Python: Λίστες λιστών

Επιμέλεια: Α.Δρακόπουλος

Πίνακας περιεχομένων

1. Θεωρία: Λίστες Λιστών στην Python

Στην Python, μία **λίστα λιστών** (list of lists) είναι μία δομή δεδομένων που χρησιμοποιείται για την αναπαράσταση **δισδιάστατων πινάκων**, πλεγμάτων (grids), χαρτών, πινάκων παιχνιδιών, αποθηκευμένων δεδομένων σε 2D μορφή και γενικότερα κάθε πληροφορίας που έχει δύο διαστάσεις.

Παράδειγμα δημιουργίας 2D λίστας:

```
world = [
    [".", ".", ".", "*"],
    [".", "#", ".", "."],
    ["*", ".", ".", "."],
    [".", ".", "#", "."]
]
```

Στο παραπάνω παράδειγμα, ο πίνακας world είναι τετραγωνικός 4×4.
Για την εργασία, το world έχει μέγεθος 10×10.

Πρόσβαση στα στοιχεία: world[row][col]

Η πρόσβαση στα στοιχεία γίνεται με δύο δείκτες:

- **row** → δηλώνει τη γραμμή (0 έως ROWS-1)
- **col** → δηλώνει τη στήλη (0 έως COLS-1)

Παράδειγμα:

```
value = world[2][0]
```

Αυτό σημαίνει:

- Πήγαινε στη γραμμή 2
- Από εκεί πάρε το στοιχείο της στήλης 0
- Αποθήκευσέ το στη μεταβλητή value

Αν world[2][0] είναι `"*"`, τότε value == `"*"`.

Αλλαγή τιμής σε κελί

```
world[row][col] = "."
```

Εδώ αλλάζουμε το περιεχόμενο του κελιού (π.χ. όταν συλλέγεται αντικείμενο).

Περιήγηση σε όλο το πλέγμα – διπλή επανάληψη

```
for r in range(ROWS):
    for c in range(COLS):
        print(world[r][c])
```

Γείτονες κελιού – θεμελιώδης έννοια για BFS και movement

Τα 4 βασικά βήματα:

- (r-1, c) → πάνω
- (r+1, c) → κάτω
- (r, c-1) → αριστερά
- (r, c+1) → δεξιά

Σημαντικά σημεία που οι φοιτητές πρέπει να κατανοήσουν

1. Το world είναι **λίστα από λίστες**, όχι δισδιάστατος πίνακας τύπου C/C++.
2. Η πρόσβαση world[row][col] **γίνεται σε δύο στάδια**.
3. Η αλλαγή στοιχείου γίνεται άμεσα στην ίδια δομή: δεν δημιουργούνται αντίγραφα.
4. Η πρόσβαση εκτός ορίων προκαλεί σφάλμα IndexError → απαιτείται έλεγχος ορίων.
5. Η περιήγηση στο world γίνεται συνήθως με δύο βρόχους ή με λίστα από ζεύγη μετατοπίσεων.

---

2. Πέντε Παραδείγματα με Εκφώνηση & Λύση

Τα παραδείγματα είναι *ομόλογα* των απαιτήσεων της εργασίας, χωρίς όμως να αποκαλύπτουν άμεσα τον αλγόριθμο BFS ή τη λύση της άσκησης.

Παράδειγμα 1 – Αναζήτηση Συμβόλου σε 2D Λίστα

Εκφώνηση

Δίνεται 2D λίστα που αναπαριστά πλέγμα χαρακτήρων. Να γραφεί πρόγραμμα που βρίσκει όλες τις θέσεις όπου υπάρχει το σύμβολο `"*"`. Να εκτυπώνονται οι θέσεις ως ζεύγη (row, col).

Λύση

```
world = [
    [".", "*", "."],
    [".", ".", "."],
    ["*", ".", "*"]
]

positions = []
for r in range(len(world)):
    for c in range(len(world[0])):
        if world[r][c] == "*":
            positions.append((r, c))

print("Found items at:", positions)
```

Αποτέλεσμα:

```
Found items at: [(0,1), (2,0), (2,2)]
```

Παράδειγμα 2 – Αλλαγή περιεχομένου σε κελί

Εκφώνηση

Δέσμευσε ότι όποτε ένα κελί περιέχει `"*"`, πρέπει να αντικαθίσταται με `"."`. Υλοποίησε την αλλαγή σε όλο το grid.

Λύση

```
for r in range(len(world)):
    for c in range(len(world[0])):
        if world[r][c] == "*":
            world[r][c] = "."
```

Αυτό προσομοιώνει την “αλλαγή κατάστασης” όπως αυτή που κάνει το ρομπότ στην εργασία.

