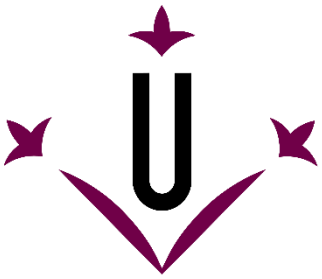
Pràctica 1: Encoding

Algorítmica i complexitat



Universitat de Lleida

Segura Paz, Aleix

Serrano Ortega, Aniol

Índex

Introducció

En aquesta pràctica s'han realitzat una sèrie de programes que donat un arxiu de text el pot codificar o descodificar. En el cas del codificador es passa una determinada clau formada per caràcters alfanumèrics que determina com ha de ser el xifratge, com a segon argument es passa l'arxiu de text a xifrar i com a últim paràmetre l'arxiu que conté la descodificació correcta realitzada en una clau determinada.

D'altra banda, en el descodificador es passen una clau de caràcters alfanumèrics que determina com s'ha de desxifrar el text que es passa com a segon paràmetre. L'últim paràmetre fa referència a l'arxiu de text desxifrat donada una determinada clau.

La implementació del codificador i del descodificador s'ha realitzat tant en versió iterativa com recursiva.

Codi iteratiu

Descripció pseudocodi

El codi del codificador en pseudocodi és el següent:

```
function encoder(T : string of char) is
      text = “”
      j = 0
      for letter in T do
            key_index ← j % len(key)
            if letter in alphabet or ALPHABET then
                  key_char ← key[key_index]
                  shift_value ← letters[key_char]
                  new_char ← alphabet[(letters[letter] + shift_value)
                                    % len(alphabet)]
                  if letter in ALPHABET then
                        new_char ← NEW_CHAR
                  text ← text + new_char
                  j ← j + 1
                  endif
            else
                  text ← text + letter
            endif
      return text
```

En l'anterior pseudocodi, T representa el text a codificar que passem per paràmetre. El text a retornar s'inicialitza com un text buit("") i per cada lletra que tingui T anirem afegint el caràcter de T codificat amb l'ajuda del caràcter de *key* que toqui en aquell moment. Inicialitzem un index j a 0 que ens servirà per a saber quin caràcter de *key* hem d'utilitzar en cada volta per xifrar el caràcter (*letter*) de T.

El recorregut de T funciona de la següent manera: primer de tot obtenim l'índex que conté el caràcter de la clau que utilitzem per xifrar. Seguidament comencem amb el tractament de la lletra de T actual a tractar. Mirem si la lletra es troba en l'alfabet de caràcters tant si és en minúscules com en majúscules o no. En el cas que no es trobi concatenarem a text aquella lletra, que realment correspondrà a un caràcter no vàlid de l'alfabet, com per exemple un espai en blanc. Per altra banda, si es troba en l'alfabet, ens guardarem a *key_char* el caràcter de la clau que utilitzem per xifrar, obtindrem el valor de desplaçament associat al caràcter i guardarem en *new_char* el caràcter xifrat resultat de desplaçar-li positivament a la lletra actual el valor de desplaçament associat al caràcter de la clau actual. Finalment mirem si pertany a l'alfabet de les majúscules, si pertany concatenem a *text* el *new_char* en majúscules i si no, en minúscules. Un cop concatenat desplacem el valor de *j* per a utilitzar el següent caràcter de la clau per a xifrar la següent lletra.

El codi del descodificador en iteratiu en pseudocodi és el següent:

```
function decoder(T : string of char) is
      text = ""
      j = 0
      for letter in T do
            key_index ← j % len(key)
            if letter in alphabet or ALPHABET then
                  key_char ← key[key_index]
                  shift_value ← letters[key_char]
                  new_char ← alphabet[(letters[letter] - shift_value)
                                    % len(alphabet)]
                  if letter in ALPHABET then
                        new_char ← NEW_CHAR
                  text ← text + new_char
                  j ← j + 1
                  endif
            else
                  text ← text + letter
            endif
      return text
```

