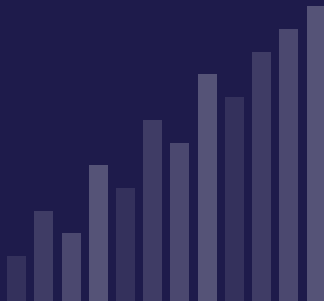

AULA 17 · BLOCO 2: DICIONÁRIOS E ESTRUTURAS LINEARES

Filas de prioridade

Seção 6.5

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Definir as operações de uma fila de prioridade
- ▶ Implementar inserção, extração do máximo e aumento de chave num heap
- ▶ Comparar heap, vetor ordenado e vetor desordenado como filas de prioridade
- ▶ Medir $\Theta(n \lg n)$ contra $\Theta(n^2)$ em n inserções e n extrações



Teoria

O tipo abstrato, as quatro operações e seus custos no heap



Prática em C

Fila de prioridade em C com comandos da entrada



Experimento

Heap contra vetores: $n \lg n$ contra n^2



Conexões

Sistemas operacionais, redes, simulação e algoritmos de grafos

O que é uma fila de prioridade

Mantém um conjunto S de elementos, cada um com uma **chave**. Fila de máximo:

- ▶ $\text{INSERT}(S, x)$: insere x .
- ▶ $\text{MAXIMUM}(S)$: devolve o elemento de maior chave.
- ▶ $\text{EXTRACT-MAX}(S)$: remove e devolve o de maior chave.
- ▶ $\text{INCREASE-KEY}(S, x, k)$: aumenta a chave de x para k .

A fila de mínimo é simétrica (EXTRACT-MIN , DECREASE-KEY).

	Insert	Extract-Max	Increase-Key
vetor desordenado	$O(1)$	$\Theta(n)$	$O(1)$
vetor ordenado	$\Theta(n)$	$O(1)$	$\Theta(n)$
heap	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$

Extrair o máximo

O último elemento ocupa o lugar da raiz e **desce**. Custo: $O(\lg n)$, a altura do heap.

Heap-EXTRACT-MAX(*A*)

```
1 se A.tamanho < 1 então  
2   erro "heap vazio"  
3 max ← A[1]  
4 A[1] ← A[A.tamanho]  
5 A.tamanho ← A.tamanho − 1  
6 MAX-HEAPIFY(A, 1)  
7 retorne max
```

Aumentar uma chave e inserir

A chave aumentada **sobe** trocando com o pai.

MAX-Heap-INSERT(A , $chave$):
aumenta o tamanho, coloca $-\infty$ na nova folha e chama Heap-INCREASE-KEY. Ambas $O(\lg n)$.

Heap-INCREASE-KEY(A , i , $chave$)

```
1 se  $chave < A[i]$  então
2   erro "nova chave é menor"
3  $A[i] \leftarrow chave$ 
4 enquanto  $i > 1$  e  $A[pai(i)] < A[i]$  faça
5   troque  $A[i]$  com  $A[pai(i)]$ 
6    $i \leftarrow pai(i)$ 
```

Aumento de chave passo a passo

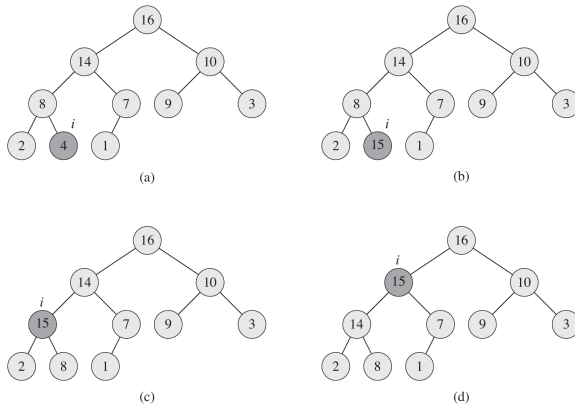


Figura 6.5 de Cormen et al. (2009) — Heap-INCREASE-KEY muda o 4 para 15, que sobe até abaixo da raiz.



Prática em C

- ▶ Heap com vetor que dobra de tamanho (`realloc`)
- ▶ sobe e desce: as duas metades do heap
- ▶ Comandos: `i x, m, x, a k v`

Fila de prioridade em C (1/4)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Heap de máximo com índices 0..tam-1. Comandos lidos da entrada:
5    i x (insere x), m (mostra o máximo), x (extrai o máximo),
6    a k v (aumenta a chave da posição k para v), f (fim). */
7 typedef struct { int *A; int tam, cap; } Fila;
8
9 void troca(int A[], int i, int j) { int t = A[i]; A[i] = A[j]; A[j] = t; }
10
11 void desce(Fila *f, int i) {                                     // MAX-HEAPIFY
12     for (;;) {
13         int e = 2 * i + 1, d = e + 1, maior = i;
14         if (e < f->tam && f->A[e] > f->A[maior]) maior = e;
15         if (d < f->tam && f->A[d] > f->A[maior]) maior = d;
16         if (maior == i) return;
```

Fila de prioridade em C (2/4)

```
17     troca(f->A, i, maior);
18     i = maior;
19 }
20 }
21
22 void sobe(Fila *f, int i) {                // laço de HEAP-INCREASE-KEY
23     while (i > 0 && f->A[(i - 1) / 2] < f->A[i]) {
24         troca(f->A, i, (i - 1) / 2);
25         i = (i - 1) / 2;
26     }
27 }
28
29 void insere(Fila *f, int x) {              // MAX-HEAP-INSERT
30     if (f->tam == f->cap) {                // vetor cheio: dobra a capacidade
31         f->cap = f->cap ? 2 * f->cap : 4;
32         f->A = realloc(f->A, f->cap * sizeof(int));
```

Fila de prioridade em C (3/4)

