Universidade Federal do Pará
ICEN
FACOMP

Relatório da Atividade II

Aluno: Pedro da Silva Mendes
Aluno: Itallo Fernando Cavalcante do Nascimento
Professor: Dr. Denis Lima do Rosário

Fevereiro
2026

Universidade Federal do Pará
ICEN
FACOMP

Relatório

Relatório da segunda atividade da disciplina de de Projetos de Algoritmos II, do curso de Bacharelado em Ciência da Computação da Universidade Federal do Pará.

Professor: Dr. Denis Lima do Rosário

Alunos: Pedro da Silva Mendes, e Itallo Fernando Cavalcante do Nascimento

Fevereiro
2026

Conteúdo

1 Informações

A seguinte atividade foi feita em dupla, por Pedro Mendes e Itallo Cavalcante, na conta do Hacker Rank, @pedromends_30, além disso, os códigos-fontes utilizados como resposta final aos desafios podem ser encontrado neste repositório.

2 Jesse and Cookies

O problema consiste em aumentar a doçura de um conjunto de cookies até que todos atinjam um valor mínimo k. Para isso, em cada passo, os dois cookies menos doces são combinados, gerando um novo cookie com doçura.

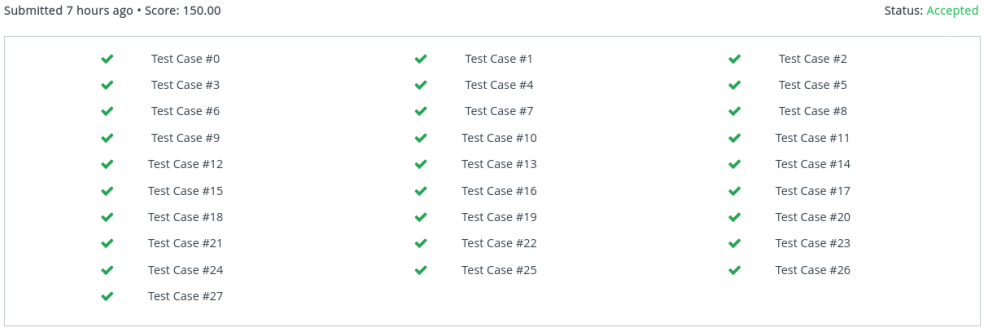
$$\text{doçura} = 1 \times (\text{menos doce}) + 2 \times (\text{segundo menos doce})$$

Esse processo se repete até que todos os cookies tenham doçura $\geq k$ ou até que não seja mais possível realizar combinações. O objetivo é descobrir o número mínimo de operações necessárias.
A todo momento, precisamos acessar os menores valores do conjunto.

2.1 Implementação

A primeira solução foi direta: a cada iteração do laço, a lista de cookies era ordenada.
Depois disso, os dois menores valores eram removidos, combinados conforme a regra, e o novo valor era inserido novamente na lista. A cada combinação, a variável de contagem de operações era incrementada.
Essa abordagem funcionou corretamente e passou nos testes, mas tinha um problema claro de eficiência: a lista estava sendo ordenada várias vezes sem necessidade.
Isso elevava muito o custo do algoritmo, levando a uma complexidade aproximada de $O(n^2 \log n)$. Percebendo isso, e considerando que o problema exigia acesso constante ao menor elemento, foi adotada uma segunda abordagem utilizando um heap mínimo. A lista inicial foi transformada em heap com heapify da biblioteca heapq do Python.
Com isso, a cada passo, os dois menores elementos eram removidos diretamente do heap, combinados, e o novo valor era inserido novamente.
Essa estratégia eliminou as ordenações repetidas e reduziu a complexidade para $O(n \log n)$.
As duas implementações passaram nos testes do HackerRank, mas a segunda se mostrou muito mais eficiente após a análise.

