

# Προγραμματιστική Άσκηση 1

Γρύλλιας Γεράσιμος 1084651

---

## Υλοποίηση Αλγορίθμου Kruskal & Πειραματική αξιολόγηση συγκριτικά με την MIN\_SPANNING\_TREE της LEDA

---

Μερικά λόγια για την υλοποίηση:

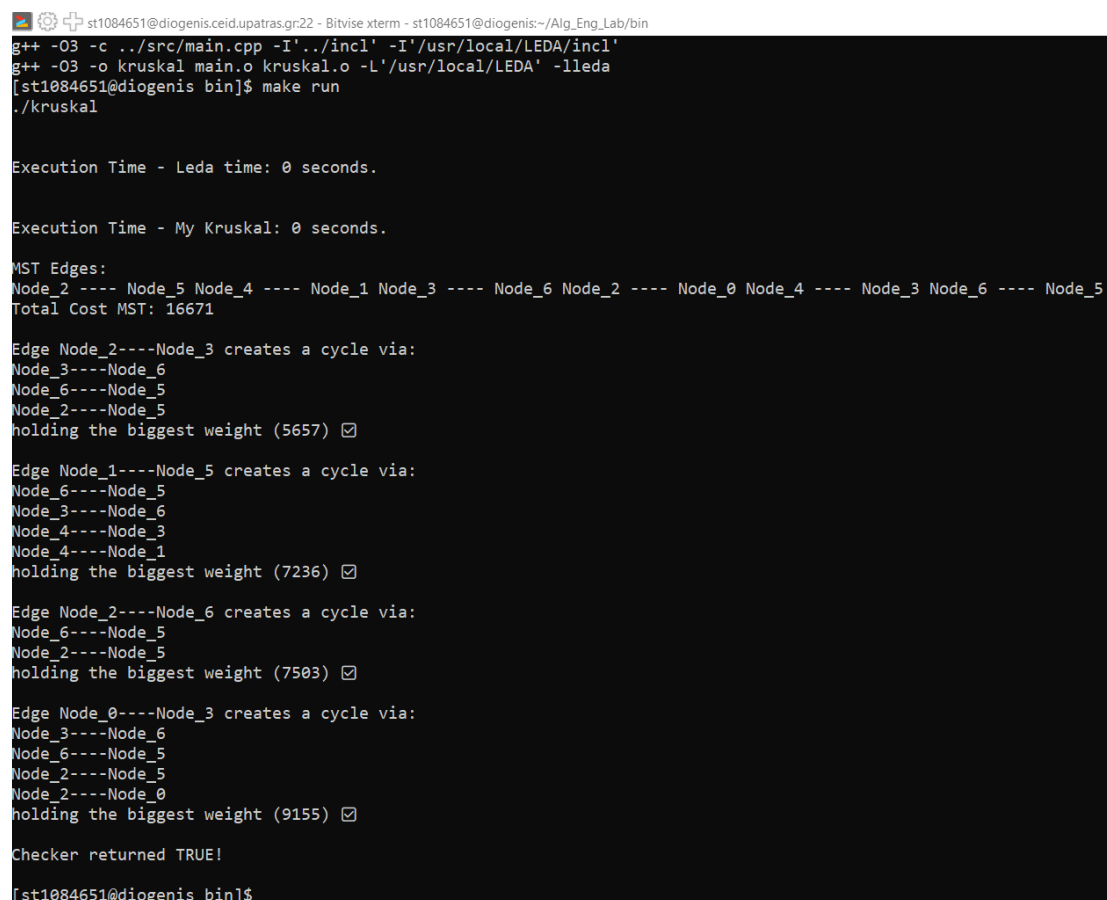
Πρώτα, οι ακμές του γράφου ταξινομούνται με βάση το βάρος τους. Κάθε κόμβος τοποθετείται αρχικά σε μια δική του λίστα (με το δικό του id). Στη συνέχεια, για κάθε ακμή ελέγχεται αν οι δύο κόμβοι της ανήκουν σε διαφορετικές συνιστώσες (first[]). Αν ναι, η ακμή προστίθεται στο ελάχιστο συνδεδετικό δέντρο και οι αντίστοιχες λίστες ενώνονται. Με αυτόν τον τρόπο, αποφεύγεται η δημιουργία κύκλων και χτίζεται σταδιακά το ελάχιστο συνδεδετικό δέντρο του γράφου.