En l'anterior pseudocodi, T representa el text a descodificar que passem per paràmetre. El text a retornar s'inicialitza com un text buit("") i per cada lletra que tingui T anirem afegint el caràcter de T descodificat amb l'ajuda del caràcter de *key* que toqui en aquell moment. Inicialitzem un index j a 0 que ens servirà per a saber quin caràcter de *key* hem d'utilitzar en cada volta per xifrar el caràcter (*letter*) de T.
El recorregut de T funciona de la següent manera: primer de tot obtenim l'índex que conté el caràcter de la clau que utilitzem per desxifrar. Seguidament comencem amb el tractament de la lletra de T actual a tractar. Mirem si la lletra es troba en l'alfabet de caràcters tant si és en minúscules com en majúscules o no. En el cas que no es trobi concatenarem a text aquella lletra, que realment correspondrà a un caràcter no vàlid de l'alfabet, com per exemple un espai en blanc. Per altra banda, si es troba en l'alfabet, ens guardarem a *key_char* el caràcter de la clau que utilitzem per xifrar, obtindrem el valor de desplaçament associat al caràcter i guardarem en *new_char* el caràcter xifrat resultat de desplaçar-li negativament a la lletra actual el valor de desplaçament associat al caràcter de la clau actual. Finalment mirem si pertany a l'alfabet de les majúscules, si pertany concatenem a *text* el *new_char* en majúscules i si no, en minúscules. Un cop concatenat desplacem el valor de *j* per a utilitzar el següent caràcter de la clau per a xifrar la següent lletra.

Cost empíric

Pel que fa al cost tant de *encoder* com de *decoder* parlant en termes generals tenim un cost de *O(n)*, on *n* és la llargada del text a xifrar o desxifrar. Per a cada caràcter del text s'afegeix una volta al bucle. Per tant, *O(n)* és el cost dels algorismes en el pitjor dels casos. Per altra banda en cas de passar-se un text buit com a paràmetre el cost seria *O(1)* ja que no entraríem en el bucle i es retornaria "".

Però podem desgranar el codi una mica més i analitzar-lo més exhaustivament. Prendrem per exemple la funció *encoder*, on tenim les següents línies de codi amb els seus respectius costos:

Codi	Cost
text = “”	O(1): assignació
j = 0	O(1): assignació
for letter in cyphered_text	O(n): bucle
key_index = j % len(key)	O(1): assignació
if letter in alphabet or letter in alphabet.upper()	O(1): accés a un diccionari clau-valor
key_char = key[key_index]	O(1): assignació per accés directe
shift_value = letters[key_char]	O(1): assignació per accés directe
new_char = alphabet[(letters[letter]+ shift_value) % len(alphabet)	O(1): assignació per accés directe
if letter in alphabet.upper()	O(1): accés a un diccionari clau-valor
new_char = new_char.upper()	O(1): assignació
text += new_char	O(1): concatenació
j += 1	O(1): suma
text += letter	O(1): concatenació
return text	O(1): retorn

Cost experimental (extra)

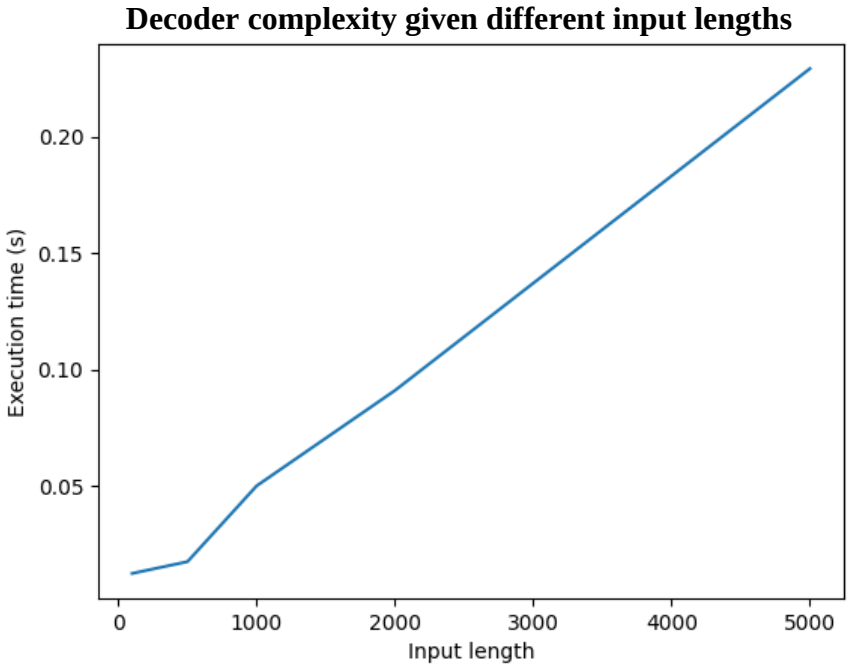


Figura1. Complexitat de la funció decoder en la versió iterativa. Font: elaboració pròpia via matplotlib.

