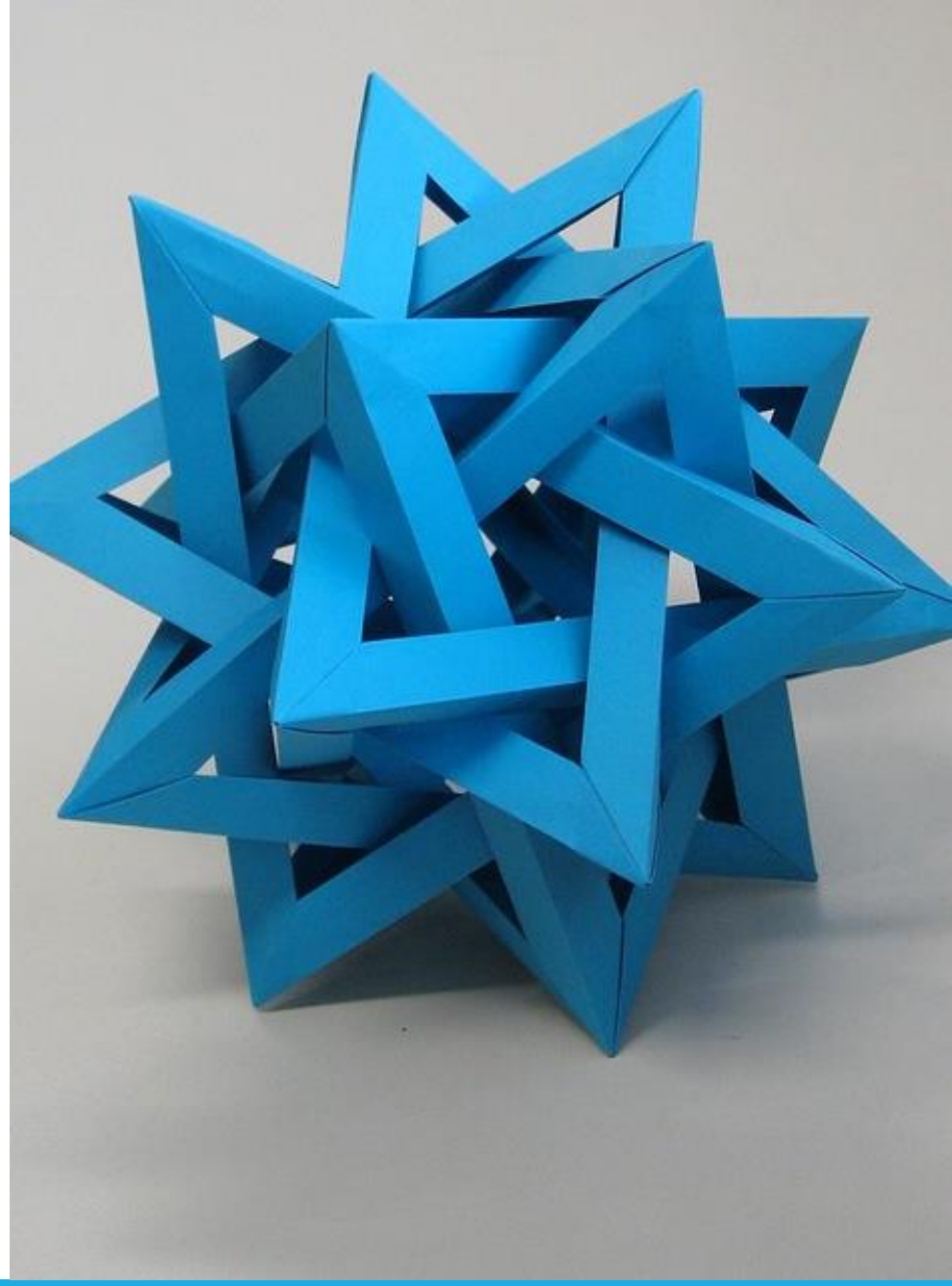
Politecnico di Torino

Dipartimento
di Automatica e Informatica

Laboratorio 11

STRUTTURE COMPLESSE DI DATI, OPERAZIONI SUGLI INSIEMI



Esercizio 1

Esercizio 1. Scrivete un programma che conti le occorrenze di ciascuna parola presente in un file di testo. Successivamente, migliorate il programma in modo che visualizzi le 100 parole più comuni (in caso di parità alla posizione 100, è indifferente che parola si stampa). [P8.2] [P8.3]

