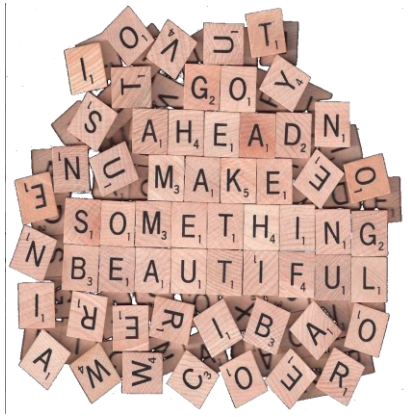
Práctica Final: Cifras y Letras



Cifras:

Dado el conjunto *6, 8, 10, 9, 4, 75* obtener el valor 835

- 8/4 = 2
- 9+2 = 11
- 75*11= 825
- 825+10 = 835

Letras:

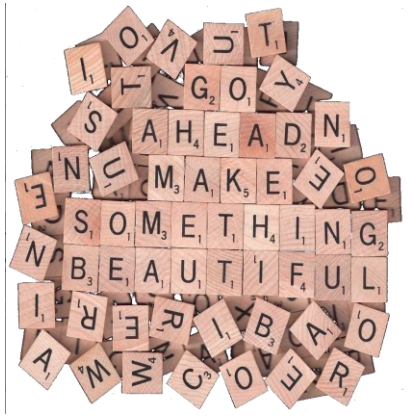
Dadas las letras

T I E O I T U S

obtener la palabra de mayor longitud

- otitis
- tiesto

Práctica Final: Cifras y Letras



Letras:

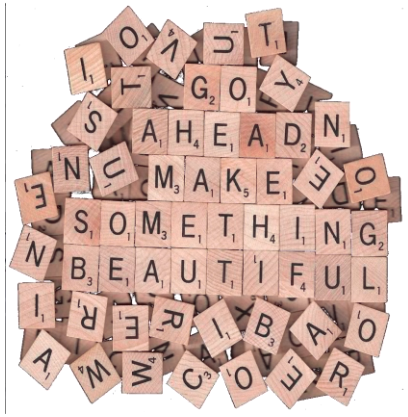
Dadas las letras

T I E O I T U S

obtener la palabra de mayor longitud

- otitis
- tiesto

Práctica Final: Cifras y Letras



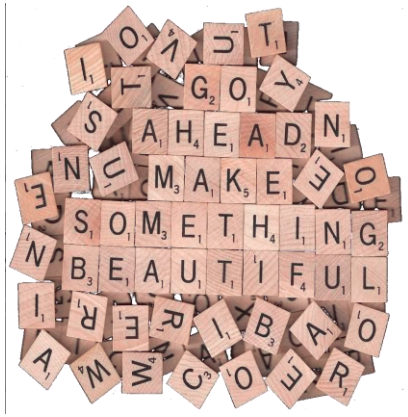
Letras. Programas a desarrollar

1. testdiccionario: Testea la clase diccionario

2. letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.

3. cantidad_letras: obtiene por cada letra el numero de veces que aparece en un diccionario y puntuación.

Práctica Final: Cifras y Letras



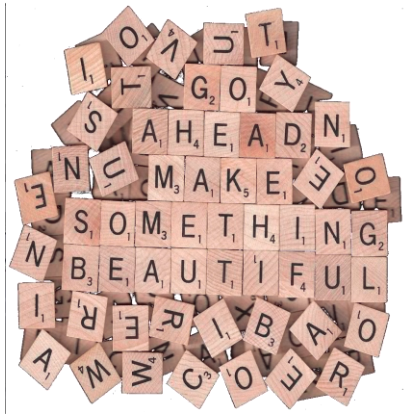
Letras. Programas a desarrollar

1. testdiccionario: Testea la clase diccionario

2. letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.

3. cantidad_letras: obtiene por cada letra el numero de veces que aparece en un diccionario y puntuación.

