CABuilder: Documentação

Release 2.0

Elloá B. Guedes e Nicoli Araújo

16 de Agosto de 2015

Sumário

Conteúdo

CAPÍTULO 1

Overview

Este projeto tem como objetivo analisar a qualidade de autômatos celulares para gerar números pseudo-aleatórios.

A ferramenta CABuilder consiste em um software que instancia autômatos celulares de tipos elementar e totalístico, e usa-os para criar imagens e arquivos de texto com sequências de números. As sequências de números devem passar por baterias de testes estatísticos para certificar a sua pseudo-aleatoriedade.

Um autómato celular é um modelo computacional que consiste de uma matriz de células que fazem auto-reprodução. Nota-se que a aplicação de tais estruturas para gerar sequências de números pseudo-aleatórios não tem registos literárias. Além disso, essas seqüências de números pseudo-aleatórios seria usado principalmente em criptografia, simulações e algoritmos de modelagem de sistemas operacionais.

Página do projeto: href http://nicoliaraujo.github.io/paic-autocel/ CABuilder.

Responsável: Nicoli Araujo

Orientadora: Elloa B. Guedes

CAPÍTULO 2

Módulo CellularAutomata

Criado em 04/01/2015

@author: Nicoli Araújo @author: Elloá B. Guedes

Este módulo contém as classes que definem um Autômato Celular. Superclasse CellularAutomata - Define métodos e atributos gerais de um CA Classe ElementaryCode - Instancia métodos e atributos específicos para autômatos celulares com k = 2 Classe TotalisticCode - Instancia métodos e atributos para autômatos celulares com k > 2

2.1 Superclasse CellularAutomata

class cellularautomata.modules.CellularAutomata.**CellularAutomata**(*rule*, *k*, *seed*)

SuperClasse que define autômatos celulares.

Um automato celular é um vetor multidimensional composto por células que detém um estado. O estado de cada célula é determinado por uma regra numérica que considera os estados das celulas vizinhas.

__init__(*rule*, *k*, *seed*)

Construtor do autômato celular.

Aqui, são instanciados a regra (rule), o número de estados(k), e o primeiro estado da célula que se encontra exatamente no centro do vetor (seed). Após isso, é criado um dicionário que associa o estado da vizinhança de uma célula com o seu próprio (dictRule).

-rule (int) : número da regra a que o autômato obedece. -k (int) : número de estados que cada célula do autômato pode ter. Os estados são inteiros de 0 a k - 1. -seed (int) : estado inicial da célula central do autômato celular

Após isso, é criado um dicionário que associa o estado da vizinhança de uma célula com o seu próprio (dictRule).

-dictRule (dict (int -> int)) : dicionário que relaciona os estados de uma vizinhança (inteiros de 0 a 7) ao estado de uma célula (bits de rule na base k). -type (str) : tipo do autômato celular. Inicialmente, é vazio, ou genérico.

__str__()

Retorna o nome do autômato celular.

__weakref__

list of weak references to the object (if defined)

getState(*chave*)

Retorna dictRule[chave].

chave (int) - um inteiro de 0 a 7, que armazena uma das oito possíveis combinações de estados da vizinhança de uma célula.

```
>>> ca.setDictRule(200, 3)
{0: 2, 1: 0, 2: 1, 3: 1, 4: 2, 5: 0, 6: 0, 7: 0}
```

ca.getState(0) 2 ca.getState(6) 0 ca.getState(4) 2

k

Retorna k.

setDictRule(*rule*, *k*)

Cria o dicionário de 8 chaves dictRule, que relaciona a vizinhança com o estado de uma célula.

Cada chave é um inteiro de 0 a 7 que representa uma soma dos estados de uma vizinhança. As chaves guardam inteiros que vão de 0 a k-1, que representam os possíveis estados da célula a partir da vizinhança designada na chave. Cada estado é um bit da string de tamanho 8 ruleInKBase. É nesta variável que rule escrita na base k é guardada.

Retorna dictRule (int -> int)

```
>>> ca.setDictRule(30, 2)
{0: 0, 1: 1, 2: 1, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0}
```

```
>>> ca.setDictRule(45, 3)
{0: 0, 1: 0, 2: 2, 3: 1, 4: 0, 5: 0, 6: 0, 7: 0}
```

```
>>> ca.setDictRule(200, 3)
{0: 2, 1: 0, 2: 1, 3: 1, 4: 2, 5: 0, 6: 0, 7: 0}
```

2.2 Classe ElementaryCode

class cellularautomata.modules.CellularAutomata.**ElementaryCode**(*rule*)

Subclasse de CellularAutomata: Define autômatos celulares do tipo elementar.

