

# Trabalho 01 - Máquina de maionese

stco02 2025s1

Data para a formação dos grupos (mesmo que de 1): 09/04/2025

Data de entrega: 25/04/2025

Data para entrega atrasada sem penalidade: 02/05/2025

Importante:

- **Não** olhem códigos de outros grupos ou da internet. Exceto os fornecidos e feitos em aula.
- Os trabalhos poderão ser feitos individualmente ou em duplas. Em qualquer caso é necessário preencher esse formulário para a formação de grupos (mesmo que seja um grupo de 1 só pessoa): <https://forms.gle/f9CPZ1CTqjpr76q39>
- TODOS os membros do grupo devem participar e compreender completamente a implementação.
- Em caso de plágio, fraude ou tentativa de burlar o sistema será aplicado nota 0 na disciplina aos envolvidos.
- Alguns alunos serão solicitados individualmente para explicar com detalhes a implementação.
- Passar em todos os testes não é garantia de tirar a nota máxima. Sua nota ainda depende do cumprimento das especificações do trabalho, qualidade do código, clareza dos comentários, boas práticas de programação e entendimento da matéria demonstrada em possível reunião.
- O líder do grupo deverá submeter, até a data de entrega, o seu código na plataforma [runcodes.hokama.com.br](https://runcodes.hokama.com.br).
- Você deverá implementar seu programa em linguagem Python.
- É esperado e faz parte do aprendizado vocês terem algumas dificuldades como Problemas com Falha de Segmentação, Loops Infinitos, etc. Então não deixe para os últimos dias!



Stardew Valley é um jogo de fazendinha que combina elementos de simulação, RPG e aventura. Nele, o jogador herda uma fazenda abandonada e deve cultivá-la, plantar diversas culturas, cuidar de animais e expandir suas terras. Além da vida rural, o jogo permite explorar cavernas repletas de criaturas e minerais, pescar e interagir com os moradores

da Vila Pelicano. No jogo você pode fabricar diversos itens e equipamentos, que usam diversos itens em sua receita.

(parte A) Um jogador achou que seria interessante fazer um aplicativo para consultar quais os itens necessários para cada receita. (parte B) E também para saber em quais receitas um determinado item pode ser usado.

## Parte A

Se o jogador quiser saber quais os ingredientes necessários para fabricar uma “Jarra de conserva” ele digita no terminal o nome da receita precedido de um r (de receita) e depois Enter.



```
r Jarra de conserva
```

E o programa deverá imprimir o nome da receita e a lista dos ingredientes e suas quantidades (na ordem que será dada no arquivo):

```
Jarra de conserva  
Madeira 50  
Pedra 40  
Carvão 8
```

Neste trabalho você receberá um arquivo `craft.txt` com as receitas que precisam ser lidas e deverá implementar uma **tabela hash de tamanho fixo 29**, para armazenar as receitas. A chave é o próprio nome da receita, e. g. “Jarra de conserva”, e o dado armazenado é uma lista dos ingredientes e suas quantidades.

Dado um par (chave, dado) a posição na tabela que esse par deve ser armazenado deve ser calculado com a seguinte função de hash, vista em aula, e depois tirando o módulo 29:

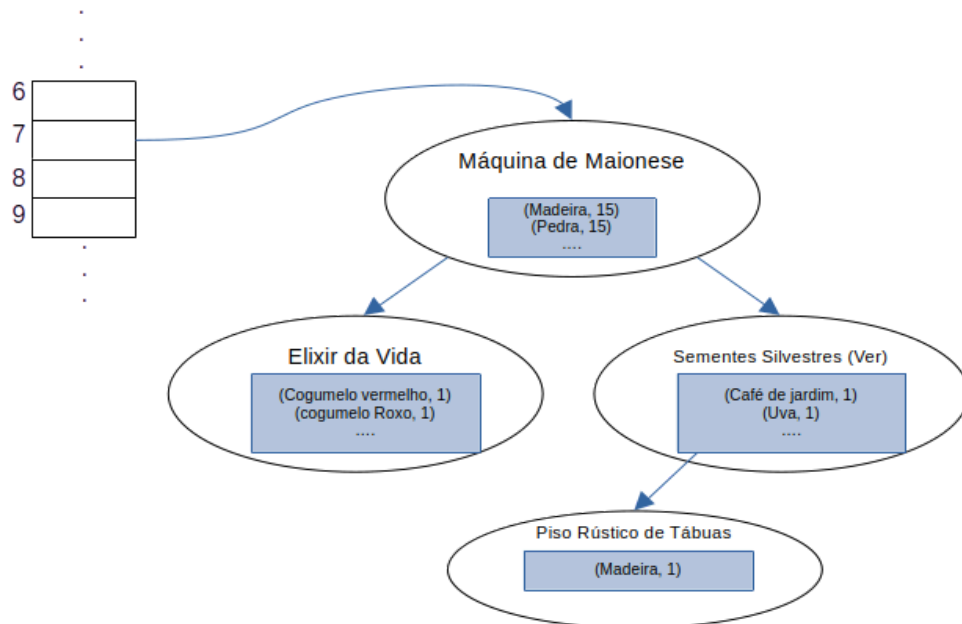
```
mult = 1  
hash_value = 0  
for c in s:  
    hash_value += mult * ord(c)  
    mult += 1  
return hash_value
```

Existem muito mais de 29 receitas então, haverá conflitos. Os conflitos deverão ser resolvidos usando uma **Árvore Binária de Busca**. Ou seja, cada uma das 29 posições da tabela terá uma Árvore Binária de Busca própria, contendo os pares (chave, dado) que foram mapeados para aquela posição. Os novos elementos da árvore devem ser inseridos em folhas, e não é necessário se preocupar com balanceamento nesse trabalho.