Esercizio 1 – *il codice Python*

```
# Read the file name from the user and open the file.
filename = input("Enter the name of a file: ")
inf = open(filename, "r")

# Create a new empty dictionary.
counts = {}

# Count the words in the file.
for line in inf:
    words = line.split()
    for word in words:
        if word in counts:
            counts[word] = counts[word] + 1
        else:
            counts[word] = 1

# Close the file.
inf.close()

# Display the counts.
for word in sorted(counts):
    print("%s: %d" % (word, counts[word]))
```

Esercizio 1 – *il codice Python*

```python
# Read the file name from the user and open the file.
filename = input("Enter the name of a file: ")
inf = open(filename, "r")

# Create a new empty dictionary.
counts = {}

# Count the words in the file.
for line in inf:
    words = line.split()
    for word in words:
        if word in counts:
            counts[word] = counts[word] + 1
        else:
            counts[word] = 1

# Close the file.
inf.close()

# Display the counts.
for word in sorted(counts):
    print("%s: %d" % (word, counts[word]))
```



```python
# Read the file name from the user and open the file.
filename = input("Enter the name of a file: ")
inf = open(filename, "r")

# Create a new empty dictionary.
counts = {}

# Count the words in the file.
for line in inf:
    words = line.split()
    for word in words:
        if word in counts:
            counts[word] = counts[word] + 1
        else:
            counts[word] = 1

# Close the file.
inf.close()

# Find the count for the 100th word.
sorted_counts = sorted(counts.values())
if len(sorted_counts) < 100:
    count_100 = 1
else:
    count_100 = sorted_counts[len(sorted_counts) - 100]

# Display the counts of all words with frequencies about the 100th word.
count = 0
for word in sorted(counts):
    if counts[word] > count_100:
        print("%s: %d" % (word, counts[word]))
        count = count + 1

# Display enough words with frequency equal to the 100th word so that a total
# of 100 words are displayed.
for word in sorted(counts):
    if counts[word] == count_100 and count < 100:
        print("%s: %d" % (word, counts[word]))
        count = count + 1
```

Esercizio 2

Esercizio 2. Scrivete un programma che chieda all'utente di fornire due stringhe, per poi visualizzare (evitando ripetizioni di caratteri nella stampa):

- i caratteri che compaiono in entrambe le stringhe;
- i caratteri che compaiono in una stringa ma non nell'altra;
- le lettere che non compaiono in nessuna delle due stringhe.

Suggerimento: trasformare una stringa in un insieme di caratteri. [P8.9]

Esercizio 2 – *il codice Python*

```python
# Gather input from the user and store them as sets.
set1 = set(input("Enter the first string: ").upper())
set2 = set(input("Enter the second string: ").upper())

# Find and display the characters common to both strings.
common = set1.intersection(set2)
print("The characters that are in both strings:")
for ch in sorted(common):
    print(ch, end=" ")
print()

# Find and display the characters in string 1 but not string 2.
diff = set1.difference(set2)
print("The characters in string 1 that are not in string 2:")
for ch in sorted(diff):
    print(ch, end=" ")
print()

# Find and display the characters that are not in either string.
all_chars = set("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
print("The letters that are not in either string:")
neither = all_chars.difference(set1.union(set2))
for ch in sorted(neither):
    print(ch, end=" ")
print()
```

Esercizio 3

Esercizio 3. Un array sparso è una sequenza di numeri, la maggior parte dei quali è zero. Un modo efficiente per memorizzare un array sparso è un dizionario, nel quale le chiavi sono le posizioni in cui sono presenti valori diversi da zero, e i valori sono i corrispondenti valori nella sequenza. Per esempio, la sequenza 0 0 0 0 0 4 0 0 0 2 9 sarebbe rappresentata dal dizionario `{5:4, 9:2, 10:9}`. Scrivete una funzione, `sparseArraySum`, i cui argomenti sono due di tali dizionari, a e b, e che restituisca un array sparso che ne sia il vettore *somma*: il suo valore nella posizione i è la somma dei valori di a e b nella posizione i. I dizionari del programma chiamante *non* devono essere modificati. [P8.22]