Práctica Final: Cifras y Letras



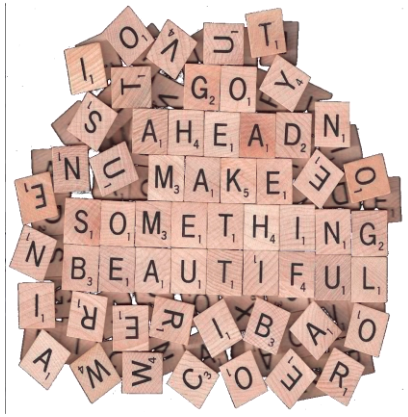
Letras. test_diccionario

TDA. Diccionario.- Almacena en memoria una secuencia de palabras válidas del idioma español.

Fichero con palabras (data/diccionario.txt)

```
a
aaronica
aaronico
ab
abab
ababillarse
ababol
abaca
abacera
abaceria
abacero
abacial
abaco
abad
abada
abadejo
...
```

Práctica Final: Cifras y Letras



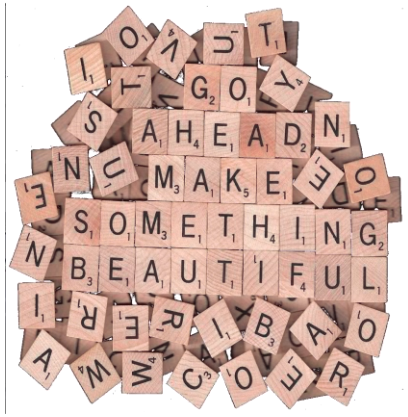
Módulos: dictionary (dictionary.cpp y dictionary.h)

class Dictionary {

private:
  set<string> words;
public:

1. Dictionary();
2. ~Dictionary(){}
3. void clear();
4. unsigned int size() const;
5. bool empty() const;
6. bool exists(const string &val);
7. bool erase(const string &val);
8. friend istream &operator>>(istream &is, Dictionary &dic);
9. friend ostream &operator<<(ostream &os, const Dictionary &dic);
10. int getOccurrences(const char c)const;
11. int getTotalLetters()const;
12. vector<int>getWordsLength(int length);
13. class iterator {
14. …
15. };//class iterator
16. class const_iterator {
17. …
18. };//class const_iterator
19. iterator find(const string & w);
20. pair<iterator,bool> insert(const string &val);
21. pair<iterator, iterator> range_prefix(const string
22. &val);
23. iterator begin() ;
24. const_iterator begin()const;
25. iterator end() ;
26. const_iterator end()const;
27. };//class Dictionary

Práctica Final: Cifras y Letras

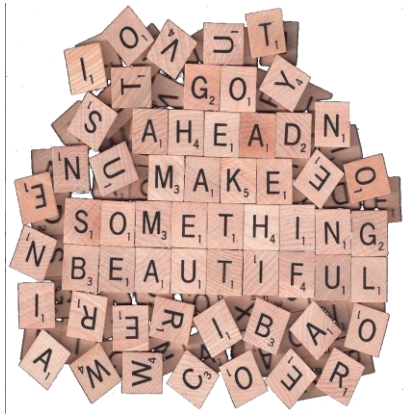


Módulos: dictionary (dictionary.cpp y dictionary.h)

```
class Dictionary {

private:
  set<string> words;
public:
 …
 class iterator{
 private:
  set<string>::iterator it;
 public:
   iterator();
   bool operator==(const iterator &i)const;
   bool operator!=(const iterator &i)const;
   iterator& operator++();
   const string & operator *();
  …
 };
}
```

Práctica Final: Cifras y Letras

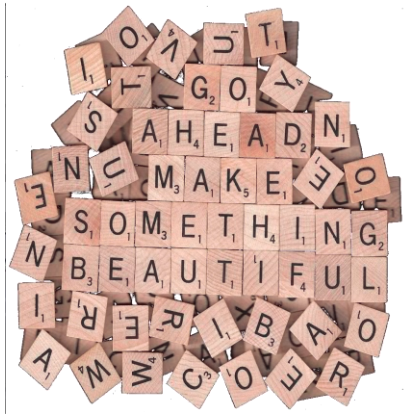


Módulos: dictionary (dictionary.cpp y dictionary.h)

```
class Dictionary {

private:
  set<string> words;
public:
 …
 class const_iterator{
 private:
  set<string>::const_iterator it;
 public:
   const_iterator();
   bool operator==(const const_iterator &i)const;
   bool operator!=(const const_iterator &i)const;
   const_iterator& operator++();
   const string & operator *();
  …
 };
}
```

