Relatório da resolução de "Os alquimistas se reúnem"

Vitor Rafael Gonçalves Bastos Borges*
Faculdade de Informática — PUCRS

5 de fevereiro de 2024

Resumo

Este artigo descreve o problema apresentado pelo T2, que é a necessidade do cálculo do valor de uma receita que – utilizando hidrogênio e passando por diversos outros elemento químicos – produz ouro. O artigo também apresenta a resolução proposta pelo autor, referenciando ambas as abstrações pensadas e as soluções elaboradas.

O Problema

O problema apresentado pelo T2 mostra um cenário onde diversos alquimistas necessitam da análise de receitas que transformam hidrogênio em ouro por entremeio da troca de valores transmutadas entre diferentes elementos até um valor final de ouro.

A Figura 1 demonstra uma ilustração utilizando elementos genéricos entre hidrogênio e ouro, com valores também genéricos em suas arestas.

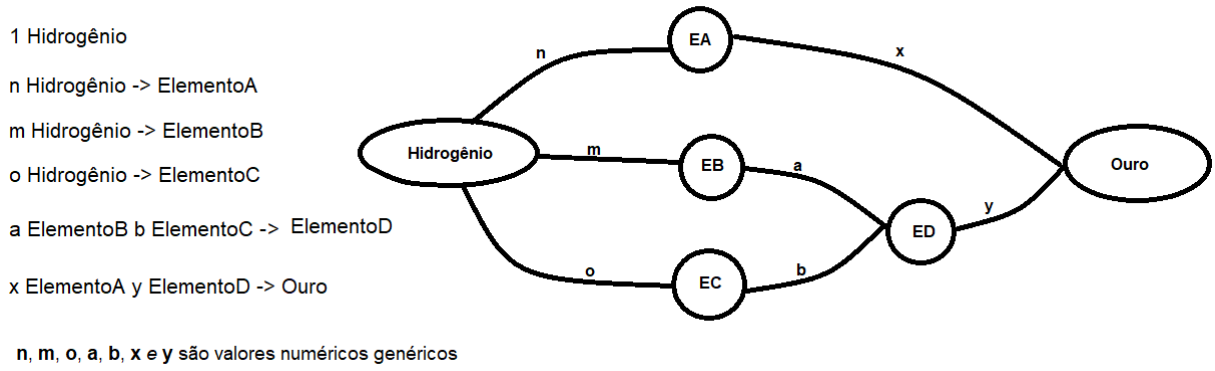

Figura 1: Exemplo da visualização dos elementos e arestas.

A Solução

A apresentação da solução é dividida entre a sua abstração e sua implementação, sendo abstração a conceptualização e a resolução de forma teórica do problema apresentado, e a implementação sua aplicação técnica utilizando os conceitos conceptualizados durante a abstração para a criação de um algorítimo.

*v.rafael02@edu.pucrs.br,

A Abstração

Na abordagem inicial à resolução, a Figura 1 será empregada como exemplo para esclarecer o processo de pensamento que conduziu à solução.

Se agimos com a suposição de que $Hidrogenio = 1$, podemos sucessivamente supor que $ElementoA = n \times Hidrogenio$, $ElementoB = m \times Hidrogenio$ **e** $ElementoC = o \times Hidrogenio$, e então podemos observar que $ElementoD = a \times ElementoB + b \times ElementoC$, então supomos que $Ouro = x \times ElementoA + y \times ElementoD$, logo, podemos finalmente montar a seguinte equação:

$$Ouro = x * (n * 1) + y * (a * [m * 1] + b * [o * 1])$$

Sendo *Ouro* o custo de *Hidrogênio* necessário para transmutação.

Consideramos que elementos podem possuir dependências que são declaradas apenas depois do elemento atual ser declarado, por exemplo, *Elemento X* tem dependência de *Elemento Y*, mas *Elemento X* é declarado antes de *Elemento Y*, nesse caso é interessante buscar declarar *Elemento Y* antes de *Elemento X*. Observa-se também a possibilidade de funcionamento recursivo, por exemplo *Elemento X* depende de *Elemento Y*, e *Elemento Y* depende de *Elemento Z*, mas ambos os casos são referências a elementos que são declarados posteriormente. Isto é relevante tendo em mente a formatação da lista de elementos proposta pelo problema apresentado.

