Trabalho Prático 0

Aluna: Ingrid Rosselis Sant'Ana Da Cunha Matrícula: 2016430936

Introdução:

O problema do TP0 é resolver uma expressão em notação polonesa reversa com os operadores apagados, dado a expressão e o total, resultando em várias possibilidades da sequência dos operadores + e *, que satisfazem a expressão. Para solucioná-lo é necessário um entendimento sobre o funcionamento da notação e um conhecimento amplo das estruturas de dados e técnicas de programação, pois a execução pode se tornar bem custosa dependendo do tamanho da entrada e da solução utilizada.

A solução implementada neste trabalho baseou-se na forma usual para a resolução desse tipo de problema: insere-se os símbolos retirados da entrada um-a-um em uma pilha e quando o símbolo for um operador, retira-se os dois últimos operandos inseridos e faz a operação. Por fim, coloca-se o resultado da operação de volta na pilha. Deste modo, o elemento restante da pilha será a solução da expressão. Para lidar com a característica específica de cada operador ser + ou *, foi observado que poderia ser usada a mesma lógica, apenas colocando cada resposta possível em uma lista da pilha.

Metodologia:

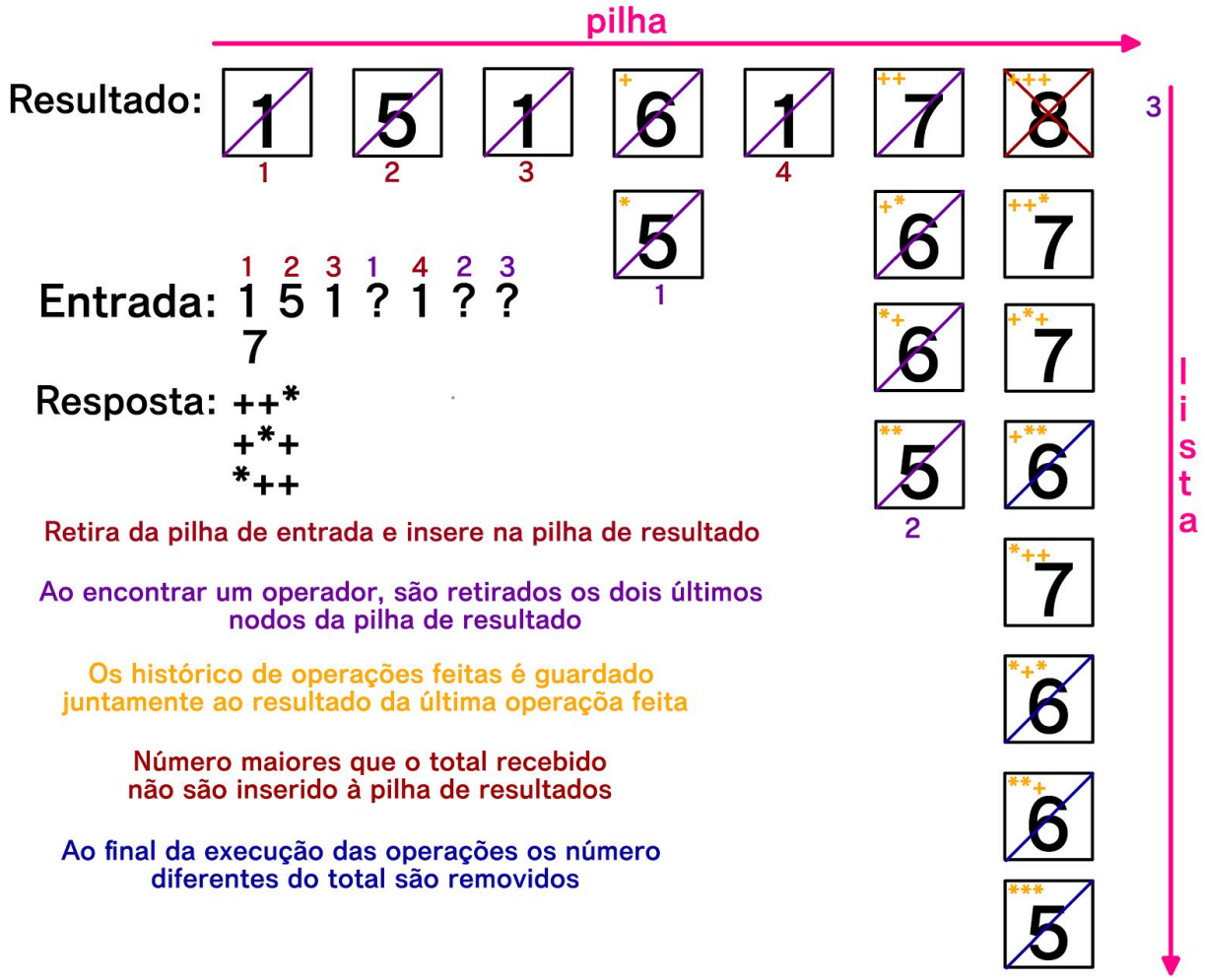
A implementação do programa é composta por duas estruturas de dados: pilha e lista. Cada nodo da pilha possui uma lista e cada nodo da lista possui o resultado de uma operação e uma string para guardar o histórico das operações feitas. A entrada é recebida com alocação estática de tamanho 200. O projeto foi dividido em 7 arquivos: 2 para lista, 2 para pilha, 2 para funções auxiliares e 1 para o programa principal.