Σύγκριση με MIN\_SPANNING\_TREE:

Ο αλγόριθμος της LEDA είναι πιο αποδοτικός από τη δική μου υλοποίηση Kruskal από τη σκοπιά του χρόνου εκτέλεσης, αν και η διαφορά είναι πολύ μικρή σε μικρά γραφήματα (έως μηδαμινή). Σε αυτά τα γραφήματα, και οι δύο αλγόριθμοι είναι επαρκώς γρήγοροι. Ωστόσο, για μεγαλύτερα γραφήματα (π.χ. χιλιάδες κόμβους και ακμές), αναμένεται ότι η LEDA θα έχει σαφώς καλύτερη κλιμάκωση και σταθερότερη απόδοση. Για την πειραματική μέτρηση του χρόνου εκτέλεσης, καταγράφηκε ο χρόνος πριν και μετά την εκτέλεση κάθε αλγορίθμου, χρησιμοποιώντας τις συναρτήσεις χρονισμού του λειτουργικού συστήματος.

Η διαφορά των δύο χρονικών στιγμών υπολογίστηκε και αντιπροσωπεύει τον συνολικό χρόνο εκτέλεσης του αλγορίθμου.

### SCREENSHOT ΑΠΟ ΔΟΚΙΜΗ ΓΡΑΦΟΥ ΜΕ 7 NODES ΚΑΙ 10 EDGES:



```
st1084651@diogenis.ceid.upatras.gr:22 - Bitwise xterm - st1084651@diogenis:~/Alg_Eng_Lab/bin
g++ -O3 -c ../src/main.cpp -I'../incl' -I'/usr/local/LEDA/incl'
g++ -O3 -o kruskal main.o kruskal.o -L'/usr/local/LEDA' -lleda
[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0 seconds.

Execution Time - My Kruskal: 0 seconds.

MST Edges:
Node_2 ---- Node_5 Node_4 ---- Node_1 Node_3 ---- Node_6 Node_2 ---- Node_0 Node_4 ---- Node_3 Node_6 ---- Node_5
Total Cost MST: 16671

Edge Node_2----Node_3 creates a cycle via:
Node_3----Node_6
Node_6----Node_5
Node_2----Node_5
holding the biggest weight (5657) ☑

Edge Node_1----Node_5 creates a cycle via:
Node_6----Node_5
Node_3----Node_6
Node_4----Node_3
Node_4----Node_1
holding the biggest weight (7236) ☑

Edge Node_2----Node_6 creates a cycle via:
Node_6----Node_5
Node_2----Node_5
holding the biggest weight (7503) ☑

Edge Node_0----Node_3 creates a cycle via:
Node_3----Node_6
Node_6----Node_5
Node_2----Node_5
Node_2----Node_0
holding the biggest weight (9155) ☑

Checker returned TRUE!
[st1084651@diogenis bin]$
```

