Curso de Tecnologia em Sistemas de Computação
Disciplina: Estrutura de Dados e Algoritmos
AP3 - Primeiro Semestre de 2015

Nome -
Assinatura -

Observações:

1. Prova sem consulta e sem uso de máquina de calcular.
2. Use caneta para preencher o seu nome e assinar nas folhas de questões e nas folhas de respostas.
3. Você pode usar lápis para responder as questões.
4. Ao final da prova devolva as folhas de questões e as de respostas.
5. Todas as respostas devem ser transcritas nas folhas de respostas. As respostas nas folhas de questões não serão corrigidas.

---

1. Forneça as definições dos seguintes conceitos:

   (a) (1,5) Limite Inferior de um Problema

   Resposta: Um limite inferior para um problema P é uma função f, tal que qualquer algoritmo que resolve P executa pelo menos $\Omega(f)$ passos, ou seja, a complexidade de pior caso de qualquer algoritmo que resolva P é $\Omega(f)$.

   (b) (1,5) Árvore Binária de Busca

   Resposta: Uma árvore binária de busca é uma árvore binária (árvore em que o nó raiz possui duas subárvores binárias, *esquerda* e *direita*, ou é vazia) vazia ou que todos os elementos da subárvore esquerda são menores que o elemento pertencente à raiz, cujo valor é menor que todos os elementos da subárvore direita. Além disso, as subárvores esquerda e direita da raiz são árvores binárias de busca.

2. Responda os itens a seguir:

   (a) (1,0) Desenhe uma árvore binária de busca que seja **estritamente binária, de altura 4, e com o menor número possível de nós**. Não se esqueça de colocar os valores das chaves dentro de cada nó.

   Resposta: Lembrando que uma árvore estritamente binária de busca é uma árvore binária de busca em que cada nó possui exatamente 0 ou 2 filhos, podemos observar na Figura 1 uma representação de árvore estritamente binária de busca de altura 4.

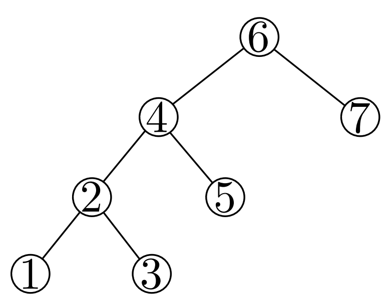


Figura 1: Exemplo de árvore estritamente binária de busca de altura 4 com o menor número de elementos.

(b) (1,0) Para a árvore acima, escreva a sequência que corresponde à ordem dos nós visitados no **percurso em pós-ordem**.

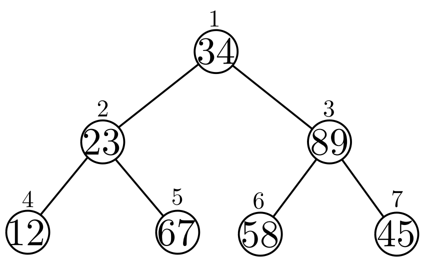
Resposta: Lembrando que no percurso em pós-ordem a visita aos nós da árvore são feitas seguindo a ordem do filho esquerdo em pós-ordem, filho direito em pós-ordem e raiz, obtemos que a sequência de visitas da árvore representada na Figura 1 como: $\langle 1, 3, 2, 5, 4, 7, 6 \rangle$.

3. (1,5) Explique as vantagens e desvantagens das seguintes estruturas de dados, em relação à *remoção* de um elemento:
(a) lista sequencial não ordenada; (b) lista encadeada não ordenada.

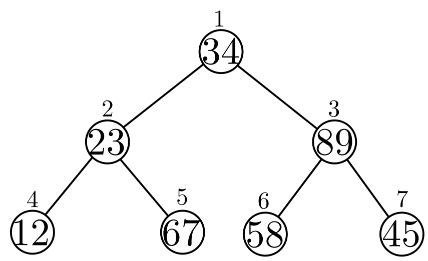
Resposta: Ambos os algoritmos de remoção de um elemento em listas sequenciais e encadeada não ordenadas possuem complexidade de pior caso igual a $O(n)$. A complexidade de pior caso é dada pela busca do elemento a ser removido, onde devemos percorrer a lista em busca do mesmo, em ambos os casos. Porém, a remoção na lista sequencial consome $O(n)$ passos se quisermos manter a lista resultante na ordem inicial dos elementos, já que é necessário copiar todos os elementos posteriores ao da posição onde houve a remoção. O mesmo não é necessário em listas encadeadas, já que a alteração do ponteiro do nó antecessor aquele nó p que contém o elemento a ser removido para o sucessor $p+1$ é feita em tempo constante, $O(1)$, sendo a mesma complexidade para liberar o nó p para a memória. Podemos eliminar a necessidade de efetuarmos a cópia dos elementos posteriores aquele encontrado ao movermos a posição do último elemento para esta posição e diminuindo o tamanho da lista em uma unidade. Porém, nas listas encadeadas faz-se necessário a liberação da memória da posição a ser removida, enquanto a lista sequencial não o faz. Assim, do ponto de vista da remoção de elementos, a lista sequencial é melhor empregada em situações em que há poucas remoções, caso contrário haverá muita memória desperdiçada, o que não ocorre em listas encadeadas.