Práctica Final: Cifras y Letras



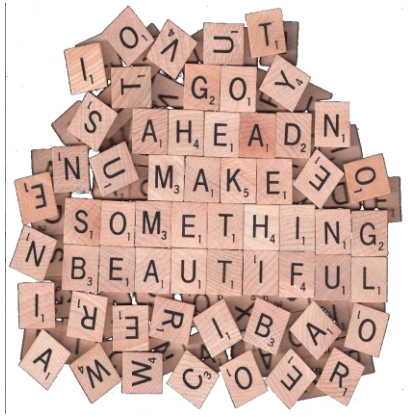
Letras. Programas a desarrollar

1. testdiccionario: Testea la clase diccionario

prompt% testdiccionario spanish

El fichero testdiccionario.cpp lo tenéis en la carpeta src.

Práctica Final: Cifras y Letras



Letras. Programas a desarrollar

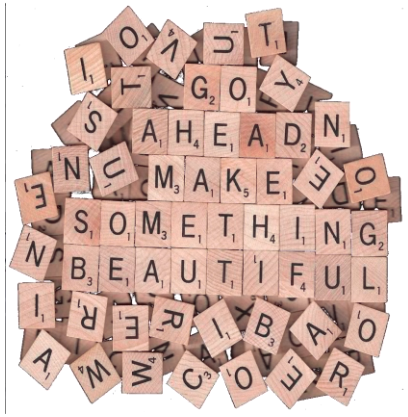
1. testdiccionario: Testea la clase diccionario

2. letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.

3. cantidad_letras: obtiene por cada letra el numero de veces que aparece en un diccionario y puntuación.

Práctica Final: Cifras y Letras

letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.



Las letras son**: T I E O I T U S**
Dime tu solucion: tieso
tieso Puntuacion: 5

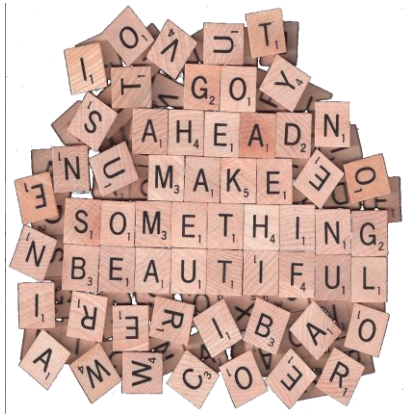
Mis soluciones son:
otitis Puntuacion: 6
tiesto Puntuacion: 6

Mejor Solucion:tiesto

¿Quieres seguir jugando[S/N]?N

Práctica Final: Cifras y Letras

letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.



Las letras son**: T I E O I T U S**
Dime tu solucion: tieso
tieso Puntuacion: 5

Mis soluciones son:
otitis Puntuacion: 6
tiesto Puntuacion: 6

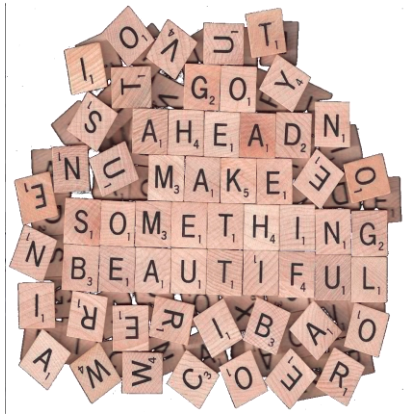
Mejor Solucion:tiesto

¿Quieres seguir jugando[S/N]?N

prompt% letras data/diccionario.txt data/letras.txt 8 L

Práctica Final: Cifras y Letras

letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.