Esercizio 3 – *il codice Python*

```python
def main():
    a = {5: 4, 9: 2, 10: 9}
    b = {5: 4, 8: 4, 10: 4}

    print("a is", a)
    print("b is", b)
    print("The sum of a and b is:", sparseArraySum(a, b))


## Compute the sum of two sparse arrays.
#  @param a the first array to include in the sum
#  @param b the second array to include in the sum
def sparseArraySum(a, b):
    retval = dict(a)
    for key in b:
        if key in retval:
            retval[key] = retval[key] + b[key]
        else:
            retval[key] = b[key]
    return retval


# Call the main function.
main()
```

Esercizio 4

Esercizio 4. Realizzate il *crivello di Eratostene*: una funzione che calcola i numeri primi, nota anche nell'Antica Grecia. Scegliete un numero intero, n: questa funzione calcolerà tutti i numeri primi fino a n. Per prima cosa, inserite in un insieme tutti i numeri interi da 2 a n, poi eliminate dall'insieme tutti i multipli di 2, tranne 2 (cioè 4, 6, 8, 10, 12, …), quindi eliminate tutti i multipli di 3, tranne 3 (cioè 6, 9, 12, 15, …) e proseguite così, cancellando ogni volta i multipli del valore minimo presente nell'insieme, fino al numero $\sqrt{n}$. I numeri rimasti nell'insieme sono quelli richiesti. [P8.4]

Esercizio 4 – *il codice python*

```python
from math import sqrt, floor

# Read input from the user.
n = int(input("Enter a positive integer: "))

# Start with all numbers from 2 to n.
primes = set(range(2, n + 1))
for i in range(2, floor(sqrt(n)) + 1):
    for j in range(i**2, n+1, i):
        primes.discard(j)

# Display primes.
print("The primes less than or equal to", n, "are:")
for p in sorted(primes):
    print(" ", p)
```

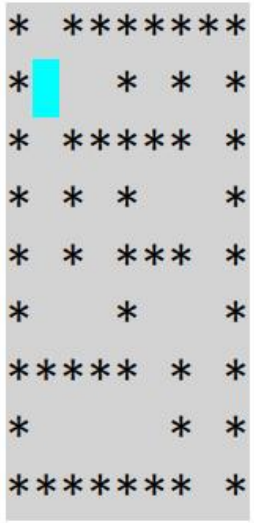
Esercizio 5

Esercizio 5. Scrivete un programma “censore”, che legga un file (bad_words.txt) contenente “parolacce”, come “sesso”, “droga”, “C++” e così via, inserendole in un insieme, per poi leggere un altro file di testo: il programma deve riscrivere il file letto, generandone un altro nel quale tutte le lettere di ciascuna parolaccia (comprese quelle nelle sotto-parole contenenti parolacce) siano state sostituite da un numero di asterischi pari alla sua lunghezza. [P8.14]

Esercizio 5 – *il codice python*

Esercizio 7

Esercizio 7. Scrivete un programma che legga un file di testo contenente l'immagine di un labirinto, come questo:



dove gli asterischi sono muri invalicabili e gli spazi sono corridoi percorribili. Generate un dizionario le cui chiavi siano tuple del tipo (riga, colonna) delle posizioni dei corridoi e i cui valori siano insiemi di posizioni di corridoi adiacenti. Nell'esempio di labirinto qui presentato, (1, 1) (quadratino azzurro) ha come corridoi adiacenti {(1, 2), (0, 1), (2, 1)}. Visualizzate il dizionario. [P8.20]

Esercizio 7 – *il codice python*

```python
## LAB11_es7
#
# Apre il file e lo legge
inf = open("maze.txt", "r")
maze = []
for line in inf:
    maze.append(line.rstrip())

# Chiude il file
inf.close()

# Crea il dizionario
positions = {}
for row in range(len(maze)) :
    for col in range(len(maze[row])) :
        if maze[row][col] == " " :
            positions[(row, col)] = set()
            # Controlla a sinistra
            if row > 0 and maze[row - 1][col] == " " :
                positions[(row, col)].add((row - 1, col))
            # Controlla a destra
            if row < len(maze) - 1 and maze[row + 1][col] == " " :
                positions[(row, col)].add((row + 1, col))
            # Controlla in alto
            if col > 0 and maze[row][col - 1] == " " :
                positions[(row, col)].add((row, col - 1))
            # Controlla in bassp
            if col < len(maze[row]) - 1 and maze[row][col + 1] == " " :
                positions[(row, col)].add((row, col + 1))
```

```python
# Visualizza il dizionario
for i in sorted(positions) :
    print(i, ":", positions[i])
```

