Filas-com-prioridades

Imagine um conjunto S de números. Os elementos de S são às vezes chamados *chaves* ou *prioridades* (especialmente se há outros dados associados a cada elemento de S). Uma *fila-com-prioridades* (ou fila priorizada, ou *priority queue*) é um tipo abstrato de dados que permite executar as seguintes operações sobre S:

- encontrar um elemento máximo de S,
- extrair um elemento máximo de S,
- inserir um novo número em S,
- aumentar o valor de um elemento de S,
- diminuir o valor de um elemento de S.

Há uma variante dessa definição em que "máximo" é substituído por "mínimo". Para distinguir uma variante da outra, diremos que a primeira é uma fila-com-prioridades *decrescente* (ou "de máximo") e a segunda é uma fila-com-prioridades *crescente* (ou "de mínimo").

Filas-com-prioridades (crescentes e descrescentes) têm um papel fundamental na implementação eficiente de diversos algoritmos célebres, como o algoritmo Heapsort, o algoritmo de Dijkstra, o algoritmo de Prim, e o algoritmo de Huffman.

Esta página é inspirada na seção 6.5 de CLRS.

Implementações de fila-com-prioridades

Não é difícil imaginar maneiras de implementar uma fila-com-prioridades. Nas implementações mais óbvias, algumas das operações ficam rápidas mas as outras ficam lentas. O desafio é inventar uma implementação em que *todas* as operações sejam rápidas.

Exercícios 1

1. Discuta a implementação de uma fila-com-prioridades decrescente num vetor que armazena as chaves em ordem arbitrária.
2. Discuta a implementação de uma fila-com-prioridades decrescente num vetor que armazena as chaves em ordem crescente.

Implementação em heap

No que segue, trataremos de filas-com-prioridades decrescentes, mas é fácil adaptar toda a discussão às filas crescentes.

Eis uma maneira muito eficiente de implementar uma fila-com-prioridades decrescente: mantenha o conjunto S de chaves em um max-heap. Vamos mostrar a seguir como cada uma das operações da fila-com-prioridades decrescente é executada. Suporemos que o max-heap fica armazenado em um vetor $A[1..n]$.

Máximo

É fácil encontrar (e devolver) o valor de um elemento máximo do max-heap $A[1..n]$ não vazio (ou seja, com $n \geq 1$):

```
ENCONTRA-MÁXIMO (A, n)
1  devolva A[1]
```

Remoção de máximo

O seguinte algoritmo remove um elemento máximo do max-heap não vazio $A[1..n]$ e devolve o valor desse elemento:

```
EXTRAIA-MAX (A, n)
1  max := A[1]
2  A[1] := A[n]
3  n := n−1
4  CORRIGE-DESCENDO (A, n, 1)
5  devolva max
```

O algoritmo CORRIGE-DESCENDO foi discutido na página *A estrutura heap*.

Inserção

O algoritmo abaixo insere um número c no max-heap $A[1..n]$ (ou seja, acrescenta um novo elemento, com valor c, ao vetor) e rearranja o vetor para que ele volte a ser um max-heap:

```
INSERE-NA-FILA (A, n, c)
1  A[n+1] := c
2  CORRIGE-SUBINDO (A, n+1)
```

O algoritmo CORRIGE-SUBINDO foi discutido na página *A estrutura heap*.

Aumento do valor de uma chave

O algoritmo seguinte recebe um max-heap não vazio $A[1..n]$, um índice i no intervalo $1..n$ e um número $c \geq A[i]$. O algoritmo altera para c o valor de $A[i]$ e rearranja o vetor para que ele volte a ser um max-heap:

```
AUMENTA-CHAVE (A, i, c)
1  A[i] := c
```

```
2   CORRIGE-SUBINDO (A, i)
```

Redução do valor de uma chave

O algoritmo abaixo recebe um max-heap não vazio $A[1..n]$, um índice i no intervalo $1..n$ e um número $c \leq A[i]$. O algoritmo altera para c o valor de $A[i]$ e rearranja o vetor para que ele volte a ser um max-heap:

```
DIMINUA-CHAVE (A, n, i, c)
1   A[i] := c
2   CORRIGE-DESCENDO (A, n, i)
```

Construção de um heap

Para transformar um vetor arbitrário $A[1..n]$ em max-heap (de maneira muito eficiente), use o algoritmo CONSTRÓI-MAX-HEAP, já discutido na página *A estrutura heap*.

Desempenho

Os algoritmos acima são todos muito rápidos: no pior caso, cada um consome tempo proporcional à altura do heap. Como a altura é $\lfloor \lg n \rfloor$ o consumo de tempo está em $O(\lg n)$ no pior caso.

Exercícios 2

1. Prove que os algoritmos ENCONTRA-MÁXIMO, EXTRAIA-MAX, INSERE-NA-FILA, AUMENTA-CHAVE e DIMINUA-CHAVE estão corretos.

2. ★ [CLRS] Escreva uma implementação eficiente da operação REMOVE com parâmetros (A, n, i). Ela deve remover o nó i do max-heap $A[1..n]$ e armazenar os elementos restantes, em forma de max-heap, no vetor $A[1..n-1]$.

3. ★ Reescreva o código do algoritmo HEAPSORT (veja a página *Ordenação: Heapsort*) usando as operações de manipulação de heap definidas acima.

4. ★ Escreva implementações das operações de uma fila-com-prioridades "de mínimo"

5. Escreva um algoritmo que receba um max-heap $A[1..n]$ e um número x e devolva um índice i tal que $A[i] = x$ ou devolva 0 se tal i não existe. Procure escrever um algoritmo eficiente. Qual o consumo de tempo de seu algoritmo?

6. [CLRS] Suponha dado um conjunto $A_1, A_2, \ldots, A_n$ de vetores numéricos crescentes. (Você pode trocar "vetor" por "lista encadeada" ou por "arquivo" se desejar.) Digamos que cada A_i tem t_i elementos. Queremos imprimir todos os $t = t_1 + t_2 + \ldots + t_n$ números, em ordem decrescente. Dê um algoritmo que consuma tempo $O(t \lg n)$ para fazer o serviço. (Faz sentido dizer que o consumo de seu algoritmo está em $O(n \lg n)$? Faz sentido dizer que está em $O(t \lg t)$?)

Objetos e chaves

Em geral, os elementos do vetor A são objetos arbitrários. Um dos atributos de cada objeto é um número que chamaremos *chave* (= *key*). O valor da chave determina a prioridade do objeto na fila. Na descrição acima, para simplificar, cada objeto consiste na chave e nada mais.

Veja o verbete Priority_queue na Wikipedia.

Veja meus slides, página 55.
Veja também minha página no site *Estruturas de Dados*.

---

www.ime.usp.br/~pf/analise_de_algoritmos/
Atualizado em 2020-05-08
Paulo Feofiloff
Departamento de Ciência da Computação
Instituto de Matemática e Estatística da USP