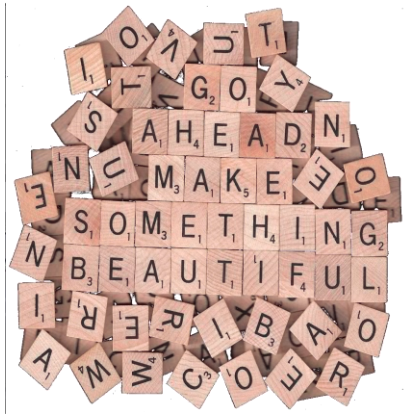


Fichero con palabras (data/diccionario.txt)

```
a
aaronica
aaronico
ab
abab
ababillarse
ababol
abaca
abacera
abaceria
abacero
abacial
abaco
abad
abada
abadejo
…
```

Práctica Final: Cifras y Letras

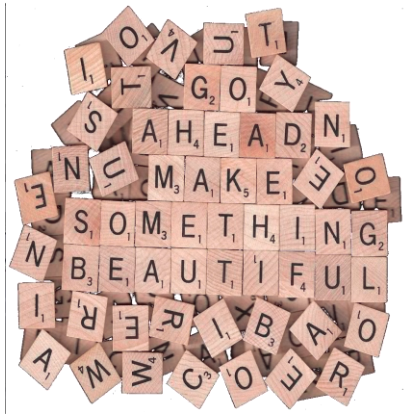
letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.



Fichero con letras(data/letras.txt)

#Letra	Cantidad	Puntos
A	16	1
B	2	3
C	6	3
D	5	2
E	10	1
F	2	4
G	2	2
H	1	4
I	9	1
J	1	8
K	1	6
L	5	1
M	4	3
N	7	1
O	10	1
P	3	3
Q	1	5
R	10	1
S	5	1
T	6	1
U	4	1
V	2	4
W	1	8
X	1	8
Y	1	4
Z	1	10

Práctica Final: Cifras y Letras

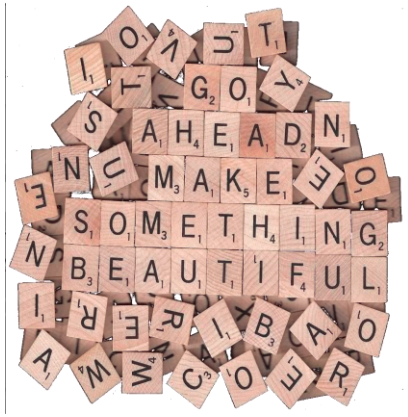


letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.

Módulos. Dictionary, LettersSet, LettersBag, Solver

Práctica Final: Cifras y Letras

letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.



Módulos. Dictionary, **LettersSet**, LettersBag, Solver

```
class LettersSet{
private:
  map<char, LetterInfo> charSet;
…
```

```
structLetterInfo
  int cantidad;
  int puntos;
```

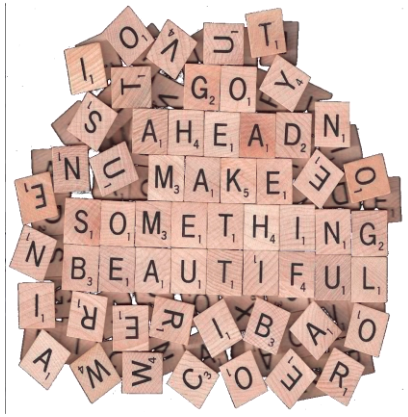
se almacena en

#Letra	Cantidad	Puntos
A	16	1
B	2	3
C	6	3
D	5	2
E	10	1
F	2	4
G	2	2
H	1	4
I	9	1
J	1	8
…		

Fichero

Práctica Final: Cifras y Letras

letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.



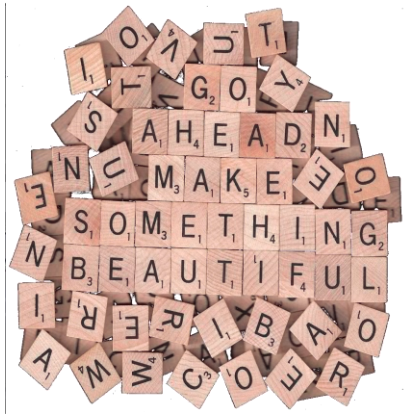
Módulos. Dictionary, LettersSet, **LettersBag**, Solver

```
class LettersBag {
private:
  vector<char> letters;
….
```

Contiene tantas letras como indica una instancia de la clase LettersSet. Una letra se repite tantas veces como indica su cantidad.

