Estrutura de Dados II (ED2)

Aula 16 – Filas com Prioridade

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (UFES)

(Material baseado nos slides do Professor Eduardo Zambon)

Introdução

- Filas com prioridade (*priority queues*) são uma generalização de várias coleções de dados.
- **Aula de hoje**: apresentação das diferentes implementações de filas com prioridades.
- **Objetivos**: compreender as aplicações de filas com prioridade.

Referências

Chapter 9 – Priority Queues and Heapsort
R. Sedgewick

Coleções

Coleção: tipo de dado que armazena um grupo de itens.

data type	core operations	data structure
stack	PUSH, POP	*linked list, resizing array*
queue	ENQUEUE, DEQUEUE	*linked list, resizing array*
priority queue	INSERT, DELETE-MAX	*binary heap*
symbol table	PUT, GET, DELETE	*binary search tree, hash table*
set	ADD, CONTAINS, DELETE	*binary search tree, hash table*

Fila com prioridade

Coleções: inserção e remoção de itens. Qual item remover?

- Pilha (*stack*): remove o item mais recente.
- Fila (*queue*): remove o item mais antigo.
- Fila aleatória (*randomized queue*): remove um item aleatório.
- Fila com prioridade (*priority queue*): remove o maior (ou menor) item.

operation	*argument*	*return value*
insert	P	
insert	Q	
insert	E	
remove max		Q
insert	X	
insert	A	
insert	M	
remove max		X
insert	P	
insert	L	
insert	E	
remove max		P

Fila com prioridade

Critério de comparação (prioridade) dos itens depende da aplicação da fila.

Uma fila com prioridade generaliza todas as estruturas anteriores:

- Pilha: prioridade de um item recém inserido é a altura da pilha +1.
- Fila: prioridade é um ticket de espera que começa em 0 e sempre é incrementado de 1 quando um novo elemento é adicionado. Remoção é pelo menor ticket.
- Fila aleatória: prioridade de um item recém inserido é um número aleatório.

Fila com prioridade: API

Exigência: itens são genéricos, mas precisam ser comparáveis, i.e., implementar uma função `less()`.

```
#include <stdbool.h>
#include "item.h"

// Argument is maximum expected number of items.
void PQ_init(int);     // Creates an empty priority queue.

void PQ_insert(Item); // Inserts an item in the priority queue.

// Removes and returns the largest item.
Item PQ_delmax();    // Dual op delmin also possible.

Item PQ_max();       // Returns the largest item. Dual: min.

bool PQ_empty();     // Tests if the queue is empty.
int  PQ_size();      // Number of entries in the priority queue.

void PQ_finish();    // Cleans up the queue.
```

Permite chaves duplicadas: remove qualquer uma.

Fila com prioridade: aplicações

- Simulação baseada em eventos: clientes em uma fila, simulação de colisão de partículas.
- Otimização discreta: *bin packing*, escalonamento.
- Inteligência artificial: busca A*.
- Redes de computadores: *web cache*.
- Sistemas operacionais: balanceamento de carga, tratamento de interrupções.
- Compressão de dados: códigos de Huffman.
- Busca em grafos: algoritmo de Dijkstra, algoritmo de Prim.
- Filtro de *spam*: filtro Bayseano de *spam*.
- Estatística: mediana online de *stream* de dados.
- ...

Fila com prioridade: exemplo de programa cliente

Desafio: Encontrar os *M* maiores itens em uma sequência de *N* itens.

- Detecção de fraudes: isolar grandes transações financeiras.
- Monitoramento (NSA): marcar documentos suspeitos.

Restrição: Não há memória suficiente para armazenar os *N* itens.

```
// Items are money transactions, comparable by amount ($).
while (has_next(Item_Stream)) {
    Item item = read_next(Item_Stream);
    PQ_insert(item);     // Use a min-oriented PQ.
    if (PQ_size() > M) {
        PQ_delmin();     // PQ now contains largest M items.
    }
}
```

Fila com prioridade: exemplo de programa cliente

Desafio: Encontrar os M maiores itens em uma sequência de N itens.

