MC202ABCD – Estrutura de Dados

1 Filas de prioridade com duas extremidades

Vimos que uma **fila de prioridade** (de máximo) é uma estrutura de dados S para armazenar elementos, cada um com uma prioridade, que suporta as seguintes operações:

- verificar se S é vazio,
- devolver o elemento de maior prioridade em S,
- remover o elemento de maior prioridade em S e
- inserir um elemento com prioridade k em S.

Uma **fila de prioridade com duas extremidades** (*double ended priority queue*) é uma estrutura de dados S para armazenar elementos, cada um com uma prioridade, que suporta as seguintes operações:

- verifica se S é vazio
- devolver o elemento de maior prioridade em S,
- devolver o elemento de menor prioridade em S,
- remover o elemento de maior prioridade em S,
- remover o elemento de menor prioridade em S e
- inserir um elemento com prioridade k em S.

Vimos que filas de prioridade podem ser implementadas usando *heaps* (de máximo). Heaps de mínimos podem ser implementados de modo semelhante.

Filas de prioridade com duas extremidades também podem ser implementadas usando o que chamamos de *heaps de intervalos* como veremos a seguir.

2 Heaps de intervalos

Um **heap de intervalos** é uma árvore binária (quase) completa onde os nós estão organizados da seguinte forma:

1. cada elemento possui uma prioridade que suporemos ser um inteiro positivo; na descrição que segue não distinguiremos um elemento de sua prioridade (elementos distintos podem ter a mesma prioridade),

2. cada nó contém dois elementos exceto possivelmente o último nó (no percurso em largura) – o último nó contém apenas um elemento se e somente se o número de elementos for ímpar;

3. se um nó A possui dois elementos com prioridades a e b, com $a \leq b$, dizemos que A representa o **intervalo** $[\mathbf{a}, \mathbf{b}]$. Dizemos que a é o **extremo esquerdo** e que b é o **extremo direito** do intervalo $[a, b]$.

4. se $[a, b]$ é o intervalo representado por um nó A e $[c, d]$ é o intervalo representado por um filho de A, então $[c, d]$ está contido em $[a, b]$ (ou seja, $a \leq c \leq d \leq b$). Se o último nó contém apenas um elemento com prioridade c e $[a, b]$ é o intervalo representado por seu pai (se houver), então $a \leq c \leq b$.

Na Figura 1 ilustramos um heap de intervalos com 26 elementos (apenas as prioridades estão indicadas).

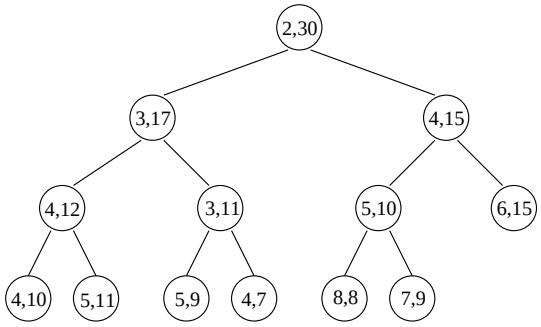

Figura 1: Heap de intervalos.

Observe que:

1. Os extremos esquerdos dos intervalos definem um heap de mínimo e os extremos direitos dos intervalos definem um heap de máximo. Quando o número de elementos é ímpar, o último elemento pode ser considerado como um membro do heap de mínimo ou do heap de máximo. A Figura 2 ilustra os heaps de mínimo e máximo embutidos no heap de intervalos da Figura 1.

2. Quando a **raiz** do heap contém dois elementos, o **extremo esquerdo** de seu intervalo é a **prioridade mínima** e o **extremo direito** é a **prioridade máxima** do heap de intervalos. Quando a raiz (e portanto, o heap) contém apenas um elemento, ele é tanto o mínimo quanto o máximo.

3. O heap de intervalos pode ser representado através de um vetor como no caso de heaps usuais. A diferença é que agora cada nó deve ser capaz de armazenar dois elementos.

4. A altura de um heap de intervalos com n elementos é $O(\log n)$.

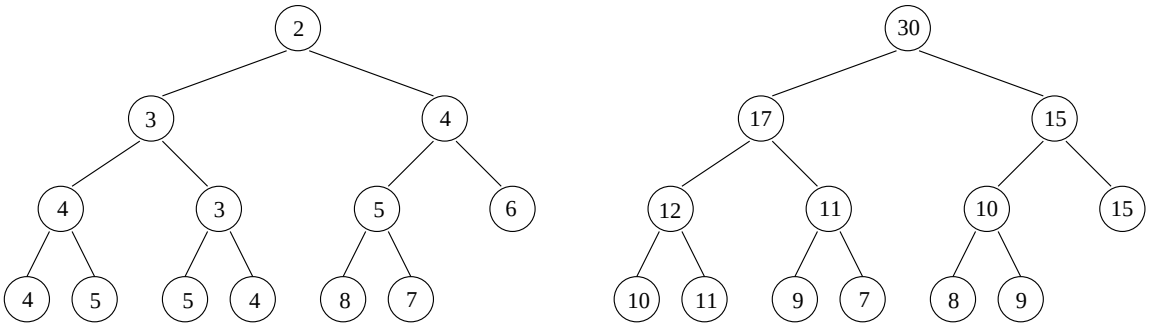

Figura 2: Heaps de mínimo e de máximo embutidos no heap da Figura 1.

