Escrevendo seu relatório

João B. Oliveira*
Escola Politécnica — PUCRS

11 de maio de 2021

Resumo

O abstract ou resumo deve dar uma descrição curta (um ou dois parágrafos) do que será tratado no texto, para que o leitor possa decidir se o assunto interessa ou não. Descreva o problema, conte da abordagem usada e a solução que você deu, fale dos resultados obtidos e mostre que o trabalho é interessante. Note também que a data está em português, com o mês em minúscula. Isso é a língua portuguesa e pode ser que seu editor de texto não saiba disso.

Para quem quer colocar dois abstracts, um em inglês e outro em português: isso é mesmo necessário para um trabalho de disciplina?

Este artigo descreve alternativas de solução para o primeiro problema proposto na disciplina de ..., que trata da representação dos inteiros de 2 até 100 na forma $2^i \div 3^j$. São apresentadas três possibilidades de solução para o problema e sua eficiência é analisada. Em seguida as representações obtidas para os números são mostradas.

Note que seu trabalho pode pedir mais que isso! Você pode ter que oferecer exemplos de uso, casos de teste, e outros itens que não foram pedidos neste trabalho!

Introdução

O papel da introdução é descrever o problema que vai ser resolvido, de onde ele veio, qual o seu contexto e se já sabemos algo a respeito dele (se alguém já tentou resolver, até onde foi, etc.) e depois descrever os passos que serão apresentados para a resolução, mostrando uma espécie de roteiro do que vem a seguir. Não esqueça de que o leitor deve entender o problema pra entender sua solução!

Se fizer parte do problema, explique o que vai ser a entrada.
Vai ler arquivo? Como é o formato dele? O que significa?
Vai guardar onde e em que estrutura?
Mostre pseudocódigo e figura da estrutura, se for adequado!

No contexto de ..., o problema que investigaremos pode ser resumido assim: dado um número $a = 1$, outros números podem ser produzidos seguindo-se as duas regras:

1. Multiplicando por 2 um número já produzido.
2. Dividindo por 3 um número já produzido e descartando a parte fracionária (divisão inteira).

Por exemplo, seguindo estas regras temos que

$$10 = 1 \times 2 \times 2 \times 2 \times 2 \times 2 \div 3 = 32 \div 3.$$

O problema a ser resolvido é determinar a representação de todos os inteiros de 2 até 100 seguindo apenas estas regras. Para a saída, os números devem ser apresentados em ordem

*joao.souza@pucrs.br

crescente, apresentando-se quantas vezes tivemos de usar cada regra. Segundo o enunciado do trabalho, a linha da saída para 10 seria a seguinte:

```
10 = 2^5 / 3^1
```

O enunciado também informa que os resultados possíveis não são únicos, muitos números podem ser obtidos em mais de uma forma, portanto deve-se dar preferência aos expoentes menores. Algumas regras e considerações adicionais são:

1. Não é permitido usar inteiros especiais, como `long`, `BigInt` e outros inteiros extra-longos. O maior tipo inteiro deve ter 32 bits.

2. É permitido usar `float, real` e `double` em tarefas auxiliares, mas não é permitido calcular potências com eles.

3. Se um número não precisa ser dividido por 3, pode-se apresentar `3^0` na saída.

Para resolver o problema proposto, analisaremos três possíveis alternativas de solução, bem como suas características, optando por uma soluçao bastante eficiente ainda que pouco intuitiva. Em seguida os resultados obtidos serão apresentados, bem como as conclusões obtidas no decorrer do trabalho.

Este texto é todo escrito no plural: fizemos, faremos, analisaremos, apresentaremos. ***Nunca*** *usamos o singular (fiz, analisei, pensei, etc), mesmo que você esteja trabalhando sozinho. Outra possibilidade, melhor ainda, é usar termos passivos: faz-se, analiza-se, demonstra-se, etc.*

O texto é em espaço simples, pois é muito chato ler um trabalho em espaço maior: achamos que quem escreveu precisava encher papel. Sobre o tamanho de letra, recomenda-se 10, 11 ou 12. Este texto foi originalmente feito em tamanho 12 porque iria ser reduzido.

Coisinhas de português: não existe espaço antes de uma vírgula, depois de dois pontos sempre vêm minúscula e coloca-se espaço em branco antes de abrir um par de parênteses. Por dentro dos parênteses não se coloca espaço, o texto vem colado neles. Procure estes detalhes no texto!

