

# Αλγοριθμικές Τεχνικές και Εφαρμογές

Φοιτητής: Παναγιώτης Κοντος

AM: mpsp2215

Η παρακάτω εργασία αφορά την υλοποίηση ενός αλγόριθμου που προσομοιώνει την διάδοση πυρκαγιάς σε ένα δάσος με χρήση του προγραμματιστικού μοντέλου των POSIX Threads.. Η εργασία έγινε με την χρήση της **Python**. Αξίζει να σημειωθεί ότι γράψαμε εκ νέου έναν fire starter αλγόριθμο απο C σε Python, τον οποίον χρησιμοποιήσαμε για να φτιάξουμε ένα fire starter Monte Carlo Simulation .

## Βήματα

Αρχικά πρέπει να εξετάσουμε πως λειτουργει ο αλγοριθμος με 1 thread, ο αλγοριθμος παιρνει σαν μεταβλητες το forest\_size, prob\_min, prob\_max, n\_trials, n\_probs και για την καθε πιθανοτητα θα υπολογισει το burn\_output, δηλαδη το ποσοστο του δασους που καιγεται καθε φορα.

```
panos@panos-H510M-K-M-2: ~/Documents/projects/monteCarlo$ ls
dynamic_threading.py  single_thread.py  static_threading.py
panos@panos-H510M-K-M-2: ~/Documents/projects/monteCarlo$ code .
panos@panos-H510M-K-M-2: ~/Documents/projects/monteCarlo$ pytho3 single_thread.py
Command 'pytho3' not found, did you mean:
  command 'python3' from deb python3 (3.10.6-1-22.04)
Try: sudo apt install <deb name>
panos@panos-H510M-K-M-2: ~/Documents/projects/monteCarlo$ python3 single_thread.py
0.0, 0.0
0.01, 5.0125313283208015e-05
0.02, 0.0003007518796992481
0.03, 0.0002506265664160401
0.04, 0.0004010025062656641
0.05, 0.0003007518796992481
0.06, 0.0006015037593984962
0.07, 0.0008020050125313282
0.08, 0.0016541353383458645
0.09, 0.0013533834586466164
0.1, 0.0011027568922305764
0.11, 0.0016040100250626565
0.12, 0.0022556390977443606
0.13, 0.0020551378446115286
0.14, 0.0019047619047619048
0.15, 0.002406015037593985
0.16, 0.004511278195488721
0.17, 0.003709273182957393
0.18, 0.002807017543859649
0.19, 0.003258145363408521
0.2, 0.0047619047619047615
0.21, 0.007619047619047619
0.22, 0.0071177944862155385
0.23, 0.006766917293233083
0.24, 0.008020050125313283
0.25, 0.009724310776942356
0.26, 0.00887218045112782
0.27, 0.01012531328320802
0.28, 0.012230576441102757
0.29, 0.016541353383458645
0.3, 0.013283208020050124
0.31, 0.014385964912280698
0.32, 0.02265664160401002
0.33, 0.024360902255639104
0.34, 0.02506265664160401
0.35, 0.03243107769423558
0.36, 0.029273182957393493
0.37, 0.040050125313283204
0.38, 0.056090225563909774
0.39, 0.05513784461152883
0.4, 0.05433583959899748
0.41, 0.08065162907268171
0.42, 0.08616541353383456
0.43, 0.06145363408521302
0.44, 0.12756892230576442
0.45, 0.15493734335839596
0.46, 0.17017543859649123
0.47, 0.2505764411027569
```

επισης μπορουμε να δουμε το δασος χρησιμοποιοντας την μεθοδο `print_forest(forest_size, forest)`

```
panos@panos-H510M-K-M-2: ~/Documents/projects/monteCarlo
XX.....XXXXX.....XXXXXX
XX.....XXX.....XXXXXX
XXX.....XXXXXX.....XXXXXX
XX.....XXXXXXXXXXXXXXXXX
0.48, 0.2770426065162907
XXXXXXXXXXXXXX.XX.X.
XXXXXXXXXXXXXX.X.....
XXXXXXXXX.....
XXXXXXXXX.X.....
XXXXXXXXX.X.X.X
XXXXXXXXX.....X
XXXXXXXXX.....XX
XXXXXXXXX.....X.X
XXXXXXXXX.....X
XXXXXXXXX.....X
XXXXXXXXX.....X
XXXXXXXXX.XX.....XX
XXXXXXXXXXXXX.....XX
XXXXXXXXXXXXX.....
XXXXXXXXXXXXX.....
XXXXXXXXXXXXX.....X
XXXXXXXXXXXXX.....X
XXXXXXXXXXXXX.....X
XXXXXXXXXXXXX.X.X.XX
XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX
0.49, 0.25679197994987474
XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX
```

