Os problemas a seguir devem ser resolvidos lendo uma linha por vez de um arquivo texto e executando a ação apropriada em cada linha em uma estrutura de dados.

A implementação deve ser eficiente para poder ler um arquivo de mais de 1 milhão de linhas e fornecer a resposta em poucos segundos.

1. Ler a entrada, linha por linha e então escrever as linhas na ordem inversa, de tal modo que a última linha lida seja a primeira a ser impressa, em seguida a penúltima, e assim em diante.

   Solução:

   ```
   Pilha p

   Ler linha de arquivo
   While linha != Nil:
     p.push (linha)
     Ler linha de arquivo

   while pilha p não estiver vazia:
     Imprime p.pop()
   ```

2. Ler as primeiras 50 linhas de uma entrada e então escrevê-las na ordem reversa. Ler as próximas 50 linhas e então escrevê-las em ordem reversa. Fazer isso até que não existam mais linhas a serem lidas, neste ponto as linhas restantes devem ser impressas em ordem reversa.

   Em outras palavras, sua saída vai começar com a linha de número 50, então e 49, 48 e assim por diante até a primeira linha. Isso será seguido pela linha de número 100, seguida pela 99m 98 até a linha 51, e assim por diante.

   Seu código nunca deverá ter armazenado mais de 50 linhas.

3. Ler a entrada uma linha por vez. Em qualquer momento, após ter lido as primeiras 42 linhas, se alguma linha é vazia (isto é, uma *string* de tamanho zero), então imprima a linha que ocorreu 42 linhas antes dela. Por exemplo, se a linha 242 é vazia, seu programa deve imprimir a linha 200. Seu programa deve ser implementado de tal maneira que ele nunca armazene mais de 43 linhas da entrada, em qualquer momento.

4. Ler a entrada linha por linha e imprimir a linha se ela não é a duplicata de alguma linha anterior. Preste atenção para que um arquivo com muitas linhas duplicadas não utilize mais memória que aquela necessária para o número de linhas únicas.

5. Ler uma linha por vez e imprimir uma linha somente se você já tiver lido esta linha anteriormente. O resultado final é que você remove a primeira ocorrência de cada linha. Preste atenção para que um arquivo com muitas linhas duplicadas não utilize mais memória que aquela necessária para o número de linhas únicas.

6. Ler uma linha por vez. Então imprimir todas as linhas ordenadas por tamanho, com a linha mais curta em primeiro. No caso em que duas linhas tenham o mesmo tamanho, resolva sua ordem usando a "regra de ordenação" usual. Linhas duplicadas devem ser impressas apenas uma vez.

7. Fazer o mesmo que no problema anterior, exceto que as linhas duplicadas devem ser impressas o mesmo número de vezes em que aparecem na entrada.

8. Ler a entrada completa, uma linha por vez, e imprimir as linhas pares (começando com a primeira linha, linha 0) seguidas pelas linhas ímpares.

9. Ler a entrada completa, uma linha por vez, e imprimi-las em uma ordem aleatória.

10. Uma *string* casada é uma sequência de caracteres {, }, (,), [, e] que estejam casados corretamente. Por exemplo, "{{()[]}}" é uma *string* casada, porém "{{()]}" não é, pois o segundo { casa com um]. Mostre que uma pilha pode ser usada para isso de tal modo que dada uma *string* de tamanho n, você possa determinar se ela é uma string casada no tempo $O(n)$.

11. Suponha que você tenha uma pilha s, que suporta somente as operações *push(x)* e *pop()*. Mostre como, usando somente uma fila FIFO, q, você pode reverter a ordem de todos os elementos em s.
