Siga o Dinheiro – Algoritmos e Estrutura de Dados II

João Miguel B. Meier, Matheus Bueno, João Pedro Krigger

Escola Politécnica — PUCRS 10 de abril de 2024

**Resumo**

Neste artigo, é apresentada a solução detalhada para um problema proposto na disciplina de Algoritmos e Estruturas de Dados II, durante o primeiro semestre do ano de 2024. O problema consiste na identificação de uma "trilha de dinheiro deixada por ladrões". Em termos mais técnicos, o desafio envolve o desenvolvimento de um algoritmo capaz de calcular a soma dos números encontrados dentro de uma matriz que inclui não apenas números, mas também caracteres especiais, tais como "/", "\", "-", "|", e "#". Cada caractere na matriz representa uma indicação do que está ocorrendo na trilha. Por exemplo, ao encontrar o caractere "#", o algoritmo deve interromper a contagem, indicando que o final da trilha foi alcançado. Esta abordagem visa fornecer uma solução eficiente para a identificação e soma dos valores ao longo da trilha, considerando os elementos numéricos e os caracteres que a definem.

# Introdução

Dentro do escopo deste projeto, identificamos dois grandes problemas que precisam ser resolvidos:

1. Desenvolvimento de um algoritmo para realizar a leitura de arquivos .txt e convertê-los em uma representação matricial.
2. Implementação de um algoritmo para percorrer a matriz, ajustando a rota da trilha de acordo com os caracteres encontrados e realizando a soma dos valores numéricos ao longo do percurso.

Iniciaremos descrevendo o contexto do primeiro problema. O objetivo é criar um algoritmo capaz de ler um arquivo .txt e armazená-lo em uma matriz. O tamanho da matriz a ser criada é indicado sempre no canto superior esquerdo do arquivo. É crucial garantir que os valores sejam armazenados corretamente para possibilitar o funcionamento adequado do algoritmo responsável por percorrer essa matriz.

No segundo problema, buscamos desenvolver um algoritmo que percorra a matriz, cujo tamanho pode variar, começando da esquerda e indo para a direita. O algoritmo deve seguir uma trajetória linear (percorrendo os caracteres "-" e "|"), mas deve mudar de direção ao encontrar os caracteres "/" ou "\", que representam uma "curva". Sempre que encontrar um número, o algoritmo deve somá-lo a um acumulador, respeitando a ordem em que foi encontrado, já que isso pode variar conforme a direção do percurso. Ao encontrar o caractere "#", o algoritmo deve encerrar sua execução, indicando o fim da trilha.

Para resolver este problema, optamos por dividir a solução em duas partes distintas, correspondentes às duas etapas identificadas na introdução.

Nesta primeira etapa da solução, descrevemos o desenvolvimento do algoritmo para a leitura de dados em arquivos .txt. Detalharemos a lógica utilizada para implementar esse algoritmo, assim como sua eficiência.

Na segunda parte da solução, descrevemos o algoritmo desenvolvido para percorrer a matriz e somar os valores encontrados ao longo do caminho. Explicamos a lógica utilizada na implementação desse algoritmo, bem como sua eficiência.

Por fim, apresentaremos os resultados obtidos por meio dos casos de teste fornecidos, bem como as conclusões derivadas durante o desenvolvimento deste trabalho.

# Solução para Leitura do Arquivo (.txt)

Ao abordarmos a solução para o problema descrito, focamos na criação de um algoritmo capaz de ler e interpretar arquivos .txt para formação de uma matriz conforme especificado.Começamos o algoritmo realizando a leitura das dimensões, em seguida vamos pulando as linhas vazias do arquivo até chegar na primeira linha em que há dados, assim que essa linha for encontrada é realizada a inicialização de uma matriz com espaços em branco, representando uma tela limpa para a inserção dos dados. Podemos ver este trecho do código abaixo:

*1 registro matriz[linhas][colunas] {*

*2 char valor*

*3 }*

*4*

*5 procedimento LER\_MATRIZ\_DE\_ARQUIVO*

*6 arquivo ← ABRIR(caminho\_do\_arquivo)*

*7 linhas, colunas ← LER\_LINHA(arquivo).separar\_e\_converter\_para\_int()*

*8 para i 0 até linhas - 1 faça*

*9 para j 0 até colunas - 1 faça*

*10 matriz[i][j].valor ' ' // Espaço como valor padrão*

*11 fim*

*12 fim*

*13 linha ← LER\_LINHA(arquivo)*

*14 enquanto linha ≠ NULL e linha.strip() = ' ' faça*

*15 linha ← LER\_LINHA(arquivo)*

*16 fim*

Posteriormente, o algoritmo avança para a etapa de leitura e processamento dos dados contidos no arquivo. À medida que os dados são lidos, eles são distribuídos pela matriz, respeitando as limitações de linhas e colunas previamente estabelecidas. Podemos ver este trecho do código abaixo:

1 enquanto linha ≠ NULL faça

2 se row ≥ linhas então

3 interromper

4 fim

5 dados ← linha.cortar\_ate(colunas + 1)

6 para col de 0 até colunas - 1 faça

7 char ← dados[col]

8 se char.strip() = ' ' então

9 matriz[row][col].valor ← ' '

10 senão

11 matriz[row][col].valor ← char

12 fim

13 fim

14 row ← row + 1

15 linha ← LER\_LINHA(arquivo)

16 fim

# Solução 1 - Percorrer a Matriz e Somar os Valores

Uma vez que a alternativa mais simples falha em quase metade dos casos, optamos por desenvolver uma abordagem mais cuidadosa, experimentando um truque bastante simples: se já tivermos os ex- poentes para certos números (obtidos, por exemplo, com o método anterior), podemos criar novos números a partir destes. Por exemplo, se sabemos que 2 = 21 30 e que 3 = 25 32, então podemos

÷ ÷

tentar produzir 6 através da combinação das duas representações: 6 = 2 3 = 21+5 30+2 = 26 32*.*

∗ ÷ ÷

Infelizmente, conferindo o resultado descobrimos que 26 32 = 64*/*9 = 7*.*1111 *. . .* e portanto não obtemos 6, e sim 7. Tabulando os resultados e prestando atenção nos valores, temos

÷

|  |  |  |  |
| --- | --- | --- | --- |
| Valor truncado | Fórmula | Valor exato | Excesso |
| 2 | 21 ÷ 30 | 2 | 0 |
| 3 | 25 ÷ 32 | 3.5555. . . | 0.5555. . . |
| 6 | 26 ÷ 32 | 7.1111. . . | 1.1111. . . |

O fenômeno parece bastante simples: como a representação de 3 carrega consigo um erro (ex- cesso) de 0.5555. . . , ao multiplicar 3 por 2 o erro também dobra, fazendo com que o valor obtido seja maior do que 6. Como se pode ver na tabela, o erro para 6 é de 1.1111. . . , exatamente o dobro de 0.5555. . . .

Desta maneira, podemos perceber que um mecanismo simples de geração de novos números a partir de outros que já são conhecidos teria, no mínimo, os seguintes problemas:

1. São necessários valores iniciais que são usados como “sementes”. Estes valores iniciais podem ser obtidos com o método anterior, por exemplo.
2. Os números sendo procurados devem ser decompostos para que se saiba que outros números devem ser multiplicados para obtê-los, da mesma forma que constatamos que 6 poderia ser produzido com 2 e 3. Esta decomposição é relativamente trabalhosa, e envolve números primos e vários tipos de testes.
3. É preciso conhecer a representação para os números primos, pois estes não podem ser obtidos multiplicando-se outros valores. Por exemplo, o valor 19 não poderá ser decomposto e sua representação tem que ser descoberta de forma independente.
4. É preciso ter controle do erro sendo transmitido entre os números, a fim de garantir que estamos chegando aos números corretos.

Por todos estes motivos optamos por não desenvolver esta solução, mas com o raciocínio feito até o momento podemos elaborar uma solução aparentemente mais complexa, mas que se revela mais simples quando analisada. Esta solução será apresentada a seguir.

