Processamento da Informação

Fabrício Olivetti de França

18 de Fevereiro de 2020

Atividade 02: Novos comandos para o IntCode

Relembrando...

Relembrando, a máquina IntCode contém um código composto por uma sequência de números inteiros que representam operações de um computador. Na aula passada introduzimos as operações de adição e multiplicação. Essa semana faremos outras operações importantes para que possamos criar programas interessantes com ela.

Opcodes e modos de leitura e escrita

As operações de adição e multiplicação eram representadas pelos inteiros 1 e 2, respectivamente. Uma sequência `[1, 7, 5, 2, ...]` indica que aplicaremos a operação `soma` somando o conteúdo da posição 7 com o conteúdo da posição 5 da memória, e armazenando na posição 2.

Esse modo é conhecido como **modo de indexação** para a leitura de argumentos e escrita do resultado. O IntCode suporta outros modos além desses. Hoje implementaremos o **modo de valor** em que o valor a ser utilizado como argumento é o próprio valor de memória `memoria[pos + n]` para o n-ésimo argumento.

Representamos o modo de operação acrescentando dígiots a esquerda do *opcode*. Por exemplo, seja o valor de *opcode* for 1001, os dois dígitos mais a direita representam o *opcode* (01), ou seja, é a operação de adição. Os dígitos seguintes representam o modo de leitura/escrita para cada argumento na sequência da direita para a esquerda. Ou seja, o primeiro argumento está no modo de indexação (0), o segundo argumento no modo valor (1) e o último, como não tem mais dígitos, assumimos como modo de indexação (0).

```
opcode = 1001

 arg 3  arg 2  arg 1  opcode
|   0  |   1  |   0  |  01  |
```

Entrada e Saída

Outra novidade de nossa máquina é o suporte a entrada e saída de dados. Além dos campos `memoria, pos, status` temos os campos `entrada` e `saida`. O campo `entrada` contém o último valor inserido pelo usuário através de algum mecanismo de entrada. O campo `saida` contém um valor de saída resultado de alguma operação e que deve ser mostrado ao usuário.

Exercício 01

Crie as funções `opcode` e `pegaModoN` no arquivo *IntCode.py* para recuperar o *opcode* e os modos de cada argumento de um inteiro. A função `opcode` deve retornar os dois últimos digitos mais a direita do número. A função `pegaModoN` recebe o inteiro correspondente ao valor de memória sem os dois últimos dígitos (somente os modos) e um inteiro indicando o modo de qual argumento quer recuperar.

Exemplo:

```
opcode(1001) == 1

pegaModoN(10, 0) == 0
pegaModoN(10, 1) == 1
pegaModoN(10, 2) == 0
```

A partir desse momento utilizaremos as funções `leiaArgN` para ler um argumento de entrada e `leiaPosEscrita` para ler a posição de memória onde escreveremos os resultados.

Exercício 02

Vamos implementar duas novas instruções para nossa máquina: `ler` (*opcode* 3) e `escrever` (*opcode* 4). A função `ler` deve armazenar o valor contido no campo `entrada` na posição de memória obitda pela função `leiaPosEscrita`. Após a operação, não se esqueça de atualizar `pos` para a próxima posição e atualizar o campo `entrada` com o valor `None`, indicando que não existe nenhum valor de entrada no *buffer*.

A função `escrever` deve atualizar o campo `saida` com o valor obtido pelo uso da função `leiaArgN`.

Não se esqueça de descomentar as linhas:

```
#elif op == 3:
#    return ler(m)
#elif op == 4:
#    return escrever(m)
```

Exercício 03

Escreva as funções `pularNZ` (*opcode* 5) e `pularZ` (*opcode* 6). A função `pularNZ` deve ler dois argumentos de entrada (usando `leiaArgN`) e, se o primeiro argumento for diferente de zero, atualize o campos `pos` para o valor do segundo argumento, caso contrário atualize `pos` para 3 posições a frente.

A função `pularZ` é análoga apenas trocando a comparação pela igualdade com o valor zero.

Não se esqueça de descomentar as linhas:

```
#elif op == 5:
#    return pularNZ(m)
#elif op == 6:
#    return pularZ(m)
```

Exercício 04

Escreva as funções `comparar` e `igual`. A função `comparar` deve ler dois argumentos de entrada (usando `leiaArgN`) e a posição de escrita (usando `leiaPosEscrita`). Se o primeiro argumento for menor que o segundo, escreva 1 na posição de escrita caso contrário escreva 0.

A função `igual` é análoga apenas trocando a comparação por igualdade.