ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS


**ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ**
**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**


# ΤΗΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ ΚΑΙ ΕΜΠΕΙΡΑ ΣΥΣΤΗΜΑΤΑ

# ΑΛΓΟΡΙΘΜΟΣ BRANCH AND BROUND


## ΟΜΑΔΑ ΑΝΑΠΤΥΞΗΣ:

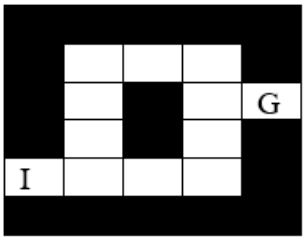
ΒΑΣΙΛΕΙΟΣ ΖΑΡΤΗΛΑΣ ΠΑΠΑΧΑΡΑΛΑΜΠΟΥΣ – **Π17168**


*Πειραιάς*
*Μάιος 2020*

- Ο αλγόριθμος αναπτύχθηκε σε γλώσσα Python μέσω PyCharm

## Επεξήγηση κώδικα:

Αρχικά θα υλοποιήσουμε το πρώτο λαβύρινθο όπου είναι ο εξής:
I → Σημείο Εκκίνησης
G→ Σημείο Τερματισμού




Αντίστοιχα στον κώδικα ο λαβύρινθος:

```
labyrinth = [[1, 1, 1, 1, 1],
             [1, 0, 0, 0, 1],
             [1, 0, 1, 0, 0],
             [1, 0, 1, 0, 1],
             [0, 0, 0, 0, 1],
             [1, 1, 1, 1, 1]]
```

όπου ένα (1) είναι τα μαύρα κουτιά το όποια δεν μπορούμε να κινηθούμε και μηδέν(0) τα λευκά όπου μπορούμε να κινηθούμε.

Η συνάρτηση children_stalker:

```
def children_stalker(route):
    x = y = 0
    coordinate = []
    childrenStates = []
    if len(route) == 1:
        coordinate = route[0]
        x = coordinate[1]
        y = coordinate[0]
    elif len(route) > 1:
        coordinate = route[-1]
        x = coordinate[1]
        y = coordinate[0]
```

με την συνάρτηση αυτή βρίσκουμε τις καταστάσεις παιδιά.

- Στο σημείο if len(route) == 1 ελέγχουμε αν στην διαδρομή μάς υπάρχει μόνο ένα σημείο για να κινηθούμε και τότε το βάζουμε για να επεκτείνουμε την διαδρομή του αλγόριθμου.
- Αν όμως έχει περισσότερα σημεία για να κινηθούμε θα πάρουμε το τελευταίο σημείο για να επεκτείνουμε την διαδρομή μας μέσω του ελέγχουν elif len(route)>1

Στο σημείο αυτό :

```
if len(route) < BestCost:
    print('ΘΑ ΠΑΜΕ:')
```

με το len(route)<BestCost: γίνεται έλεγχος αν μπορεί να γίνει κλάδεμα .Αν μπορούμε δηλαδή να φτάσουμε στο τέρμα με οποιαδήποτε επιπλέον κίνηση.

Στο σημείο:

```
if y - 1 >= 0:
    if labyrinth[y - 1][x] == 0:
        northChildrenX = x
        northChildrenY = y - 1
        if [y - 1, x] not in route:
            northChildren = route + [[northChildrenY, northChildrenX]]
            childrenStates.append(northChildren)
            print(" ΒΟΡΕΙΑ ")
```

- Στο σημείο if y-1>=0: γίνεται έλεγχος ώστε σε περίπτωση που φτάσει να είναι αρνητικό θα χρησιμοποιεί y από το τέλος της λίστας.

- Στο σημείο if labyrinth[y-1][x]==0: γίνεται έλεγχος ώστε σε περίπτωση που ισούται με μηδέν μπορεί να συνεχίσει βόρεια.

Στο σημείο αυτό :

```
if [y - 1, x] not in route:
    northChildren = route + [[northChildrenY, northChildrenX]]
    childrenStates.append(northChildren)
    print(" ΒΟΡΕΙΑ ")
```

Σε περίπτωση που η διαδρομή υπάρχει ήδη τοποθετούμε το σημείο στο τέλος της λίστα και επίσης το τοποθετούμε στις καταστάσεις παιδιά.

Στο σημείο:

```
while len(SearchFront) != 0:
    CounterReps += 1
    print("\nΑΡΙΘΜΟΣ ΕΠΑΝΑΛΗΨΕΩΝ: " + str(CounterReps))
    route = SearchFront[0]
    SearchFront.pop(0)
    if route[-1] != FinalState:
        children = children_stalker(route)
        SearchFront = children + SearchFront
        children.clear()
```

Χρησιμοποιούμε το `while len(SearchFront) != 0:` μέχρι να αδειάσει το Μέτωπο Αναζήτησης. Με την εντολή `route = SearchFront[0]` βάζουμε στην κατάσταση το πρώτο σημείο από το Μέτωπο Αναζήτησης και με την εντολή `SearchFront.pop(0)` το αφαιρούμε από το μέτωπο αναζήτησης εφόσον θα το ελέγξουμε.

