Trabalho Prático 0 - AEDS III

Aluna: Ingrid Elisabeth Spangler
Professor: Marcos Augusto M. Vieira
Data: 18/04/2017

Introdução

O problema dado é ajudar um funcionário hipotético de uma empresa, chamado João, a achar todas as possíveis combinações de operadores em uma equação que a faz verdadeira dado um resultado. Convenientemente, a notação polonesa reversa foi escolhida para expressar as fórmulas, assim eliminando as ambiguidades de prioridade de operadores existentes na notação *infixa*, e também faz com que seja possível a resolução computacional usando uma pilha. Entretanto, a solução adotada para este problema é em tempo exponencial, pois otimizações adicionais tornariam o programa muito extenso.

Solução do problema

A heurística utilizada foi uma de tentativa e erro exaustiva mas que consome pouco espaço de memória. Como há apenas dois operadores possíveis em todos os casos, uma máscara de bits foi utilizada para computar cada tentativa, com os 1s representando os *s e os 0s os +s. O algoritmo guarda a sequência dada em um vetor, com os espaços correspondentes aos operadores preenchidos por -1, valor escolhido devido à ausência de valores negativos no problema.

Exemplo: 3 2 1 ? ? vira [3,2,1,-1,-1]

O tamanho da sequência e o número de operadores também são contados durante o processo de processamento da entrada para uso em futuros loops. Um loop começa em 0 e itera até 2^m, sendo m o número total de operadores. Os valores no vetor são empilhados e quando o algoritmo encontra um -1, neste caso os últimos dois valores da pilha são desempilhados e a operação ditada por mask &(1<<n) (n o número do operador atual), 1 ou 0, é executada.

Análise de complexidade

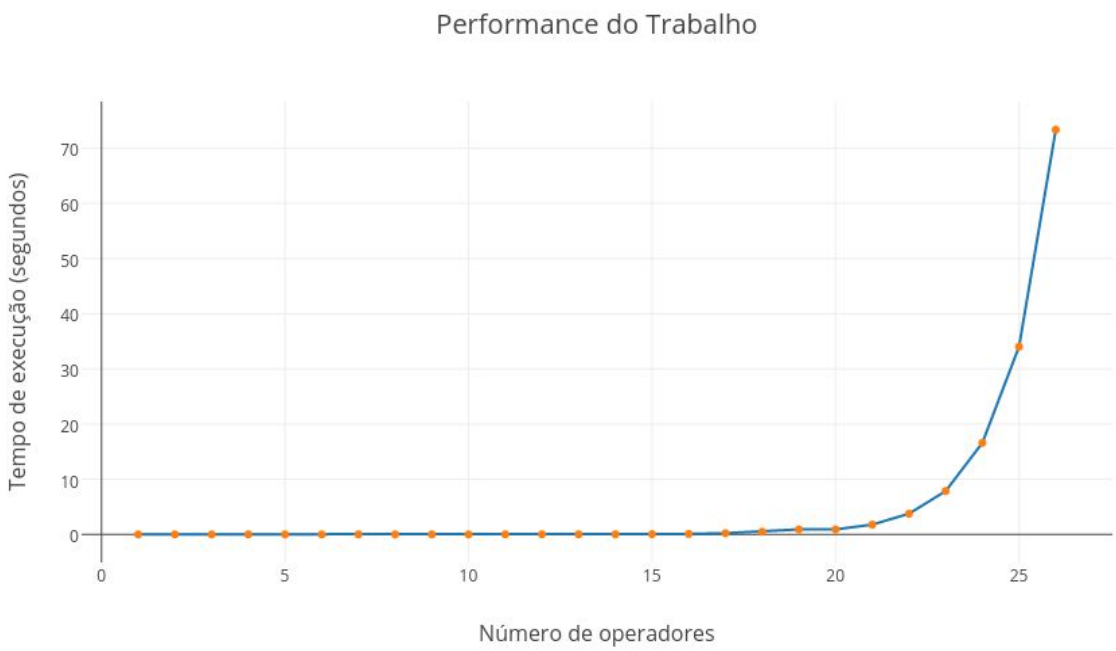
A complexidade do problema é , $\mathcal{O}(m * 2^m)$ m sendo o número de operadores lidos e 2 o número de possibilidades para estes operadores, que é um número estático neste problema. Como todas as combinações devem ser lidas, não há um melhor ou pior caso, todos os casos são testados. o espaço utilizado pelo programa não é

maior do que o espaço utilizado pelos vetores principais e algumas instâncias da pilha, que nunca cresce muito, sendo sempre $\mathcal{O}(n)$, sendo n o número de operandos na expressão. Tudo é alocado e liberado sem leaks de memória.

	Função	Complexidade
0	getnterms()	O(n)
1	fill_vector()	O(n)
2	polish()	O(n²)
3	makestring()	O(m)
4	main()	O(n² * 2^n)
5	new_stack()	O(1)
6	push()	O(n)
7	pop()	O(n)
8	check()	O(n)
9	free_stack()	O(n)

Avaliação experimental

o programa foi testado em um processador Intel Core i7 com casos que contém entre 1 e 26 operadores, os seus tempos foram:



	Operadores	Tempo
0	1	0.000220
1	2	0.000234
2	3	0.000203
3	4	0.000177
4	5	0.000206
5	6	0.000281
6	7	0.000318
7	8	0.000589
8	9	0.001108
9	10	0.001845
10	11	0.004229
11	12	0.007973
12	13	0.015777
13	14	0.020932
14	15	0.041738
15	16	0.088929
16	17	0.218199
17	18	0.468277
18	19	0.905603
19	20	0.902242
20	21	1.769244
21	22	3.749258
22	23	7.847928
23	24	16.585234
24	25	34.035572
25	26	73.347266

Detalhes da implementação e considerações extras

O problema foi desenvolvido no sistema operacional Linux, distribuição Arch (Manjaro). Foi uma boa revisão de ponteiros e estruturas de dados simples, ótima oportunidade para trabalhar com uma notação útil como a polonesa, e também para implementar lógica bitwise.