Programação II
+
Estruturas de Dados para Bioinformática

Dicionários e sets

Hugo Pacheco

DCC/FCUP
22/23

Dicionários

- Dicionários são outro tipo composto para **tabelas de associação**, analogia com dicionário Português->Inglês

- Cada **chave** mapeia num (e só num) **valor**

    - **chaves** têm que ser valores de tipos imutáveis (strings, ints, floats, tuplos ou combinações dos mesmos)

    - **valores** podem ser tipos compostos, e.g., uma palavra em Português associada a uma lista de possíveis traduções em Inglês

- Só estamos interessados na associação, ordem não interessa e pode variar!

- São objetos **mutáveis**, tal como listas

- Algoritmos eficientes para pesquisa, inserção, remoção, etc

Dicionários

- Associar frutas a quantidades
- Porque não apenas listas de pares?

Fruta	Quantidade
bananas	25
peras	10
laranjas	5

```
[("bananas",25),("peras",10),("laranjas",5)]
```

- Problemas:
  - algoritmos mais complicados e ineficientes porque ordem conta
  - é necessário percorrer a lista toda
  - lista não garante unicidade de chaves

```
[("bananas",10),("peras",10),("laranjas",5),("bananas",15)]
```

Dicionários (operações)

```
d = {"bananas":25,"peras":10,"laranjas":5}
```

- Pesquisa: obter o valor associado a uma chave

```
>>> d['bananas']
25
>>> d['kiwis']
KeyError: 'kiwis'
```

- Atualização/Inserção: altera o valor associado a uma chave

```
>>> d['bananas'] = d['bananas']+10
>>> d['kiwis'] = 5
>>> d
{'bananas': 35,…, 'kiwis': 5}
```

- Pertença: testa se uma chave existe no dicionário

```
>>> 'limao' in d
False
>>> 'bananas' in d
True
```

Dicionários (operações)

```
d = {"bananas":25,"peras":10,"laranjas":5}
```

- Valores por omissão

```
>>> d.get("bananas",0)
25
>>> d.get("kiwis",0)
0
```

- Apagar chave

```
>>> del d["peras"]
>>> d
{"bananas":25,"laranjas":5}
```

- Lista de chaves/valores

```
>>> list(d.keys())
['bananas','laranjas']
>>> list(d.values())
[25,5]
>>> list(d.items())
[('bananas',25),('laranjas',5)]
```

- Limpar dicionário

```
>>> d = {}
>>> d.clear()
```

Dicionários (iteração)

```
d = {"bananas":25,"peras":10,"laranjas":5}
```

- Há diferentes formas de percorrer um dicionário

```
# percorrer as chaves (por defeito)
for k in d:
    print(k,d[k])

# percorrer as chaves ordenadas
for k in sorted(d):
    print(k,d[k])

# percorrer pares (chave,valor)
for k,v in d.items():
    print(k,v)
```

Exemplo *Os Lusíadas*

- Contar o número de ocorrências de cada letra

```
# lê estrofes de ficheiro
with open('estrofes.txt','r') as f:
    texto = f.read()

# número de ocorrências por caracter
count = {}
for c in texto:
    if c.isalpha():
        count[c] = 1 + count.get(c,0)
print(count)
```

- Maiúsculas e acentos são caracteres diferentes

```
{'A': 1222, 'a': 29815, 'o': 26409, 'õ': 113,...}
```

Exemplo *Os Lusíadas*

- Normalizar caracteres convertendo em minúsculas e manipulando a representação unicode (package *unidecode*)

