Fila de prioridades

Estruturas de Dados 1 – 2023.2
Prof. Pedro Nuno Moura

Material adaptado da Profa. Adriana Alvim

Fila de prioridades

- Em muitas situações uma fila simples é inadequada pois o critério "primeiro a entrar/ primeiro a sair" deve ser substituído por um critério de prioridade
- Uma questão importante é encontrar uma implementação eficiente que permita colocar na fila e retirar da fila de forma relativamente rápida
- Os elementos podem chegar de forma aleatória,
  - não existe garantia de que os elementos do início da fila são aqueles que devem ser retirados primeiro

Fila de prioridades

- Uma fila de prioridades é uma estrutura de dados usada para manutenção de um conjunto de elementos,
  - cada qual com um valor associado chamado chave
- Existem dois tipos de filas de prioridades
  - as filas de prioridades máxima e
  - as filas de prioridades mínima

Fila de prioridades

- Algumas operações possíveis
  - insere um novo elemento no conjunto
  - remove o elemento com a maior (menor) chave
  - remove e retorna o elemento com a maior (menor) chave
  - informa o elemento do conjunto com a maior (menor) chave
  - aumenta (diminui) o valor da chave do elemento x para o novo valor k, que se presume ser pelo menos tão grande quanto o valor da chave atual de x
  - juntar duas listas de prioridades
  - constrói uma fila de prioridades dos elementos dados com seus valores chave

Fila de prioridades

- Exemplos de aplicações de filas de prioridade
- Máxima
  - programar tarefas em um computador compartilhado
  - a fila de prioridades máxima mantém o controle das tarefas a serem executadas e de suas prioridades relativas
  - quando uma tarefa termina ou é interrompida, a tarefa de prioridade mais alta é selecionada dentre os trabalhos pendentes, com o uso da função remove-maximo
  - um novo trabalho pode ser adicionado a lista a qualquer instante, com o uso da função insere

Fila de prioridades

- Mínima
  - pode ser usado em um simulador orientado a eventos
  - os itens na fila são eventos a serem simulados, cada qual com um tempo de ocorrência associado que serve como sua chave
  - os eventos devem ser simulados em ordem de seu momento de ocorrência, a simulação de um evento pode provocar outros eventos a serem simulados no futuro

Representação de fila de prioridades

- Lista (representada com um vetor ou lista encadeada)
  - lista linear não ordenada de valores de chave
    - inserir
      - imediato
    - remover máximo (ou mínimo)
      - é necessário percorrer a lista, tempo linear
  - lista linear ordenada de valores de chave
    - inserir
      - envolve encontrar a posição, percorre a lista, tempo linear
    - remover máximo (ou mínimo)
      - imediato, primeiro elemento

Representação de fila de prioridades

- Heap binário
  - a estrutura de dados chamada heap é a melhor forma de implementar uma lista de prioridades

- Um heap é uma sequencia de elementos com chaves

  c[1], c[2], ... c[n]

  tal que

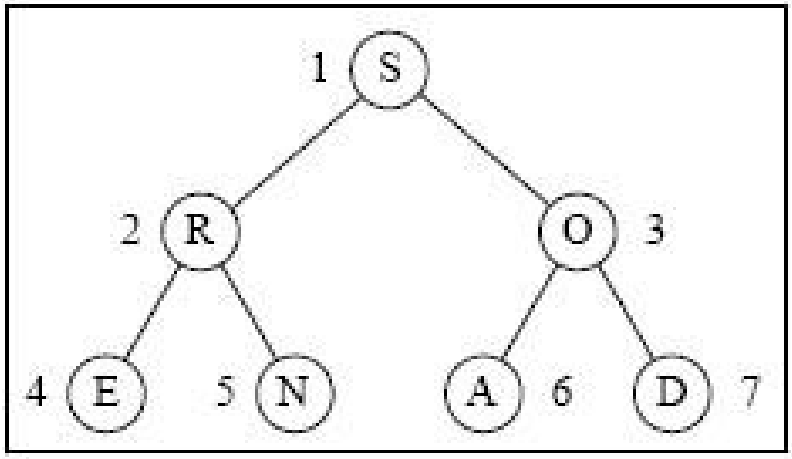
  c[i] >= c[2i]

  c[i] >= c[2i + 1]

  - para todo i = 1, …, n/2

Representação de fila de prioridades

- Heap binário
  - a definição pode ser facilmente visualizada como uma árvores binária completa, isto é
    - uma árvore onde cada nível é completado da esquerda para a direita e tem de estar cheio antes que o próximo nível seja iniciado



