TP3

Wilson Moreira Tavares

December 15, 2015

1 Introdução

O problema a ser resolvido traz o seguinte cenário: o Departamento de Nutrição solicitou ao DCC um programa que ajude na criação de um plano alimentar para pessoas que irão participar de um evento sobre alimentação saudável.

O programa executa a seguinte tarefa, é dado o número de calorias totais que podem ser consumidas por uma pessoa e o número de calorias de cada alimento entre os disponíveis para consumo. O programa deve saber se o valor total de calorias que deve ser consumido pode ser alcançado através da soma de alguns valores dentre os alimentos disponíveis.

O algoritmo deve, portanto, responder se existe uma combinação entre os valores de calorias disponíveis que somados resultam exatamente no número de calorias que um dos pacientes pode consumir.

2 NP-completo

O problema a ser resolvido de acordo com a descrição acima é conhecido e tem por nome Subset Sum. Nesta seção será provado que o mesmo pertence à classe NP-Completo e não possui um algoritmo polinomial que calcula a sua solução. Para provar que um problema pertence a NP-Completo, devemos seguir os seguintes passos:

- Provar que o problema pertence a NP, isso pode ser feito de duas formas:
  1. Apresentar um algoritmo não-determinista que calcula a solução do algoritmo em tempo polinomial.
  2. Apresentar um algoritmo determinista que verifica se uma solução está correta em tempo polinomial.
- Estando a condição acima satisfeita, é necessário provar agora que o problema é NP-Completo, para isso, devemos reduzir um problema conhecidamente NP-Completo ao problema em questão. É importante prezar pela ordem da transformação proposta. Após feita a transformação, levamos em conta a seguinte proposição:

  Se $Pdificil \rightarrow Pfacil$, então Pfacil não é fácil.

Abaixo se encontra a prova de que o problema Subset Sum é NP-Completo, seguindos as diretrizes citadas acima:

Passo 1: Para um subconjunto A, pertencente a B, temos uma soma S? Existe um algoritmo polinomial que verifica se a solução está correta, o mesmo é apresentado abaixo.

```
-----------------------------------------------------------------
Algoritmo determinista de verificação em tempo polinomial
-----------------------------------------------------------------
Para todo item Ai pertencente a A, faça
    Soma += Ai
Fim Para
Se Soma = S
    Resultado = sim
Senao
    Resultado = não
-----------------------------------------------------------------
```

Desta forma, é sabido que o algoritmo pertence a NP.

Passo 2: Para realizar o passo 2, será escolhido o problema da mochila, conhecido como Knapsack problem.

O problema da mochila consiste em um cenário onde existe um mochila de capacidade M e N itens que podem ser colocados na mochila. Os itens possuem duas propriedades que são exploradas pelo problema, peso e valor. O problema diz respeito a escolher os itens de forma a maximizar o valor dos itens que estão na mochila sem que os mesmos extrapolem a capacidade da mesma. Este problema pertence à classe dos problemas NP-Completo.

Para transformar o mesmo no problema Subset Sum, levamos em consideração a seguinte transformação:

Na entrada de um problema do tipo knapsack, tratamos tanto o valor bi de um item como o peso ai com o mesmo peso ai, desta forma, valor e peso apresentam a mesma responsabilidade sobre o resultado. Com essa mudança, a podemos verificar se os valores do subconjunto A estão abaixo do valor mochila, assim como verificar se os mesmos são iguais à capacidade da mochila, se os mesmos se igualarem ao valor da mochila, temos a resposta sim, em caso contrário, a resposta é não. É evidente que a redução proposta é feita em tempo polinomial, portanto, fica provado que o problema Subset Sum é NP-Completo.

3 Solução

A solução para resolução do problema foi usar uma estrutura de árvore de possibilidades. Onde os filhos representam o ato de incluir e não incluir o alimento na refeição. A altura da árvore é relativa ao número de alimentos do conjunto, ou seja, a árvore possui altura X, onde X é o número de alimentos que existem na lista que foi fornecida.