*Veja como esta segunda solução se apoia sobre a primeira, e é descrita em um pouco mais de detalhe. Motivo: além de ser menos óbvia, ela serve de ponte para a terceira alternativa, que será a verdadeira resposta, e aqui precisamos deixar claro como funciona o mecanismo de propagação do erro. Isto é fundamental para o que vem a seguir. Note que esta solução é descartada com uma lista de motivos, para que ninguém pergunte porque ela não foi implementada.*

# Terceira solução

Se já sabemos como o erro se propaga quando a regra da multiplicação por 2 é usada, podemos pensar em uma alternativa: em vez de procurar individualmente cada número entre 2 e 100, podemos iniciar

uma busca que vai aplicando as regras que temos e controlando o erro envolvido, anotando os números encontrados no decorrer do processamento. É claro que para isto precisamos saber exatamente como se propaga o erro quando as regras são aplicadas, mas já conhecemos o caso da primeira regra, e agora só falta analisar o que acontece quando se aplica a segunda. A maneira mais simples é fazendo um teste, iniciando com 32 e dividindo duas vezes, anotando os erros:

|  |  |  |  |
| --- | --- | --- | --- |
| Valor truncado | Fórmula | Valor exato | Erro |
| 32 | 25 ÷ 30 | 32 | 0 |
| 10 | 25 ÷ 31 | 10.6666. . . | 0.6666. . . |
| 3 | 25 ÷ 32 | 3.5555. . . | 0.5555. . . |

É bastante claro que o erro não é simplesmente dividido por três a cada aplicação da regra, pois se assim fosse não haveria erro para 10 ou 3, já que o erro inicial era zero. Assim, imaginamos que o erro para a divisão seja calculado de forma mais complicada, e depois de alguns testes a regra obtida é a seguinte: dado um número *n* com erro *en,* ao aplicarmos a regra de divisão obteremos um novo número *n/*3 com erro dado por *en/*3 + (*n* mod 3)*/*3 = (*en* + *n* mod 3)*/*3. Embora esta regra pareça complicada, podemos fazer um teste com os números anteriores:

|  |  |  |
| --- | --- | --- |
| Valor | Valor exato | Erro |
| 32 | 32 | 0 |
| 10 | 10.6666. . . | 0*/*3 + (32 mod 3)*/*3 = 0 + 2*/*3 = 0*.*6666 *. . .* |
| 3 | 3.5555. . . | 0*.*6666 *. . ./*3 + (10 mod 3)*/*3 = 0*.*2222 *. . .* + 0*.*3333 *. . .* = 0*.*5555 *. . .* |

Agora que sabemos como o erro se propaga quando as regras são aplicadas, podemos facilmente propor uma solução que recebe um valor inicial e aplica repetidamente as regras dadas, mantendo controle do erro e anotando novos valores encontrados.

*Perceba que a fórmula apresentada não foi provada nem justificada de forma sólida! Isto deve ser melhorado para que um leitor entenda* ***por que*** *ela funciona!*

Este algoritmo pode funcionar de forma recursiva, da seguinte forma:

1 *registro tab*[100] {

2 *int achei* , *int exp2* , *int exp3*

3 }

4

5 **para** *i* 1 *até* 100 *faça*

←

6 *tab*[*i*].*achei FALSE*

←

7 **fim**

8 *achados* ← 0

9

10 **procedimento** PESQUISA(*int num* , *int exp2* , *int exp3* , *real err*)

11 // *Verificamos* **se** *o erro ultrapassa* 1.

12 // *Se ultrapassa estamos falando de outro número*...

13 **se** *err* ≥ 1 *então*

14 *num num* + 1

←

15 *err err* 1

← −

16 **fim**

17

18 // *Armazenamos os expoentes*...

19 **se** *num* ≤ 100 *e tab*[*num*].*achei* ← *FALSE então*

20 *tab*[*num*].*exp2 exp2*

←

21 *tab*[*num*].*exp3 exp3*

←

22 *tab*[*num*].*achei TRUE*

←

23 *achados achados* + 1

←

24 **fim**

25

26 // *Se ainda faltam números* , *reaplicar regras*...