2.2 Testcases



3 No Prefix Set

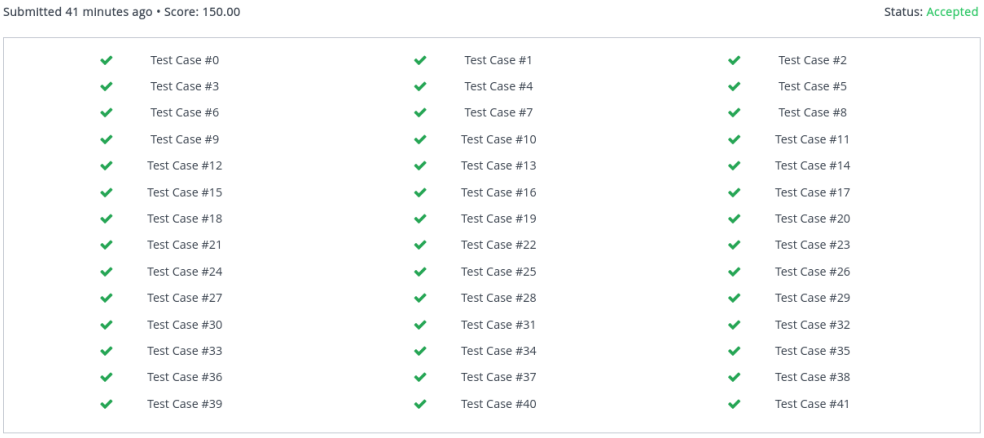
O problema apresenta uma lista de palavras formadas apenas por letras minúsculas. O conjunto é considerado GOOD SET quando nenhuma palavra é prefixo de outra. Caso contrário, é um BAD SET, e deve-se informar qual palavra violou a regra no momento da verificação.
Se duas palavras forem idênticas, elas também são consideradas prefixos entre si.
O ponto central do problema é verificar relações de prefixo entre várias strings de forma eficiente.

3.1 Implementação

Desde o início, ficou claro que a estrutura de dados mais adequada para esse tipo de verificação seria uma Trie (árvore de prefixos). Essa estrutura permite representar palavras letra a letra, compartilhando os caminhos comuns entre elas, o que facilita a detecção de prefixos.
Na primeira implementação, a ideia foi simples: construir a Trie com todas as palavras e, ao final, verificar se a quantidade de filhos do nó raiz era igual à quantidade de palavras fornecidas. A lógica era que, se cada palavra gerasse um ramo independente logo na raiz, então nenhuma seria prefixo da outra. Com isso, o programa apenas retornava GOOD SET ou BAD SET.
Essa abordagem funcionava parcialmente, mas apresentava duas falhas importantes. A primeira foi uma leitura incompleta do enunciado, que exigia também informar qual palavra causava a quebra da regra. A segunda foi perceber que a verificação apenas no nível da raiz era insuficiente, pois palavras poderiam compartilhar prefixos mais profundos na árvore sem necessaria-

mente aparecerem como filhos diretos da raiz.
Na segunda implementação, foi criado um método auxiliar que percorre a Trie enquanto cada palavra é inserida. Durante esse percurso, duas situações são verificadas: se a palavra inserida é prefixo de alguma existente, ou se alguma palavra existente é prefixo da palavra inserida.
Se ao percorrer a Trie, o marcador de fim de palavra for encontrado, significa que a palavra inserida é prefixo de alguma existente, se ao terminar de inserir a palavra, o nó atual ainda possuir filhos, significa que a palavra atual é prefixo de alguma palavra já inserida, e assim a Trie é torna-se excelente nesse caso.
Caso alguma dessas situações ocorra, o método retorna imediatamente que o conjunto é inválido e informa qual palavra causou o problema. Um segundo método apenas consome esse resultado e formata a saída conforme exigido.

3.2 Testcases



4 Conclusão

Os dois problemas evidenciaram, por caminhos distintos, a importância da escolha adequada de estruturas de dados e da interpretação completa dos requisitos propostos. Em ambos os casos, as primeiras soluções desenvolvidas eram funcionais e produziam os resultados esperados, porém apresentavam limitações que só se tornaram claras após uma análise mais criteriosa.

No problema dos cookies, a lógica da combinação estava correta desde o início, mas a forma de acessar repetidamente os menores elementos da lista gerava um custo computacional desnecessário. A substituição das ordenações sucessivas por um heap mínimo demonstrou, de forma prática, como a estrutura de dados influencia diretamente a eficiência do algoritmo.

Já no problema do conjunto de palavras, a Trie foi identificada como a estrutura mais adequada desde o começo. Entretanto, uma leitura incompleta do enunciado levou a uma verificação insuficiente, que precisou ser ajustada para atender corretamente às exigências do problema. A checagem realizada durante a inserção das palavras mostrou-se a abordagem mais precisa.

De modo geral, os exercícios reforçaram que desenvolver uma solução que funcione é apenas parte do processo. Avaliar sua eficiência, compreender o papel das estruturas de dados e garantir a aderência completa aos requisitos são aspectos igualmente essenciais para a construção de algoritmos corretos e bem elaborados.