Estrutura de Dados - 1º. período de 2017

Segunda Avaliação a Distância

1. (1,5) Elabore um algoritmo que, recebendo como entrada uma árvore binária T, calcula para cada nó v de T o número de elementos da subárvore de T enraizada em v. Cada nó v de T apontado por um ponteiro pt possui os campos $pt \uparrow .esq$ e $pt \uparrow .dir$, para apontar para os filhos esquerdo e direito, respectivamente.

   Resposta: O Algoritmo 1 calcula o número de elementos da subárvore enraizada por cada nó da árvore T dada como entrada. Supomos que T possui um campo raiz e que cada nó v de T possui um campo N_elem que guarda o número de elementos na subárvore enraizada por v. É utilizado o Procedimento 2 que calcula o campo N_elem de um dado nó v de forma recursiva, onde é feita a soma dos respectivos campos da subárvore esquerda e direta, além do próprio v no caso em que v é não vazio.

Algoritmo 1: Algoritmo que calcula o número de nós da subárvore enraizada por cada nó de T através de um procedimento recursivo.

Entrada: Árvore binária T cujos nós possuem um ponteiro tanto para o filho esquerdo quanto para o direito.

$v \leftarrow T.raiz$;
Proc(v);

Procedimento 2: Proc(v)

Entrada: Ponteiro para o nó v de uma árvore binária.
Saída: Número de elementos da subárvore enraizada por v.

$pt \leftarrow v$;
$pt.N\_elem \leftarrow 0$;
se $pt \neq \lambda$ **então**
  $pt.N\_elem \leftarrow 1 + \text{Proc}(pt \uparrow .esq) + \text{Proc}(pt \uparrow .dir)$;
retorna $pt.N\_elem$;