27 **se** *achados <* 100 *então*

28 PESQUISA(*num* 2 , *exp2* + 1 , *exp3* , *err* 2)

∗ ∗

29 PESQUISA(*num* / 3 , *exp2* , *exp3* + 1 , (*err* + *n mod* 3) / 3)

30 **fim**

31 **fim**

32

33 // *Primeira chamada*

34 PESQUISA(1 , 0 , 0 , 0)

*A solução é quase essa, mas afinal se você correr pra implementar e funcionar de cara seria muito fácil, e isso não dá pra deixar. Num artigo de verdade eu colocaria a solução certinha, mas este é uma demonstração, por isso você pode pensar a respeito.*

*Se você está usando linguagens orientadas a objetos: dar uma lista de classes que você criou* ***não*** *é explicação pra nada! Falar das classes não basta, você deve esclarecer o que elas fazem! Ou seja, como trabalham, que algoritmos usam, como são interligadas. Pense como é pouco informativo: “. . . daí fiz umas classes A, B e C, que resolvem o problema.” Isso diz alguma coisa?*

A primeira chamada para a função que pesquisa os números deve ser feita com um número inicial, e é bastante simples determinar este número: deve ser *a*, o número fornecido inicialmente para o problema. Desta forma, passamos inicialmente o valor 1, com expoente 0 para 2 e 3 e com erro também zero. A partir daí o algoritmo é executado até que 100 valores estejam na tabela, e em seguida a tabela pode ser impressa no formato adequado.

Nesta versão de algoritmo não garantimos que estamos encontrando os menores expoentes pos- síveis, mas o algoritmo pode ser modificado para usar comparações com valores já existentes, ou manter listas de expoentes em vez de apenas uma tabela.

*Perceba que você não precisa ser a mãe do leitor, explicando cada linha do código e detalhe por detalhe. Se você acha que ja explicou o bastante e a idéia está clara, deixe que o leitor pense um pouco para entender como as coisas funcionam. Cuidado: isso* ***não*** *quer dizer que você deve deixá- lo perdido e no escuro. Você também não pode colocar cinco páginas de código e esperar que ele simplesmente leia e entenda.*

