TP3: Alimentação Saudável

Reuben Nascimento Morais

1 de dezembro de 2015

1 Introdução

O objectivo desse trabalho prático é implementar um programa que encontre combinações ótimas de alimentos para criar uma dieta com a quantidade adequada de calorias. Dada uma recomendação de um(a) nutricionista de quantas calorias consumir em cada refeição, e uma lista de alimentos, é possível montar uma refeição que atinge exatamente a quantidade desejada de calorias?

2 Análise de complexidade computacional do problema

Esse problema é conhecido na literatura como o problema da SOMA DE SUBCONJUNTOS.

Definição 2.1 (SOMA DE SUBCONJUNTOS)**.** Dado um conjunto de inteiros N e um inteiro b, existe um subconjunto $T \subset N$ tal que $\sum\limits_{t \in T} t = b$?

Lema 2.1. SOMA DE SUBCONJUNTOS $\in NP$

Demonstração. Dado o subconjunto resposta T, calcule $\sum\limits_{t \in T} t$ em tempo $O(T) = O(N)$. □

Sabendo que o problema é NP, reduziremos o problema de COBERTURA EXATA para o de SOMA DE SUBCONJUNTOS.

Definição 2.2 (COBERTURA EXATA)**.** Dada uma família S de subconjuntos de u, $|u| = t$, existe uma subfamília $T \subset S$ tal que os conjuntos em T são disjuntos e $\cup T = \cup S = u$?

Lema 2.2. COBERTURA EXATA $\propto$ SOMA DE SUBCONJUNTOS

Demonstração. Podemos representar todo subconjunto S_i como uma string de bits s_{ij}, onde o bit j é 1 caso $u_j \in S_i$. Por exemplo, para $u = 1, 2, 3, 4, 5$:

$$S_i = \{1, 2, 3\} \longrightarrow s_i = 00111$$
$$S_i = \{3, 4, 5\} \longrightarrow s_i = 11100$$

Calculamos então inteiros N_i, equivalentes a s_i na base $d = |S| + 1$:

$$N_i = \sum_{i=1}^{t} s_{ij} * d^{i-1}$$

Acabamos de representar subconjuntos como números inteiros, e união de subconjuntos como soma de inteiros. Basta então encontrar um subconjunto cuja soma representa u. Ora, u é simplesmente a string de bits com todos os bits iguais a 1, já que $\forall j u_j \in u$.

$$u = \underbrace{111\cdots1}_{t}$$

$$b = \text{u na base d} = \sum_{i=1}^{t} d^{i-1} = \frac{d^t - 1}{d - 1}$$

Na base d, isso significa que cada bit 1 da soma corresponde a um único elemento de um único subconjunto, e portanto os subconjuntos são disjuntos. Uma interseção não nula entre subconjuntos resultaria em uma string do tipo 11211. De fato, no pior caso teríamos:

$$S = \{\{1\}, \{1,2\}, \{1,2,3,...\}, ..., \{1,2,3,...,x\}\}$$
$$N = \{000\cdots001, 000\cdots011, 000\cdots111, ..., 111\cdots111\}(\text{base } d)$$

E a soma de todos os N subconjuntos, que contém todos os elementos em u mas não é uma resposta para o problema já que seus subconjuntos não são disjuntos, seria o valor correspondente à seguinte sequência de dígitos, que é diferente de b:

$$(d-1)(d-2)(d-3)\cdots(1)$$

Temos então a conversão de uma entrada (S, t) do problema Cobertura exata para uma entrada (N, b) do problema Soma de subconjuntos. A conversão de cada subconjunto para um número inteiro é feita em $O(t)$, e portanto toda a conversão é feita em tempo $O(St)$. □

Teorema 2.3. Cobertura exata *é NP-completo[1]. A partir dos lemas 2.1 e 2.2,* Soma de subconjuntos *também é NP-completo.*

3 Modelagem do problema

O problema é muito bem estudado, e existem várias alternativas propostas para resolvê-lo. Uma solução simples baseada em programação dinâmica de pior caso $O(Nb)$ é ensinada em cursos de algoritmos[2]. Além disso, várias soluções paralelas existem na literatura, porém a maioria com complexidade de espaço exponencial, o que limita a aplicabilidade na prática.

Com o interesse de focar no uso de programação paralela para resolver o problema, foi implementada uma solução trivial do problema, uma busca exaustiva no espaço de soluções de tamanho 2^N, como descrito no algoritmo 1.

Algoritmo 1: Força bruta

Entrada: Conjunto N, soma b.
Saída: *true* caso exista um subconjunto de N cuja soma é b, *false* caso contrário.

```
for i ← 0 to 2^|N| − 1 do
    Soma ⟵ 0
    for j ← 0 to |N| do
        if bit(i, j) then                    /* bit(i,j) é o j-ésimo bit de i */
            Soma ← Soma + N_j
        end
    end
    if Soma = b then
        i é a solução do problema.
        return true
    end
end
return false
```

A partir dessa solução, podemos dividir o espaço de busca e executar várias buscas em paralelo, como descrito no algoritmo 2.