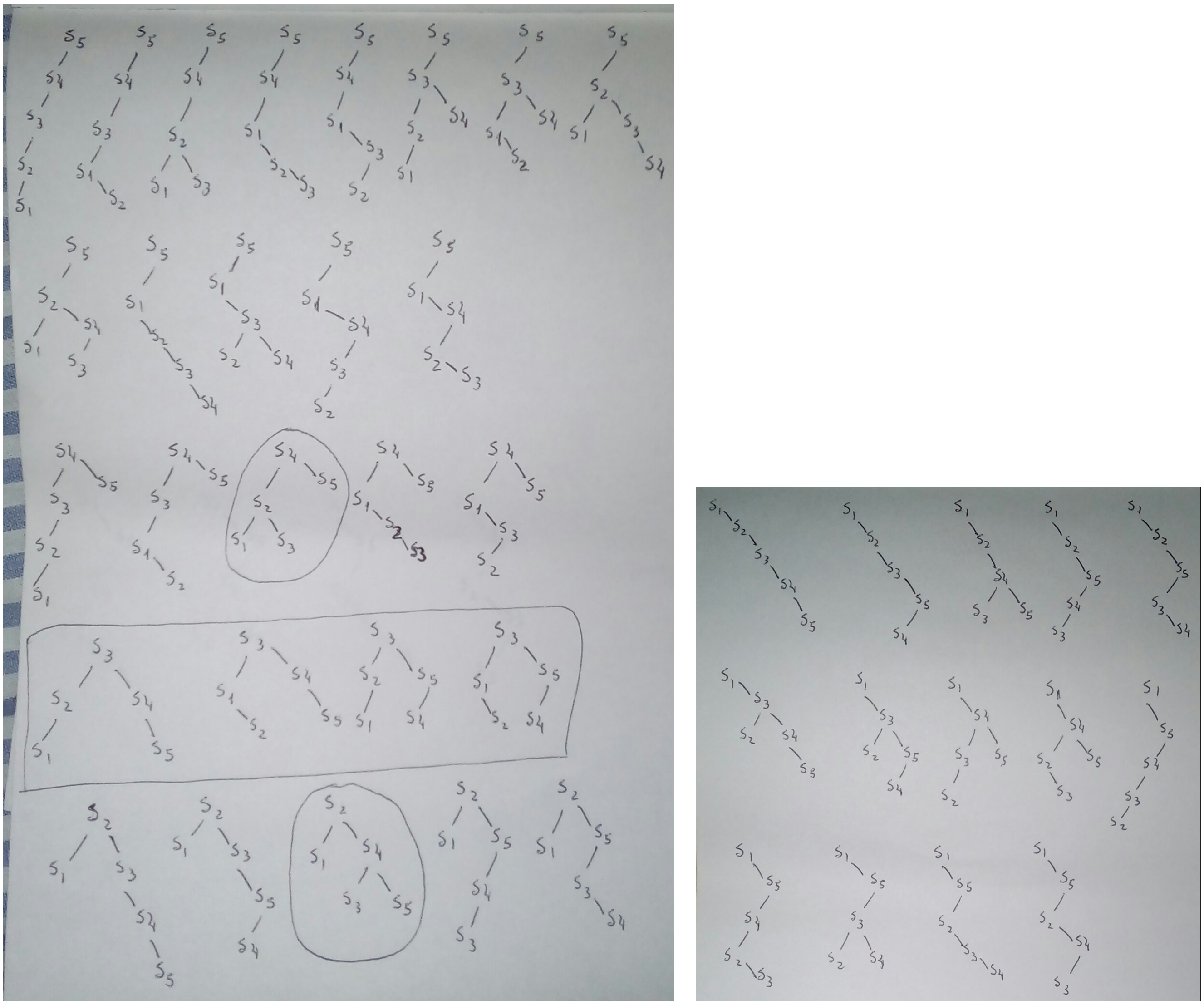
---

2. (1,5) Dado um conjunto de chaves $S = \{s_1, s_2, \ldots, s_5\}$, em que $s_1 < s_2 < \cdots < s_5$, desenhe todas as possíveis árvores binárias de busca para este conjunto de chaves.

   *Resposta:*A Figura 1 mostra todas as possíveis árvores binárias de busca com cinco valores distintos.

3. (1,5) Desenhe todas as árvores AVL possíveis para o conjunto de chaves $S = \{1, 2, 3, 4, 5\}$.

   Resposta: As árvores circuladas na Figura 1 representam todas as possíveis árvores AVL com cinco valores distintos.



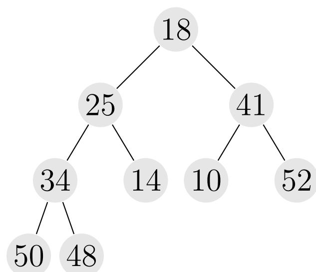
(a) (b)

Figura 1: Todas as possíveis árvores binárias de busca com cinco valores distintos.

4. (1,5) Prove ou dê contra-exemplo para a seguinte afirmação: Para qualquer conjunto de chaves e qualquer valor $d > 1$, sempre existe uma árvore B de ordem d que armazena estas chaves.

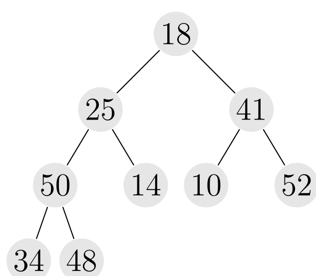
   Resposta: Vamos provar por contradição que a afirmação é falsa. Seja um contra-exemplo minimal C com relação à remoção de qualquer elemento. Seja x o maior elemento pertencente a C. Pela minimalidade de C, existe uma árvore B de ordem d, digamos T, formada pelos elementos de C exceto por x. Apenas precisamos mostrar que a inserção de x em T gera uma nova árvore B com o mesmo parâmetro d. Pela definição de árvore B e escolha de x, x deve ser inserido na última posição da última página P de T. Se P possui no máximo $2d-1$ elementos, então x pode ser inserido em P, obtendo uma nova árvore B de mesma altura e quantidade de páginas. Assim podemos supor que P possui $2d$ elementos, o que leva a uma divisão de P em duas novas páginas com exatamente d elementos cada uma e a subida do elemento central x' de P para a página pai de P, P'. O mesmo procedimento ocorre agora com a inserção de x' em P' e assim sucessivamente até que se encontre uma página ancestral com menos de $2d$ elementos, ou até que seja necessário gerar uma nova página raiz com apenas um elemento. Dessa forma conseguimos obter uma nova árvore B de ordem d contendo todos os elementos iniciais de C, o que contradiz a minimalidade de C.

5. (2,0) Execute o método de ordenação por heap ("heapsort"), aplicando-o às seguintes prioridades (nesta ordem): $18, 25, 41, 34, 14, 10, 52, 50, 48$. Desenhe as configurações sucessivas da árvore durante o processo de ordenação.

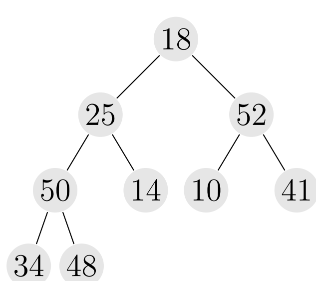


Aplicando o comando *arranjar(n)*.