Em um ElementaryCode, as células podem apresentar apenas dois estados ao levar em consideração os estados das três células vizinhas da iteração imediatamente anterior.

__init__(*rule*)

Construtor da classe ElementaryCode.

Estende o construtor de CellularAutomata.

Aqui, são instanciados rule, k, type e seed. É implementado o conceito de apenas dois estados, dando-se a k o valor 2 e a seed o valor 1 . Também é alterado o tipo do autômato celular, assumindo-se sua característica elementar.

getNext(*b1*, *b2*, *b3*)

Retorna dictRule[b1b2b3], sendo b1b2b3 os três parâmetros concatenados.

Sobrescreve CellularAutomata.getNext(b1, b2, b3).

Método que recebe o estado de tres vizinhas, concatena-os para transformá-los em um binário, e os transforma no número inteiro correspondente, que é utilizado como chave do dictRule.

Retornar o valor ao qual a chave esta associada no dictRule.

```
>>> ec = ElementaryCode(45)
```

```
>>> ec.dictRule
{0: 1, 1: 0, 2: 1, 3: 1, 4: 0, 5: 1, 6: 0, 7: 0}
```

```
>>> ec.getNext(1, 0, 0)
0
```

```
>>> ec.getNext(0, 1, 1)
1
```

```
>>> ec.getNext(1, 1, 1)
0
```

2.3 Classe TotalisticCode

class cellularautomata.modules.CellularAutomata.**TotalisticCode**(*rule*, *k*, *seed*)

Subclasse que define os autômatos do tipo totalistico.

Um TotalisticCode pode ter mais de dois estados possíveis para cada célula. Além disso, para definir o estado de uma célula, é feita a média entre as três células vizinhas da iteração imediatamente anterior.

__init__(*rule*, *k*, *seed*)

Construtor da subclasse TotalisticCode

Estende o método construtor de CellularAutomata.

Também é instanciado o tipo do autômato: Totalístico

getNext(*b1*, *b2*, *b3*)

Retorna dictRule[b1+b2+b3]

Sobrescreve CellularAutomata.getNext(b1, b2, b3).

Define o estado de uma célula a partir do de três vizinhas, armazenados em b1, b2 e b3. Retorna o valor no dictRule referente à soma dos três estados fornecidos.

```
>>> tc = TotalisticCode(200, 3)
```

```
>>> tc.dictRule
{0: 2, 1: 0, 2: 1, 3: 1, 4: 2, 5: 0, 6: 0, 7: 0}
```

```
>>> tc.getNext(0, 2, 1)
1
>>> tc.getNext(0, 2, 2)
2
>>> tc.getNext(2, 2, 2)
0
```

CAPÍTULO 3

Módulo CAOutput

Criado on 04/01/2015

@author: Nicoli Araújo

@author: Elloá B. Guedes

Módulo que abriga as classes que salvam os estados de um autômato celular em uma imagem de extensão .png composta de pixels com tons que vão do preto ao branco ou um arquivo de texto composto de números inteiros.

- AutomataImage gera imagem.
- AutomataText gera um arquivo texto.

3.1 Classe AutomataImage

class cellularautomata.modules.CAOutput.**AutomataImage**(*side*, *ca*, *info*)

Classe que tem por objetivo gerar uma imagem que represente um autômato celular.

Aqui, os estados são representados por cores que vão do branco ao preto. Cada estado é uma cor. Cada coluna representa uma célula, que muda de estado em cada linha. Assim, as linhas representam um único vetor unidimensional, que tem seu estado mudado conforme o tempo passa.

__init__(*side*, *ca*, *info*)

Construtor da classe AutomataImage

Instancia o lado da imagem a ser gerada(side) e o autômato celular a que ela pertence (ca):

•side (int) : Altura da imagem, em pixels: representa a quantidade de iterações desejadas para o autômato celular, ou seja, é a passagem de tempo; Largura da imagem, em pixels.

•ca (CellularAutomata) : Autômato celular cuja regra e semente serão utilizadas para colorir a imagem.