2.1 Inserção de um elemento no heap de intervalos

Descrevemos a seguir como é o procedimento de inserção de um novo elemento no heap de intervalos. Temos dois casos dependendo do número de elementos no heap ser par ou ímpar.

O número de elementos é par

Suponha que queremos inserir um novo elemento no heap da Figura 1. Como o número de elementos é par, é necessário criar um novo nó A como indicado na Figura 3.

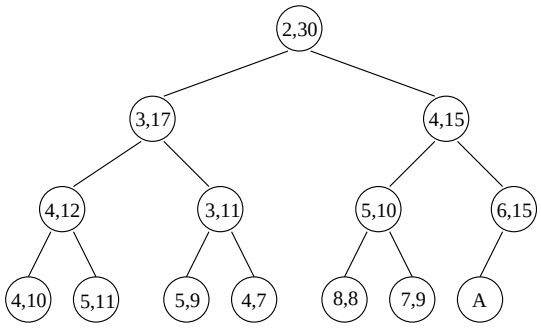

Figura 3: Heap de intervalos da Figura 1 após a inserção de um nó.

Se a prioridade do novo elemento estiver dentro do intervalo $[a, b]$ do pai, então o novo elemento pode simplesmente ser colocado em A.

Se a prioridade do novo elemento é menor que o extremo esquerdo a do intervalo do pai de A, então o novo elemento é inserido no heap de mínimo embutido no heap de intervalo (o heap de máximo permanece inalterado). Isto é feito usando os procedimentos de inserção em heaps a partir de A (Sobe_Heap para heaps de mínimo).

Se a prioridade do novo elemento é maior que o extremo direito b do intervalo do pai de A, então o novo elemento deve ser inserido no heap de máximo embutido no heap de intervalo (o

heap de mínimo parmanece inalterado). Isto é feito usando os procedimentos de inserção em heaps a partir de A (Sobe_Heap para heaps de máximo – visto em aula).

Exemplos.

1. Suponha que queremos inserir um elemento com prioridade 10 no heap da Figura 1. Então basta colocar o novo elemento no nó A. A Figura 4 mostra o heap resultante após a inserção.

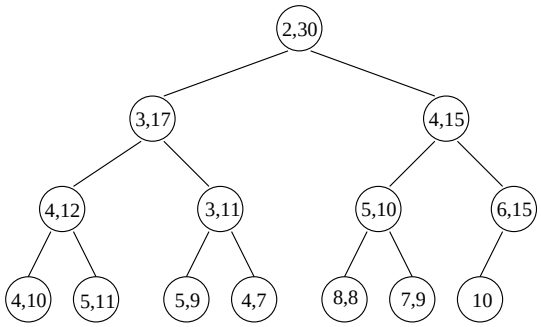

Figura 4: Heap de intervalos resultante da inserção do elemento 10 no heap de intervalos da Figura 3.

2. Suponha que queremos inserir um elemento com prioridade 3 no heap da Figura 1. Percorremos então um caminho a partir de A em direção à raiz, movimentando (atualizando) extremos esquerdos até que passemos da raiz ou cheguemos a um nó cujo extremo esquerdo seja ≤ 3. O novo elemento é então inserido neste nó que agora não tem extremo esquerdo. A Figura 5 mostra o intervalo de heap resultante após a inserção.

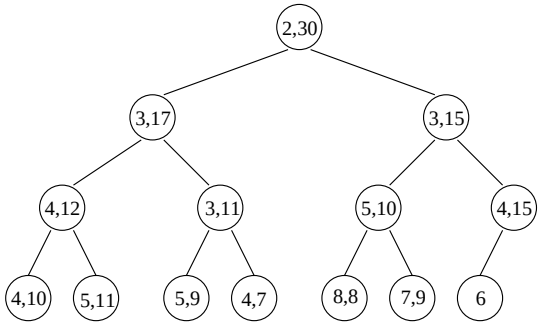

Figura 5: Heap de intervalos resultante da inserção do elemento 3 no heap de intervalos da Figura 3.

