Importante! Alguns exercícios talvez não façam sentido porque ainda não tiveram a aula sobre o assunto. Provavelmente está lista será atualizada, mas vou esquecer de avisá-los. Assim, lembre-se de consultá-la de vez em quando.

Lista de Exercícios 2

1. Considere um árvore rubro-negra esquerdista (ARNE) inicialmente vazia. Desenhe as árvores obtidas após a inserção dos elementos 12,6,7,3,1,9,16,15. Escreva 12N se o nó que tem a chave 12 é negro e 12R se é vermelha. Desenhe a árvore resultante **após cada inserção** (você não precisa mostrar as rotações nem propagação de cores, apenas o resultado final da inserção). Consulte `https://codetube.vn/visual/redblacktree/`.
2. Considere um **heap de máximo** com n elementos armazenado em um vetor. Em quais posições podem estar:
   (a) o segundo maior elemento?
   (b) o terceiro maior elemento?
   (c) o menor elemento?
   (d) o segundo menor elemento?
   (e) o terceiro menor elemento?
3. Um professor de Estrutura de Dados solicitou como tarefa uma biblioteca de manipulação de heaps (de máximo). Suponha a seguinte declaração:

```
typedef struct {
    char nome[20]; /* whatever */
    int chave;
} Item;
typedef struct {
    Item *v;
    int n, tam;
} FP, *p_fp;
```

   Além das funções tradicionais, foi exigida uma implementação da função `RemoveHeap(fprio, i)` que remove o elemento da posição i do vetor `fprio->v`. O algoritmo abaixo foi proposto por vários alunos. Infelizmente, o algoritmo **não** está totalmente correto.

```
RemoveHeap(p_fp fprio, i) {
  troca(&(fprio->v[i]), &(fprio->v[n-1]));
  fprio->n--;
```

```
    desce_no_heap(fprio, i);
  }
```

(a) Mostre onde está o erro do algoritmo acima ou então mostre um exemplo onde ele não funciona. Veja que não estou interessado em erros de sintaxe ou coisa parecida, mas sim de uma falha conceitual.

(b) Conserte o algoritmo acima, de modo que o número de operações executadas seja ainda proporcional à altura do heap.

4. Considere um heap de máximo com as operações de inserção e remoção do máximo vistas em aula. Desenhe a árvore (representação do heap) resultante

   1. após as inserções das chaves: 7, 9, 12, 10, 4, 11, 13, 2, 8, 3, 20,
   2. seguido de duas remoções de máximo.

5. Um heap binário é um caso especial de heap k-ário, onde cada nó pode ter até k filhos e as chaves dos filhos são menores ou iguais à chave do pai. Uma representação por árvore completa de um heap k-ário tem 1 nó no nível 0, k nós no nível 1, k^2 nós no nível 2 etc exceto possivelmente no último nível, em que os nós estão mais à esquerda possível. Como você implementaria um heap ternário ($k = 3$) de máximo? Descreva como você calcularia os índices dos filhos do nó na posição i. Escreva funções para inserção e remoção do máximo nesta TAD.

6. Abaixo estão dois códigos de algoritmos que supostamente ordenam um vetor. Eles funcionam?

```
void exchange_sort(int A[ ], int n) {
   int i, j;
   for (i=0; i<n; i++)
      for (j=i+1; j<n; j++)
         if (A[i] > A[j]) troca(&A[i], &A[j]);
}

void como_eh_que_pode_sort(int A[ ], int n) {
   int i, j;
    for (i=0; i<n; i++)
       for (j=0; j<n; j++)
          if (A[i] < A[j]) troca(&A[i], &A[j]);
}
```

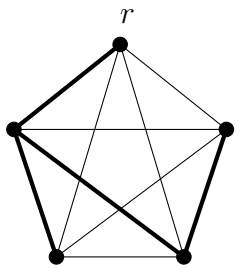
7. Suponha que mantemos uma tabela de hashing com M entradas com função de hashing $h(x) = 11k \bmod M$ onde k é a posição da primeira letra do nome x no

alfabeto. Por exemplo, se x=Ana então $k = 0$; se x=Bruno, então $k = 1$ etc. As colisões são tratadas usando encadeamento separado (as listas ligadas não estão ordenadas). Suponha que inserimos em uma tabela inicialmente vazia os nomes "Edu", "Ana", "Son", "Yap", "Quem", "Ugo", "Tom", "Ian", "Oto", "Neo" com $M = 5$.

