Obs. Nos exercícios com AB, vamos assumir que temos

```
typedef struct STnode* link;
struct STnode { Item item; link l, r; int N; };  /* nó da AB */
static link head, z;        /* pointeiro para raiz e 'cauda' */
```

Outra obs. Os exercícios abaixo são principalmente conceituais e não envolvem programação. Na prova, acho que pedirei uma ou outra codificação, mas tentarei me concentrar em questões como as desta lista.

1. Considere o seguinte código:

```
void sortR(link h)
{ if (h == z) return;
  sortR(h->l); <imprima h->item>; sortR(h->r);
}
void STsort() { sortR(head); }
```

O que faz uma chamada de `STsort()` em uma AB genérica? Ilustre sua resposta em um exemplo também.

2. Considere o seguinte código:

```
link rotR(link h)
  { link x = h->l; h->l = x->r; x->r = h; return x; }
link rotL(link h)
  { link x = h->r; h->r = x->l; x->l = h; return x; }
```

O que fazem as chamadas `rotR(h)` e `rotL(h)` genericamente? Ilustre sua resposta em exemplos específicos também.

3. [12.72 de Sedgewick] Lembre que o campo `N` de um nó `h` é usado para manter o número de nós na subárvore de raiz `h`. Considere as funções `rotR()` e `rotL()` acima e note que elas não estão atualizando este campo apropriadamente. Altere-as de forma que isto seja feito.

4. Considere o seguinte código:

```
Item selectR(link h, int k)
  { int t = h->l->N;
    if (h == z) return NULLitem;
    if (t > k) return selectR(h->l, k);
    if (t < k) return selectR(h->r, k-t-1);
    return h->item;
  }
Item STselect(int k) { return selectR(head, k); }
```

O que faz a chamada `STselect(k)`?

5. Considere agora o seguinte código:

```
link partR(link h, int k)
  { int t = h->l->N;
    if (t > k) { h->l = partR(h->l, k); h = rotR(h); }
    if (t < k) { h->r = partR(h->r, k-t-1); h = rotL(h); }
    return h;
  }
```

O que faz a chamada `partR(head, k)`?

6. Continuando com a questão anterior, considere a seguinte função:

```
link balanceR(link h)
  { if (h->N < 2) return h;
    h = partR(h, h->N/2);
    h->l = balanceR(h->l);
    h->r = balanceR(h->r);
    return h;
  }
```

O que faz a chamada `partR(head)`?

7. [13.8 de Sedgewick] Suponha que inserimos, nesta ordem, as chaves

```
E A S Y Q U T I O N
```

em uma ABB incialmente vazia. Suponha que usamos inserção aleatória (`prog13.2`). Suponha também que a função aleatória que é usada é um tanto estrenha, de modo que a rotina de inserção acaba executando inserção na raiz toda vez que um item é inserido em uma árvore com um número ímpar de elementos. Desenhe as ABBs resultantes após cada inserção.

8. [13.25 de Sedgewick] Suponha que inserimos, nesta ordem, as chaves

```
E A S Y Q U T I O N
```

em uma árvore inicialmente vazia, usando inserção splay. Desenhe a árvore após cada inserção.

9. [13.27 de Sedgewick] Implemente `STsearch()` com splaying.

10. [13.39 de Sedgewick] Suponha que inserimos, nesta ordem, as chaves

```
E A S Y Q U T I O N
```

em uma árvore 2-3-4 inicialmente vazia. Desenhe a árvore após cada inserção.

11. Suponha que usamos árvores rubro-negras para implementar árvores 2-3-4. Qual é a altura máxima de uma árvore rubro-negra que contém `N` ítens? Justifique sua resposta.

12. [13.48 de Sedgewick] Suponha que usamos árvores rubro-negras para implementar árvores 2-3-4. Suponha que inserimos, nesta ordem, as chaves

```
E A S Y Q U T I O N
```

em nossa árvore. Desenhe a árvore após cada inserção.

13. Considere a seguinte função de hashing:

```
int hash(char *v, int M)
  { int h = 0, a = 128;
    for (; *v != '\0'; v++) h = (a*h + *v) % M;
    return h;
  }
```

Suponha que o tamanho `M` da tabela que está sendo usada é `1024`. O usuário percebe que muitas colisões ocorrem quando esta função de hashing é utilizada para processar as palavras que ocorrem em um dado texto (isto é, o espalhamento esperado não ocorre). Você teria uma explicação para este fenômeno? Em particular, quando ocorre desta função devolver o mesmo valor para chaves distintas?