Στο `if route[-1] != FinalState:` ελέγχουμε αν το τελευταίο σημείο που υπάρχει στην κατάσταση δεν ισούται με το [2,4] ακολούθως

βρίσκουμε τα παιδιά `children = children_stalker(route)` και τα βάζουμε μπροστά μπροστά στο μέτωπο αναζήτησης `SearchFront = children + SearchFront` και τέλος αδειάζουμε την λίστα με τα παιδιά `children.clear()`.

Στο else:

```
else:
    print('Η ΔΙΑΔΡΟΜΗ ΟΛΟΚΛΗΡΩΘΗΚΕ')
    print('ΔΙΑΔΡΟΜΗ: ' + str(route))
    CurrentCost = len(route) - 1
    print('ΚΟΣΤΟΣ ΓΙΑ ΑΥΤΗΝ ΤΗΝ ΔΙΑΔΡΟΜΗ: ' + str(CurrentCost))
    if CurrentCost < BestCost:
        BestRoute = route
        BestCost = CurrentCost
```

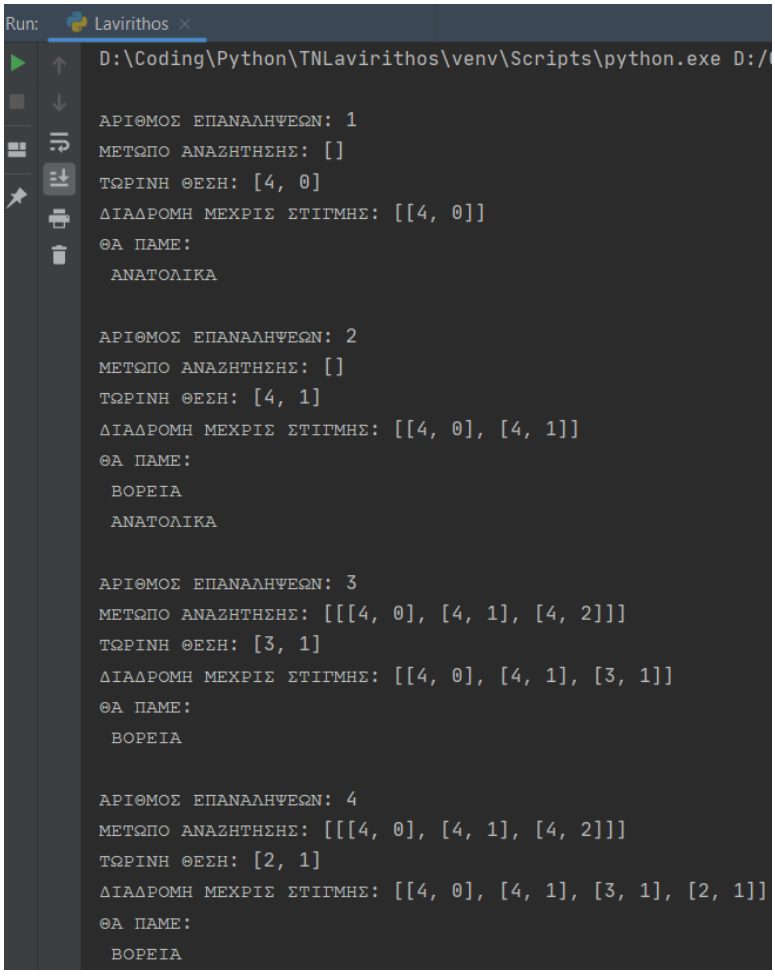
σε περίπτωση που φτάσαμε στο [2,4] υπολογίζουμε το κόστος με `CurrentCost = len(route) - 1` επειδή είναι και το αρχικό σημείο [4,0]. Έπειτα ελέγχουμε `if CurrentCost < BestCost:` αν το κόστος μας είναι το ελάχιστο και ανανεώνουμε ανάλογα τις τιμές:
`BestRoute = route`
`BestCost = CurrentCost.`