3. Suponha que queremos inserir um elemento com prioridade 40 no heap da Figura 1. Percorremos então um caminho a partir de A em direção à raiz, movimentando (atualizando) extremos direitos até que passemos da raiz ou cheguemos a um nó cujo extremo direito seja ≥ 40. O novo elemento é então inserido neste nó que agora não tem extremo direito. A Figura 6 mostra o intervalo de heap resultante após a inserção.

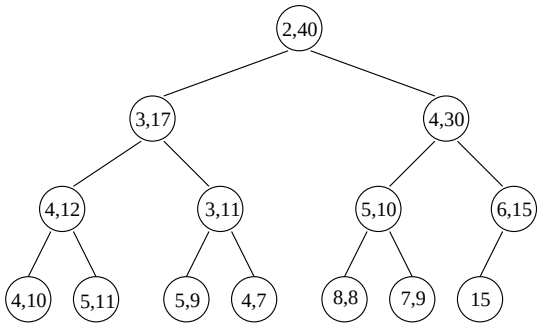

Figura 6: Heap de intervalos resultante da inserção do elemento 40 no heap de intervalos da Figura 3.

O número de nós é ímpar

Suponha agora que queremos inserir um elemento no heap de intervalos da Figura 6. Como o número de elementos é ímpar, nenhum nó novo será criado e apenas os extremos dos intervalos serão atualizados. O procedimento de inserção é análogo ao caso em que o número de elementos é par.

Seja A o último nó do heap de intervalos. Se a prioridade do novo elemento está dentro do intervalo $[a, b]$ do pai de A (na Figura 6, $[a, b] = [6, 15]$) então o novo elemento é colocado em A (ele torna-se então extremo esquerdo ou direito do novo intervalo).

Se a prioridade do novo elemento é menor que o extremo esquerdo a do intervalo do pai de A, então o novo elemento é inserido no heap de mínimo embutido no heap de intervalo; caso contrário, o novo elemento é inserido no heap de máximo embutido no heap de intervalos. A Figura 7 mostra o heap de intervalos resultante após a inserção de um elemento com prioridade 32 no heap de intervalos da Figura 6.

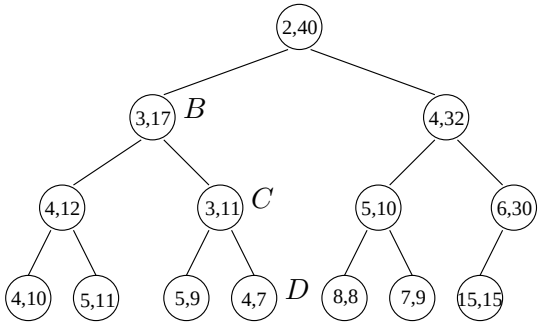

Figura 7: Heap de intervalos resultante da inserção do elemento 32 no heap de intervalos da Figura 6.

2.2 Remoção do elemento de prioridade mínima

A remoção do elemento mínimo do heap de intervalos pode feita analisando os seguintes casos:

1. se o heap de intervalos estiver vazio, a operação falha;
2. se o heap tiver apenas um único elemento, este elemento é devolvido e o heap de intervalos torna-se vazio;
3. se o heap de intervalos possui mais que um elemento, o elemento correspondente ao extremo esquerdo do intervalo da raiz é devolvido. Esse elemento é removido da raiz. Se a raiz é o último nó do heap de intervalos, nada mais precisa ser feito. Caso contrário, removemos o elemento correspondente ao extremo esquerdo p do último nó do heap (se este nó contém apenas um elemento, ele é removido) e p (mais precisamente, o elemento correspondente a p) é reinserido no heap de mínimo embutido no heap de intervalos começando a partir da raiz. Note entretanto que à medida que descemos no heap, pode ser necessário trocar o valor p corrente com o extremo direito q do intervalo do nó sendo examinado para garantir que $p \leq q$. A reinserção é feita usando a mesma estratégia procedimento Desce_Heap descrito em aula.

Vejamos como remover o elemento mínimo do heap de intervalos da Figura 7.