Representação de fila de prioridades

- Heap binário (continuação)
  - árvore binária completa
    - o primeiro nó é chamado raiz
    - os nós são numerados de 1 a n, linha por linha, começando da raiz e movendo da esquerda para a direita em cada linha
    - o nó $k/2$ é o pai do nó k, para $1 < k \leq n$
    - os nós $2k$ e $2k + 1$ são os filhos à esquerda e à direita do nó k, para $1 \leq k \leq k/2$

Representação de fila de prioridades

- Heap binário (continuação)
  - as chaves na árvore satisfazem a condição do heap
    - o valor da chave de cada nó da árvore é maior (menor) ou igual aos dos seus filhos
    - em especial, a raiz sempre contém o maior (menor) elemento

Representação de fila de prioridades

- Heap binário (continuação)
  - uma árvore binária completa pode ser representada por um vetor
  - a representação é extremamente compacta
  - permite caminhar pelos nós da árvore facilmente

1	2	3	4	5	6	7
S	*R*	*O*	*E*	*N*	*A*	*D*

  - os filhos de um nó i estão nas posições 2i e 2i + 1
  - o pai de um nó i está na posição i / 2
  - na representação do heap em um vetor, a maior (menor) chave está sempre na posição 1 do vetor

Representação de fila de prioridades

```
public class BinHeapMin{
   private int n;                  /* Numero de elementos no heap */
   private int tam;                /* Tamanho do heap */
   private int[] vetor;            /* vetor com elementos */

   /* Constrói heap vazio. */
   public BinMinHeap(int tamanho)
   {
        n = 0;
        tam = tamanho;
        vetor = new int[tamanho+1];
   }
```

Representação de fila de prioridades

```
/* Constrói heap a partir de vetor v. */
public BinMinHeap(int tamanho, int[] v)
{
      tam = tamanho;
      vetor = new int[tamanho+1];
      n = tamanho;

      for( int i = 0; i < tamanho; i++ )
            vetor[ i + 1 ] = v[ i ];

      constroiHeap();
}
```

Representação de fila de prioridades

- Heap binário (continuação)
  - um algoritmo elegante para construir o heap foi proposto por Floyd em 1964
  - o algoritmo não necessita de nenhuma memória auxiliar
  - dado um vetor v[1], v[2], . . . , v[n]
  - os itens v[n/2 + 1], v[n/2 + 2], . . . , v[n] formam um heap
  - neste intervalo não existem dois índices i e j tais que j = 2i ou j = 2i + 1

1	2	3	4	5	6	7
S	*R*	*O*	*E*	*N*	*A*	*D*

Fila de prioridades

- Heap binário (continuação)

	1	2	3	4	5	6	7
Chaves iniciais:	*O*	*R*	*D*	*E*	*N*	*A*	*S*
Esq = 3	*O*	*R*	***S***	*E*	*N*	*A*	***D***
Esq = 2	*O*	*R*	*S*	*E*	*N*	*A*	*D*
Esq = 1	***S***	*R*	***O***	*E*	*N*	*A*	*D*

  - os itens de v[4] a v[7] formam um heap
  - o heap é estendido para a esquerda (esq = 3), englobando o item v[3], pai dos itens v[6] e v[7]
  - a condição de heap é violada
    - o heap é refeito trocando os itens D e S

Fila de prioridades

- Heap binário (continuação)

	1	2	3	4	5	6	7
Chaves iniciais:	*O*	*R*	*D*	*E*	*N*	*A*	*S*
Esq = 3	*O*	*R*	**S**	*E*	*N*	*A*	**D**
Esq = 2	*O*	*R*	*S*	*E*	*N*	*A*	*D*
Esq = 1	**S**	*R*	**O**	*E*	*N*	*A*	*D*

  - o item R é incluindo no heap (esq = 2), o que não viola a condição de heap

Fila de prioridades

- Heap binário (continuação)

	1	2	3	4	5	6	7
Chaves iniciais:	*O*	*R*	*D*	*E*	*N*	*A*	*S*
Esq = 3	*O*	*R*	**S**	*E*	*N*	*A*	**D**
Esq = 2	*O*	*R*	*S*	*E*	*N*	*A*	*D*
Esq = 1	**S**	*R*	**O**	*E*	*N*	*A*	*D*