As implementações da lista e da pilha foram feitas de maneira simples e intuitiva: as estruturas de dados não possuem um nodo cabeça, um novo nodo da lista sempre é inserido no final e um novo nodo da pilha é sempre inserido na primeira posição. A escolha da estrutura de dados lista para compor um nodo da pilha se deve à tentativa de fazer a solução clássica do problema servir também para o problema específico em questão. Apesar da lista não ser a estrutura que melhor otimizaria a quantidade de dados recebidas das operações, ela permite uma robustez e uma solução precisa para cada instância do problema, não havendo casos onde uma lista não poderia acomodar as soluções possíveis.

São utilizadas durante o programa duas pilhas: uma para entrada e uma para os resultados. A pilha de entrada é apenas usada para receber os dados na forma que eles serão usados no programa, distribuindo-os depois de acordo com as operações feitas. A pilha de resultados trata toda a solução prática do programa e recebe todos os resultados parciais. Ao fim do programa, a pilha resultados possuirá um único nodo com a lista de todos os resultados possíveis para o problema.

Como uma forma de lidar com a quantidade de dados inseridos na pilha e nas listas é feita uma comparação com o resultado da expressão a cada operação feita, visando diminuir a quantidade de dados inseridos. Para diminuir a quantidade de nodos na pilha de resultados, a cada dois operandos da pilha suas listas são removidas para realizar operações e ao fim são desalocadas.

Abaixo segue uma imagem resumindo como uma instância do problema é tratada até chegar a seu resultado, enfatizando as partes mais interessantes do algoritmo:



Análise de Complexidade:

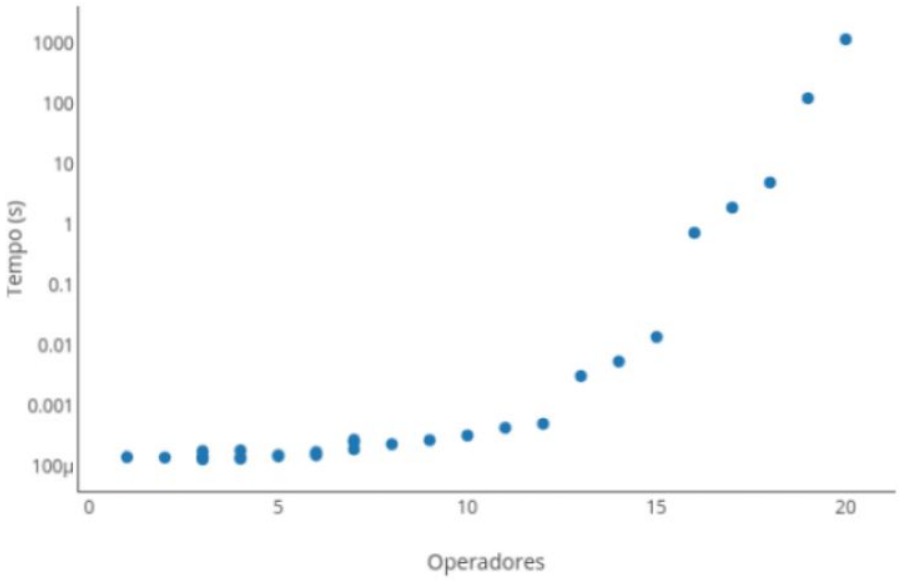
O ponto principal no problema para a análise de complexidade é o número de comparações feitas durante a execução do problema, sendo que cada comparação é originada de uma operação. As informações mais importantes para a análise são o número de operadores e em qual posição eles se encontram em relação aos outros. Como cada operador encontrado na pilha de entrada resulta em duas operações, a lista de soluções possíveis em cada execução tem um tamanho de 2^n, onde n é a ordem de aparição do operador na pilha de entrada. Isto é, como no exemplo apresentado na seção anterior, o primeiro operador gera uma lista de 2^1 nodos, o segundo de 2^2 nodos e o terceiro de 2^3 nodos. Cada lista também pode ser comparada com as outras listas, então o número de nodos total no fim a operação, nestes casos, se torna 2^{2n}, onde n é a posição do operador mais à direita. Ou seja, as execuções mais custosas no código são: dois loops encadeados que fazem duas comparações a cada execução e a comparação final da lista gerada com o resultado, onde essa lista chega a 2^{2n} nodos. As outras operações feitas no código giram em torno de inverter e tratar a entrada, logo elas estão na ordem de m comparações, onde m é o tamanho da entrada, que não passa de 200 caracteres, incluindo espaços.

Baseado nessas informações e sabendo que cada nodo é gerado de uma operação e uma comparação, a complexidade de tempo e espaço do pior caso é da ordem de $O(2^{2n})$ e em casos esperados é da ordem $O(2^n)$.

Avaliação experimental:

Para analisar o tempo de execução do programa foram usados os casos de teste toys disponibilizados junto com a especificação do trabalho e mais alguns testes que aumentam o número de operadores até 20. Segue abaixo o gráfico plotado de acordo com os dados obtidos, junto com uma tabela que relaciona o número de operadores e o tempo de execução em segundos:

Operadores	Tempo (s)
1	0.000127
2	0.000125
3	0.000126
3	0.000125
3	0.00016
3	0.000117
4	0.000164
4	0.000124
4	0.00012
5	0.000137
5	0.000131
6	0.000154
6	0.000136
6	0.000145
7	0.000246
7	0.000224
7	0.000171
8	0.000208
9	0.000242
10	0.00029
11	0.000388
12	0.000453
13	0.002806
14	0.004897
15	0.012479
16	0.664696
17	1.740241
18	4.529526
19	113.138145
20	1072.238559



É interessante reparar que o tempo de execução aumenta exponencialmente conforme aumenta-se o número de operadores na expressão, explicitando a complexidade obtida na seção anterior. Os testes com mesmo número de operadores também sofrem mudanças de desempenho, dado o tipo de entrada. Foi observado que as expressões com maior quantidade de 1s tem um tempo maior de execução, assim como aquelas que possuem mais operadores encadeados (são aquelas que resultam em uma complexidades $O(2^{2n})$).

Conclusão:

Neste trabalho foi solicitado que se resolvesse o problema de expressões em notação polonesa reversa com operadores apagados. A solução utilizada é baseada na solução original deste problema, com todos os operadores explícitos. Para este caso do problema foi utilizada uma pilha de listas para armazenar todas as possibilidades de resultados.

As maiores dificuldades neste trabalho prático foram lidar com a quantidade de alocações de memória feitas durante a execução e otimizar a solução ainda usando as estruturas de dados e a lógica propostas inicialmente. Ao fim do trabalho foi observado que o número de comparações feitas aumenta exponencialmente em relação ao número de operadores. Assim, a complexidade de tempo e espaço está na ordem de $O(2^n)$, uma complexidade exponencial.