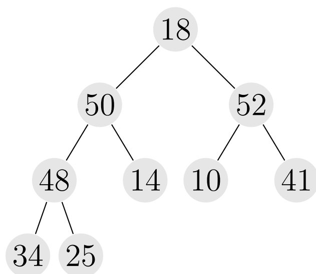
Descer 34:



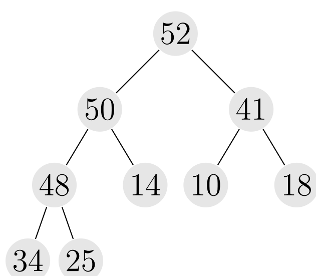
Descer 41:



Descer 25:

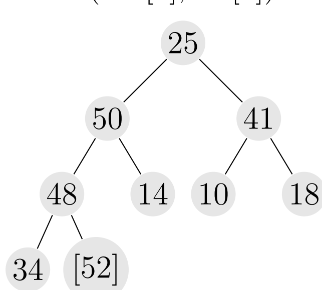


Descer 18 (heap obtido):

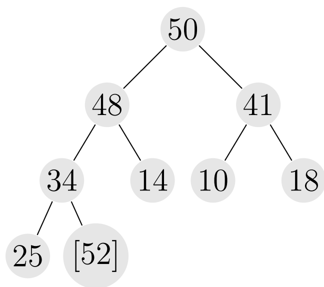


m:=9

trocar(TB[1],TB[9])

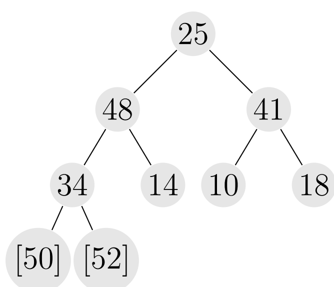


Descer 25:

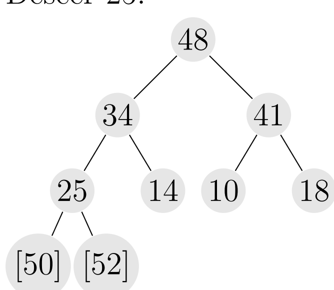


m:=8

trocar(TB[1],TB[8])

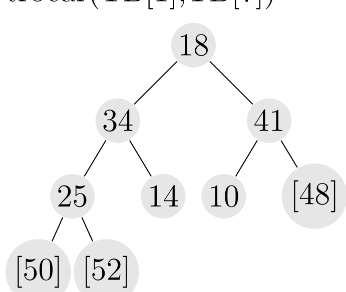


Descer 25:

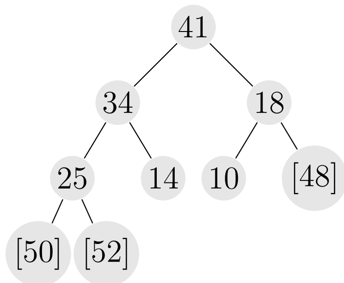


m:=7

trocar(TB[1],TB[7])

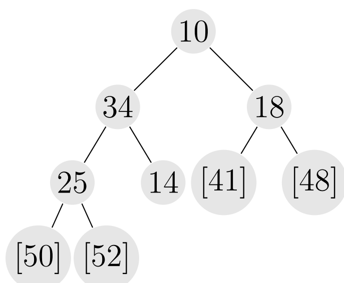


Descer 18:

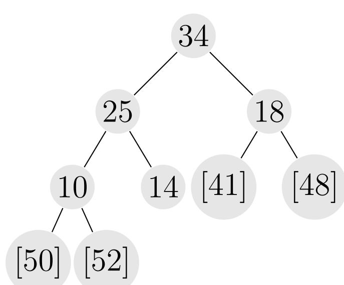


m:=6

trocar(TB[1],TB[6])

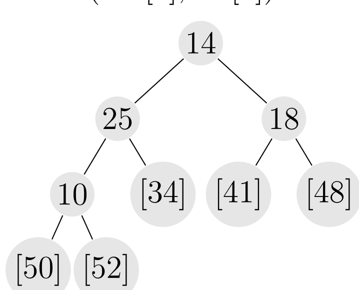


Descer 10:

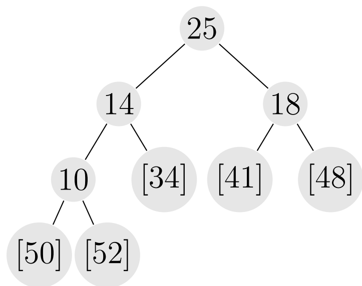


m:=5

trocar(TB[1],TB[5])