- Contar o número de ocorrências de cada letra outra vez

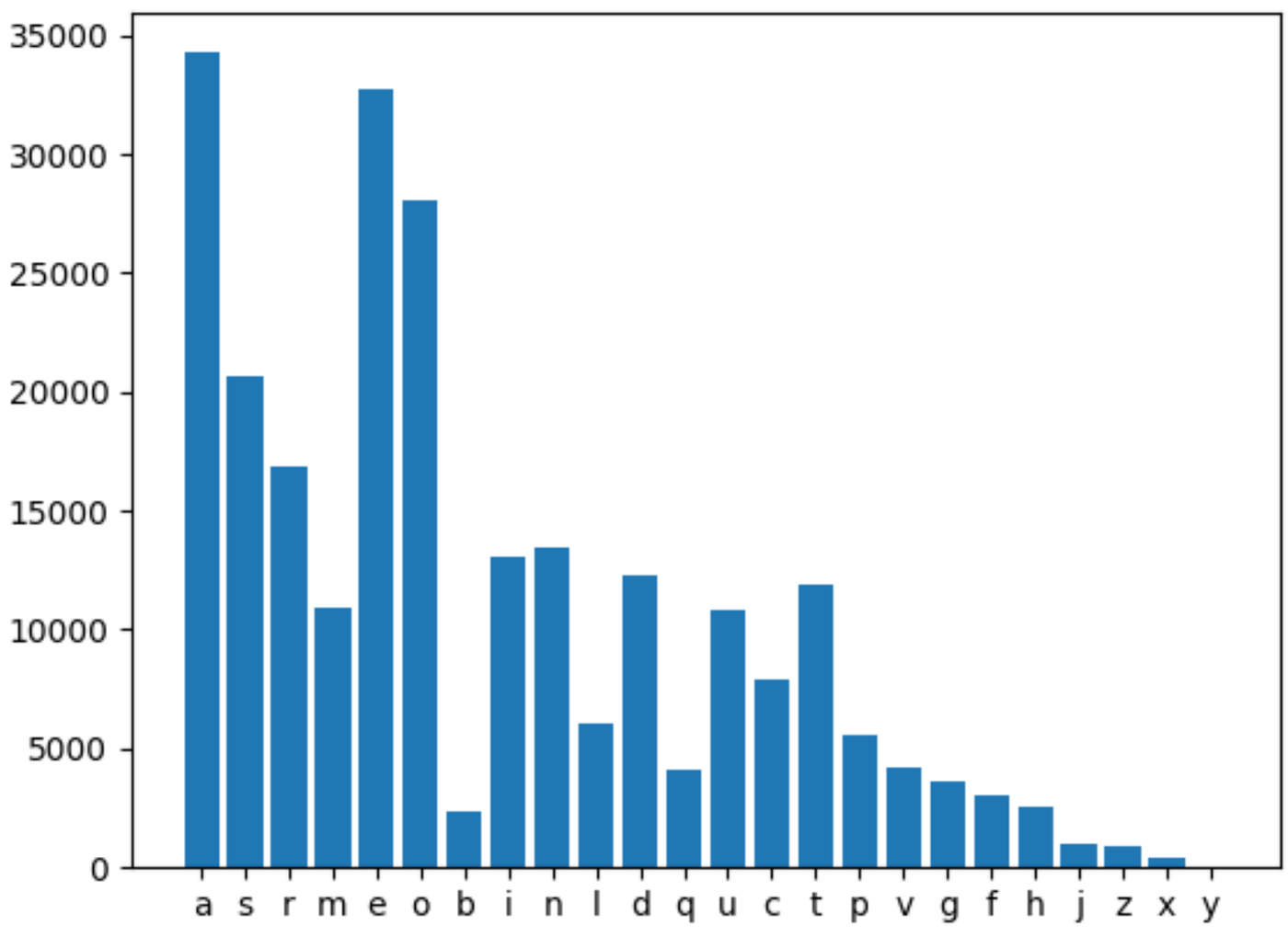
```
import unidecode as uni
def normaliza(c):
    return (uni.unidecode(c.lower()))

count = {}
for c in texto:
    if c.isalpha():
        c = normaliza(c)
        count[c] = 1 + count.get(c,0)
print(count)
```

Exemplo *Os Lusíadas*

- Desenhar um histograma (vamos estudar gráficos mais tarde)

```
import matplotlib.pyplot as plt
plt.bar(count.keys(),
count.values())
plt.show()
```



Conjuntos (sets)

- Sets são outro tipo composto para **conjuntos** tal como na matemática

- **Não** têm ordem **nem** repetidos

- São objetos **mutáveis**, tal como listas e dicionários

- São essencialmente dicionários sem valores

- Algoritmos eficientes para pertença, inserção, remoção, etc

Sets

- Criar sets

```
set() # {} está reservado para dicts
{1,2,3} # set não vazio
set([1,2,4.5]) # conversão de sequência
set((1,2,True)) # conversão de sequência
set("abcd") # conversão de sequência
```

- Elimina repetidos por defeito

```
>>> set([1,2,4,2,4])
{1,2,4}
```

Sets (operações)

```
s = {'bananas','peras','laranjas'}
```

- Como não têm ordem, indexação **não** tem significado ⇒ erro

```
>>> s['peras']
TypeError: 'set' object is not subscriptable
```

- Inserção

```
>>> s.add('kiwis')
{'kiwis', 'bananas', 'peras'}
>>> s.update(['abacate','diospiro'])
{'diospiro', 'peras', 'bananas', 'abacate', 'kiwis'}
```

- Remoção

```
>>> s.discard('peras')
{'abacate', 'bananas', 'diospiro', 'kiwis'}
>>> s.discard('laranja') #ignora se não existir
{'abacate', 'bananas', 'diospiro', 'kiwis'}
>>> s.remove('laranja') #erro se não existir
KeyError: 'laranja'
```