### ΠΑΡΑΤΗΡΗΣΗ:

Κατά την εκτέλεση του προγράμματος, πραγματοποιήθηκε αξιολόγηση τόσο της αποδοτικότητας όσο και της ορθότητας της υλοποίησης του αλγορίθμου Kruskal. Συγκεκριμένα, παρατηρήθηκε ότι τόσο ο δικός μας αλγόριθμος όσο και η ενσωματωμένη συνάρτηση MIN\_SPANNING\_TREE της LEDA ολοκλήρωσαν την εύρεση του ελάχιστου γεννητικού δέντρου (MST) σε χρόνο εκτέλεσης 0 δευτερολέπτων για το συγκεκριμένο μικρό γράφημα, γεγονός που αποδεικνύει την αποδοτικότητα της προσέγγισής μας. Το συνολικό κόστος του παραγόμενου MST ήταν 16671 και όλες οι ακμές του τυπώθηκαν ορθά. Στη συνέχεια, χρησιμοποιήθηκε ο αλγόριθμος ελέγχου (checker) που υλοποιήθηκε, ο οποίος επιβεβαίωσε την ορθότητα του MST: κάθε ακμή που δεν ανήκει στο MST δημιουργεί έναν μοναδικό κύκλο κατά την προσθήκη της, και σε κάθε τέτοιο κύκλο η

συγκεκριμένη ακμή είχε βάρος μεγαλύτερο ή ίσο από όλες τις ακμές του κύκλου που ανήκαν στο MST. Ο έλεγχος επιστράφηκε επιτυχώς (TRUE), επιβεβαιώνοντας ότι η κατασκευή του ελάχιστου γεννητικού δέντρου ήταν σωστή.

#### SCREENSHOT ΑΠΟ ΔΟΚΙΜΗ ΓΡΑΦΟΥ ΜΕ 500 NODES ΚΑΙ 2NLOGN EDGES:

(ΕΔΩ ΜΠΗΚΑΝ ΣΕ ΣΧΟΛΙΟ ΟΙ ΑΝΑΛΥΤΙΚΕΣ ΕΝΤΟΛΕΣ ΕΚΤΥΠΩΣΗΣ ΓΙΑ ΕΥΕΛΙΞΙΑ)

```
st1084651@diogenis.ceid.upatras.gr:22 - Bitwise xterm - st1084651@dio
Execution Time - Leda time: 0 seconds.

Execution Time - My Kruskal: 0 seconds.

MST Edges:

Total Cost MST: 172695

Checker returned TRUE!

[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0 seconds.

Execution Time - My Kruskal: 0 seconds.

MST Edges:

Total Cost MST: 171362

Checker returned TRUE!

[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0 seconds.

Execution Time - My Kruskal: 0.01 seconds.

MST Edges:

Total Cost MST: 178414

Checker returned TRUE!

[st1084651@diogenis bin]$
```

#### ΠΑΡΑΤΗΡΗΣΗ:

Σε αυτήν την περίπτωση, η LEDA εξακολουθεί να ολοκληρώνει την εύρεση του MST σε 0 δευτερόλεπτα, ενώ η δική μας υλοποίηση του Kruskal εμφάνισε (σε ένα γράφο από τις 3 δοκιμές) χρόνο εκτέλεσης 0.01 δευτερολέπτων, γεγονός που εξακολουθεί να δείχνει πολύ καλή αποδοτικότητα για το μέγεθος του προβλήματος. Το συνολικό κόστος του MST που υπολογίστηκε ήταν 178414. Και πάλι, ο έλεγχος (checker) επιστράφηκε επιτυχώς (TRUE), επιβεβαιώνοντας ότι και για μεγάλο πλήθος κόμβων η υλοποίησή μας λειτουργεί σωστά και παράγει το ορθό ελάχιστο γεννητικό δέντρο.

---

## Υλοποίηση Ελεγκτή “Checker”

---

Για την υλοποίηση του ελεγκτή, δημιουργήθηκε μια συνάρτηση τύπου bool με την εξής λογική:

Πρώτα από όλα, αφού κάνουμε hide τις ακμές που δεν ανήκουν στο MST και τις αποθηκεύσουμε στη λίστα nonMSTEdges (για μετέπειτα επεξεργασία) κάνουμε έναν αρχικό έλεγχο για το αν υπάρχει κύκλος στο MST που δημιουργήθηκε από τον Kruskal. **Η λογική είναι η εξής:** Για κάθε μία ακμή στο MST, κάνε τη hide προσωρινά,