•info (str) : A imagem é salva na pasta /imgoutput/, subpasta do seu tipo, com o nome igual a sua regra. Caso deseje-se acrescentar outra informação, deve-se declará-la como uma string na variável info.

A partir desses parâmetros, o construtor instancia:

•image (Image) : cria uma imagem em tons de cinza, de tamanho (side x side) de pixels brancos.

•dictColor: dict (int -> int) : Dicionário que alia cada um dos k estados do autômato a uma cor de 0 a 255. As chaves são números de 0 a k-1, e os valores guardados por elas são as cores que representam cada estado.

Então, o método setImage() edita a imagem conforme a evolução do ca, e o método save() a salva no formato .png, na pasta padrão (/Output/imgoutput/).

__weakref__
list of weak references to the object (if defined)

getSite(*x*, *y*)
Retorna o novo estado (ou seja, um número de 0 a k-1) de uma célula na posicao (x,y).

Dada as coordenadas (x,y) da imagem do auntômato celular, são salvos os estados das células nas posições (x-1, y-1), (x, y-1) e (x+1, y-1). Estes estados são passados ao automata, que retorna o estado resultante da célula na posição (x,y).

```
>>> tc = TotalisticCode(210, 3, 1)
>>> AutomataImage = AutomataImage(3, tc, '.png')
```

```
>>> AutomataImage.dictColor
{0: 255, 1: 127, 2: 0}
```

```
>>> AutomataImage.getSite(2, 1)
1
```

```
>>> AutomataImage.getSite(3, 2)
1
```

```
>>> AutomataImage.getSite(2, 2)
2
```

putFirstPixel(*seed*)
Método que põe o primeiro pixel na imagem.

- firstPixel(int) : Número do estado desejado, ou seja, chave da cor desejada no dictColor.

putPixel(*state*, *x*, *y*)
Põe nas coordenadas informadas a cor correspondente a state no dictColor.

Dado um estado de 0 a k-1, transforma-o na cor correspondenten no dictColor e coloca na imagem na posição (y,x)

save()
Salva a imagem criada no formato ".png".

Método que salva a imagem criada no caminho path, com o formato ".png". No nome do arquivo de imagem salvo consta o nome do autômato.

searchSite(*color*)
Retorna o estado (state) que guarda a cor desejada no dictColor

- color (int) : Uma cor de 0 a 255

Retorna state (int) se dictColor[state] = color

```
>>> AutomataImage.dictColor
{0: 255, 1: 0}
```

```
>>> AutomataImage.searchSite(255)
0
```

```
>>> AutomataImage.searchSite(0)
1
```

setDictColor(*k*)
Retorna um dicionario de k cores, que relaciona cada cor a um valor que varia de 0 a k-1

- k (int) : Número de estados do autômato.

```
>>> AutomataImage.setDictColor(2)
{0: 255, 1: 0}
```

```
>>> AutomataImage.setDictColor(3)
{0: 255, 1: 127, 2: 0}
```

```
>>> AutomataImage.setDictColor(4)
{0: 255, 1: 170, 2: 85, 3: 0}
```

```
>>> AutomataImage.setDictColor(5)
{0: 255, 1: 191, 2: 127, 3: 63, 4: 0}
```

setFirstPixel (*seed*)
Define a cor do primeiro pixel a ser colocado na imagem a partir do estado inical do ca.

•seed (int): Um número de 0 a k-1 que representa o estado inicial do autômato celular.

setImage (*seed*)
Gera a imagem que representa o autômato celular.

Edita a imagem criada na iniciação de acordo com o autômato. Em t = 0 (primeira linha) põe o primeiro pixel da cor correspondente à semente dada. Começando em t = 1 (ou na segunda linha), são atualizados os estados de todas as células do autômato.

•seed (int) : estado da célula central do vetor em t = 0. Inteiro de 0 a k-1.

tryGetSite (*x*, *y*)
Retorna o estado correspondente à cor nas coordenadas (x, y) da imagem do autômato celular.

O método captura a cor de um pixel na posição (x, y) e retorna seu estado (state) de acordo com o dictColor. Se não houver pixel em (x,y), retorna o estado 0.

```
>>> AutomataPicture = AutomataPicture(3, 3, 210, 3, 1)
```

```
>>> AutomataImage.dictColor
{0: 255, 1: 127, 2: 0}
```

```
>>> AutomataImage.tryGetSite(3, 2)
0
```

```
>>> AutomataImage.tryGetSite(2, 2)
2
```

```
>>> AutomataImage.tryGetSite(2, 1)
1
```

3.2 Classe AutomataText

class cellularautomata.modules.CAOutput.**AutomataText** (*size*, *it*, *ca*, *info*)
Classe que gera um arquivo de texto que armazena os estados de um autômato celular.