# Resultados

Depois de implementar o algoritmo acima em linguagem C e executá-lo num tempo de aproximada- mente meio segundo, obtivemos os seguintes resultados:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | = | 2^0 | / | 3^0 | 26 | = | 2^19 / 3^9 | | | 51 | = | 2^58 / 3^33 | | | 76 | = | 2^76 / 3^44 | | |
| 2 | = | 2^1 | / | 3^0 | 27 | = | 2^46 / 3^26 | | | 52 | = | 2^39 / 3^21 | | | 77 | = | 2^57 / 3^32 | | |
| 3 | = | 2^5 | / | 3^2 | 28 | = | 2^8 / 3^2 | | | 53 | = | 2^20 / 3^9 | | | 78 | = | 2^38 / 3^20 | | |
| 4 | = | 2^2 | / | 3^0 | 29 | = | 2^16 / 3^7 | | | 54 | = | 2^66 / 3^38 | | | 79 | = | 2^19 / 3^8 | | |
| 5 | = | 2^4 | / | 3^1 | 30 | = | 2^62 / 3^36 | | | 55 | = | 2^47 / 3^26 | | | 80 | = | 2^84 / 3^49 | | |
| 6 | = | 2^9 | / | 3^4 | 31 | = | 2^24 / 3^12 | | | 56 | = | 2^9 / 3^2 | | | 81 | = | 2^65 / 3^37 | | |
| 7 | = | 2^6 | / | 3^2 | 32 | = | 2^5 / 3^0 | | | 57 | = | 2^74 / 3^43 | | | 82 | = | 2^130 / 3^78 | | |
| 8 | = | 2^3 | / | 3^0 | 33 | = | 2^13 / 3^5 | | | 58 | = | 2^55 / 3^31 | | | 83 | = | 2^46 / 3^25 | | |
| 9 | = | 2^8 | / | 3^3 | 34 | = | 2^59 / 3^34 | | | 59 | = | 2^17 / 3^7 | | | 84 | = | 2^27 / 3^13 | | |
| 10 = | | 2^5 / 3^1 | | | 35 | = | 2^21 | / | 3^10 | 60 | = | 2^82 | / | 3^48 | 85 | = | 2^92 | / | 3^54 |
| 11 = | | 2^13 / 3^6 | | | 36 | = | 2^48 | / | 3^27 | 61 | = | 2^63 | / | 3^36 | 86 | = | 2^73 | / | 3^42 |
| 12 = | | 2^10 / 3^4 | | | 37 | = | 2^10 | / | 3^3 | 62 | = | 2^44 | / | 3^24 | 87 | = | 2^54 | / | 3^30 |
| 13 = | | 2^18 / 3^9 | | | 38 | = | 2^56 | / | 3^32 | 63 | = | 2^25 | / | 3^12 | 88 | = | 2^35 | / | 3^18 |
| 14 = | | 2^7 / 3^2 | | | 39 | = | 2^18 | / | 3^8 | 64 | = | 2^71 | / | 3^41 | 89 | = | 2^16 | / | 3^6 |
| 15 = | | 2^23 / 3^12 | | | 40 | = | 2^64 | / | 3^37 | 65 | = | 2^52 | / | 3^29 | 90 | = | 2^81 | / | 3^47 |
| 16 = | | 2^4 / 3^0 | | | 41 | = | 2^45 | / | 3^25 | 66 | = | 2^33 | / | 3^17 | 91 | = | 2^146 / 3^88 | | |
| 17 = | | 2^20 / 3^10 | | | 42 | = | 2^26 | / | 3^13 | 67 | = | 2^14 | / | 3^5 | 92 | = | 2^62 / 3^35 | | |
| 18 = | | 2^9 / 3^3 | | | 43 | = | 2^53 | / | 3^30 | 68 | = | 2^79 | / | 3^46 | 93 | = | 2^43 / 3^23 | | |
| 19 = | | 2^17 / 3^8 | | | 44 | = | 2^15 | / | 3^6 | 69 | = | 2^60 | / | 3^34 | 94 | = | 2^24 / 3^11 | | |
| 20 = | | 2^44 / 3^25 | | | 45 | = | 2^80 | / | 3^47 | 70 | = | 2^41 | / | 3^22 | 95 | = | 2^89 / 3^52 | | |
| 21 = | | 2^6 / 3^1 | | | 46 | = | 2^42 | / | 3^23 | 71 | = | 2^22 | / | 3^10 | 96 | = | 2^154 / 3^93 | | |
| 22 = | | 2^14 / 3^6 | | | 47 | = | 2^23 | / | 3^11 | 72 | = | 2^68 | / | 3^39 | 97 | = | 2^70 / 3^40 | | |
| 23 = | | 2^22 / 3^11 | | | 48 | = | 2^69 | / | 3^40 | 73 | = | 2^49 | / | 3^27 | 98 | = | 2^51 / 3^28 | | |
| 24 = | | 2^30 / 3^16 | | | 49 | = | 2^31 | / | 3^16 | 74 | = | 2^30 | / | 3^15 | 99 | = | 2^32 / 3^16 | | |
| 25 | = | 2^11 | | / 3^4 | 50 | = | 2^12 | / | 3^4 | 75 | = | 2^95 | / | 3^56 | 100 = 2^97 / 3^57 | | | | |

Após a obtenção destes valores, estes foram testados com ferramentas capazes de lidar com os grandes números envolvidos, e todos os resultados foram confirmados corretos.

*Por favor, não esqueça de apresentar os resultados, afinal deve ser isso que seu leitor quer ver!*

# Conclusões

As abordagens iniciais, mesmo oferecendo apenas resultados parciais, contribuíram bastante para o entendimento do problema e abriram caminho para a solução definitiva. Esta se mostrou bastante simples e eficiente, embora tenha exigido um estudo de como o erro se propaga quando as regras são aplicadas a novos números. A solução implementada não se preocupou em achar sempre os menores expoentes possíveis, mas mesmo assim ela foi bastante eficiente e os expoentes encontrados não são excessivamente grandes (foram feitas comparações com o primeiro algoritmo apresentado, e as diferenças foram poucas). Além disso, a solução recursiva adotada foi razoavelmente elegante e clara, não precisando de uma implementação complicada.