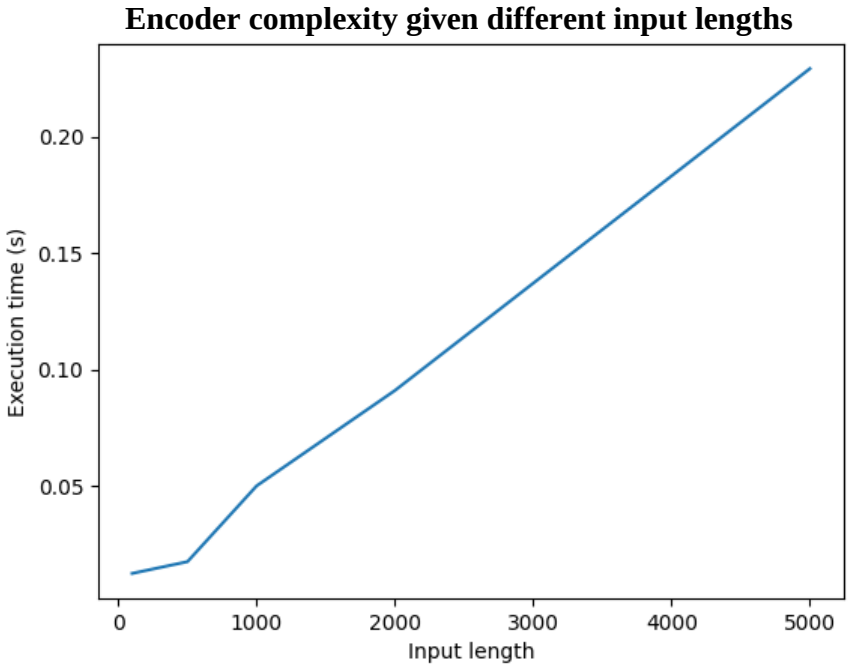


Figura2. Complexitat de la funció encoder en la versió iterativa. Font: elaboració pròpia via matplotlib.

Nota: Realització dels gràfics mitjançant les execucions:

.\decoder_it.py abc E1_encoded_text.txt E1_decoded_text.txt

.\encoder_it.py abc E1_plain_text.txt E1_encoded_text.txt

Codi recursiu

Descripció pseudocodi

El codi del descodificador en recursiu en pseudocodi és el següent:

```
function decoder(T : string of char, K: key, J: index) is
      if not T:
            return “”
      letter ← T[0]
      key_index ← j % len(key)
      if letter in alphabet or ALPHABET then
            key_char ← key[key_index]
            shift_value ← letters[key_char]
            new_char ← alphabet[(letters[letter] - shift_value)
                              % len(alphabet)]
            if letter in ALPHABET then
                  new_char ← NEW_CHAR
            endif
            return new_char + decoder(T[1:], K, J + 1)
      else
            return letter + decoder(T[1:], K, J)
      endif
```

En l'anterior pseudocodi, *T* representa el text a descodificar que passem per paràmetre, *K* la clau que utilitzem per desxifrar i *J* l'índex per seleccionar el caràcter que ajuda a desxifrar. Si es compleix que *T* és *False*, és a dir, és tracta d'un text buit, tenim aquí el nostre cas simple. On retornem "", la cadena buida.

Per altra banda tenim el cas recursiu, que es més complex. Primer de tot s'emmagatzema en *letter* el primer caràcter del text, ja que és el caràcter que toca tractar. En *key_index* guardem la posició del caràcter de la clau que toca utilitzar per a desxifrar. Seguidament avaluem si *letter* es un caràcter vàlid de l'alfabet (tant minúscules com majúscules) o simplement és un altre tipus de caràcter (com ara un espai en blanc). Els casos son ben diferents. Si és tracta d'un altre tipus de caràcter tant sols hem de retornar *letter* + la crida recursiva que fa referencia a la funció *decoder* però ara el text a xifrar es *T[1:]*, és a dir, com que ja hem tractat *letter* avancem un caràcter en el text (*T*). Per altra banda si el caràcter efectivament pertany a l'alfabet guardarem a *key_char* el caràcter de la clau que utilitzem per xifrar, obtindrem el valor de desplaçament associat al caràcter i guardarem en *new_char* el caràcter xifrat resultat de desplaçar-lo negativament a la lletra actual el valor de desplaçament associat al caràcter de la clau actual. Finalment mirem si pertany a l'alfabet de les majúscules, si pertany transformem *new_char* a majúscules, sinó, no fem res. Finalment retornem *new_char* + la crida recursiva que fa referencia a la funció *decoder* però ara el text a xifrar es *T[1:]* i també utilitzarem el següent caràcter de la clau per xifrar, és a dir, *j + 1*.