A estrutura abordada trabalha todas as possibilidades de soma entre os valores fornecidos, portanto, através dessa estrutura é possível fornecer a resposta desejada. Um detalhe desagradável e que pode ser observado na imagem abaixo é o fato de que um valor é repetido sempre no seu filho à direita, aumentando o número de itens na árvore de possibilidades.

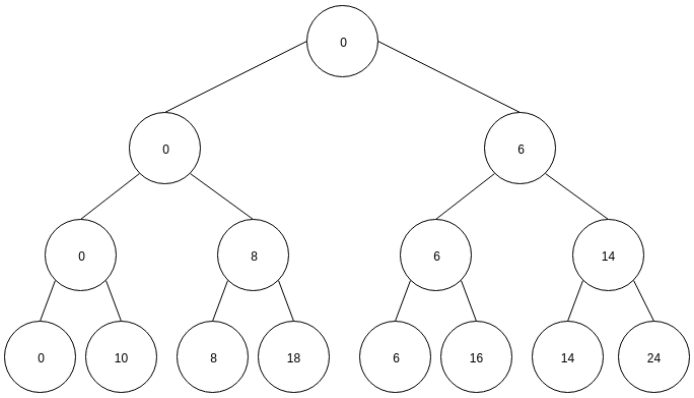

Figure 1: Exemplo para um vetor com os elementos 6, 8 e 10. Onde o filho à esquerda não inclui o alimento e o filho da direita inclui.

Para melhorar tal cenário foi proposta uma mudança na árvore, onde não haveriam valores repetidos[1], mas essa mudança não foi feita por dificuldade de implementação. A forma com a qual essa árvore que foi proposta foi pensada é de que o apontador *naoCome* da árvore de possibilidades na verdade apontaria para o ato de comer o segundo alimento, após o próximo que estaria presente na lista. A árvore ficaria disposta da seguinte forma:

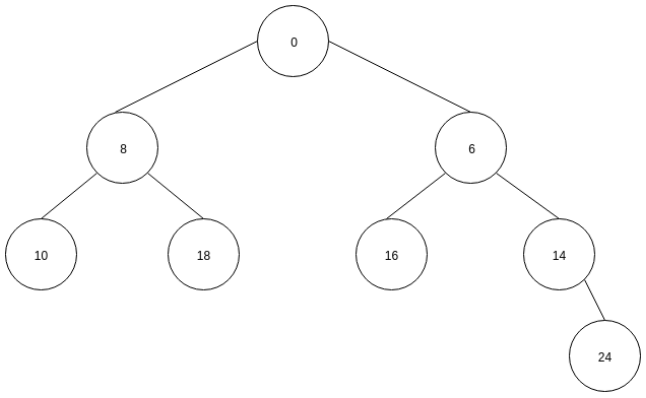

Figure 2: Exemplo para um vetor com os mesmos elementos 6, 8 e 10.

[1] Nesse caso, valores repetidos se referem à repetição de possibilidades, não a valores iguais.

Vemos abaixo um pseudocódigo onde é possível observar a forma de funcionamento do algoritmo que foi implementado:

```
-----------------------------------------------------------------
criaArvore (arvore, arranjo calorias, valorPai,
    nivel, limite, come, numAlimentos, achou)
-----------------------------------------------------------------
criaArvore
{
    Se come = 0
        arvore.valor = valorPai
    Senao Se come = 1
        arvore = valorPai + calorias[nivel]
    Fim Se

    Se arvore.valor = limite
        achou = 1
    Fim Se

    Se nivel<numAlimentos e arvore.valor < limite
        criaArvore (arvore.naoCome, calorias, arvore.valor,
            nivel+1, limite, 0, numAlimentos, achou);
        criaArvore (arvore.come, calorias, arvore.valor,
            nivel+1, limite, 1, numAlimentos, achou);
    Fim Se
}

-----------------------------------------------------------------
```

4 Paralelismo

Para a solução supracitada foi usado um paralelismo de dados, onde as instâncias foram paralelizadas, sendo distribuídas entre as threads criadas, de quantidade definida de acordo com o valor passado como parâmetro.