Primeira solução

Como este problema não tem entrada de dados, pulamos a parte de contar como ler a entrada, o que fazer com ela, etc.

Depois de considerar o problema, podemos perceber que o valor inicial $a = 1$ serve apenas para fornecer um número inicial sobre o qual podem ser aplicadas as regras para gerar outros números, e portanto seu valor não tem importância especial. Com esta constatação, e sabendo que em um inteiro de 32 bits podemos armazenar números até 2^{31} ou até 3^{20} (obtivemos estes dados através de testes), podemos propor uma solução bastante simples: bastaria criar frações $2^i/3^j$ com todas as possibilidades de expoentes para 2^i (ou seja, $0 \leq i \leq 31$) e para 3^j (ou seja, $0 \leq j \leq 20$), verificando quais números são gerados. O total de números a serem testados é de apenas $32 * 21 = 672$, portanto o algoritmo deve ser executado com grande velocidade. Podemos criar uma tabela para manter os resultados obtidos e seus expoentes e a cada resultado encontrado seus expoentes são armazenados. Um algoritmo implementando esta idéia seria parecido com este:

```
struct {
  int achei, exp2, exp3;
} tab[101];
```

```
para i de 1 a 100 tab[i].achei = FALSE;

para i de 0 a 31
  para j de 0 a 20 {
    num = 2^i / 3^j;
    // Se está na faixa desejada e ainda nao encontramos...
    if (num <= 100 e tab[num].achei == FALSE) {
      tab[num].achei = TRUE;
      tab[num].exp2 = i;
      tab[num].exp3 = j;
    }
  }
```

Ao final basta imprimir a tabela, colocando-a no formato adequado de saída (este algoritmo é simples e não será mostrado aqui).

Você não deve contar cada linha do código ("daí incrementa o i, depois faz $j = i + 1$, depois..."), mas sim *como* ele foi pensado e vai funcionar!

Também é interessante perceber que a solução apresentada até o momento acha os menores expoentes possíveis, mas precisa manter o controle de que números já foram encontrados, para não escrever expoentes maiores sobre valores já armazenados. Uma maneira muito simples de descartar este tipo de controle é fazendo o loop externo **decrescente**, garantindo assim que os menores expoentes serão encontrados por último, e permitindo a escrita sobre valores já encontrados. Esta solução seria assim:

```
struct {
  int exp2, exp3;
} tab[101];

para i de 31 a 0
  para j de 20 a 0 {
    num = 2^i / 3^j;
    if (num <= 100) {
      tab[num].exp2 = i;
      tab[num].exp3 = j;
    }
  }
```

Esta idéia é bastante simples e eficiente, porém um teste mostra que ela acha apenas 53 dos 99 números desejados. Com isto podemos concluir que os outros 46 valores devem ser obtidos com expoentes maiores, e os números envolvidos não poderão ser representados em inteiros de 32 bits.

Esta seção dá uma idéia inicial que já faz alguma coisa, analisa-a de forma rápida e diz porque ela não serve. Veja que não foi usada uma linguagem de programação com todos os detalhezinhos! Os algoritmos estão em uma linguagem simplificada, que permite que se entenda rapidamente o que deve ser feito, sem excesso de detalhes. Não é Java, nem C++, nem é uma impressão de tela.

Perceba também que o código não é cheio de explicações óbvias (esta linha incrementa o contador, esta linha calcula o número), mas é clara o bastante para que outra pessoa entenda o que é feito e reproduza o algoritmo em sua linguagem preferida. Uma falha: terminamos dizendo que achamos 53 dos 99 números, mas sem dizer quais foram nem falar do tempo necessário para encontrá-los. ***Moral****: quando terminar de falar de uma solução, apresente os resultados, comente o tempo e as (possíveis) falhas!!*

Segunda solução

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 expoentes para certos números (obtidos, por exemplo, com o método anterior), podemos criar novos números a partir destes. Por exemplo, se sabemos que $2 = 2^1 \div 3^0$ e que $3 = 2^5 \div 3^2$, então podemos tentar produzir 6 através da combinação das duas representações: $6 = 2 * 3 = 2^{1+5} \div 3^{0+2} = 2^6 \div 3^2$.