Esercizio 8

Esercizio 8. Completate il programma dell'esercizio precedente in modo che trovi una via d'uscita dal labirinto a partire da un punto qualsiasi.
Costruite un nuovo dizionario le cui chiavi siano le posizioni dei corridoi e i cui valori siano tutti uguali alla stringa "?".
Poi scandite le chiavi di tale dizionario. Per ogni chiave che si trova al *confine* del labirinto (cioè rappresenta un'uscita), sostituite la "?" con un valore ("N", "E", "S" o "W") che indichi la direzione del percorso di uscita.
A questo punto, ripetete la scansione delle chiavi il cui valore è rimasto "?" e verificate se hanno almeno un adiacente il cui valore non sia "?", usando il primo dizionario per localizzare gli adiacenti. Trovato uno di tali adiacenti, sostituite la stringa "?" con la direzione dell'adiacente.
Se durante una di tali scansioni non riuscite a compiere nessuna di queste sostituzioni, interrompete l'algoritmo.
Infine, visualizzate il labirinto così ottenuto: in ogni posizione di corridoio sarà presente la direzione che porta più rapidamente a un'uscita. Ad esempio:

```
*N*******
*NWW*?*S*
*N*****S*
*N*S*EES*
*N*S***S*
*NWW*EES*
*****N*S*
*EEEEN*S*
*******S*
```

[P8.21]

Esercizio 8 – *il codice python*

```python
## LAB11_es8

# Apre il file e lo legge
inf = open("maze.txt", "r")
maze = []
for line in inf:
    maze.append(line.rstrip())

# Chiude il file
inf.close()

# Crea il dizionario
positions = {}
for row in range(len(maze)) :
    for col in range(len(maze[row])) :
        if maze[row][col] == " " :
            positions[(row, col)] = set()
            # Controlla a sinistra
            if row > 0 and maze[row - 1][col] == " " :
                positions[(row, col)].add((row - 1, col))
            # Controlla a destra
            if row < len(maze) - 1 and maze[row + 1][col] == " " :
                positions[(row, col)].add((row + 1, col))
            # Controlla in alto
            if col > 0 and maze[row][col - 1] == " " :
                positions[(row, col)].add((row, col - 1))
            # Controlla in basso
            if col < len(maze[row]) - 1 and maze[row][col + 1] == " " :
                positions[(row, col)].add((row, col + 1))
```

Esercizio 8 – *il codice python*

```
31  # Visualizza il dizionario
32  for i in sorted(positions) :
33      print(i, ":", positions[i])
34
35  # Crea dizionario con la via d'uscita
36  escape = {}
37  for key in positions :
38      escape[key] = "?"
39  # Sostituisce i bordi con la direzione d'uscita
40  for (r, c) in escape :
41      if r == 0 :
42          escape[(r, c)] = "N"
43      if r == len(maze) - 1 :
44          escape[(r, c)] = "S"
45      if c == 0 :
46          escape[(r, c)] = "W"
47      if c == len(maze[r]) :
48          escape[(r, c)] = "E"
49
50  # Finchè non vengono più fatte modifiche
51  changed = True
52  while changed :
53      changed = False
54      for (r, c) in escape :
55          if escape[(r, c)] == "?" :
56              for (er, ec) in positions[(r, c)] :
57                  if escape[(er, ec)] != "?" :
58                      if er < r :
59                          escape[(r, c)] = "N"
```

Esercizio 8 – *il codice python*

```
                elif er > r :
                    escape[(r, c)] = "S"
                elif ec < c :
                    escape[(r, c)] = "W"
                else :
                    escape[(r, c)] = "E"
                changed = True

# Aggiorna il dizionario con le direzioni di escape
for row in range(len(maze)) :
    for col in range(len(maze[row])) :
        if maze[row][col] == " " :
            maze[row] = maze[row][:col] + escape[(row, col)] + maze[row][col + 1 : ]

# Visualizza il labirinto
for row in maze:
    print(row)
```