O Implementação

Para finalmente resolver o problema descrito, fora criado um algoritmo que iterasse sobre todos os elementos da lista formatada nas quais estão os elementos, e os adicionasse em locais de memória para que pudessem ser referenciados por outros elementos que dependem destes. Isto é, fora criado método que crie diversos valores baseado nas equações da abstração e guarde estes valores junto dos elementos especificados aos quais estes valores pertencem, sendo o resultado final guardado junto da variável *Ouro*, então sendo a variável *Ouro* igual ao valor de *Hidrogênio* necessário para sua criação, este resultado final também estando na mesma linha de pensamento da equação ilustrada acima.

Para guardar os elementos, foi adotada uma lista de sequência de objetos*(arraylist)*, baseadas em uma equação de hash *valor % 10*, *valor* sendo determinado pela primeira letra do nome do elemento, e os objetos sendo listas encadeadas onde de fato guardamos os elementos e seus valores.

Também é relevante destacar o desafio enfrentado durante o processo de criação, que diz respeito aos elementos com dependências declaradas após o próprio elemento. Para abordar essa situação, fora desenvolvido um método semelhante ao principal, mas utilizado apenas quando um elemento necessário para a criação de outro é referenciado. É relevante considerar que esse método é recursivo, considerando a possibilidade de elementos que são dependências também possuírem suas próprias dependências.

Os Resultados

Os resultados de testes, como ilustrados na Figura 2, mostram o nome do arquivo testado e o resultado da equação, junto ao número de vezes que o método principal é utilizado[1] para adição de elementos e o número de vezes que o método recursivo é chamado, *CalculateHidrogenio* e *CalculateElemento* respectivamente.

Adicionalmente, é relevante considerar que a estrutura de armazenamento dos elementos foi projetada de forma a facilitar sua referência, proporcionando agilidade no processo de criação de elementos

[1] O método *CalculateHidrogenio é chamado apenas uma vez.*

que dependem uns dos outros. Essa abordagem contribui para uma execução mais eficiente do algoritmo, otimizando a resolução de dependências entre os elementos.

É pertinente também levar em consideração que a soma das utilizações do método principal e das chamadas do método recursivo é igual ao número de linhas do caso de teste que estes referenciam.

Nome do Arquivo	Valor de Hidrogênio	contCalculateHidrogenio	chamCalculateElemento
casoa5.txt	102384	56	0
casoa20.txt	800458	161	3
casoa40.txt	8259715	253	7
casoa60.txt	1641780	150	3
casoa80.txt	69922657113	317	31
casoa120.txt	721329020388177	325	70
casoa180.txt	8428729268811505	318	125
casoa240.txt	437216915509229458856327604	282	179
casoa280.txt	125827827947272728199292986125 21	240	222
casoa320.txt	958071316502331277458702860158 2	223	249
casoa360.txt	250818967539157804697656758842792 05	180	296
casoa400.txt	633752962893093757554665901597229376 66	153	327

Figura 2: Resultados dos casos de teste.

Percebe-se que, como esperado, o número de utilizações dos métodos cresce em relação ao número de linhas que os casos de teste possuem. No entanto, é mais significativa a proporcionalidade entre o número de utilizações do método principal e das chamadas do método recursivo, que tendem a se ajustar de acordo com a quantidade de vezes que um elemento possui dependências declaradas posteriormente.

Os casos iniciais tendem a favorecer a utilização do método principal, então percebemos que na medida em que avançamos os casos de teste começam a chamar cada vez mais o método recursivo, até que nos casos finais vemos uma tendência a favorecer a utilização do método recursivo para declaração de elementos. Acredita-se que isso ocorre devido a maneira como as listas dos casos de teste foram geradas, os casos finais tendem a favorecer a inclusão de elementos que referenciam dependências declaradas posteriormente, necessitando então da utilização do método recursivo.

Como considerações finais na implementação do algorítimo, acredita-se que seja possível a implementação de diferentes formas que possam ser otimizadas em relação a este, minimizando obstáculos que afetam sua performance.

A Conclusão

A vista do que foi explicitado, é possível concluir que a abstração do problema auxiliou na produção de sua solução. Ao implementar os métodos e as listas para o armazenamento de elementos, apresenta-se resultados interessantes, e consistentes em relação aos casos de teste.

Dessa forma, mostram o funcionamento do algorítimo em relação as chamadas dos métodos implementados tanto na versão principal quanto na sua versão recursiva, baseados em suas abstrações, também reitero as dificuldades declaradas, mesmo a solução fazendo conforme o pedido, ainda há espaço para melhorias.

Portanto, espera-se que o algorítimo contribua para a otimização dos custos de produção de ouro pelos alquimistas na Grande Convenção.