Embora não tenham sido mostrados neste artigo, foram feitos testes com outros valores iniciais (*a* = 1) e para números indo até bem mais do que 100, o algoritmo também funcionou bem.

/

Acreditamos ter desenvolvido uma solução interessante e barata a um problema relativamente complexo, já que em uma linguagem de programação usual não existe suporte direto para tratar com os números envolvidos, e conseguimos determinar resultados corretos que envolvem grandes núme- ros.

*Suas conclusões não precisam ser geniais nem mudar o mundo. Veja que a parte inteligente do artigo já foi feita, e veio antes! Agora é a hora de fechar e dizer o que você acha que poderia vir depois, etc etc. E se você está pensando em dizer coisas como “Provavelmente existem soluções*

*melhores”, então diga ao menos o que você pensa que pode melhorar (por exemplo, neste artigo eu gostaria de evitar o uso de números em ponto flutuante e preferiria trocar por algum tipo de controle com inteiros). Se você não der opções e só disser que dá pra fazer melhor, está se diminundo de graça. Pra que isso?*

*A bibliografia abaixo não foi usada neste artigo, e serve apenas de exemplo para a formatação.*

*Outros exemplos estão na página da biblioteca da PUCRS.*

# Referências

1. Brassard, G.; Bratley, P: “**Fundamentals of algorithmics**”. Prentice Hall, Englewood Cliffs, New Jersey, 1996.
2. Cormen, T. H.; Leiserson, E. C.; Rivest, R. L.: **“Introduction to Algorithms”**. Mc-Graw Hill Book Co., The MIT Electrical Engineering and Computer Science Series, Cambridge, 1990.
3. Graham, R. L.; Knuth, D. E.; Patashnik, O.: **“Concrete mathematics”**. Addison-Wesley, Rea- ding, 1989.
4. Moret, B. M.; Shapiro, H. D.: “**Algorithms from P to NP: design and efficiency**”. Addison- Wesley, Reading, 1990.
5. Lueker, G. S.: “*Some techniques for solving recurrences*”. Computing Surveys, Vol. 12, no. 4, dezembro 1980.
6. Rawlins, Gregory J. E.: “**Compared to what? An introduction to the analysis of algorithms**”, Computer Science Press, New York, 1992.
   1. Para ajudar na missão ingrata de fazer um trabalho agradável para a leitura, este *checklist* pode ajudar:

Q Artigos não tem capa.

Q Coloquei título neste trabalho? Coloquei meu nome?

Q Usei um espaçamento legal e tamanho de fonte decente ou tem só cinco linhas por página e as letras parecem manchetes?

Q O trabalho tem uma introdução, para que se entenda do que estou falando? Eu contei o que ia resolver?

Q O desenvolvimento é coerente, ou parece que recortei trechos de jornal e de meus colegas e grudei tudo?

Q Dá pra entender minhas explicações ou tem que ter poderes paranormais?

Q Tem o mínimo de código possível? Não dá pra ter ainda menos? Dá pra não ter nenhum? Se não tem nenhum, será que deveria ter um pouco mostrando as partes mais importantes? (Geralmente deveria)

Q Se tem código, eu fico falando durante páginas e páginas entorpecentes e sem fim pra explicar o que ele faz, linha por linha? Posso ser **claro** sem ser **chato**?

Q Se o trabalho pedia a resposta para algum problema, será que esqueci de dar o resultado?

Tinha que dar vários exemplos e testes?

Q Usei figuras pra esclarecer coisas que eram confusas, como aquelas listas que usei e agora eu mesmo mal consigo entender? Como vou querer que outra pessoa entenda?

Q Minhas figuras são claras? Mostram coisas acontecendo passo a passo ou estão jogadas no papel? As legendas são descritivas?

Q Os dados ficam melhor se forem apresentados em tabelas? Tenho gráficos de tempo e desempenho, se preciso? Com legendas, eixos, unidades, ou está uma bagunça feita no Excel?

Q Lembrei de colocar minhas conclusões ao final do trabalho?

Q Coloque as minhas fontes de informação na bibliografia?