Infelizmente, conferindo o resultado descobrimos que $2^6 \div 3^2 = 64/9 = 7.1111\ldots$ e portanto não obtemos 6, e sim 7. Tabulando os resultados e prestando atenção nos valores, temos

Valor truncado	Fórmula	Valor exato	Excesso
2	$2^1 \div 3^0$	2	0
3	$2^5 \div 3^2$	3.5555...	0.5555...
6	$2^6 \div 3^2$	7.1111...	1.1111...

O fenômeno parece bastante simples: como a representação de 3 carrega consigo um erro (em excesso) 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 da primeira solução 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 simples para números pequenos como os que temos, mas é trabalhosa para números grandes e envolve números primos e vários tipos de testes.

3. É sempre 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. Por outro lado, alguns dos primos podem ser produzidos com números já conhecidos por causa do excesso. Por exemplo, tentar produzir 6 deu um resultado válido para 7.

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 inicia em $a = 1$ e vai aplicando as regras que temos e ao mesmo tempo controlando o erro envolvido, anotando os números encontrados no decorrer do processamento. Em outras palavras, ela sai procurando para ver o que encontra.

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 (multiplicação por 2) e agora só falta analisar o que acontece quando se aplica a segunda (dividir por 3). A maneira mais simples é fazendo um teste, iniciando com 32 e dividindo duas vezes por 3, anotando os erros:

Valor truncado	Fórmula	Valor exato	Erro
32	$2^5 \div 3^0$	32	0
10	$2^5 \div 3^1$	10.6666...	0.6666...
3	$2^5 \div 3^2$	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 e_n, ao aplicarmos a regra de divisão obteremos um novo número $n/3$ com erro dado por $e_n/3 + (n \bmod 3)/3 = (e_n + n \bmod 3)/3$. Embora esta regra pareça complicada, podemos fazer um teste com os números anteriores:

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!

Valor	Valor exato	Erro
32	32	0
10	10.6666...	$0/3 + (32 \bmod 3)/3 = 0 + 2/3 = 0.6666\ldots$
3	3.5555...	$0.6666\ldots/3 + (10 \bmod 3)/3 = 0.2222\ldots + 0.3333\ldots = 0.5555\ldots$

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.

Este algoritmo pode funcionar de forma recursiva, apresentado a seguir em duas partes:

```
// Parte 1: a tabela para guardar os resultados à medida que
// forem sendo encontrados. Precisamos também de um contador de
// quantos números foram achados.
struct {
  int achei, exp2, exp3;
} tab[101];
para i de 1 a 100 tab[i].achei = FALSE;
int achados = 0;
```

Em seguida o algoritmo para iniciar em um número `num` feito com expoentes `exp2` e `exp3` e erro `err` e depois aplicando as duas regras para produzir novos números:

```
// Aplicacao de regras
pesquisa(int num, int exp2, int exp3, real err) {
  // Verificamos se o erro ultrapassa 1.
  // Se ultrapassa estamos falando do próximo numero
  se (err >= 1) {
    num = num + 1;
    err = err - 1;
  }
  // Armazenamos os expoentes se é um resultado novo
  se (num <= 100 e tab[num].achei == FALSE) {
    tab[num].exp2 = exp2;
    tab[num].exp3 = exp3;
    tab[num].achei = TRUE;
    achados++;
  }
  // Se ainda faltam numeros, reaplicar regras...
  se (achados < 100) {
    pesquisa(num * 2, exp2 + 1, exp3, err * 2);
    pesquisa(num / 3, exp2, exp3 + 1, (err + num mod 3) / 3);
    }
}

// Primeira chamada
pesquisa(1, 0, 0, 0);
```

A solução é quase essa, mas alguns detalhes foram escondidos. Num artigo de verdade eu colocaria a solução certinha, mas este é uma demonstração, por isso você pode pensar a respeito pra descobrir o que está faltando.

Se você está usando linguagens orientadas a objetos: dar uma lista de classes que você criou ***não*** *é explicação! Falar das classes não basta, você deve esclarecer o que elas fazem e como elas funcionam para construir a sua solução! 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 possíveis, 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 ficar 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. Dizer "a explicação está a seguir" e dar pedaços de programa como se explicasse alguma coisa é uma péssima ideia.*

Resultados

Depois de implementar o algoritmo acima em linguagem C e executá-lo num tempo de aproximadamente 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. Embora os resultados estejam corretos, para alguns deles os expoentes encontrados não são os menores possíveis:

Valor encontrado	Opção melhor
42 = 2^26 / 3^13	42 = 2^7 / 3^1
64 = 2^71 / 3^41	64 = 2^6 / 3^0
75 = 2^95 / 3^56	75 = 2^11 / 3^3
85 = 2^92 / 3^54	85 = 2^8 / 3^1

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 ($a \neq 1$) e para números indo até bem mais do que 100, o algoritmo também funcionou bem.