αξιζει να σημειωθει οτι ο αλγοριθμος ξεκινει καθε φορα απο τυχαio σημειο και οτι τα X συμβολιζουν τα δεντρα ενω τα . τα καμμενα δεντρα.

Πως αυτος ο αλγοριθμος μπορει να πραγματοποιηθει παραλληλα με στατικη ανάθεση φόρτου εργασίας μεταξύ των νημάτων?

Αρχικα θα εχουμε μια νεα παραμετρο που θα ειναι ο αριθμος των νημάτων με ονομα THREADS, επειτα τα διαιρεσουμε τον αριθμο των πιθανοτητων (n\_probs) με τον αριθμο των threads, και θα δοσουμε στο καθε thread το αναλογο φορτιο.

```

threads = list()
step = NUMS//THREADS
for i in range(THREADS):
    start = step*i
    if(i!=0):
        start = (step*i)+1
    end = step*(i+1)
    # If it is the last
    if(i==THREADS-1):
        end=NUMS
    t = threading.Thread(target=thread_function, args=(i, start, end))
    threads.append(t)
    t.start()
t.join()

```

Επομένως το κάθε thread θα εκτελεί το ίδιο method με πριν αλλά αυτήν την φορά για διαφορετικό range πιθανοτήτων.

```

.....
Execution time: 5.698615312576294
panos@panos-H510M-K-M-2:~/Documents/projects/monteCarlo$ python3 static_threading.py
Thread 0: 0 - 5
0.0, 0.0
0.01, 0.00010025062656641603
0.03, 0.00045112781954887213
0.04, 0.0003007518796992481
0.05, 0.0007518796992481202
Thread 1: 6 - 10
0.06, 0.0006015037593984962
0.07, 0.0008521303258145363
0.09, 0.0011027568922305764
0.1, 0.0017543859649122805
Thread 2: 11 - 15
0.11, 0.0018045112781954885
0.12, 0.0017042606516290725
0.14, 0.003007518796992481
0.15, 0.0036591478696741853
Thread 3: 16 - 20
0.16, 0.003007518796992481
0.17, 0.003057644110275689
0.19, 0.004661654135338346
0.2, 0.004711779448621553
Thread 4: 21 - 25
0.21, 0.004611528822055138
0.22, 0.007017543859649122
0.24, 0.008020050125313283
0.25, 0.007167919799498747
Thread 5: 26 - 30
0.26, 0.00857142857142857
0.27, 0.014536340852130318
0.29, 0.01769423558897243
0.3, 0.014285714285714282
Thread 6: 31 - 35
0.31, 0.022706766917293227
0.32, 0.02050125313283208
0.34, 0.029874686716791974
0.35, 0.027669172932330822
Thread 7: 36 - 40
0.36, 0.04250626566416038
0.37, 0.04726817042606516
0.39, 0.07182957393483706
0.4, 0.059548872180451094
Thread 8: 41 - 45
0.41, 0.08641604010025063
0.42, 0.11899749373433582
0.44, 0.11744360902255638
0.45, 0.1876691729323308
Thread 9: 46 - 50
0.46, 0.2328822055137844
0.47, 0.2937844611528822
0.49, 0.28385964912280703
0.5, 0.3916791979949874

```

Το πρόβλημα με αυτήν την υλοποίηση είναι το load balancing καθώς τα threads που θα παρουν τις τελευταίες πιθανότητες θα κάνουν και την περισσότερη ώρα. Για να μην γίνεται αυτό θα πρέπει να βρούμε έναν τρόπο να φορτίο να ισομοιραζεται.

Πως αυτος ο αλγοριθμος μπορεί να πραγματοποιηθει παραλληλα με δυναμική ανάθεση φόρτου εργασίας μεταξύ των νημάτων?