Relatório do Trabalho 2 de Teoria da Computação
Maurício Konrath e Thiago Xikota

Máquina 1a

Algoritmo

Dado um conjunto de caracteres representado por uma sequência de entrada w:

1. Varra a fita da esquerda para a direita, copiando todas as ocorrências do caractere 'a' para a terceira fita, parando após copiar o último 'a'.
2. Varra a fita da esquerda para a direita, copiando todas as ocorrências do caractere 'b' para a segunda fita, parando após encontrar o último 'b'. Se encontrar algum 'a', rejeite a sequência.
3. Se ler um caractere 'C', mova a fita 1 para a direita e percorra toda a fita 2 da direita para a esquerda, movendo a terceira fita para a esquerda para cada elemento da fita 2. Se, em algum momento, encontrar um espaço vazio na terceira fita e um 'b' na segunda fita, rejeite a sequência. Se encontrar espaços vazios nas três fitas, aceite a sequência.
4. Se ler um caractere 'C', mova a fita 1 para a direita e percorra toda a fita 2 da esquerda para a direita, movendo a terceira fita para a esquerda para cada elemento da fita 2. Se, em algum momento, encontrar um espaço vazio na terceira fita e um 'b' na segunda fita, rejeite a sequência. Se encontrar espaços vazios nas três fitas, aceite a sequência.
5. Volte ao passo 3.

Análise de complexidade

1. O primeiro passo percorre todos os elementos 'a' da primeira fita, o que resulta em uma complexidade de ordem 'n'.
2. No segundo passo, percorremos todos os elementos 'b' da primeira fita, o que também tem uma complexidade de ordem 'n'.
3. O terceiro passo envolve percorrer completamente a segunda fita para a esquerda, o que resulta em uma complexidade de ordem 'n'.
4. No quarto passo, percorremos completamente a segunda fita para a direita, o que também tem uma complexidade de ordem 'n'.
5. O quinto passo repete os passos 3 e 4 'c' vezes, resultando em uma complexidade de ordem 'n'.

Portanto, a complexidade total é dada por n + n + n(n + n) = 2n + 2n^2, o que resulta em uma complexidade de ordem 'n^2', representada por O(n^2).

Máquina 1b

Algoritmo

Dado uma sequência de caracteres representada por uma string de entrada 'w':

1. Percorra toda a fita da esquerda para a direita, marcando um '0' a cada dois caracteres.
2. Se, após o passo 1, a fita contiver apenas um '0', aceite a sequência.
3. Se, após o passo 1, a fita contiver mais de um '0' e o número de '0's for ímpar, rejeite a sequência.
4. Retorne o cabeçalho de leitura para o elemento mais à esquerda da fita.
5. Volte ao passo 1.

Análise de complexidade

1. O passo 1 percorre toda a fita uma vez, resultando em uma complexidade de ordem 'n'.
2. No passo 2, verificamos se há apenas um '0'. No entanto, não é necessário percorrer a fita novamente, pois podemos adicionar um estado adicional que aceita se encontrarmos um espaço vazio após o primeiro '0'. Portanto, a complexidade é constante, ou seja, O(1).
3. O passo 3, assim como o passo 2, envolve apenas uma verificação adicional, independentemente do tamanho da entrada. Podemos adicionar um estado adicional que rejeita se encontrarmos um espaço vazio em um número ímpar de '0's. O custo de percorrer toda a fita já foi considerado no passo 1, portanto a complexidade é constante, ou seja, O(1).
4. O passo 4 percorre a fita de volta do final à direita para o primeiro item à esquerda, resultando em uma complexidade de ordem 'n'.
5. O passo 5 retorna ao passo 1, mas o passo 1 reduz pela metade o número de '0's a cada repetição. A única maneira de sair do loop do passo 5 é através do passo 2 ou 3. Portanto, no pior caso, devemos repetir o passo 1 o suficiente para que reste apenas um '0', ou seja, repetir o passo 1 'x' vezes, onde 'x' resolve a equação $n/2^x = 1$, sendo n a entrada. Resolvendo a equação, temos $x = \log(n)$. Portanto, a complexidade é $\log(n)$.

Considerando que vamos repetir o passo 1 (complexidade O(n)) $\log(n)$ vezes, a complexidade total é $\log(n) * (n + 1 + 1 + n)$, resultando em uma complexidade de $O(n \log(n))$.

Máquina 2b

Algoritmo (Máquina Antiga)
Dado um input representado pela primeira fita e uma segunda fita para manter uma cópia de 'xi' facilitando a comparação com os demais elementos na entrada:

1. Inicie a leitura da primeira fita.
2. Ao encontrar o caractere '#', marque-o como 'C'.
3. Se encontrar outro caractere '#', aceite o input.
4. Se encontrar o caractere '1', copie-o para a segunda fita e marque-o como 'a'.
5. Se encontrar o caractere '0', copie-o para a segunda fita e marque-o como 'b'.
6. Repita os passos 4 e 5 até encontrar outro caractere '#'.
7. Retorne ao início da primeira fita.
8. Inicie a leitura da primeira fita novamente, ignorando os caracteres 'a' e 'b', e compare o conteúdo das duas fitas até encontrar um caractere '#'.
9. Se houver divergência entre as fitas, continue comparando.
10. Se houver divergência em todas as comparações, rejeite o input.
11. Se não houver divergência, retorne ao início da primeira fita, substituindo os caracteres 'a' e 'b' por '1' e '0', respectivamente.
12. Ignore o caractere 'C' e o conteúdo entre 'C' e o próximo caractere '#'.
13. Volte ao passo 4, repetindo o processo até chegar à última palavra a ser comparada.