4. (2,0) Desenhe e explique os passos intermediários do algoritmo de construção de um *heap* (lista de prioridades) que é executado em tempo $O(n)$, para o seguinte vetor de entrada: 34, 23, 89, 12, 67, 58, 45.

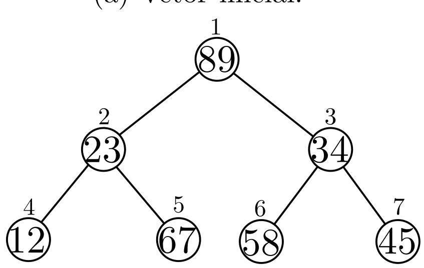
Resposta: A Figura 2a representa o vetor de entrada em forma de árvore binária, onde cada índice do vetor é indicado acima de um nó e o valor de um nó está representado no mesmo. Executamos os passos $Descer(\lfloor \frac{i}{2} \rfloor, i)$, para cada i de n até 1, onde n é o tamanho do vetor de entrada, de modo a construir um heap cujo primeiro elemento seja o maior de todos. Como $n = 7$, a Figura 2b mostra o heap resultante da operação *Descer* aplicada ao nó de índice 3. Como ele maior que seus dois filhos, então o heap não é alterado. A Figura 2c mostra o resultado da operação *Descer* aplicado ao primeiro elemento, obtendo assim o heap representado na Figura 2d, onde o elemento de valor 89 passa a ser o primeiro elemento, enquanto o valor 34 "desce"para a posição 6. A Figura 2e mostra o resultado da operação *Descer* aplicado ao elemento da posição 2, que troca de lugar com o elemento da posição 5. Finalmente, aplicamos *Descer* ao primeiro elemento do heap. Como o valor nesta posição é o maior de todos, o algoritmo termina.



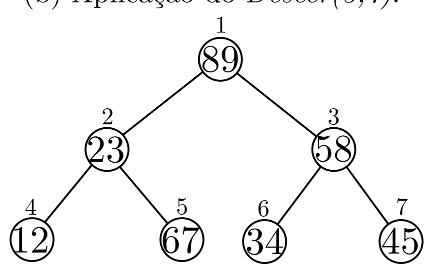
(a) Vetor inicial.



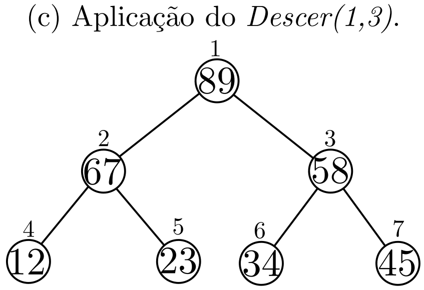
(b) Aplicação do *Descer(3,7)*.



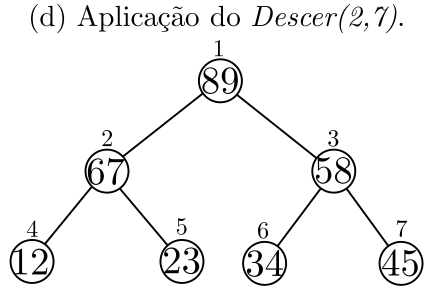
(c) Aplicação do *Descer(1,3)*.



(d) Aplicação do *Descer(2,7)*.



(e) Aplicação do *Descer(1,2)*.



(f) Aplicação do *Descer(1,1)*.

Figura 2: Execução do algoritmo linear de construção de heap a partir do vetor de entrada.

5. (1,5) Construa uma árvore de Huffman para o seguinte conjunto símbolos $\{s_1, \ldots, s_8\}$, onde cada s_i possui frequência f_i, com valores respectivamente iguais a:
$f_1 = 5$, $f_2 = 4$, $f_3 = 1$, $f_4 = 10$, $f_5 = 2$, $f_6 = 12$, $f_7 = 1$, $f_8 = 2$.

   Descrever, passo a passo, as computações efetuadas pelo algoritmo de Huffman.

   Resposta: De acordo com o algoritmo de Huffman, todas os símbolos são representados nas folha da árvore de Huffman e um nó interno contém o custo das subárvores esquerda e direita, dado pela soma de cada uma. O algoritmo inicia com um nó para cada símbolo, como na Figura 3.