Práctica Final: Cifras y Letras

letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.



Módulos. Dictionary, LettersSet, **LettersBag**, Solver

```
class LettersBag {
private:
  vector<char> letters;
....
```

Contiene tantas letras como indica una instancia de la clase LettersSet. Una letra se repite tantas veces como indica su cantidad.

#Letra	Cantidad	Puntos
A	16	1
B	2	3
C	6	3
D	5	2
E	10	1
F	2	4
G	2	2
H	1	4
I	9	1
J	1	8
…		

Fichero:letras.txt

```
LettersSet Ls;
ifstream fin ("letras.txt");
fin>>Ls;
LetterBag Lb(Ls);
```

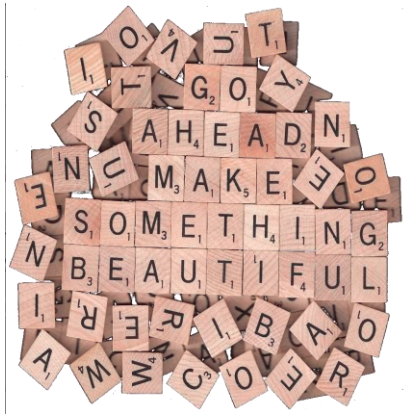
16

Lb.letters contendrá

{A,A,….A,B,B,C,C,..C,D,D…D,…..J…}

Práctica Final: Cifras y Letras

letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.



Módulos. Dictionary, LettersSet, **LettersBag**, Solver

```
class LettersBag {
private:
  vector<char> letters;
….
```

Métodos de interés:

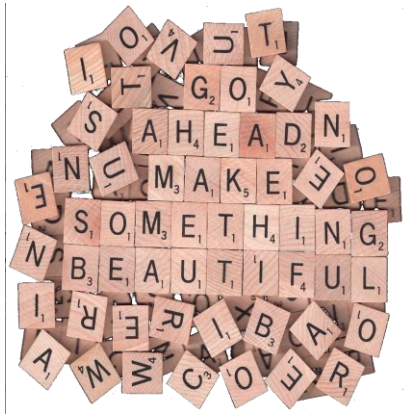
- LettersBag(const LettersSet &letterSet);
- char extractLetter()
- vector<char> extractLetter(int num)

Otros métodos típicos:

- size
- insert
- erase
- clear
- etc

Práctica Final: Cifras y Letras

letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.



Módulos. Dictionary, LettersSet, LettersBag, **Solver**

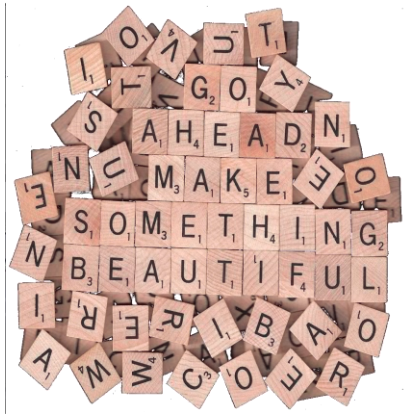
```
class Solver{
private:
  Dictionary dictionary;
  LettersSet letters_set;
....
```

TDA Solver ofrece las soluciones del juego de Letras. Para ello dado un diccionario, un conjunto de letras, permite adaptar diferentes configuraciones del juego como

- El número de letras aleatorios para jugar
- El modo de juego:
  - Longitud (L). Busca la solución de mayor longitud
  - Puntuación (P).Busca la palabra que sumando los puntos de las letras que la constituyen acumulan el valor más alto.

Práctica Final: Cifras y Letras

letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.