Por exemplo, considere essas quatro receitas e seus respectivos valores de hash.

```
Máquina de maionese: 18886  
Elixir da vida: 9722  
Sementes Silvestres (Ver): 29384  
Piso Rústico de Tábuas: 27006
```

Tirando o mod 29 de cada um desses valores chegamos no mesmo valor 7. Portanto as 4 receitas estarão armazenadas na posição 7, em uma árvore de busca binária, sendo que para decidir se um item deve estar a direita ou à esquerda, você deve comparar **as chaves diretamente** e não o hash value. Conforme ilustrado a seguir.



Para verificar se tudo foi armazenado corretamente, caso o usuário digite p r (de **print** receitas) o programa imprime toda a tabela Hash com o nome das receitas (as chaves). Caso nenhuma receita esteja naquela posição, ela imprime “None”, caso tenha uma ou mais receita, ela imprime a árvore binária correspondente, no formato red (pré-order):

(<raiz>, <subárvore esquerda>, <subárvore direita>)

Por exemplo a árvore binária da posição 7 da figura anterior seria impressa da seguinte forma:

(Máquina de maionese, (Elixir da vida, None, None), (Sementes Silvestres (Ver), (Piso Rústico de Tábuas, None, None), None))

Com todas as receitas carregadas o começo da tabela hash será assim (use o zoom para enxergar melhor):

```

p r
(Fertilizante Premium, (Caminho de cristal, None, (Espantalho, None, None)), (Vaso de Jardim, (Totem de Teletransporte: Ilha, None, None), None))
(Apiário, None, (Jarra de conserva, None, (Semente de Fibra, None, (Totem de Teletransporte: Deserto, None, None))))
(Imã, (Isca, (Braseiro de toco de árvore, None, None), None), None)
(Totem de teletransporte: Montanhas, None, None)
(Prensa de queijo, (Caminho de cascalho, (Boia-armadilha, None, (Braseiro dourado, None, None)), (Cesto de minhocas, None, (Moedor de Ossos, None, None))), None)
(Café de Jardim, (Totem de Teletransporte: Ilha, None, None), None)
  
```

## Parte B

Para que o jogador consiga ver onde um item pode ser usado, ele digita o nome do item precedido de i (de item) e depois enter. Por exemplo, se ele quiser saber em quais receitas o item “Essência nula” é usado ele digita:



```
i Essência nula
```

E então é impresso o nome do item para confirmação e cada receita que ele aparece (na ordem que será dada no arquivo).

```
Essência nula  
Megabomba  
Anel de irídio
```

Para trazer esse resultado rapidamente para o jogador você deverá armazenar uma **segunda tabela hash**. Nessa segunda tabela as chaves são os nomes dos itens e o dado armazenado é uma lista com as receitas no qual o item é utilizado. Sugiro que você crie as duas tabelas simultaneamente, para cada receita lida, você acessa a lista de cada item e adiciona o nome da receita. Como a lista é guardada por referência você pode acessar e modificar ela.

Para verificar se os itens foram armazenados corretamente se o usuário digitar “p i” (de print itens) o programa imprime a Tabela hash de itens seguindo o mesmo formato da Tabela hash de Receitas:

```
p i  
(Geléia do Rio, (Diamante, None, None), (Sementes Silvestres (Qualquer), None, None))  
(Sementes de Jasmim-azul, None, None)  
(Geléia da Caverna, (Bulbo de Tulipa, None, None), None)  
(Barra de ferro, (Barra Radioativa, None, None), (Uva, (Mármore, (Minério de irídio, None, None), None), None))  
(Minério de cobre, (Geléia do Mar, (Carne de inseto, None, (Cesto de minhocas, None, None)), None), (Minério de ferro, None, (Raiz-Forte, None, None)))  
(Facão, None, (Qualquer coiza, (Coqueado vermelho, None, None), None))
```

## Observações

Caso o usuário digite uma receita ou um item que não existe na Tabela (ou digite o nome errado) o programa imprime o nome que o usuário digitou e “Não encontrado” na linha seguinte e segue normalmente esperando o próximo comando.

```
r Mini-Forja
```

```
Mini-Forja  
Não encontrado
```

i Tijolo

Tijolo  
Não encontrado

Por fim, se o usuário digita “q” (de **Q**uit) o programa fecha corretamente.

q

Seu programa não deve imprimir nenhuma mensagem adicional na tela como “Digite seu comando:” ou “Digite r para procurar receita:” já que será comparado as saídas de cada programa.

- Se você não tiver certeza se alguma coisa é permitida ou não no trabalho, pergunte ao Hokama!