(a) Desenhe a configuração da tabela após as inserções.

(b) Desenhe a configuração da tabela após as inserções, agora supondo que as listas ligadas são mantidas ordenadas em ordem alfabética. A sua resposta depende da ordem da inserção dos nomes?

8. Considere o grafo abaixo. As arestas em negrito formam uma árvore. Verifique se esta árvore pode ser obtida através de uma *busca em largura* ou de uma *busca em profundidade* começando em r. Se for possível, diga a ordem em que os vértices foram visitados.



9. Indique a para cada vértice v quem é o pai e qual é a distância da oriegem até v que resultam da execução da busca em largura do grafo orientado da Figura 1. usando o vértice 3 como origem.

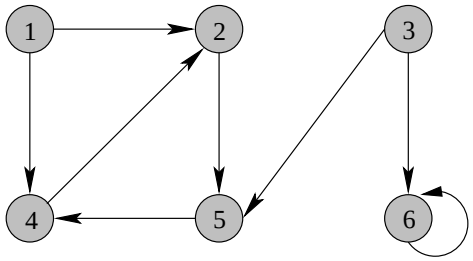


Figura 1: Grafo orientado.

10. Repita o exercício anterior agora com a execução da busca em largura do grafo não-orientado da Figura 2 usando o vértice u como origem.
11. Mostre como busca em profundidade funciona sobre o grafo indicada da Figura 3. Suponha que na lista de adjacência de cada vértice, seus vizinhos aparecem em ordem alfabética. Indique as arestas da árvore no desenho.

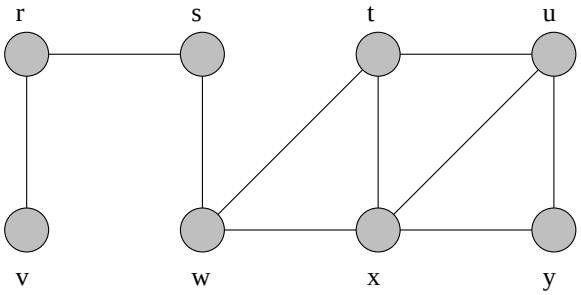


Figura 2: Grafo não-orientado.

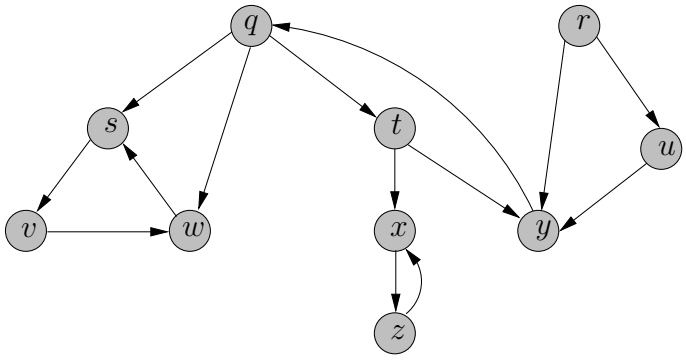


Figura 3: Grafo orientado.

12. Execute o algoritmo de Dijkstra para o grafo orientado da Figura 4 com fonte s. Faça o mesmo agora com fonte z.

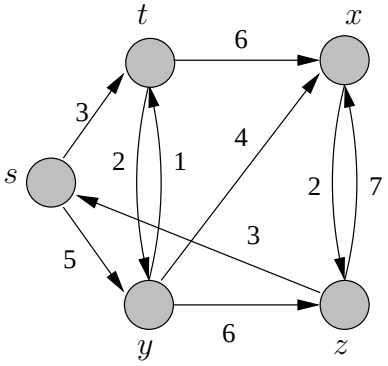


Figura 4: Grafo orientado com pesos/custos.

13. Desenhe a árvore B de ordem 5 resultante da inserção dos seguintes elementos: 0,

10, 20, 30, 1, 11, 21, 31, 2, 12, 22, 32, 3, 13, 23, 33, 4, 14, 24, 34, 5, 15, 25, 35, 6, 16, 26, 36, 7, 17, 27, 37, 8, 18, 28, 38, 9, 19, 29, 39. Suponha que a árvore está inicialmente vazia.

14. Considere a árvore B resultante das inserções indicadas na questão anterior. Desenhe a árvore B resultante da remoção de todos os múltiplos de 3 em ordem crescente.