Módulos. Dictionary, LettersSet, LettersBag, **Solver**

```
class Solver{
private:
  Dictionary dictionary;
  LettersSet letters_set;
....
```

getSolutions(const vector<char> & available_letters,bool score_game)

Input

available_letters: letras escogidas aleatoriamente de una LettersBag

score_game: false si se juega por longitud, true por puntos

Output

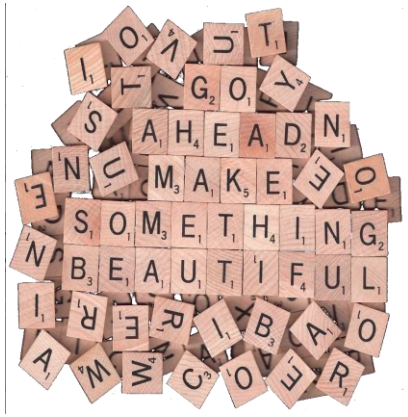
out: vector de string

Pasos a seguir:

1. Por cada palabra p en el diccionario
    1. Si se puede **construir** con available letters
        - Si es mejor a la mejor solución vaciar el vector out y poner la palabra p en out
        - Si p es igual en longitud a la mejor solución añadir p a out
2. Devolver out

Práctica Final: Cifras y Letras

letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.



Módulos. Dictionary, LettersSet, LettersBag, **Solver**

```
class Solver{
private:
  Dictionary dictionary;
  LettersSet letters_set;
....
```

getSolutions(const vector<char> & available_letters,bool score_game)

Input

available_letters: letras escogidas aleatoriamente de una LettersBag

score_game: false si se juega por longitud, true por puntos

Output

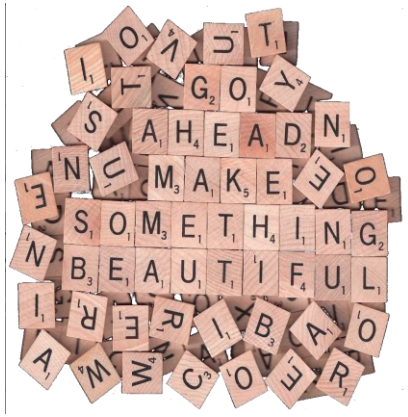
out: vector de string

Pasos a seguir:

1. Por cada palabra p en el diccionario
    1. Si se puede **construir** con available letters
        - Si es mejor a la mejor solución vaciar el vector out y poner la palabra p en out
        - Si p es igual en longitud a la mejor solución añadir p a out
2. Devolver out

PENSAR EN OPTIMIZACIONES

Práctica Final: Cifras y Letras



Letras. Programas a desarrollar

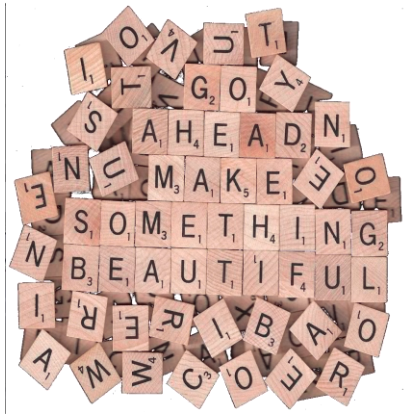
1. testdiccionario: Testea la clase diccionario

2. letras : dada una secuencia de letras obtiene la palabra válida de mayor longitud o puntuación.

3. cantidad_letras: obtiene por cada letra el número de veces que aparece en un diccionario y puntuación.

Práctica Final: Cifras y Letras

Letras. cantidad_letras



Ficheros de entrada

a
aaronica
aaronico
ab
abab
ababillarse
…

diccionario.txt

#Letra	Cantidad	Puntos
A	16	1
B	2	3
C	6	3
D	5	2
E	10	1
F	2	4
G	2	2
H	1	4
I	9	1
J	1	8
…		

letras.txt

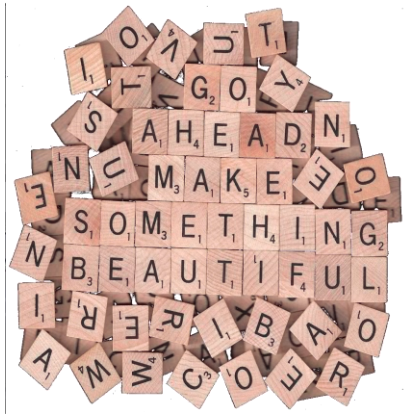
cantidad_letras

salida.txt

Letra	Cantidad	Puntos
a	96567	1
b	12873	4
c	37858	2
d	25903	3
e	65755	2
f	7424	4
g	11634	4
…		
w	36	10
x	1159	6
y	1119	6
z	5447	4