Algoritmo 2: Força bruta paralela

Entrada: Conjunto N, soma b, número de processadores P.
Saída: $true$ caso exista um subconjunto de N cuja soma é b, $false$ caso contrário.

$Total \longleftarrow 2^{|N|} - 1$
$Início \longleftarrow 0$
$Final \longleftarrow -1$
for $i \leftarrow 1$ **to** P **do**
  $Início \leftarrow Final + 1$
  $Final \leftarrow Total * (i/P) - 1$
  Cria thread e executa força bruta de $Início$ a $Final$
end
Espera todas as threads terminarem, retorna $true$ caso qualquer thread tenha retornado $true$

---

4 Análise teórica do custo assintótico

A análise do loop principal é trivial: 2^N soluções são exploradas, cada uma em tempo $O(N)$. O tempo total sequencial do programa é $O(N2^N)$. Ao dividir o espaço de busca entre P processadores, a busca é executada em tempo

$$O(\frac{N2^N}{P})$$

4.1 Análise teórica do custo assintótico de espaço

As únicas alocações do programa são para armazenar a entrada ($O(N)$), e uma estrutura de tamanho $O(1)$ para armazenar informações relevantes a cada thread ($O(P)$). O custo total é

$$O(N + P)$$

5 Análise de experimentos

Para realizar a análise experimental, foram geradas entradas de tamanhos 10, 20, 30, 40, 50, todas elas com resultado "não", de forma a medir o tempo de execução no pior caso. Para medir o tempo de execução do código, foi utilizada a API `mach_absolute_time`. Cada instância foi testada 3 vezes e o tempo de execução foi obtido retirando a média do tempo das 3 execuções. Os testes foram repetidos com 1, 2 e 4 threads. Os testes foram executados em uma máquina com processador Intel Core i7 2.6 GHz e 16GB de memória RAM.

A figura 5 demonstra tanto o tempo de execução exponencial quanto o paralelismo linear: o tempo de execução com N threads aproxima com precisão o tempo de execução com 1 thread dividido por N.

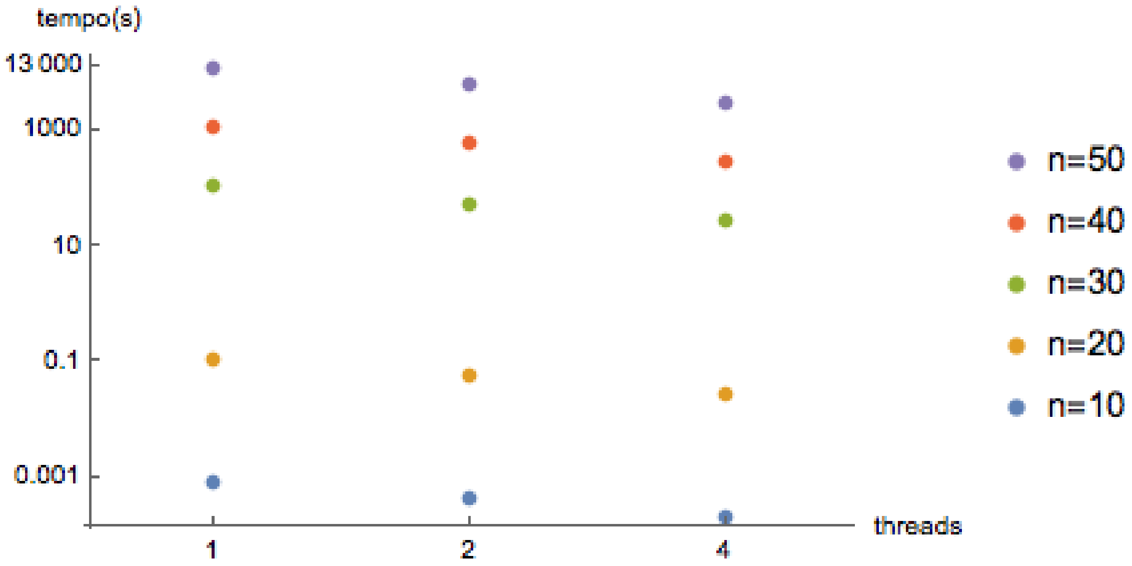

Figura 1: Tempo de execução para 5 tamanhos de entrada com 1, 2 e 4 threads. O eixo tempo tem escala logarítmica.

Referências

[1] Richard M Karp. *Reducibility among combinatorial problems.* Springer, 1972.

[2] Thomas H Cormen. *Introduction to algorithms.* MIT press, 2009.