <- nesse ponto temos 101 elementos na PQ
    PQdelmax();
}
```

OU

```
while(scanf("%d", &x) == 1)
{
    if(x < PQespiamax())
    {
        PQworkaround(x);
    }
}
```

<- depende do fixup

<- troca o max com algum dos filhos da fila de prioridade e faz o fixdown

```
Item PQespiamax()
{
    return pq[1];
}
```

```
Void PQworkaroundMax(Item x)
{
    pq[1] = x;
    fixdown(pq, 1, N);
}
```

// 100 elementos em intervalo fechado [0, 99]; [1, 100];

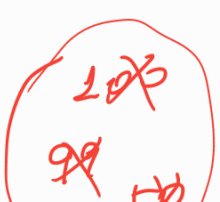
// IMPRESSÃO

// Solução do Hugo

```
sort(pq, 1, 100);
for(int i=1; i<101; i++)
    printf("%d\n", pq[i]);
```

// Solução do Joao Pedro

```
Void imprimeheap()
{
    if(PQempty()) return;
    IH
```



int x = PQ.delMax();
imprimeHeap();
printf("%d\n", x);
}

IH
x=99
IH
x=50
IH
print(50)
print(99)
print(100)

→ PQsortSimple — O(N) (potencialmente)
↳ Percorre um vetor não ordenado e insere cada elemento em uma f.p. de prioridade
↳ O(lg(N)) — O(N · lg N)
↳ Depois percorre a f.p. de prioridades, removendo o maior elemento e inserindo no vetor
O(N · lg N)
O(N lgN + N lgN) = O(2N lgN)

→ Custo adicional de espaço
N

→ IntroSort
↳ qsort
↳ std::sort
2 * lg N

Começo com o QuickSort, no pior caso ele para e termina com o HeapSort.