Ordem de crescimento:

implementation	time	space
sort	$n \log n$	n
elementary PQ	$m\,n$	m
binary heap	$n \log m$	m
best in theory	n	m

Fila com prioridade: implementações elementares

Implementações elementares de uma fila com prioridade:

- Itens desordenados: insere no final, busca para remover.
- Itens ordenados: busca para inserir, remove o último.
- Implementação em ambos os casos: lista encadeada ou *array*.

operation	*argument*	*return value*	*size*	*contents (unordered)*	*contents (ordered)*
insert	P		1	P	P
insert	Q		2	P Q	P Q
insert	E		3	P Q E	E P Q
remove max		Q	2	P E	E P
insert	X		3	P E X	E P X
insert	A		4	P E X A	A E P X
insert	M		5	P E X A M	A E M P X
remove max		X	4	P E M A	A E M P
insert	P		5	P E M A P	A E M P P
insert	L		6	P E M A P L	A E L M P P
insert	E		7	P E M A P L E	A E E L M P P
remove max		P	6	E M A P L E	A E E L M P

Fila com prioridade: custo das implementações

Desafio: Implementar todas as operações de forma eficiente.

Ordem de crescimento:

implementation	insert	del max	max
unordered array	1	n	n
ordered array	n	1	1
goal	$\log n$	$\log n$	$\log n$

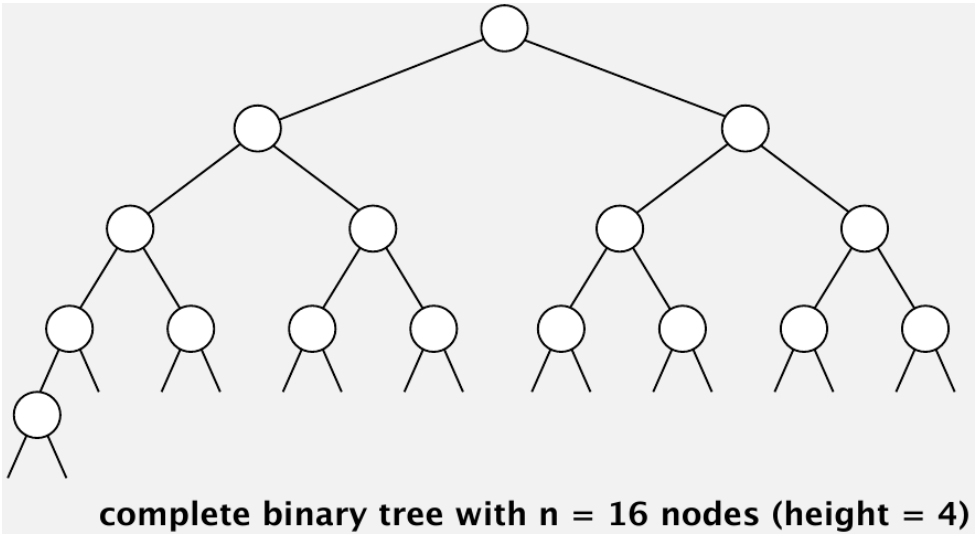
Solução: *array* parcialmente ordenado.

Haep binário

Árvore binária completa

Árvore binária: formada por nós com *links* para as sub-árvores da esquerda e direita.

Árvore completa: árvore perfeitamente balanceada, exceto no último nível.



Propriedade: Altura de uma árvore binária completa com N nós é $\lfloor \lg N \rfloor$.

Justificativa: Altura só aumenta quanto N é uma potência de 2.

Árvore binária completa na natureza

Heap binário: representação

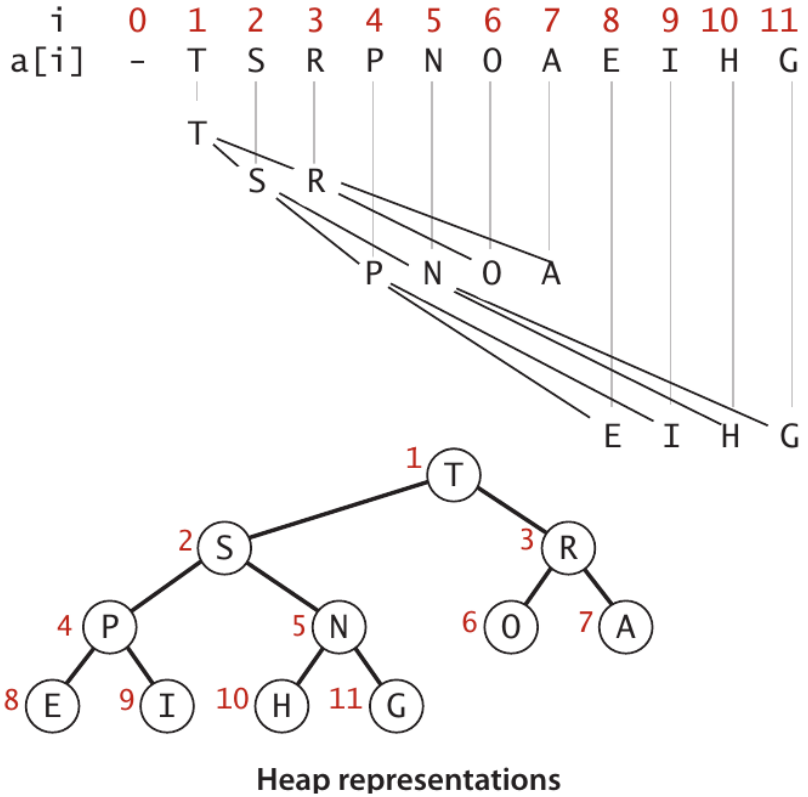
Heap binário: representação em *array* de uma árvore binária completa que é *heap-ordered*.