Análise de complexidade

1. O passo 1 tem um tempo de computação proporcional ao tamanho da entrada, ou seja, O(n), onde n é o tamanho da entrada.
2. O passo 2 tem um tempo de computação constante, independentemente do tamanho da entrada.
3. O passo 3 também tem um tempo de computação constante.
4. O passo 4 tem um tempo de computação constante.
5. O passo 5 tem um tempo de computação constante.
6. O passo 6 tem um tempo de computação proporcional ao tamanho da palavra atualmente analisada, denotado por i.
7. O passo 7 tem um tempo de computação proporcional ao tamanho da entrada, ou seja, O(n).
8. O passo 8 tem um tempo de computação proporcional ao tamanho da entrada, ou seja, O(n).
9. O passo 9 tem um tempo de computação constante.
10. O passo 10 tem um tempo de computação constante.
11. O passo 11 tem um tempo de computação proporcional ao tamanho da entrada, ou seja, O(n).
12. O passo 12 tem um tempo de computação constante.

13. O passo 13 irá repetir a partir do passo 4 n vezes, resultando em um tempo de computação de n * (2i + 3n + c). Considerando que i é assintoticamente ignorável e ignorando as constantes de soma e multiplicação, a complexidade resultante é O(n^2).

Portanto, a complexidade do algoritmo implementado pela máquina é O(n^2).

Algoritmo (Máquina Nova)

A máquina nova implementa a seguinte ideia: para cada palavra i, de 1 até n, ela copia wi para a segunda fita e apaga wi da primeira fita. Em seguida, ela compara todas as outras palavras da primeira fita com wi. Se uma palavra for igual a wi, ela é marcada com um caractere especial que será ignorado se encontrado novamente (ou seja, passa por cima de palavras já vistas e iguais). Essa abordagem permite comparar cada palavra individualmente com as demais, mantendo apenas uma cópia de wi na segunda fita. Assim, a máquina pode identificar e marcar as palavras que são iguais a wi sem precisar percorrer toda a fita novamente.

1. O passo 1 percorre a primeira fita.
2. Ao encontrar um símbolo '#', ele é apagado da fita.
3. Se encontrar outro símbolo '#', o algoritmo aceita (isso ocorre apenas na primeira execução, correspondendo ao caso '##!').
4. Se encontrar o símbolo '1', ele é copiado para a segunda fita e apagado da primeira.
5. Se encontrar o símbolo '0', ele é copiado para a segunda fita e apagado da primeira.
6. Os passos 4 e 5 são repetidos até encontrar outro símbolo '#'.
7. Em seguida, o algoritmo percorre a primeira fita, ignorando os símbolos 'a', 'b' e 'C', e compara cada palavra com o conteúdo da segunda fita.
8. Ao encontrar uma palavra igual à da segunda fita, o algoritmo volta ao início da palavra e marca o símbolo '#' como 'C'.
9. Ao encontrar o símbolo '1' na palavra encontrada, ele é marcado como 'a'.
10. Ao encontrar o símbolo '0' na palavra encontrada, ele é marcado como 'b'.
11. Os passos 9 e 10 são repetidos até encontrar outro símbolo '#' ou 'C'.
12. Os passos 7 a 11 são repetidos até chegar na última palavra a ser comparada.
13. Se ao final da fita houver apenas divergências entre as palavras, o algoritmo rejeita.
14. O algoritmo volta à fita e verifica se não há mais símbolos '0' e '1'. Se não houver, ele aceita.
15. Os passos 1 a 2 e 4 a 15 são repetidos.

Análise de complexidade

1. A etapa inicial tem uma complexidade de tempo computacional igual a n, onde n representa o tamanho da entrada.
2. A segunda etapa possui uma complexidade de tempo computacional constante.
3. A terceira etapa possui uma complexidade de tempo computacional constante.
4. A quarta etapa possui uma complexidade de tempo computacional constante.

5. A quinta etapa possui uma complexidade de tempo computacional constante.
6. A sexta etapa possui uma complexidade de tempo computacional igual a 2i, onde i é o tamanho da palavra analisada.
7. A sétima etapa possui uma complexidade de tempo computacional igual a n (na primeira iteração do loop principal, diminuindo em 1 a cada iteração).
8. A oitava etapa possui uma complexidade de tempo computacional igual a i, onde i é o tamanho da palavra analisada.
9. A nona etapa possui uma complexidade de tempo computacional constante.
10. A décima etapa possui uma complexidade de tempo computacional constante.
11. A décima primeira etapa possui uma complexidade de tempo computacional igual a 2i, onde i é o tamanho da palavra analisada.
12. A décima segunda etapa irá repetir as etapas de 7 a 12 até alcançar a última palavra a ser comparada, tendo uma complexidade de tempo computacional igual a ((n + (n - (n - 1))) + 5i + c)/2, uma vez que o valor de n diminui em 1 a cada iteração do loop da décima quinta etapa.
13. A décima terceira etapa possui uma complexidade de tempo computacional constante.
14. A décima quarta etapa possui uma complexidade de tempo computacional igual a n.
15. A décima quinta etapa possui uma complexidade de tempo computacional igual a n * ((n + (n - (n - 1))) + 5i + c)/2 + c.

Considerando i como o tamanho de uma palavra do conjunto de palavras de entrada, podemos considerar i como negligenciável em termos assintóticos. Também podemos ignorar as constantes de soma e multiplicação, resultando em n * (n + (n - n + 1))/2, que pode ser simplificado para 1/2 * n^2 + n. Como n^2 define o comportamento do algoritmo, a complexidade deste algoritmo é O(n^2).