Sets (operações)

- Operações matemáticas de conjuntos
  - União ($S_1 \cup S_2$) `s1 | s2`
  - Interseção ($S_1 \cap S_2$) `s1 & s2`
  - Diferença (S_1 / S_2) `s1 - s2`
  - Pertença ($x \in S$) `x in s`
  - Subset ($S_1 \subseteq S_2$) `s1.issubset(s2)`

Dicionários como sets

- É possível definir operações análogas de conjuntos sobre dicionários
- **União**: temos que decidir o que fazer com valores repetidos

```
# função f une 2 valores repetidos
def uniao(f,d1,d2):
    d = d1.copy()
    for k in d2:
        if k in d: d[k] = f(d[k],d2[k])
        else: d[k] = d2[k]
    return d

# união de dois dicionários somando valores
>>> d1 = {'a':1,'b':2,'c':3}
>>> d2 = {'a':2,'c':1,'d':1}
>>> uniao(lambda x,y:x+y,d1,d2)
{'a': 3, 'b': 2, 'c': 4, 'd': 1}
```

Dicionários como sets

- É possível definir operações análogas de conjuntos sobre dicionários
- **Interseção**: temos que decidir o que fazer com valores repetidos

```
# função f une 2 valores repetidos
def intersecao(f,d1,d2):
    d = {}
    for k in d1:
        if k in d2: d[k] = f(d1[k],d2[k])
    return d

# interseção de dois dicionários somando valores
>>> d1 = {'a':1,'b':2,'c':3}
>>> d2 = {'a':2,'c':1,'d':1}
>>> intersecao(lambda x,y:x+y,d1,d2)
{'a': 3, 'c': 4}
```

Dicionários como sets

- É possível definir operações análogas de conjuntos sobre dicionários
- **Diferença**:

```
def diferenca(d1,d2):
    d = {}
    for k in d1:
        if k not in d2: d[k]=d1[k]
    return d
```

```
>>> d1 = {'a':1,'b':2,'c':3}
>>> d2 = {'a':2,'c':1,'d':1}
>>> diferenca(d1,d2)
{'b': 2}
```

Exemplo (pangrama)

- Um pangrama é uma palavra que utiliza todas as letras do alfabeto
- E.g., em Português: *“Fidel exporta whiskey, vinho, queijo azul, caju, manga e nabo.”*

```
def pangrama(frase):
    alfabeto = set("abcdefghijklmnopqrstuvwxyz")
    chars = set(normaliza(frase))
    return alfabeto.issubset(chars)
```

Exemplo (heterograma)

- Um heterograma é uma palavra que utiliza cada letra do alfabeto no máximo uma vez

- E.g., em Inglês: *“The big dwarf only jumps.”*

```
def alpha(c): return c.isalpha()
def heterograma(frase):
    chars = list(filter(alpha,normaliza(frase)))
    return len(set(chars)) == len(chars)
```

Exemplo *Os Lusíadas*

- Estudar o vocabulário utilizado em *Os Lusíadas*

```
# lê estrofes de ficheiro
with open('estrofes.txt','r') as f:
    texto = f.read()

# set de palavras
vocabulario = set(normaliza(texto).split())
```

Exemplo (DNA→RNA)

- Converter uma sequência de DNA em RNA

```
bases =  {'A':'U', 'T':'A', 'C':'G', 'G':'C'}
def base2rna(b): return bases.get(b)
def dna2rna(dna):
    return ''.join(map(base2rna,dna))

print(dna2rna('ATCG'))
# UAGC
```

Exemplo (Gene Ontology)

- Considerem uma Gene Ontology de exemplo, retirada deste tutorial
  - Uma linha por gene
  - PZid GO:id nome

```
PZ7180000020811_DVU     GO:0003824      GJ12748 [Drosophila virilis]
PZ7180000020752_DVU     GO:0003824      GI16375 [Drosophila mojavensis]
PZ7180000034678_DWY     GO:0003824      hypothetical protein YpF1991016_1335 [Ye
PZ7180000024883_EZN     GO:0006548      sjchgc01974 protein
PZ7180000024883_EZN     GO:0004252      sjchgc01974 protein
PZ7180000024883_EZN     GO:0004500      sjchgc01974 protein
```

- Encontrar gene com maior número de identificadores

```
with open('PZ.annot.txt','r') as f:
    linhas = f.read().splitlines()

genes = {}
for linha in linhas:
    id,go,name = linha.split('\t')
    genes[name] = genes.get(name,set()) | {id,go}

maxg = max(genes,key=lambda g: len(genes[g]))
print(maxg,genes[maxg])
```