Acreditamos ter desenvolvido uma solução interessante e rápida 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 números.

Suas conclusões não precisam ser geniais nem mudar o mundo. Veja que a parte inteligente do artigo já foi feita. 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, mas não sei se é uma boa ideia). Se você não der opções e só disser que dá pra fazer melhor, está se diminuindo 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"**. McGraw 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, Reading, 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.

Outros detalhes

(Coisas que não entram num artigo, mas que vou escrever aqui porque deve ser ditas em algum lugar).

1. Em primeiríssimo lugar: você está na universidade e portanto deve ter ido à escola por uns doze anos. Você teve aulas de português por cerca de sete desses anos, várias vezes por semana e ainda prestou prova para o vestibular. Seu editor de texto, por outro lado, não teve aula nenhuma e provavelmente foi (meio) adaptado do inglês. Mostre um pouco de respeito por si mesmo e não acredite em tudo o que ele diz pra você.

2. Acentos ainda existem em português e deverão continuar por um tempo. Use-os.

3. Note a consistência no que está escrito: o valor inicial a, por exemplo, está sempre escrito da mesma forma, em itálico, para que o leitor não encontre cinco tipos de letra a diferentes e tenha de lembrar o que cada uma significa. Isto mostra que o autor teve cuidado ao escrever.

4. Note que os fontes são sempre consistentes: texto em Times, matemática em *itálico*, algoritmos em `typewriter`. Não há fontes maiores ou menores, nem paragrafos com fonte diferente, margem ou espaçamento maiores ou menores do que os outros.

5. Veja também que as potências estão sempre escritas corretamente no texto: x^y. Não se usam notações como $x\hat{}y$ porque elas são feias. Esta notação só é usada quando se fala do formato de saída, pois ele é exigido desta forma. Da mesma forma, quando são necessários subscritos eles devem ser escritos corretamente: A_i, b_{ij} etc. Escrever Aj ou bij é um mau sinal.

6. Se algo está nas referências deve estar citado no texto, e se está citado no texto tem de aparecer nas referências!! A maneira correta de citar uma referência é: "Segundo Fulano [3] e Beltrano [4], o algoritmo precisa..."

7. Tenha cuidado com seu texto, ele deve transmitir a impressão de que foi escrito com todo o cuidado do mundo. Dedique tempo a ele e tente deixá-lo melhor, pois sua reputação (e nota) dependem disso.

8. Este artigo foi escrito usando LyX, um editor de textos fazendo uso de TeX e LaTeX, que são sem dúvida os melhores formatadores de texto do mundo. Tudo isto é software livre rodando sob Linux. Aprenda a usá-los, vai fazer bem. Outras opções são TeXmaker, TeXnicCenter, Texstudio, TeXworks, Papeeria, Overleaf, Authorea e Kile. Algumas são na Web, outras você instala.

9. Para ajudar na missão ingrata de fazer um trabalho agradável para a leitura, este *checklist* pode ajudar:

   - ☐ Artigos não tem capa.
   - ☐ Coloquei título neste trabalho? Coloquei meu nome?
   - ☐ Usei um espaçamento legal e tamanho de fonte decente ou tem só cinco linhas por página e as letras parecem manchetes?
   - ☐ O trabalho tem uma introdução, para que se entenda do que estou falando? Eu contei o que ia resolver?
   - ☐ O desenvolvimento é coerente, ou parece que recortei trechos de jornal e de meus colegas e grudei tudo?
   - ☐ Dá pra entender minhas explicações ou tem que ter poderes paranormais?

- □ 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)
- □ 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**?
- □ Se o trabalho pedia a resposta para algum problema, será que esqueci de dar o resultado? Tinha que dar vários exemplos e testes?
- □ 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?
- □ Minhas figuras são claras? Mostram coisas acontecendo passo a passo ou estão jogadas no papel? As legendas são descritivas? Tem legendas?
- □ 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?
- □ Lembrei de colocar minhas conclusões ao final do trabalho?
- □ Coloque as minhas fontes de informação na bibliografia?