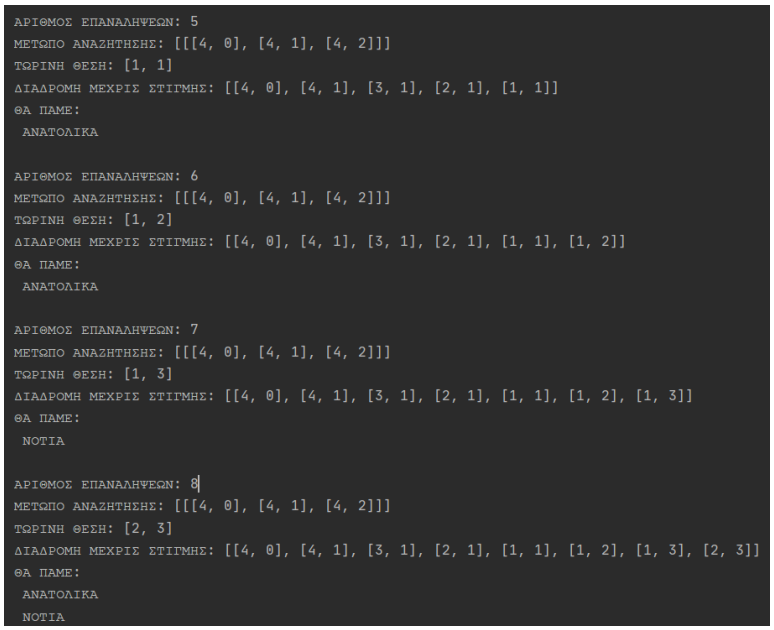
Τέλος:

```
if BestCost == sys.maxsize:
    print("ΚΑΤΙ ΠΗΓΕ ΣΤΡΑΒΑ")
else:
    print("\nΤΕΛΟΣ ΑΛΓΟΡΙΘΜΟΥ")
    print("ΕΛΑΧΙΣΤΟ ΚΟΣΤΟΣ " + str(BestCost) + " ΜΕ ΜΟΝΟΠΑΤΙ " + str(BestRoute) + ".")
```

Αν το κόστος είναι ίδιο με το maxsize τότε κάτι πήγε στραβά και αλλιώς θα τυπώσουμε τα αποτελέσματα στην οθόνη!

Παραδείγματα εκτέλεσης κώδικα:



```
ΑΡΙΘΜΟΣ ΕΠΑΝΑΛΗΨΕΩΝ: 14
ΜΕΤΩΠΟ ΑΝΑΖΗΤΗΣΗΣ: []
ΤΩΡΙΝΗ ΘΕΣΗ: [2, 3]
ΔΙΑΔΡΟΜΗ ΜΕΧΡΙΣ ΣΤΙΓΜΗΣ: [[4, 0], [4, 1], [4, 2], [4, 3], [3, 3], [2, 3]]
ΘΑ ΠΑΜΕ:
 ΒΟΡΕΙΑ
 ΑΝΑΤΟΛΙΚΑ

ΑΡΙΘΜΟΣ ΕΠΑΝΑΛΗΨΕΩΝ: 15
ΜΕΤΩΠΟ ΑΝΑΖΗΤΗΣΗΣ: [[[4, 0], [4, 1], [4, 2], [4, 3], [3, 3], [2, 3], [2, 4]]]
ΤΩΡΙΝΗ ΘΕΣΗ: [1, 3]
ΔΙΑΔΡΟΜΗ ΜΕΧΡΙΣ ΣΤΙΓΜΗΣ: [[4, 0], [4, 1], [4, 2], [4, 3], [3, 3], [2, 3], [1, 3]]
ΘΑ ΠΑΜΕ:
 ΔΥΤΙΚΑ

ΑΡΙΘΜΟΣ ΕΠΑΝΑΛΗΨΕΩΝ: 16
ΜΕΤΩΠΟ ΑΝΑΖΗΤΗΣΗΣ: [[[4, 0], [4, 1], [4, 2], [4, 3], [3, 3], [2, 3], [2, 4]]]
ΤΩΡΙΝΗ ΘΕΣΗ: [1, 2]
ΔΙΑΔΡΟΜΗ ΜΕΧΡΙΣ ΣΤΙΓΜΗΣ: [[4, 0], [4, 1], [4, 2], [4, 3], [3, 3], [2, 3], [1, 3], [1, 2]]
ΚΛΑΔΕΜΑ, ΚΑΛΥΤΕΡΟ ΚΟΣΤΟΣ: 8

ΑΡΙΘΜΟΣ ΕΠΑΝΑΛΗΨΕΩΝ: 17
Η ΔΙΑΔΡΟΜΗ ΟΛΟΚΛΗΡΩΘΗΚΕ
ΔΙΑΔΡΟΜΗ: [[4, 0], [4, 1], [4, 2], [4, 3], [3, 3], [2, 3], [2, 4]]
ΚΟΣΤΟΣ ΓΙΑ ΑΥΤΗΝ ΤΗΝ ΔΙΑΔΡΟΜΗ: 6

ΤΕΛΟΣ ΑΛΓΟΡΙΘΜΟΥ
ΕΛΑΧΙΣΤΟ ΚΟΣΤΟΣ 6 ΜΕ ΜΟΝΟΠΑΤΙ [[4, 0], [4, 1], [4, 2], [4, 3], [3, 3], [2, 3], [2, 4]].

Process finished with exit code 0
```