Práctica Final: Cifras y Letras

Letras. cantidad_letras



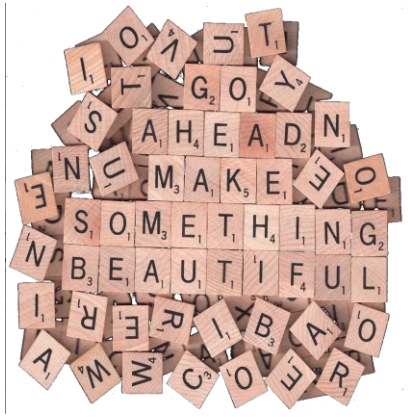
L: LettersSet

D: Dictionary

Pasos a seguir.-

1. min = infinito
2. Para cada letra l en L
3. Obtener el número de apariciones de l en D. Sea este numero a(l)
4. Obtener la proporción de la letra con respecto al total de apariciones para todas las letras
5. porcentaje(l) = a(l)/total
   if (min>porcentaje(l)) min = porcentaje(l)
6. min_log= -log10(min)
7. Para cada letra l en L
8. Escribir en el fichero de salida la tripleta
9. **(l,a(l), 10*(-log10(porcentaje(l))/min_log)**

Práctica Final: Cifras y Letras



Cifras

Inputs

C=conjunto de números {1,2,3,4,5,6,7,8,9, 10, 25, 50, 75, 100}

S=conjunto de 6 números extraídos de C de forma aleatoria

objetivo= numero como máximo de tres cifras, que se pretende lograr con S y operaciones aritmeticas {+,-,*,/}

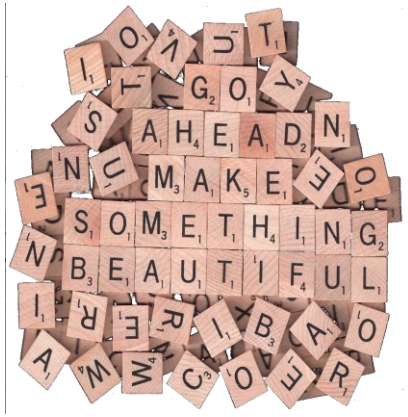
best: mejor solución hasta el momento. Se compone de ***valor*** alcanzado y ***operaciones*** acumuladas.

actual: la solución que tenemos hasta el momento. Se compone del ***valor*** alcanzado y **operaciones** acumuladas

Cifras (S,objetivo,actual,best)

1. Si **actual**.valor == **objetivo**
2. **best** = **actual**
3. return
4. Si actual ***es mejor*** que best
5. **best** = **actual**
6. Para cada numero **n** en **S**
7. **restantes** = **S**-{**n**}
8. operaciones =GeneraOperaciones(**actual**, **n**)
9. Para cada operación **op** en operaciones
10. Cifras(**restantes,objetivo,op,best**)

Práctica Final: Cifras y Letras



Cifras

GeneraOperaciones (actual, n)

return {<actual.valor+n,actual.operaciones+”actual.valor+n”>,

<actual.valor-n,actual.operaciones+”actual.valor-n”>,

<actual.valor*n,actual.operaciones+”actual.valor*n”>,

<actual.valor/n,actual.operaciones+”actual.valor/n”>}

Consideraciones con GeneraOperaciones:

1. Si la división no es exacta no se incluyen en GeneraOperaciones
2. Si se divide por 0 no se incluye en GeneraOperaciones
3. Si se obtiene una diferencia negativa no se incluye en GeneraOperaciones