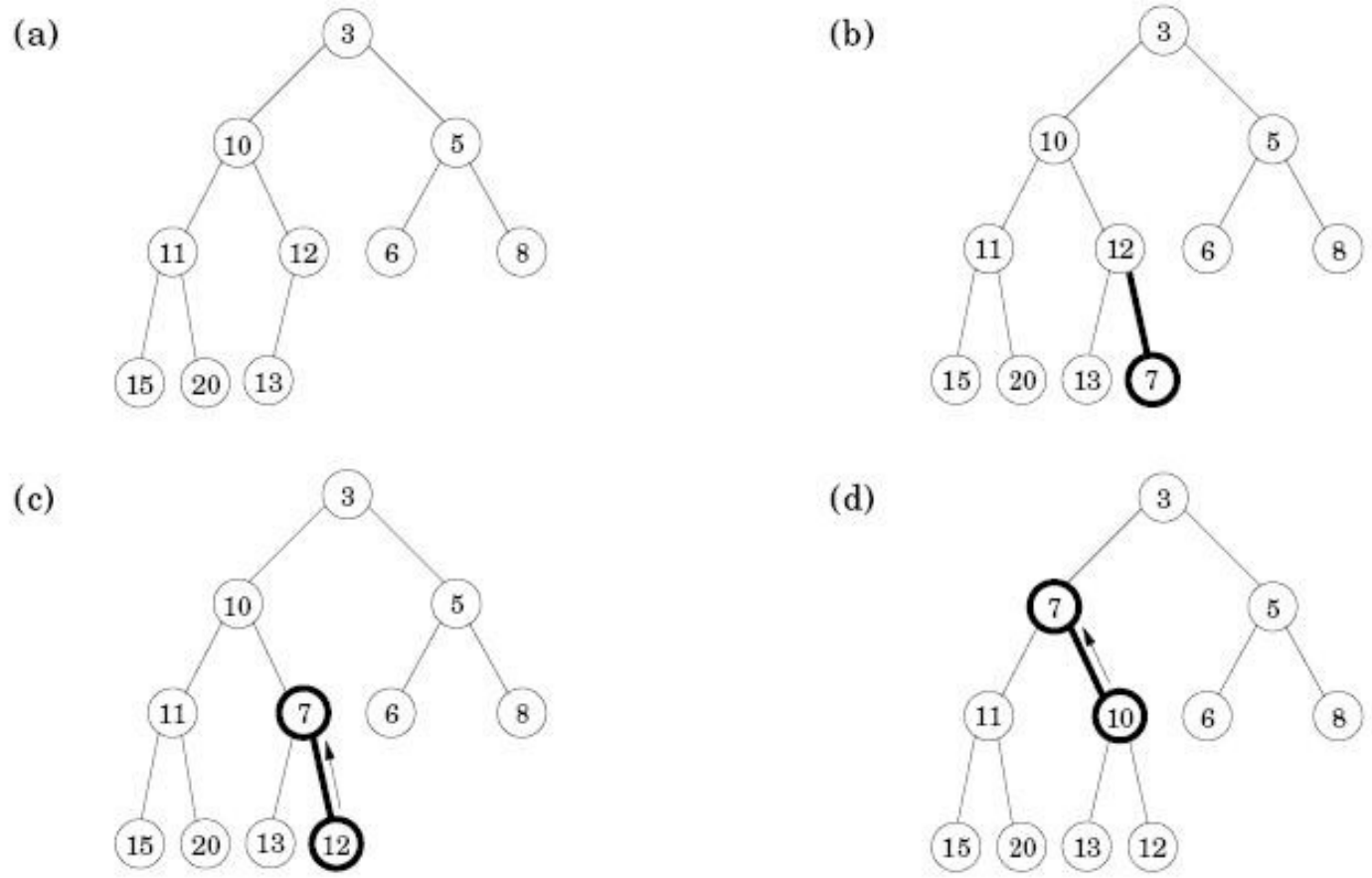
  - o item O é incluindo no heap (esq = 1)
  - a condição de heap é violada
    - o heap é refeito trocando os itens O e S,
    - encerrando o processo

Heap binário

- Inserir um elemento (heap mínimo)
  - coloque o novo elemento na parte inferior da árvore (primeira posição disponível), o que equivale a ultima posição do vetor, e depois faça o elemento subir
    - quer dizer, se ele é maior (menor) do que seu pai, troque um pelo outro e repita
  - o número de trocas é no máximo igual a altura da árvore, que é log n, quando existem n elementos na árvore

Heap binário

- Inserir um elemento (heap mínimo)
  - movimentos de subida, quando o elemento 7 é inserido



Heap binário

- Inserir um elemento (heap mínimo)