και έλεγξε αν υπάρχει άλλη διαδρομή από το source στο target (αυτό γίνεται με τη βοήθεια του BFS). Αν υπάρχει κύκλος, ο αλγόριθμος Kruskal επέστρεψε λάθος αποτέλεσμα (ο Checker θα το εντοπίσει και θα το επισημάνει). Η επόμενη φάση του Checker είναι ο έλεγχος ότι κάθε ακμή non-MST δημιουργεί κύκλο. **Η λογική είναι η εξής:** Για κάθε ακμή εκτός του MST (δηλαδή για κάθε ακμή από τον πίνακα nonMSTEdges), εφάρμοσε BFS στο MST(!), ώστε να βρεθεί μια διαδρομή από το source στο target. Αν βρεθεί, (dist[endNode] != -1), το επόμενο βήμα είναι να ελέγξουμε αν το βάρος της είναι μεγαλύτερο από τα επιμέρους βάρη της διαδρομής που ανακαλύψαμε προηγουμένως (αυτό γίνεται με τη βοήθεια του pred[] και μιας μεταβλητής currentNode τύπου node που διαπερνά τους κόμβους της διαδρομής λόγω του while (G.source(edd) != startNode && G.target(edd) != startNode)). Αν το βάρος της non-MST ακμής είναι μεγαλύτερο από το μεγαλύτερο βάρος των επιμέρους ακμών, συνεχίζουμε με την επόμενη ακμή του πίνακα nonMSTEdges.

#### SCREENSHOT ΑΠΟ ΤΟ ΓΡΑΦΗΜΑ ΤΗΣ ΑΣΚΗΣΗΣ 5 ΤΟΥ ΕΡΓΑΣΤΗΡΙΟΥ:

##### (ΟΡΘΟΣ ΚΑΙ ΛΑΝΘΑΣΜΕΝΟΣ KRUSKAL ΑΝΤΙΣΤΟΙΧΑ)

```
st1084651@diogenis.ceid.upatras.gr:22 - Bitwise xterm - st1084651@diogenis:~/Alg_Eng_Lab
[st1084651@diogenis bin]$ make run
./kruskal

MST Edges:
A ---- B E ---- F B ---- C A ---- D C ---- F
Total Cost MST: 16

Edge B----D creates a cycle via:
A----D
A----B
holding the biggest weight (5) ☒

Edge C----D creates a cycle via:
A----D
A----B
B----C
holding the biggest weight (7) ☒

Edge F----D creates a cycle via:
A----D
A----B
B----C
C----F
holding the biggest weight (8) ☒

Edge B----E creates a cycle via:
E----F
C----F
B----C
holding the biggest weight (9) ☒

Checker returned TRUE!

[st1084651@diogenis bin]$
```

##### ΛΑΘΟΣ KRUSKAL (MST ΠΟΥ ΚΑΝΕΙ ΚΥΚΛΟ):

```
st1084651@diogenis.ceid.upatras.gr:22 - Bitwise xterm - st1084651@diogenis:~/Alg_Eng_Lab/bin
[st1084651@diogenis bin]$ make run
./kruskal

MST Edges:
A ---- B E ---- F B ---- C A ---- D B ---- D C ---- F C ---- D F ---- D B ---- E
Total Cost MST: 45

There is a cycle in the MST...
Checker returned FALSE :(

[st1084651@diogenis bin]$
```