O estado de cada célula é representado por um número inteiro de 0 a k-1, sendo k o número de estados possíveis do autômato.Cada bit representa uma célula, que muda de estado em cada linha. Assim, as linhas representam um único vetor unidimensional, que tem seu estado mudado conforme o tempo passa. Portanto, o arquivo mostrará uma matriz size x it, em que size é a largura e it é a altura.

__init__ (*size*, *it*, *ca*, *info*)
Construtor da classe AutomataText

Instancia o tamanho do vetor a ser gerado (size), a quantidade de iterações que ocorrerão (it) e o autômato celular que se deseja representar (ca):

- size (int) : Largura do vetor
- it (int) : Quantidade de iterações, ou seja, é a passagem de tempo
- ca (CellularAutomata) - Autômato celular cuja regra será utilizada para alterar os estados de cada célula.
- info (Str) : O arquivo é salvo na pasta /original/, subpasta do seu tipo, com o nome igual a sua regra. Caso deseje-se acrescentar outra informação, deve-se declará-la como uma string na variável info.

A partir desses parâmetros, o construtor instancia:

- file (File) : Arquivo de texto (.txt) composto de uma matriz de dimensões size x it de bits. Cada bit tem valor de 0 a k-1.
- firstBit (int) : Estado que a célula central do vetor deve ter no início do crescimento. Varia de 0 a k-1.

Então, o método setFile altera os estados das células do vetor it vezes. Em seguida, salva os estados em file.

__weakref__
list of weak references to the object (if defined)

getBits(*line*, *column*)
Calcula o estado de uma célula na posição (line x column). Para isso, pega o estado da vizinhança da célula na iteração anterior.

Retorna o estado da célula no tempo atual.

```
>>> ec = ElementaryCode(45)
>>> ectext = AutomataText(5, 10, e, '')
>>> ectext.array
[[ 0.  0.  1.  0.  0.]
 [ 1.  0.  1.  0.  1.]
 [ 0.  1.  1.  1.  1.]
 [ 1.  1.  0.  0.  0.]
 [ 1.  0.  0.  1.  1.]
 [ 0.  0.  0.  1.  0.]
 [ 1.  1.  0.  1.  0.]
 [ 1.  0.  1.  1.  0.]
 [ 1.  1.  1.  0.  0.]
 [ 1.  0.  0.  0.  1.]]
>>> ectext.getBits(5, 4)
0
```

```
>>> ec = ElementaryCode(63)
>>> ectext = AutomataText(5, 10, e, '')
>>> ecext.array
[[ 0.  0.  1.  0.  0.]
 [ 1.  1.  1.  1.  1.]
 [ 0.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.]
 [ 0.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.]
 [ 0.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.]
 [ 0.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.]]
```

```
>>> ectext.getBits(5, 4)
1
```

```
>>> tc = TotalisticCode(1074, 3, 1)
>>> tctext = AutomataText(10, 10, tc, '')
>>> tctext.array
[[ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  1.  1.  0.  0.  0.]
 [ 0.  0.  0.  1.  2.  0.  2.  1.  0.  0.]
 [ 0.  0.  1.  0.  0.  1.  0.  0.  1.  0.]
 [ 0.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 2.  2.  0.  0.  0.  0.  0.  0.  0.  2.]
 [ 1.  1.  2.  0.  0.  0.  0.  0.  2.  2.]
 [ 1.  1.  0.  2.  0.  0.  0.  2.  1.  1.]
 [ 0.  2.  0.  2.  2.  0.  2.  0.  1.  2.]
 [ 1.  2.  1.  1.  1.  1.  2.  0.  0.  0.]]
>>> tctext.getBits(5, 4)
```

setFile (*firstBit*, *info*)

Salva os estados do autômato no arquivo de texto.

Edita o arquvio de texto criado na iniciação de acordo com o autômato. Em t = 0 (primeira linha) põe o primeiro bit com o valor firstBit. Começando em t = 1 (ou na segunda linha), são atualizados e salvos os estados de todas as células do autômato até que tenha sido feito o número de iterações solicitado.