Árvore binária *heap-ordered*:

- Chaves nos nós.
- Chave do pai nunca é menor que as chaves dos filhos.

Representação por *array*:

- Índices começam de 1.
- Tome os nós em *level order*.
- Nenhum ponteiro é necessário!



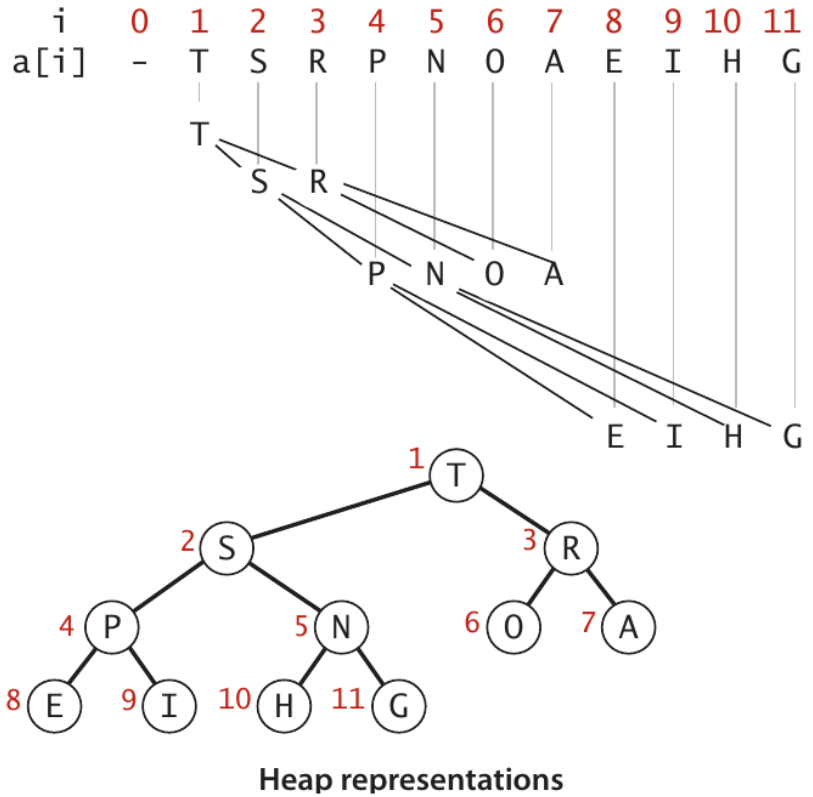
Heap representations

Heap binário: propriedades

Maior chave: está em `a[1]`, raiz da árvore.

Caminhando na árvore: basta usar os índices.

- Pai de um nó no índice k está em $k/2$.
- Filhos de um nó no índice k estão em $2k$ e $2k + 1$.



Heap representations

Heap binário: demonstração

- Operações sobre o *heap* devem manter a propriedade de *heap-ordered*.
- Inserção e remoção violam temporariamente a ordenação.
- Inserção: adiciona o nó no final, propaga para cima (*swim up*, *fix up*, *bottom-up heapify*).
- Remoção do máximo: troca a raiz com o último nó, propaga para baixo (*sink down*, *fix down*, *top-down heapify*).

Ver arquivo `24DemoBinaryHeap.mov`.