El codi de codificador en recursiu en pseudocodi és el següent:

```
function encoder(T : string of char, K: key, J: index) is
      if not T:
            return ""
      letter ← T[0]
      key_index ← j % len(key)
      if letter in alphabet or ALPHABET then
            key_char ← key[key_index]
            shift_value ← letters[key_char]
            new_char ← alphabet[(letters[letter] + shift_value)
                              % len(alphabet)]
            if letter in ALPHABET then
                  new_char ← NEW_CHAR
            endif
            return new_char + encoder(T[1:], K, J + 1)
      else
            return letter + encoder(T[1:], K, J)
      endif
```

En l'anterior pseudocodi, *T* representa el text a codificar que passem per paràmetre, *K* la clau que utilitzem per desxifrar i *J* l'índex per seleccionar el caràcter que ajuda a desxifrar. Si es compleix que *T* és *False*, és a dir, és tracta d'un text buit, tenim aquí el nostre cas simple. On retornem "", la cadena buida.

Per altra banda tenim el cas recursiu, que es més complex. Primer de tot s'emmagatzema en *letter* el primer caràcter del text, ja que és el caràcter que toca tractar. En *key_index* guardem la posició del caràcter de la clau que toca utilitzar per a desxifrar. Seguidament avaluem si *letter* es un caràcter vàlid de l'alfabet (tant minúscules com majúscules) o simplement és un altre tipus de caràcter (com ara un espai en blanc). Els casos son ben diferents. Si és tracta d'un altre tipus de caràcter tant sols hem de retornar *letter* + la crida recursiva que fa referencia a la funció *encoder* però ara el text a xifrar es *T[1:]*, és a dir, com que ja hem tractat *letter* avancem un caràcter en el text (*T*). Per altra banda si el caràcter efectivament pertany a l'alfabet guardarem a *key_char* el caràcter de la clau que utilitzem per xifrar, obtindrem el valor de desplaçament associat al caràcter i guardarem en *new_char* el caràcter xifrat resultat de desplaçar-lo positivament a la lletra actual el valor de desplaçament associat al caràcter de la clau actual. Finalment mirem si pertany a l'alfabet de les majúscules, si pertany transformem *new_char* a majúscules, sinó, no fem res. Finalment retornem *new_char* + la crida recursiva que fa referencia a la funció *encoder* però ara el text a xifrar es *T[1:]* i també utilitzarem el següent caràcter de la clau per xifrar, és a dir, *j + 1*.

Cost empíric

Pel que fa al cost tant de *encoder* com de *decoder*, ara en el cas recursiu, parlant en termes generals tenim també un cost de *O(n)*, on *n* és la llargada del text a xifrar o desxifrar. Per a cada caràcter del text s'afegeix una volta al bucle. Per tant, *O(n)* és el cost dels algorismes en el pitjor dels casos. Per altra banda en cas de passar-se un text buit com a paràmetre el cost seria *O(1)* ja que no entraríem en el bucle i es retornaria "".

Però podem desgranar el codi una mica més i analitzar-lo més exhaustivament. Prendrem per exemple la funció *decoder*, on tenim les següents línies de codi amb els seus respectius costos:

Codi	Cost
if not cyphered_text	O(1): verificació booleana
return “”	O(1): retorn
Letter = cyphered_text[0]	O(1): assignació
key_index = j % len(key)	O(1): assignació
if letter in alphabet or letter in alphabet.upper()	O(1): accés a un diccionari clau-valor
key_char = key[key_index]	O(1): assignació per accés directe
shift_value = letters[key_char]	O(1): assignació per accés directe
new_char = alphabet[(letters[letter]-shift_value) % len(alphabet)	O(1): assignació per accés directe
if letter in alphabet.upper()	O(1): accés a un diccionari clau-valor
new_char = new_char.upper()	O(1): assignació
return new_char + decoder(cyphered_text[1:], key, j + 1)	O(n): crida recursiva
O bé:	**O bé:**
return letter + decoder(cyphered_text[1:], key, j)	O(n): crida recursiva

Cost experimental (extra)

No ha estat possible realitzar la mateixa execució de codi per a generar les gràfiques de la complexitat dels algorismes en la versió recursiva donat que el intèrpret de Python llença l’excepció *RecursionError*. Tot i així es pot afirmar que les gràfiques haurien de sortir molt semblants ja que com hem vist en ambdós versions el cost és *O(n)*.

Conclusions

Tot i saber que casi sempre el codi es millorable en termes de cost i complexitat i que si re-busquem igual podrem estalviar-nos alguna variable o alguna repetició de codi creiem que la implementació que hem realitzat es prou bona, ja que el mínim cost possible per el problema que se’ns planteja(en el pitjor cas) es *O(n)* degut a que per a xifrar cada caràcter d’un text s’ha de recórrer cada un dels caràcters que formen el text i per tant *n* és la mida del text. El disseny del nostre algoritme compleix aquesta solució.