- firstBit (int) : estado da célula central do vetor em t = 0. Inteiro de 0 a k-1.

tryGetBit (*i*, *j*)

Pega o estado de uma célula na posição (i,j) do array.

Retorna um inteiro de 0 a k-1 que representa o estado da célula.

```
>>> ec = ElementaryCode(45)
>>> ectext = AutomataText(5, 10, e, '')
>>> ectext.array
[[ 0.  0.  1.  0.  0.]
 [ 1.  0.  1.  0.  1.]
 [ 0.  1.  1.  1.  1.]
 [ 1.  1.  0.  0.  0.]
 [ 1.  0.  0.  1.  1.]
 [ 0.  0.  0.  1.  0.]
 [ 1.  1.  0.  1.  0.]
 [ 1.  0.  1.  1.  0.]
 [ 1.  1.  1.  0.  0.]
 [ 1.  0.  0.  0.  1.]]
>>> ectext.tryGetBit(4, 6)
0
```

```
>>> ec = ElementaryCode(63)
>>> ectext = AutomataText(5, 10, e, '')
>>> ecext.array
[[ 0.  0.  1.  0.  0.]
 [ 1.  1.  1.  1.  1.]
 [ 0.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.]
 [ 0.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.]
 [ 0.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.]
 [ 0.  0.  0.  0.  0.]
```

```
 [ 1.  1.  1.  1.  1.]]
>>> ectext.tryGetBit(4, 6)
0
```

```
>>> tc = TotalisticCode(1074, 3, 1)
>>> tctext = AutomataText(10, 10, tc, '')
>>> tctext.array
[[ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  1.  1.  0.  0.  0.]
 [ 0.  0.  0.  1.  2.  0.  2.  1.  0.  0.]
 [ 0.  0.  1.  0.  0.  1.  0.  0.  1.  0.]
 [ 0.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 2.  2.  0.  0.  0.  0.  0.  0.  0.  2.]
 [ 1.  1.  2.  0.  0.  0.  0.  0.  2.  2.]
 [ 1.  1.  0.  2.  0.  0.  0.  2.  1.  1.]
 [ 0.  2.  0.  2.  2.  0.  2.  0.  1.  2.]
 [ 1.  2.  1.  1.  1.  1.  2.  0.  0.  0.]]
>>> tctext.tryGetBit(4,6)
1
```

writeLine(*array*, *line*)

Escreve uma linha de um array em file. Transfere os valores inteiros de cada posição, sem separador.

- array (Array ou List): vetor unidimensional do mesmo tamalho de self.size
- line (int): Linha do array que se deseja escrever no arquivo

CAPÍTULO 4

Module ParserImgText

Created on 03/03/2015

@author: Nicoli Araújo

Neste módulo estão classes que manipulam os arquivos de saída das classes AutomataText e AutomataImage. As classes desde módulo transcrevem arquivos e imagens para arquivos compostos de sequências de npumeros binários ou decimais, para que possam ser avaliados por um software que teste sua pseudo-aleatoriedade.

- ImgtoText pega uma imagem de um autômato celular totalístico ou elementar e a transforma em um arquivo de texto.
- Parser edita a forma como os números estão dispostos em um arquivo de texto.

4.1 Classe ImgtoText

class cellularautomata.modules.ParserImgText.**ImgtoText**(*caK*, *caImageName*, *info*)

Classe que pega uma imagem de um Autômato Celular e a transfere para um arquivo de texto. Utilizando o dicionário de cores do AutomataImage, transfere para um arquivo de texto o estado de cada célula, representado como uma cor na imagem.

Atenção: imagens com formatos que sofrem perda com a compressão não poderão ser utilizadas nesta classe.

__init__(*caK*, *caImageName*, *info*)

Construtor da classe ImgtoText

Instancia a quantidade de estados que o autômato que originou a imagem tem (caK - int) e o nome da imagem que se deseja transcrever para arquivo:

•caK (int) : número k>=2 de estados possíveis para as células do autômato que gerou a imagem

•caImageName (str) : string que descreve o nome da imagem, com o tipo. Ex: '45k3.png'. Não há a necessidade de dar-se a localização da imagem desejada.