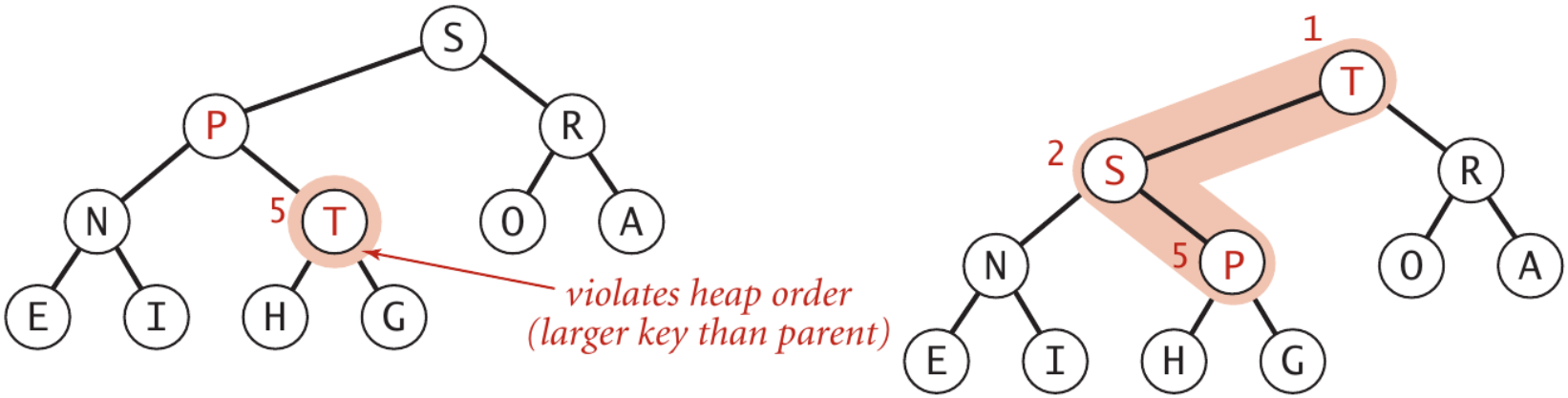
Heap binário: promoção

Situação: uma chave está maior que o seu pai.

Para reparar o invariante da estrutura (*heap-order*):

- Troque as chaves do filho e do pai.
- Repita até restaurar a ordenação.

```
void fix_up(Item *a, int k) { // swim up
    while (k > 1 && less(a[k/2], a[k])) {
        exch(a[k], a[k/2]);
        k = k/2;
    }
}
```

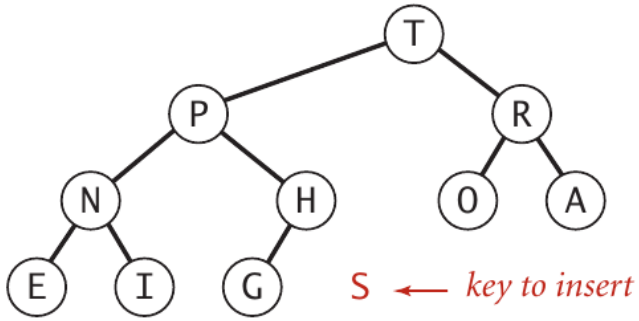


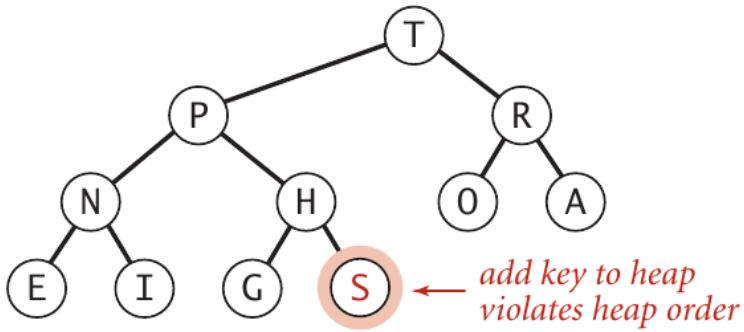
Heap binário: inserção

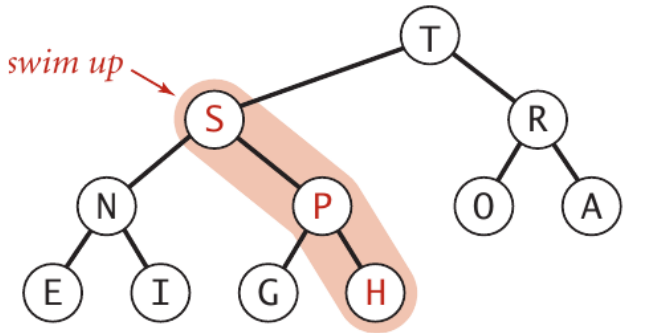
Inserção: Adiciona a chave no fim, depois `fix_up` (*swim*).