```
33     }
34     f->A[f->tam++] = x;
35     sobe(f, f->tam - 1);
36 }
37
38 int extrai_max(Fila *f) {                                // HEAP-EXTRACT-MAX
39     int max = f->A[0];
40     f->A[0] = f->A[--f->tam];
41     desce(f, 0);
42     return max;
43 }
44
45 void aumenta(Fila *f, int k, int v) {                    // HEAP-INCREASE-KEY
46     if (v < f->A[k]) { printf("nova chave é menor\n"); return; }
47     f->A[k] = v;
48     sobe(f, k);
```

Fila de prioridade em C (4/4)

```
49 }
50
51 int main(void) {
52     Fila f = {NULL, 0, 0};
53     char cmd[4];
54     int x, k;
55     while (scanf("%3s", cmd) == 1 && cmd[0] != 'f') {
56         if (cmd[0] == 'i' && scanf("%d", &x) == 1) insere(&f, x);
57         else if (cmd[0] == 'm' && f.tam) printf("maior %d\n", f.A[0]);
58         else if (cmd[0] == 'x' && f.tam) printf("extraído %d\n", extrai_max(&f));
59         else if (cmd[0] == 'a' && scanf("%d %d", &k, &x) == 2 && k < f.tam)
60             aumenta(&f, k, x);
61     }
62     free(f.A);
63     return 0;
64 }
```



Experimento

- ▶ Mesma sequência de operações nas três filas
- ▶ Hipótese: $\Theta(n \lg n)$, $\approx n^2/4$ e $n(n-1)/2$
- ▶ n de 1.000 a 16.000

Três implementações, mesmo trabalho

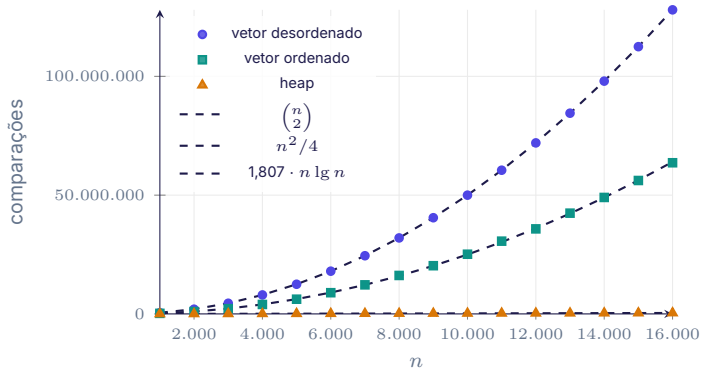
Hipótese. Cada operação no heap custa $O(\lg n)$: total $\Theta(n \lg n)$. No vetor ordenado, cada inserção desloca em média metade dos elementos: $\approx n^2/4$. No desordenado, extrair o máximo examina todos: exatamente $n(n-1)/2$ comparações.

Método

1. Os mesmos n números aleatórios nas três filas
2. Contador de comparações entre chaves
3. n inserções, depois n extrações
4. n de 1.000 a 16.000

EXPERIMENTO

Comparações das três filas



n	medido	razão
1.000	499.500	1,000
4.000	7.998.000	1,000
7.000	24.496.500	1,000
10.000	49.995.000	1,000
13.000	84.493.500	1,000
16.000	127.992.000	1,000

razão = medido $\div$ $\binom{n}{2}$

✓ desvio máx. 0,0%

O heap quase some no gráfico: para $n = 16.000$, faz 404 mil comparações contra 64 e 128 milhões.

EXPERIMENTO

O que o experimento mostrou

O vetor desordenado faz exatamente $n(n - 1)/2$ comparações e o ordenado, $n^2/4$; o heap fica em $\approx 1,8 n \lg n$. Para $n = 16.000$, o heap é **150 vezes** mais barato que o vetor ordenado. A estrutura certa muda a ordem de crescimento.

>_ Reproduza: `gcc -O2 aula17_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Sistemas operacionais
- ▶ Redes de computadores
- ▶ Simulação de eventos
- ▶ Compressão de dados

Onde isso aparece na computação

Sistemas operacionais

Escalonadores escolhem o próximo processo pela prioridade; temporizadores do kernel disparam o evento mais próximo: EXTRACT-MIN num heap.

Redes de computadores

Roteadores com QoS servem primeiro os pacotes de voz e vídeo; o roteamento OSPF roda Dijkstra com fila de prioridade em cada roteador.

Simulação de eventos

Simuladores (filas de banco, tráfego, epidemias) processam eventos em ordem de tempo: o próximo evento sai de uma fila de mínimo.

Compressão de dados

O código de Huffman (aula 34) junta repetidamente os dois símbolos menos frequentes, retirados de uma fila de prioridade.

Para casa

Exercícios

1. (6.5-1) Ilustre Heap-EXTRACT-MAX em $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.
2. (6.5-2) Ilustre MAX-Heap-INSERT($A, 10$) no mesmo heap.
3. (6.5-8) Implemente Heap-DELETE(A, i) em $O(\lg n)$.
4. (6.5-9) Intercale k listas ordenadas em $O(n \lg k)$ com uma fila de mínimo.
5. Altere a fila para guardar pares (prioridade, nome de tarefa) e simule um escalonador.

Leitura

Cormen et al., seção 6.5.

Código

josedihego.net/estruturas