•info (str) :

Utilizando-se estes parâmetros, o construtor instancia: - catype (str) : a partir do caK, o autômato tem seu tipo detectado e classificado em Elementar o Totalistico - size (int) : tamanho da imagem que deve ser manipulada - dictTxt (dict) : dicionário que relaciona as cores que cada pixel da imagem pode ter com os estados possíveis do autômato, que deverão ser escritos no arquivo. - filename (str) : nome do arquivo resultante. É utilizado como base o nome da imagem

A seguir, o construtor chama o método imgtotxt(), que transcreve os estados das células do autômato representados na imagem para o arquivo de texto.

__weakref__
list of weak references to the object (if defined)

imgToTxt()
Método que transcreve os estados das células representados na imagem para um arquivo de texto.

O método passa em cada pixel na imagem, utiliza o dictTxt para encontrar o estado numérico correspondente à cor e escreve este estado numérico no arquivo de texto. Cada linha da imagem corresponde a uma linha do arquivo.

setDictTxt(*k*)
Constroi o dictColors de trás pra frente.

Retorna dictTxt: Dict (int->int). Cada cor é utilizada como chave par guardar um dos k estados (0 a k-1) que uma célula pode apresentar.

4.2 Classe Parser

class cellularautomata.modules.ParserImgText.**Parser**(*oldFilePath*)
Classe que altera a disposição dos dados de um arquivo de texto composto apenas de números inteiros, e armazena essas alterações em um novo arquivo.

__init__(*oldFilePath*)
Construtor da classe Parser

Parâmetro de inicialização:

- oldFilePath (str) :indica o nome e a localização do arquivo que se deseja alterar desde as subpastas de /txtoutput/

A partir deste parâmetro, o construtor instancia:

- strData (str) : string que armazena o conteúdo a ser modificado e escrito no novo arquivo

__weakref__
list of weak references to the object (if defined)

binary()
Transforma todos os bits que estão no arquivo em binário.

removeSeparator(*strDeleted*)
Retiram um caractere ou uma string (separator) do conteúdo do arquivo.

•strDeleted (string): String que será retirada do arquivo

setBitsSeparation(*separator*, *qtBits*)
Método que adiciona uma string separator a cada qtBits do arquivo.

•separator (str) : String que será adicionada com o propósito de separar os bits do arquivo

•qtBits (int) : Quantidade de bits entre um separator e outro

setNewFile(*info*)
Escreve todas as mudanças realizadas no conteúdo do arquivo antigo em um novo arquivo de texto localizado em /txtoutput/treated/, com o mesmo nome do antigo, porém com as informações adicionais requeridas.

Parâmetro:

- info (str) : informação que deve ser adicionada ao nome do novo arquivo a ser gerado

A partir de self.oldFilePath e info, é instanciado o nome e a localização do novo arquivo.

- newFilePath (str) : localização e nome do novo arquivo. É instanciado utilizando o método setPath(). Arquivos criados nesta classe ficam localizados na pasta /treated/

Então, as alterações são repassadas para o novo arquivo.

setPath (*oldFilePath*, *info*)

A partir do caminho do arquivo, define o tipo do autômato celular (e a pasta na qual o novo arquivo deve ser armazenado), e o nome que o novo arquivo deve ter.

•oldFilePath (str) : indica a localização e o nome do arquivo de texto

•info (str) - O novo arquivo é salvo na pasta /treated/, subpasta do seu tipo, com o nome igual a sua regra. Caso deseje-se acrescentar outra informação, deve-se declará-la como uma string na variável info.

Classe IntKBase

class util.IntKBase.**IntKBase**(*num*, *b*)

Classe que abriga o método publicado por Mark Borgerding no dia 15/02/2010 no site StackOverflow.

Atributos da classe:

num (int) - número que que deve ser transformado b (int) - base na qual num deve ser escrito numInBase (string) - num escrito na base b

intKbase(*x*, *b*)

Retorna x na base b.

Método que transforma x em um número na base b, e o retorna.

b deve ser estar entre 2 e 32, ou seja, 2 < = b < = 32. b também pode ser 64, mas somente 64.

```
>>> IntKBase(1200, 60).numInBase
'UA'
```

```
>>> IntKBase(12, 34).numInBase
'C'
```

```
>>> IntKBase(12, 2).numInBase
'1100'
```

```
>>> IntKBase(12, 8).numInBase
'14'
```

retorna rets

Indices and Tables

- genindex
- modindex
- search

C