- Primeiro, o elemento com prioridade 2 é removido da raiz. A seguir, o extremo esquerdo 15 é removido do último nó e começamos a reinserção a partir da raiz com $p = 15$.
- Note que $p = 15$ é menor ou igual ao extremo direito do intervalo da raiz. O menor dos filhos da raiz no heap de mínimo é 3 e está no nó B. Como este elemento é menor ou igual a $p = 15$, movemos o elemento 3 de B para a raiz (3 torna-se extremo esquerdo do intervalo da raiz) e prosseguimos a partir do nó B (filho esquerdo da raiz) com $p = 15$.
- Como p é menor ou igual ao extremo direito de B ($15 \leq 17$) não precisamos trocar seus valores. O menor dos filhos de B no heap de mínimo é 3 e está nó C. Como $3 \leq 15$ movemos 3 de C para B (3 torna-se extremo esquerdo do intervalo de B) e prosseguimos a partir de C com $p = 15$.
- Como p é maior que o extremo direito de C ($15 > 11$), trocamos os dois valores e assim, 15 torna-se extremo direito do intervalo de C e $p = 11$. O menor dos filhos de C no heap de mínimo é 4 e está no nó D. Como este elemento é menor ou igual a $p = 11$, movemos o elemento 4 de D para C e prosseguimos a partir de D com $p = 11$.
- Como p é maior que o extremo direito de D ($11 > 7$), trocamos os dois valores e assim, 11 torna-se extremo direito do intervalo de D e $p = 7$. Como D é uma folha, o valor corrente $p = 7$ é inserido em D como o extremo esquerdo do seu intervalo. A Figura 8 mostra o heap de intervalos resultante após a remoção do mínimo do heap da Figura 7.

Observação. Note que pode haver vários elementos com a mesma prioridade. O procedimento acima remove então apenas **um** elemento com prioridade mínima (o que está na raiz do heap

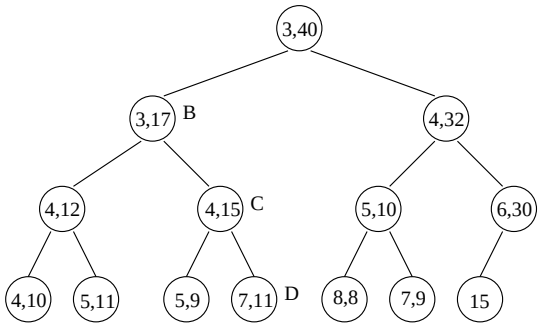

Figura 8: Heap de intervalos resultante da remoção do mínimo no heap de intervalos da Figura 7.

de intervalos). Isto **não** é inconsistente com a definição de heap de intervalo descrita aqui. Para remover **todos** os elementos de prioridade mínima, o procedimento é executado várias vezes. No **laboratório** suporemos que as prioridades são **distintas**.

2.3 Remoção do elemento de prioridade máxima

Este é análogo ao procedimento de remoção de mínimo com as modificações apropriadas.

2.4 Consumo de tempo

Todas as operações descritas podem ser implementadas de modo a garantir que o tempo gasto em cada uma é proporcional à altura do heap de intervalos.

3 A Atividade

Você foi contratado por um escritório de advocacia para resolver um impasse: nele trabalham dois advogados, João e Maria. João gosta de ajudar os seus clientes, não cobrando-os, mas por isso tem a filosofia de atender sempre primeiro quem tem menos dinheiro. Já Maria cobra de seus clientes e para maximizar seu ganho sempre dá prioridade aos clientes mais ricos (infelizmente há mais Marias que Joãos nesse mundo...). Você pode assumir que nunca haverá 2 clientes com a mesma quantia em mãos no escritório, e que a quantia de dinheiro é sempre um inteiro. Além disso, a capacidade máxima é de 130 clientes simultaneamente no escritório. Assim, sua tarefa é ajudar os advogados, dizendo quem é o próximo a ser atendido por cada um, quando um deles termina de atender um cliente.

3.1 Entrada de Dados

A entrada de dados corresponderá um caractere correspondente à operação seguido das opções relacionadas:

Caractere	**Primeiro Argumento**	**Segundo Argumento**	**Função**
A	dinheiro	nome_cliente	Corresponde à entrada do cliente nome_cliente no escritório, que possui a quantidade de dinheiro indicada. O cliente deve ser inserido no Heap, e sua quantidade de dinheiro deve ser utilizada como critério de prioridade.
J			João vai atender um cliente: o cliente que tiver menos dinheiro neste momento no escritório deve ser removido do Heap para ser atendido
M			Maria vai atender um cliente: o cliente que tiver mais dinheiro neste momento no escritório deve ser removido do Heap para ser atendido
#			Deve ser mostrado quantos clientes se encontram no escritório, para que os advogados saibam se devem apressar o atendimento ou não.

3.2 Saída de Dados

Para cada operação mostrada acima o programa deverá escrever uma linha:

Caractere	**Função**
A	Deve imprimir "Escritorio cheio" quando não houver mais espaço no escritório.
J	Deve imprimir "Joao vai atender nome_cliente (dinheiro)" ou "Escritorio vazio" quando não houver clientes.
M	Deve imprimir "Maria vai atender nome_cliente (dinheiro)" ou "Escritorio vazio" quando não houver clientes.
#	Deve imprimir "Ha # clientes", com a quantidade de clientes no lugar de #.

3.3 Observações

O programa deve ser implementado em C e DEVE utilizar o algoritmo descrito acima para implementar o heap. O número máximo de submissões é 15 e o prazo máximo é até 1 de Novembro de 2012.