14. [14.16 de Sedgewick] Suponha que usamos uma tabela de hashing com resolução de colisões por encadeamento. Suponha que inserimos `N` ítens em nossa tabela, que está inicialmente vazia. No pior caso (isto é, tendo muito azar com a função de hashing), quanto tempo pode demorar este processo? Suponha agora que resolvemos manter as nossas listas em ordem crescente de chaves; quanto tempo pode demorar este processo? (Nesta questão, estamos interessados em saber se o tempo é proporcional a `N`, proporcional a $\mathtt{N}^2$, proporcional a $\mathtt{N} \log \mathtt{N}$, etc).

15. [14.17 de Sedgewick] Suponha que estamos utilizando uma tabela de hashing com `M` entradas, com resolução de colisões por encadeamento (as `M` listas não são mantidas em ordem). Suponha que usamos a função de hashing que devolve $11k \bmod \mathtt{M}$ quando a entrada é a k-ésima letra do alfabeto (por exemplo, `C` corresponde a $k = 3$). Suponha que inserimos em uma tabela iniclamente vazia, nesta ordem, as chaves

```
E A S Y Q U T I O N
```

onde $\mathtt{M} = 5$. Desenhe diagramas que ilustram este processo de inserção.

16. [14.18 de Sedgewick] Continuando com a questão anterior, desenhe a configuração final da tabela agora supondo que mantemos as nossas listas em ordem crescente das chaves. A sua resposta depende da ordem de inserção das chaves?

17. Considere o seguinte código:

```
void STdelete(Item item)
  { int j, i = hash(key(item), M); Item v;
    while (!null(i))
      if eq(key(item), key(st[i])) break;
      else i = (i+1) % M;
    if (null(i)) return;
    st[i] = NULLitem; N--;
    for (j = i+1; !null(j); j = (j+1) % M, N--)
      { v = st[j]; st[j] = NULLitem; STinsert(v); }
  }
```

A função acima implementa a remoção de um item de uma tabela de hashing, onde usamos probing linear para a resolução de colisões. O que ocorreria se omitíssemos o `for` desta função? Por que a nossa tabela deixaria de funcionar?

18. [14.24 de Sedgewick] Quanto tempo pode demorar, no pior caso, a inserção de `N` ítens em uma tabela de hashing incialmente vazia, se usamos resolução de colisões por probing linear?

19. [14.25 de Sedgewick] Suponha que estamos utilizando uma tabela de hashing com `M` entradas, com resolução de colisões por probing linear. Suponha que usamos a função de hashing que devolve $11k \bmod \mathtt{M}$ quando a entrada é a k-ésima letra do alfabeto (por exemplo, `C` corresponde a $k = 3$). Suponha que inserimos, nesta ordem, as chaves

```
E A S Y Q U T I O N
```

em uma tabela inicialmente vazia, onde $\mathtt{M} = 16$. Desenhe diagramas que ilustram este processo de inserção.

20. [14.26 de Sedgewick] Faça a questão anterior com $\mathtt{M} = 10$.

21. [9.1 de Sedgewick] Este é um exercício sobre filas de prioridade. Na seqüência

```
P R I O * R * * I * T * Y * * * Q U E * * * U * E
```

uma letra significa a inserção daquela letra em uma fila e um asterisco significa a remoção do item de maior prioridade desta fila. Qual é a seqüência de chaves devolvida pelas 12 operações de remoção acima?

22. [9.3 e 9.4 de Sedgewick] Explique como usar uma fila de prioridades para implementar uma pilha. Explique como usar uma fila de prioridades para implementar uma fila.

23. [9.22 de Sedgewick] Suponha que a fila de prioridades da Questão 21 é implementada usando-se heaps. Desenhe os 25 heaps resultantes após cada uma das 25 operações daquela questão.

24. O algoritmo heapsort demora tempo $O(n \log n)$ para ordenar um vetor de n ítens. Dê um argumento preciso para justificar esta asserção.

25. Esta questão baseia-se na rotina de partição do quicksort. Considere o seguinte código:

```
select(Item a[], int l, int r, int k)
  { int i;
    if (r <= l) return;
    i = partition(a, l, r);
    if (i > k) select(a, l, i-1, k);
    if (i < k) select(a, i+1, r, k);
  }
```

Qual é o efeito da chamada `selection(a, 0, N-1, N/2)`?

26. Considere o seguinte código:

```
select(Item a[], int l, int r, int k)
{ while (r > l) {
    int i = partition(a, l, r);
    if (i >= k) r = i-1;
    if (i <= k) l = i+1;
  }
}
```

Qual é o efeito da chamada `selection(a, 0, N-1, N/2)`?