Figura 3: Conjunto de nós contendo um símbolo em cada.

   A cada iteração, fazemos a escolha gulosa em relação ao custo das árvores em questão, onde escolhemos sempre os dois menores deles para compor a nova árvore através da *operação +*. Caso exista apenas uma árvore faltante, o algoritmo termina. Como os dois menores valores de frequência dados valem ambos 1, obtemos a Figura 4 como resultado da *operação +*:

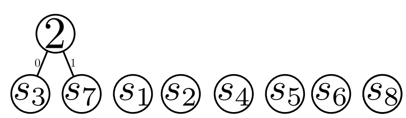


Figura 4: Passo 1.

   Em seguida os dois menores valores disponíveis são aqueles dados pela nova árvore gerada no passo anterior e o nó que contém o símbolo s_5, por exemplo, resultando na Figura 5:

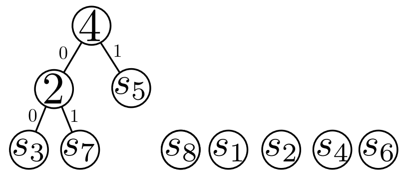


Figura 5: Passo 2.

Escolhemos agora a árvore obtida no passo anterior e o nó contendo o símbolo s_8, obtendo a Figura 6:

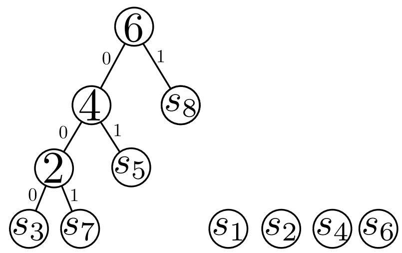


Figura 6: Passo 3.

Como os dois menores valores das árvores disponíveis são dados pelos nós contendo s_1 e s_2, unimos os mesmos em uma nova árvore, como na Figura 7:

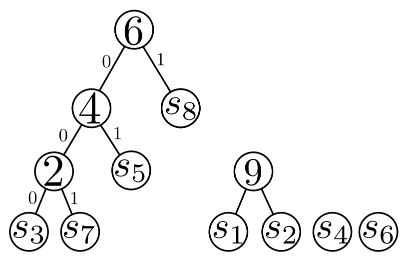


Figura 7: Passo 4.

Em seguida unimos as duas árvores resultantes das operações anteriores, já que suas raízes contém os menores valores disponíveis. Obtemos assim a Figura 8:

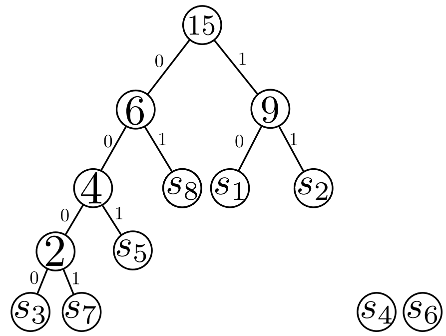


Figura 8: Passo 5.

Os menores valores disponíveis encontram-se nos nós com símbolos s_4 e s_6. Logo criamos uma nova árvore através da *operação* + com esses nós, obtendo a Figura 9:

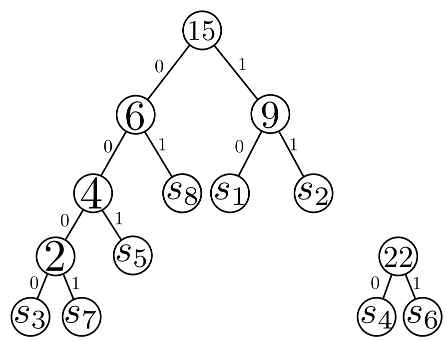


Figura 9: Passo 6.

Finalmente, como existem apenas duas árvores restantes, não temos escolha e fazemos a união das mesmas obtendo uma árvore de Huffman ótima, representada na Figura 10.

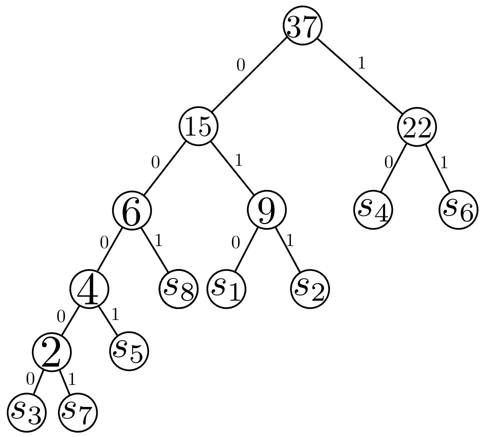


Figura 10: Árvore de Huffman ótima.