---

## Πειραματική Αξιολόγηση

---

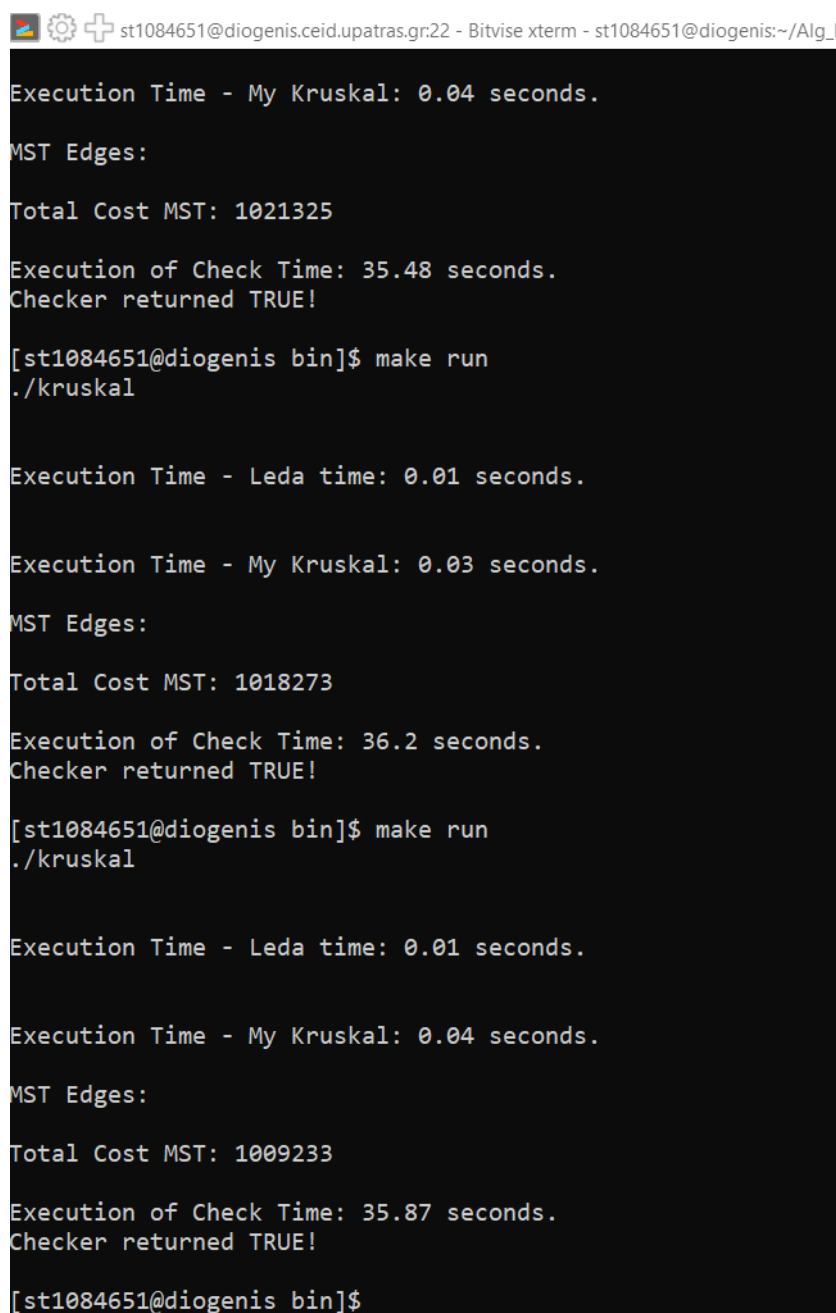
Για πιο πλήρη αξιολόγηση της διαδικασίας, προστέθηκε και χρονομέτρηση για τη μέτρηση του χρόνου εκτέλεσης του ελεγκτή.

### 1) Τυχαία συνεκτικά γραφήματα με μέγεθος κορυφών $n \in \{4000, 8000, 16000\}$ :

Για μέγεθος κορυφών  $n=4000$ : Τα κόστη των ακμών ορίστηκαν ως τυχαίοι ακέραιοι αριθμοί στο διάστημα  $[10, 10000]$ . Η διαδικασία υλοποιήθηκε ως εξής: χρησιμοποιήθηκε η συνάρτηση `random_graph` της LEDA. Κάθε κόμβος έλαβε μοναδικό όνομα της μορφής "Node\_id" και σε κάθε ακμή αποδόθηκε τυχαίο βάρος χρησιμοποιώντας τη δομή `random_source`.