Παράδειγμα 3 – Μέτρηση αντικειμένων σε grid 10×10

Εκφώνηση

Δίνεται πίνακας 10×10. Να γραφεί πρόγραμμα που μετρά πόσα αντικείμενα `"*"` υπάρχουν συνολικά.

Λύση

```
count = 0
for r in range(10):
    for c in range(10):
        if world[r][c] == "*":
            count += 1
print("Total items:", count)
```

Αυτό το μοτίβο είναι απαραίτητο όταν αποφασίζουμε αν “τελείωσαν τα αντικείμενα”.

Παράδειγμα 4 – Έλεγχος ορίων πριν από πρόσβαση

Εκφώνηση

Να γραφεί συνάρτηση η οποία δέχεται συντεταγμένες (r, c) και επιστρέφει αν είναι εντός ορίων ενός grid 10×10.

Λύση

```
def in_bounds(r, c):
    return 0 <= r < 10 and 0 <= c < 10

print(in_bounds(3, 7))  # True
print(in_bounds(12, 5)) # False
```

Η έννοια αυτή είναι κρίσιμη όταν κινούμαστε στο world ή όταν παράγουμε γείτονες.

Παράδειγμα 5 – Εύρεση γειτονικών κελιών (χωρίς εμπόδια)

Εκφώνηση

Να γραφεί συνάρτηση που επιστρέφει όλα τα γειτονικά κελιά ενός κελιού (r, c) στο πλέγμα, όπου επιτρέπονται μόνο οι κινήσεις πάνω/κάτω/αριστερά/δεξιά.

Λύση

```
def neighbors(r, c):
    res = []
    for dr, dc in [(-1,0),(1,0),(0,-1),(0,1)]:
        nr, nc = r + dr, c + dc
        if 0 <= nr < 10 and 0 <= nc < 10:
            res.append((nr, nc))
    return res

print(neighbors(5, 5))
```

Αυτό το μοτίβο είναι το θεμέλιο για αλγορίθμους αναζήτησης, συμπεριλαμβανομένου και του BFS.

Όλες οι ασκήσεις στοχεύουν σε δεξιότητες που απαιτούνται στην εργασία (2D λίστες, neighbors, valid cells, αναζήτηση, scanning, counting, path marking, boundaries), αλλά **δεν χρησιμοποιούν BFS ή reconstruct_path**.

Ασκήσεις

Άσκηση 1 — Έλεγχος εγκυρότητας κελιού (valid cell check)

Εκφώνηση

Να γραφτεί συνάρτηση `valid_cell(grid, r, c)` που επιστρέφει `True` αν το κελί (r,c) είναι εντός ορίων και δεν είναι εμπόδιο `"#"`.
Αλλιώς να επιστρέφει `False`.

Λύση

def valid_cell(grid, r, c):

ROWS = len(grid)

COLS = len(grid[0])

if 0 <= r < ROWS and 0 <= c < COLS:

return grid[r][c] != "#"

return False

Σχόλια

- Ένας φοιτητής πρέπει να γνωρίζει πως ένα grid αποτελεί έναν *έμμεσο γράφο* – τα εμπόδια είναι “απαγορευμένες κορυφές”.

Άσκηση 2 — Μέτρηση αντικειμένων σε 2D grid

Εκφώνηση

Δεδομένο ένα grid 10×10, γράψτε συνάρτηση `count_items(grid)` που μετρά πόσα `"*"` υπάρχουν συνολικά.

Λύση

```
def count_items(grid):
    count = 0
    for row in grid:
        for cell in row:
            if cell == "*":
                count += 1
    return count
```

Σχόλια

- Χρήσιμη για κατανόηση world scanning, όχι για pathfinding.

Άσκηση 3 — Εύρεση όλων των θέσεων που περιέχουν ένα σύμβολο

Εκφώνηση

Να γραφτεί συνάρτηση `find_all(grid, symbol)` που επιστρέφει λίστα με όλα τα `(r,c)` όπου υπάρχει το `symbol`.

Λύση

```
def find_all(grid, symbol):
    results = []
    for r in range(len(grid)):
        for c in range(len(grid[0])):
            if grid[r][c] == symbol:
                results.append((r, c))
    return results
```

Σχόλια

- Χρήσιμο για εντοπισμό πολλών στόχων.
- Δεν χρησιμοποιεί BFS, άρα είναι ασφαλής για προετοιμασία.

Άσκηση 4 — Αντιγραφή 2D λίστας (deep copy)

Εκφώνηση

Να γραφτεί συνάρτηση `copy_grid(grid)` που επιστρέφει βαθιά αντιγραφή του grid.