Custo: No máximo $1 + \lg N$ comparações.

```
void PQ_insert(Item v) {
     N++;
     pq[N] = v;
     fix_up(pq, N);
}
```







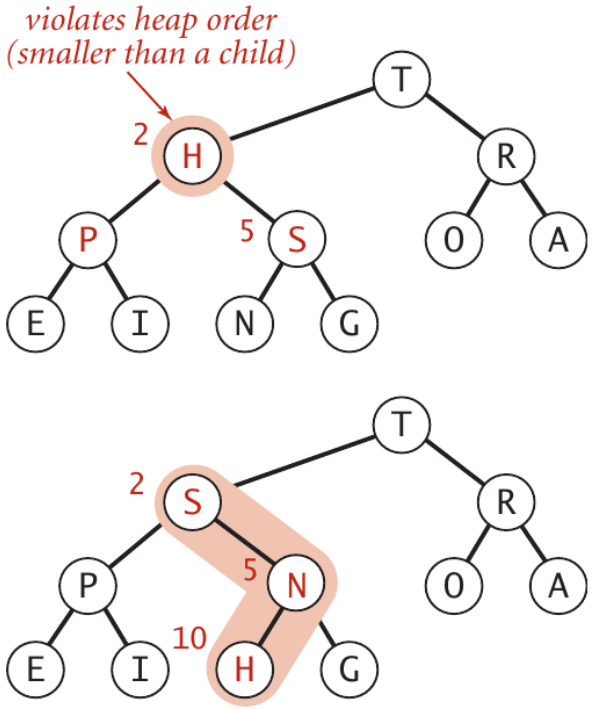
Heap binário: rebaixamento

Situação: uma chave está menor que um dos filhos.

Para reparar o invariante da estrutura (*heap-order*):

- Troque as chaves do pai com o filho maior.
- (Por que não o filho menor?)
- Repita até restaurar a ordenação.

```
void fix_down(Item *a, int sz, int k){
  while (2*k <= sz) {
    int j = 2*k;
    if (j < sz && less(a[j], a[j+1])){
      j++;
    }
    if (!less(a[k], a[j])) {
      break;
    }
    exch(a[k], a[j]);
    k = j;
  }
}
```

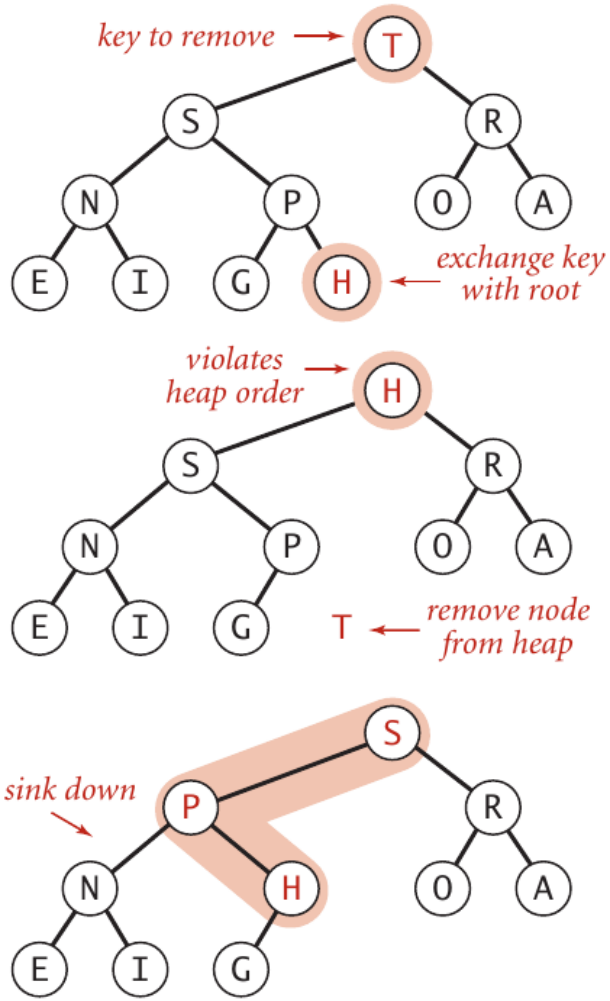


Heap binário: remover o máximo

Remover máximo: Troca a raiz com a chave no fim, depois `fix_down` (*sink*).