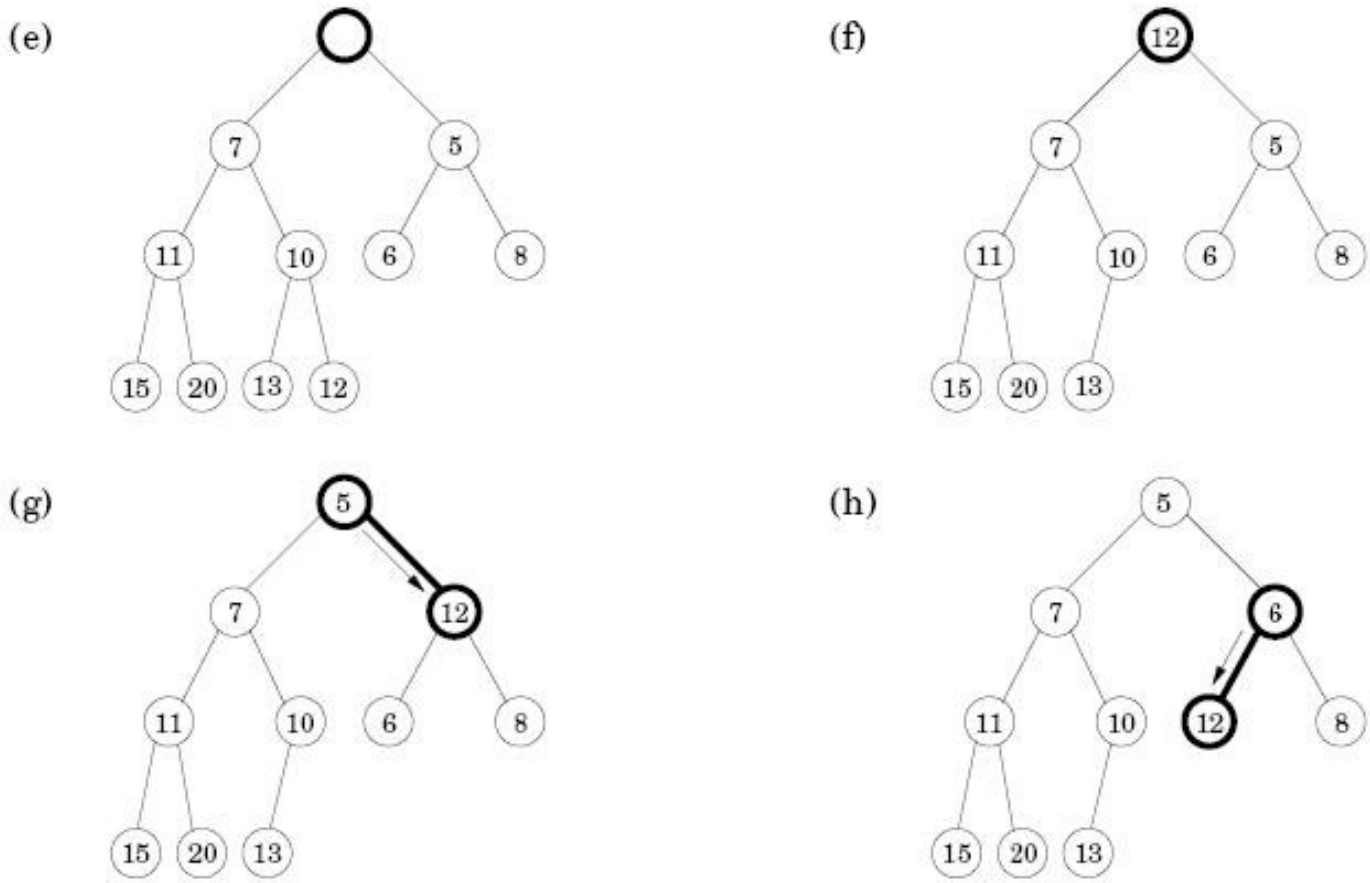
```
/* Insere item x na fila de prioridade, mantendo a propriedade do heap.
   É permitido duplicadas. */
public void insere(int x) {
  int dir;

  if ( tam == n ){
    System.out.println("Fila de prioridades cheia!”);
    return;
  }

  /* inicializa sentinela */
  vetor[ 0 ] = x;
  dir = ++n;

  /* Refaz heap (sobe elemento) */
  for( ; x < vetor[dir/2]; dir /= 2 )
    vetor[dir] = vetor[ dir/2 ];
  vetor[dir] = x;
}
```