Para que essa paralelização seja feita da forma correta, é necessário salvar os dados de leitura das instâncias, para que nada se perca ou seja lido de forma assíncrona. Por esse motivo, foi criada uma estrutura de dados onde são salvas três informações:

- O valor da soma S de calorias que deve ser encontrado.
- Um vetor de inteiros que armazena as calorias dos alimentos disponíveis.
- O resultado daquela instâcia.

Para realizar a paralelização dessa forma, foi criado um vetor de threads com T posições[2], foi criada uma estrutura de laços onde o programa executa as

[2] T é o número de threads passado por parâmetro

instâncias por lotes de threads. Suponha que o programa possua 10 instâncas e quatro threads, a execução se dará da seguinte forma:

```
 ---------------------------------
|Execução | Instâncias computadas |
|---------------------------------|
|    1    |      1, 2, 3, 4       |
|    2    |      5, 6, 7, 8       |
|    3    |         9, 10         |
 ---------------------------------
```

O funcionamento da paralelização se dá da seguinte forma:

```
-------------------------------------------------------------------
Pseudocódigo de alocação das threads
-------------------------------------------------------------------
j = 0
Para cada instância Ai faça
    Se j < numThreads
        Cria a thread para a instância i na posição j
    Fim Se
    Senão Se j == numThreads
        Espera todas as threads terminarem
        j = 0
    Fim Senão
    Se i = numInstancias
        Espera todas as threads até j terminarem
    Fim Se
    j++
Fim Para
-------------------------------------------------------------------
```

5 Análise teórica de custo assintótico

5.1 Tempo

Na execução do algoritmo proposto uma árvore binária de altura N. Uma árvore binária de altura N, possui o seguinte número de nós:

$$f(n) = \sum_{x=0}^{N} 2^x$$

.

Portanto, a complexidade de espaço é:

$$O(f(n))$$

5.2 Espaço

O custo da execução em termos de tempo se equipara ao custo de espaço em termos de notação, tendo em vista o número de nós, o custo de popular a árvore é de:

$$O(g(n))$$

onde

$$g(n) = \sum_{x=0}^{N} 2^x$$

.

O custo para verificar qual é a solução não tem impacto relevante no tempo final de execução do algoritmo, tendo em vista que a solução é verificada no momento em que a árvore é construída.

6 Análise prática de custo assintótico

Para os testes, foi utilizado o seguinte padrão de crescimento, foi fixado um limite, que é o número de calorias, e o que foi avaliado foi o tamanho dos itens, que é o que define até onde a árvore de possibilidades chegará. Os valores do eixo X dizem respeito ao tamanho máximo das calorias que serão computadas. Exemplo: Para uma instância de soma requerida 500 e caloria máxima dos alimentos 250, no eixo X o tempo de execução seria marcado no ponto 50. Para 300 no ponto 40, e assim por diante. O número de threads foi fixado em 100, que é o número de instâncias da entrada.

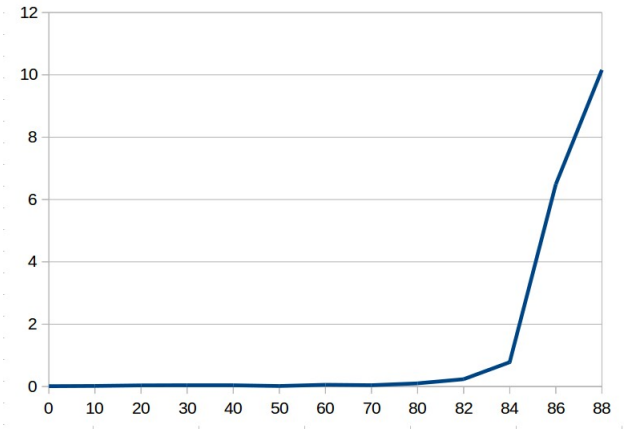

Figure 3: O eixo X representa o tamanho das entradas e o eixo Y representa o tempo de execução em segundos