Observação

Nos exercícios com listas ligadas, vamos supor que temos

```
typedef struct node *link;
struct node { Item item; link next; };
```

Exercício 1 (4.6 do Sedgewick). *Este é um exercício sobre pilhas. Na seqüência*

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

uma letra significa empilhe e um asterisco desempilhe. Qual é a seqüência de letras devolvida pelos desempilhe?

Exercício 2 (4.7 do Sedgewick). *Continuemos usando a notação do exercício anterior. Diga como devemos inserir asteriscos na seqüência* `E A S Y` *para que os desempilhe devolva as seguintes seqüências: (i)* `E A S Y`*, (ii)* `Y S A E`*, (iii)* `A S Y E`*, (iv)* `A Y S E`*, (v)* `E Y A S`*. Note que nem sempre podemos obter a saída desejada; nestes casos, argumente por que a dada saída não pode ser obtida desta forma.*

Exercício 3 (4.31 do Sedgewick). *Este é um exercício sobre filas. Na seqüência*

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

uma letra significa inserção no fim da fila e um asterisco significa remoção do começo da fila. Qual é a seqüência de letras devolvida por esta seqüência de operações em nossa fila?

O problema das panquecas

Suponha que temos, por exemplo, a seqüencia de números

```
8 2 3 5 1 4 9 7 6 0
```

Imagine que esta seqüência indica uma pilha de panquecas de 10 tamanhos distintos, empilhadas de forma que a menor está no fundo da pilha, a maior é a quarta panqueca a partir do fundo da pilha, a segunda maior está no topo, etc. As únicas operações permitidas são inverter um segmento inicial desta seqüência. Portanto, existem 9 operações não-triviais para a pilha acima; uma delas produz, por exemplo,

```
5 3 2 8 1 4 9 7 6 0
```

Podemos chamar esta operação deste exemplo de ‘operação número 4’, já que invertemos a ordem das 4 panquecas do topo.

Exercício 4 (O problema das panquecas). *O problema é o seguinte: dados um inteiro positivo* `n` *e uma permutação dos inteiros não-negativos menores do que* `n`*, determinar uma seqüência de operações (permitidas) que resultam na seqüência*

```
0 1 2 ... n - 1
```

Isto é, com as panquecas em ordem crescente, do topo para o fundo. Escreva uma função de protótipo

```
void imprima_solucao(int n, int panq[]);
```

para resolver este problema.

Exercício 5 (5.14 do Sedgewick). *Escreva uma função recursiva que remove o último elemento de uma lista ligada.*

Exercício 6. *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?*

Exercício 7 (14.16 de Sedgewick). *Suponha que usamos uma tabela de hashing com resolução de colisões por encadeamento. Suponha que inserimos* N *itens 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* N^2*, proporcional a* $N \log N$*, etc).*

Exercício 8 (14.17 de Sedgewick). *Suponha que estamos utilizando uma tabela de hashing com* $\mathtt{M}$ *entradas, com resolução de colisões por encadeamento (as* $\mathtt{M}$ *listas não são mantidas em ordem). Suponha que usamos a função de hashing que devolve* $11\mathtt{k} \bmod \mathtt{M}$ *quando a entrada é a* $\mathtt{k}$*-ésima letra do alfabeto (por exemplo,* $\mathtt{C}$ *corresponde a* $\mathtt{k} = 3$*). Suponha que inserimos em uma tabela inicialmente 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.*

Exercício 9 (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?

Exercício 10 (9.22 de Sedgewick). *Suponha que a fila de prioridades da questão acima é implementada usando-se heaps. Desenhe os 25 heaps resultantes após cada uma das 25 operações daquela questão.*

Um exercício de projeto 1

Descreva *cuidadosamente* o projeto (dê os protótipos das funções principais, com uma descrição do que elas devem fazer e como elas devem ser implementadas (não esqueça do `main()`)) de um programa que resolve o seguinte problema: (*i*) a entrada de seu programa é formada por um inteiro k e um texto T e (*ii*) a saída de seu programa deve ser a lista das k palavras mais freqüentes no texto, com suas respectivas freqüências (número de ocorrências em T). O seu projeto deve supor que a entrada T pode ser grande; por exemplo, T poderia ser um livro como *Anna Karenina* (~ 350.000 palavras).

Um exercício de projeto 1

Qual seria a *complexidade de tempo* de um programa que fosse implementado seguindo-se seu projeto? Isto é, se $\mathtt{T}$ tem $\mathtt{n}$ palavras, quanto tempo levaria seu programa para processar o par $(\mathtt{k}, \mathtt{T})$? Responda a mesma pergunta para a *complexidade de espaço* de um tal programa (quanto espaço ele usa).

Observação. Você deve argumentar em detalhe por que o programa teria estas propriedades (em particular, você deve explicitar as hipóteses que você usa em sua análise).

Um exercício de projeto 2

Dizemos que uma palavra s é um *anagrama* de uma palavra t se s pode ser obtida de t pela permutação de suas letras. Por exemplo, as palavras *triangle*, *relating*, *integral*, *altering*, e *alerting* são todas anagramas umas das outras. Descreva *cuidadosamente* o projeto de um programa que resolve o seguinte problema: a entrada é um conjunto D de palavras e a saída deve ser um conjunto S de anagramas com $S \subset D$. Ademais, o conjunto S deve ser de cardinalidade máxima.

Um exercício de projeto 2

O seu projeto deve supor que a entrada D pode ser grande; por exemplo, D poderia ser o conjunto de palavras de um dicionário. Um programa implementado de acordo com o seu projeto deve ser tal que,

- sob hipóteses razoáveis, ele leva tempo basicamente proporcional ao número de palavras em D,
- ele gasta uma quantidade de memória proporcional ao número de palavras em D.

Você deve argumentar por que o programa teria estas propriedades (em particular, você deve explicitar as hipóteses que você usa em sua análise).

Um exercício de projeto 2

Sugestão. Chame de *assinatura* de uma palavra s a cadeia de caracteres que obtemos ao ordenar as letras em s. Por exemplo, a assinatura de 'assinatura' é 'aaainrsstu'. Use este conceito.

Um exercício de projeto 3

Descreva cuidadosamente o projeto de um programa que recebe uma cadeia de caracterers T como entrada e devolve o trecho repetido mais comprido neste texto. Por exemplo, se o texto de entrada for

```
aacabcabcbabcbabac
```

então a saída de seu programa deve ser `abcbab`.

Um exercício de projeto 3

O seu projeto deve supor que a entrada T pode ser grande; por exemplo, T poderia um livro todo. Um programa implementado de acordo com o seu projeto deve ser tal que,

- sob hipóteses razoáveis, ele leva tempo basicamente proporcional a $n \log n$, onde n é o número de caracteres em T,
- ele gasta uma quantidade de memória proporcional ao número de caracteres em T.

Você deve argumentar por que o programa teria estas propriedades (em particular, você deve explicitar as hipóteses que você usa em sua análise).

Um exercício de projeto 3

Sugestão. Um *sufixo* de uma cadeia de caracteres T é um 'segmento final' dela. Os sufixos de `abcde` são

```
abcde
bcde
cde
de
e
```

e a cadeia vazia, com 0 caracteres (6 sufixos no total). Considere os sufixos de T.