Custo: No máximo $2 \lg N$ comparações.

```
Item PQ_delmax() {
     Item max = pq[1];
     exch(pq[1], pq[N]);
     N--;
     fix_down(pq, N, 1);
     return max;
}
```



Fila com prioridade: custo das implementações

Desafio: Implementar todas as operações de forma eficiente.

Ordem de crescimento:

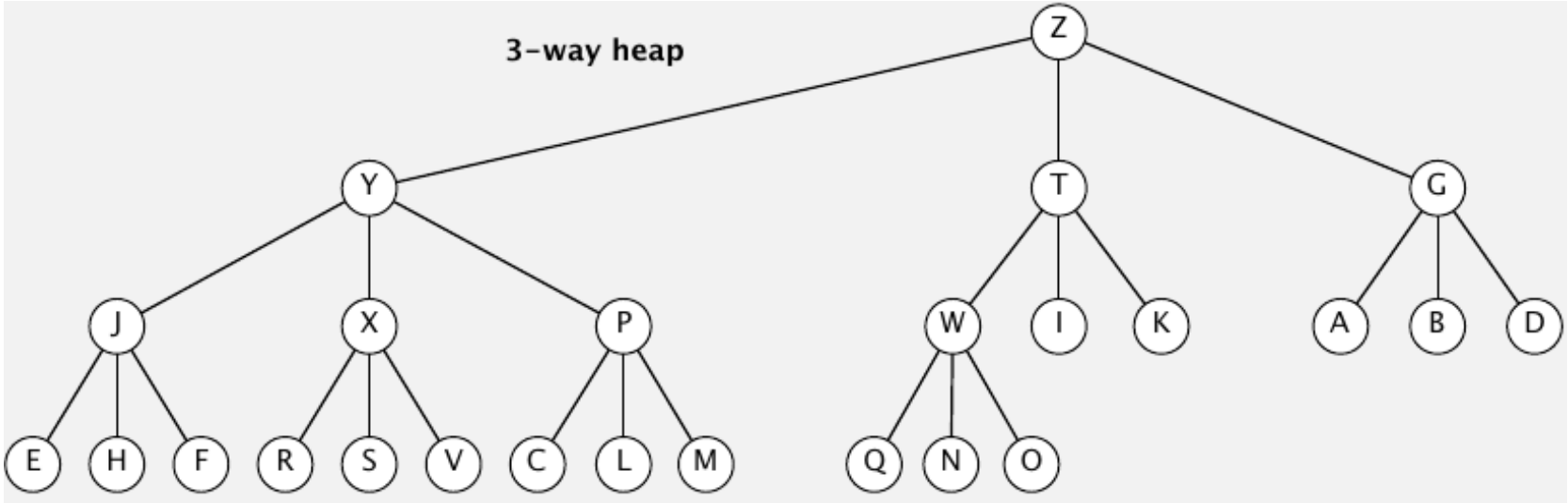
implementation	insert	del max	max
unordered array	1	n	n
ordered array	n	1	1
binary heap	$\log n$	$\log n$	1

Multiway heaps

Multiway heaps:

- Árvore completa *d-way*.
- Mesmo invariante: Chave do pai não é menor que a chave de nenhum dos *d* filhos.

Altura de uma árvore completa *d-way* com N nós é $\sim \log_d N$.



Fila com prioridade: custo das implementações

Ordem de crescimento:

implementation	insert	del max	max
unordered array	1	n	n
ordered array	n	1	1
binary heap	$\log n$	$\log n$	1
d-ary heap	$\log_d n$	$d \log_d n$	1
Fibonacci	1	$\log n$ †	1
Brodal queue	1	$\log n$	1
impossible	1	1	1

† amortized

Heap binário: considerações

Underflow e *overflow*:

- *Underflow*: emitir um erro ao se tentar remover de uma fila vazia.
- *Overflow*: aumentar o tamanho do *array* e copiar o conteúdo.
- Mudança acima leva a tempo amortizado $\log N$.

Fila com prioridade para mínimos:

- Troque `less()` por `greater()` nos códigos.
- Implementar a função `greater()`.

Outras operações:

- Remover um item arbitrário.
- Modificar a prioridade de um item.
- Podem ser implementadas de forma eficiente com `fix_up` e `fix_down`.