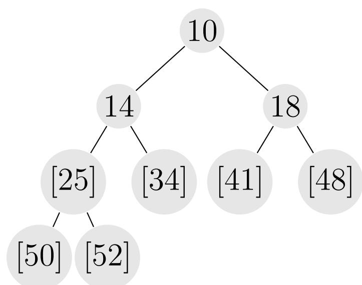


Descer 14:



m:=4

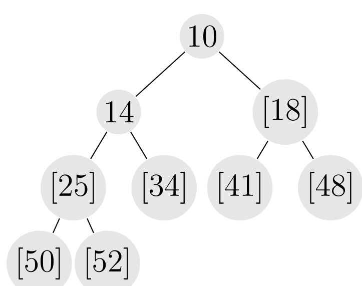
trocar(TB[1],TB[4])



Descer 10

m:=3

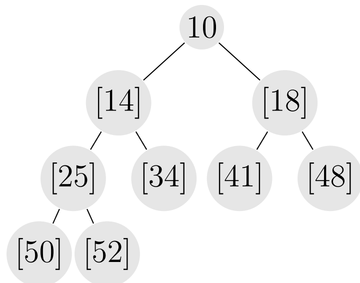
trocar(TB[1],TB[3])



Descer 10

m:=2

trocar(TB[1],TB[2])



6. (1,0) Assista às aulas sobre tabelas de espalhamento e faça uma dissertação resumida sobre como funcionam os métodos de tratamento de colisões abordados.

   Resposta: Tabelas de espalhamento servem para armazenar um conjunto de n valores de entrada em uma tabela com m posições. Como n e m podem ser bastante distintos, onde muitas vezes $n \gg m$, faz-se necessário o uso de funções de dispersão, que associa cada valor de entrada a uma posição da tabela. Eventualmente podem ocorrer colisões entre entradas distintas, ou seja, dados dois valores distintos x e y, temos $h(x) = h(y)$, onde h é uma função de dispersão. Uma função de dispersão é boa quando ela provoca poucas colisões e possui uma baixa complexidade de computação. Além disso é interessante que a função atribua igualmente os elementos a cada posição, tornando a distribuição uniforme. Para tratar colisões utiliza-se dois métodos principais: encadeamento externo e interno. Tais métodos consistem em armazenar chaves simétricas, que colidem, em uma lista encadeada.

   No encadeamento externo a tabela de espalhamento é uma lista de nós-cabeça para listas encadeadas. Para buscar uma chave x na tabela T, calcula-se $h(x) = xmodm$ e procura-se x na lista encadeada correspondente ao endereço-base $h(x)$. A inclusão de uma nova chave x é feita no final da lista encadeada correspondente ao endereço-base $h(x)$.

   No modelo por encadeamento interno, pode-se ainda separá-los em dois tipos: heterogêneo e homogêneo. Êm ambos os modelos o espaço utilizado para armazenar os elementos é apenas o da tabela, ou seja, m posições. Em outras palavras tal modelo pode ser utilizado quando $n \leq m$. No modelo heterogêneo a tabela é dividida em duas partes, uma contendo os endereços base, que são aqueles aos quais os valores armazenados são associados pela função de espalhamento, e os endereços de colisão. Cada posição na tabela contém um ponteiro para outra posição da mesma tabela, de modo que os endereços base tornam-se nós cabeça de listas encadeadas e os endereços de colisão apontam para endereços de colisão. Ao incluir um elemento, é feita a busca pelo mesmo na lista associada ao seu endereço pela função de espalhamento. Caso o mesmo não exista em tal lista o mesmo pode ser incluído ao final da lista em uma posição vaga. No modelo homogêneo não há uma diferenciação entre posições da tabela quanto a endereços base ou de colisão. Dessa forma, sempre que a função de dispersão associar um novo elemento x a uma posição i da tabela que esteja vazia, então x passa a ser o elemento da cabeça de uma nova lista encadeada. Caso a posição i já esteja ocupada por um elemento não pertencente a lista encadeada correspondente a função de dispersão, então tempos uma colisão secundária. Neste caso deve-se buscar uma posição vazia a partir do fim da tabela.

7. (1,0) Responda: como é a árvore de Huffman relativa a n frequências iguais? (Suponha que n é da forma $n = 2^k$, isto é, n é uma potência de 2.)

   Resposta: Seja f o valor de frequência de todos os elementos. Neste caso podemos ver que a cada passo podemos sempre escolher dois elementos de igual valor f de modo a formar uma nova subárvore com valor $2f$ até que todos os elementos iniciais sejam esgotados. Ao final desta etapa geramos $n/2$ subárvores com pesos idênticos a $2f$. Repetimos o procedimento gerando assim $n/4$ novas subárvores com valores idênticos a $4f$. Continuando este processo geramos um nó raiz com peso igual a $2^{\log n} \times f = n \times f$ e uma árvore de altura $\log n = k$ e cheia.