Heap binário

- Remove mínimo
  - para retornar o valor mínimo, retorna o valor da raiz (primeiro elemento do vetor)
  - para também remover este elemento do heap, pegue o último nó da árvore, o da posição mais à direita da linha inferior, (último elemento do vetor) e coloque na raiz
  - depois faça o elemento "descer" se ele é maior do que algum filho, troque-o com o menor dos filhos
    - agora, a subarvore com raiz no menor dos filhos (raiz corrente) pode violar a propriedade do heap,
    - repita o processo a partir da raiz corrente enquanto for necessário refazer a propriedade do heap

Heap binário

- Remove mínimo
  - movimentos de descida, quando removendo elemento



Heap binário

- Remove mínimo

```
/* Remove o menor item da lista de prioridades e coloca ele
em itemMin. */
public void removeMin(int itemMin)
{
    if( vazia( ) ){
     System.out.println("Fila de prioridades vazia!");
     return;
    }
    itemMin = vetor[1];
    vetor[ 1 ] = vetor[ n-- ];
    refaz( 1 , n );
}
```

Heap binário

- Usa o método refaz (conhecido como heapify na literatura)

```
/* Metodo interno para refazer o heap.
   esq eh o indice de onde inicia o processo para refazer */
public void refaz( int esq , int dir )
{
    int filho;
    int x = vetor[ esq ];
    for( ; esq * 2 <= dir; esq = filho )
    {
        filho = esq * 2;
        if( filho != dir && vetor[ filho + 1 ] < vetor[ filho ] )
            filho++;
        if( vetor[ filho ] < x )
            vetor[ esq ] = vetor[ filho ];
        else
            break;
    }
    vetor[ esq ] = x;
}
```

Heap binário

- Remove mínimo
  - refaz é uma rotina importante para a manutenção das propriedades do heap mínimo (máximo)
  - quando refaz é chamado, supomos que as árvores binárias com raízes nos filhos de v[esq] são heaps mínimos, mas que v[esq] pode ser maior que seus filhos, violando assim a propriedade de heap mínimo (máximo)
  - a função é deixar que o valor em v[esq] flutue para baixo no heap, de tal forma que a subárvore com raiz no índice esq se torne um heap mínimo (máximo)
  - em cada passo o maior entre v[esq] e seus dois filhos é determinado

Heap binário

- Remove mínimo
  - se v[esq] é o menor deles então o procedimento acaba
  - do contrário o menor dos filhos é trocado com v[esq]
  - agora, a subarvore com raiz no menor dos filhos (raiz corrente) pode violar a propriedade do heap,
  - o processo é repetido a partir da raiz corrente enquanto for necessário refazer a propriedade do heap

Heap binário

- Constrói heap
  - é possível usar o método refaz (de baixo para cima) a fim de construir um heap a partir de um vetor não ordenado
  - os elementos v[n/2 + 1], v[n/2 + 2], . . . , v[n] do vetor são todos folhas da árvore, então cada um deles é um heap de 1 elemento
  - o procedimento percorre todos os outros nós da árvore e executa refaz sobre cada um

Heap binário

- Constrói heap (conhecido como *Build Heap* na literatura)

```
/* Estabelece a propriedade de ordem do heap
a partir de um arranjo arbitrario dos itens.
*/
public void constroiHeap()
{
    for( int i = n / 2; i > 0; i-- )
        refaz( i, n );
}
```

Algoritmo Heapsort

- Algoritmo para ordenar um vetor
  - inicia construindo o heap em um vetor de n elementos
  - uma vez que o elemento máximo (ou mínimo) do vetor se encontra na raiz (posição 1 do vetor), ele pode ser colocado na sua posição correta, trocando-se este elemento por v[n]
  - se descartarmos o nó n do heap, diminuindo o tamanho do heap, o vetor v[1,n-1] pode ser facilmente transformado em um heap
    - os filhos da raiz continuam sendo um heap, mas o novo elemento da raiz pode violar a propriedade do heap
    - desta forma, é necessário chamar a operação refaz para o vetor v[1, n-1]

Algoritmo Heapsort

1. Construir o heap
2. Troque o item na posição 1 do vetor (raiz do heap) com o item da posição n
3. Use o procedimento refaz para reconstituir o heap para os itens v[1], v[2], . . . , v[n − 1]
4. Repita os passos 2 e 3 com os n − 1 itens restantes, depois com os n − 2, até que reste apenas um item

- O algoritmo está implementado na classe *BinMaxHeap* contida no projeto Java fornecido.

Referências

- Este material foi produzido baseado no material dos livros
  - T. Cormen, C. Leiserson, C, Stein e R. Rivest, Algoritmos: Teoria e Prática, Ed. Campus, 2002.
  - S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani, Algorithms.
  - Ziviani, Projeto de Algoritmos.