Λύση

```
def copy_grid(grid):
```

```
    return [row[:] for row in grid]
```

Σχόλια

- Η λανθασμένη χρήση του `grid_copy = grid` είναι συχνό λάθος.
- Στην εργασία, αλλαγές στο world μπορεί να προκαλέσουν side effects.

Άσκηση 5 — Γείτονες 4-κατευθύνσεων (χωρίς BFS)

Εκφώνηση

Να γραφτεί συνάρτηση `neighbors4(grid, r, c)` που επιστρέφει όλους τους έγκυρους (χωρίς εμπόδια) γείτονες σε 4 κατευθύνσεις.

Λύση

```
def neighbors4(grid, r, c):
    deltas = [(-1,0),(1,0),(0,-1),(0,1)]
    results = []
    for dr, dc in deltas:
        nr, nc = r + dr, c + dc
        if 0 <= nr < len(grid) and 0 <= nc < len(grid[0]):
            if grid[nr][nc] != "#":
                results.append((nr, nc))
    return results
```

Σχόλια

- Η 4-κατευθυντική γειτνίαση είναι βασικό concept των grid graphs.
- Οι φοιτητές πρέπει να το κατανοούν πλήρως πριν την BFS.

Άσκηση 6 — Αλλαγή τιμών σε 2D λίστα (mark visited)

Εκφώνηση

Δεδομένο ένα grid, γράψτε συνάρτηση `mark_cell(grid, r, c, symbol)` που αλλάζει το περιεχόμενο του κελιού `(r,c)`.

Λύση

```
def mark_cell(grid, r, c, symbol):
    if 0 <= r < len(grid) and 0 <= c < len(grid[0]):
        grid[r][c] = symbol
```

Σχόλια

- Χρήσιμο για debugging ή visual marking (π.χ. `"V"` = visited).
- Δεν αποκαλύπτει BFS logic.

Άσκηση 7 — Εύρεση κελιών σε ευθεία γραμμή

Εκφώνηση

Να γραφτεί συνάρτηση `get_row_cells(grid, r)` που επιστρέφει όλα τα κελιά της γραμμής r.

Λύση

```
def get_row_cells(grid, r):
    return grid[r][:]
```

Σχόλια

- Χρήσιμο για scan-based operations ή debugging.
- Ενισχύει κατανόηση indexing.

Άσκηση 8 — Μετατροπή grid από λίστα συμβολοσειρών σε 2D λίστα χαρακτήρων

Εκφώνηση

Να γραφτεί συνάρτηση `parse_grid(lines)` που δέχεται λίστα string όπως:

```
["..#..",
 ".#.*.",
 "....."]
```

και επιστρέφει 2D list χάρτη.

Λύση

```
def parse_grid(lines):
    return [list(row) for row in lines]
```

Σχόλια

- Πολύ χρήσιμο για διάβασμα world.txt.

Άσκηση 9 — Μετατροπή 2D grid σε string (printing)

Εκφώνηση

Να γραφτεί συνάρτηση `grid_to_string(grid)` που επιστρέφει string για εκτύπωση.

Λύση

```
def grid_to_string(grid):
    return "\n".join(" ".join(row) for row in grid)
```

Σχόλια

- Χρήσιμο για debugging world states.
- Επιτρέπει “οπτικοποίηση” της λίστας.

Άσκηση 10 — Εύρεση κοντινότερου συμβόλου με Manhattan distance (όχι BFS)

Εκφώνηση

Να γραφτεί συνάρτηση `closest_item_by_distance(grid, r, c)` που βρίσκει ποιο σύμβολο `"*"` έχει τη μικρότερη Manhattan απόσταση από το `(r,c)`.

$$\text{Manhattan distance} = |r1 - r2| + |c1 - c2|$$

Δεν πρέπει να γίνεται BFS.
Η συνάρτηση δεν εντοπίζει αν το αντικείμενο είναι προσβάσιμο.

Λύση

```
def closest_item_by_distance(grid, r, c):
    best = None
    best_dist = float("inf")
    for rr in range(len(grid)):
        for cc in range(len(grid[0])):
            if grid[rr][cc] == "*":
                d = abs(rr - r) + abs(cc - c)
                if d < best_dist:
                    best_dist = d
                    best = (rr, cc)
    return best
```

Σχόλια

- Η άσκηση ενισχύει κατανόηση μετρικών σε grids.
- Χρήσιμη για σύγκριση BFS vs heuristics (χωρίς να αποκαλύπτουμε BFS logic).