Έπειτα, εκτελέστηκε τόσο η έτοιμη υλοποίηση του Kruskal της LEDA όσο και η δική μας υλοποίηση. Τα αποτελέσματα που προέκυψαν είναι τα εξής: ο χρόνος εκτέλεσης της LEDA ήταν σταθερά 0.01 δευτερόλεπτα, ενώ ο χρόνος εκτέλεσης της δικής μας υλοποίησης κυμάνθηκε από 0.03 έως 0.04 δευτερόλεπτα. Τα συνολικά κόστη του MST στις τρεις δοκιμές ήταν 1.021.325, 1.018.273 και 1.009.233 αντίστοιχα. Και στις τρεις εκτελέσεις το αποτέλεσμα επαληθεύτηκε επιτυχώς μέσω του ελεγκτή (TRUE). Ο έλεγχος εγκυρότητας χρειάστηκε περίπου 35-36 δευτερόλεπτα για κάθε εκτέλεση σε γράφο.

#### SCREENSHOT ΓΙΑ $N=4000$ , $M=2N\log N$ :



```
st1084651@diogenis.ceid.upatras.gr:22 - Bitvise xterm - st1084651@diogenis:~/Alg_E
Execution Time - My Kruskal: 0.04 seconds.
MST Edges:
Total Cost MST: 1021325
Execution of Check Time: 35.48 seconds.
Checker returned TRUE!

[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0.01 seconds.

Execution Time - My Kruskal: 0.03 seconds.
MST Edges:
Total Cost MST: 1018273
Execution of Check Time: 36.2 seconds.
Checker returned TRUE!

[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0.01 seconds.

Execution Time - My Kruskal: 0.04 seconds.
MST Edges:
Total Cost MST: 1009233
Execution of Check Time: 35.87 seconds.
Checker returned TRUE!

[st1084651@diogenis bin]$
```

#### SCREENSHOT ΓΙΑ $N=8000$ , $M=2N\log N$ :

```
st1084651@diogenis.ceid.upatras.gr:22 - Bitvise xterm - st1084651@diog
[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0.02 seconds.

Execution Time - My Kruskal: 0.14 seconds.

MST Edges:

Total Cost MST: 1954181

Execution of Check Time: 167.81 seconds.
Checker returned TRUE!

[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0.03 seconds.

Execution Time - My Kruskal: 0.17 seconds.

MST Edges:

Total Cost MST: 1960402

Execution of Check Time: 174.82 seconds.
Checker returned TRUE!

[st1084651@diogenis bin]$ █
```

#### ΠΑΡΑΤΗΡΗΣΗ:

Η αύξηση του χρόνου εκτέλεσης, ειδικά του ελεγκτή, είναι αναμενόμενη λόγω της αυξημένης πολυπλοκότητας και του πλήθους των κόμβων και ακμών. Παρατηρείται επίσης ότι ο δικός μας αλγόριθμος Kruskal εξακολουθεί να παραμένει σε αποδεκτά επίπεδα απόδοσης σε μεγαλύτερα γραφήματα, αν και με σαφώς μεγαλύτερο χρόνο σε σχέση με τη LEDA, όπως ήταν αναμενόμενο.

#### SCREENSHOT ΓΙΑ $N=16000$ , $M=2N\log N$ :

```
st1084651@diogenis.ceid.upatras.gr:22 - Bitvise xterm [disconnected]
[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0.05 seconds.

Execution Time - My Kruskal: 0.38 seconds.

MST Edges:

Total Cost MST: 3600955

Execution of Check Time: 653.35 seconds.
Checker returned TRUE!

[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0.05 seconds.

Execution Time - My Kruskal: 0.37 seconds.

MST Edges:

Total Cost MST: 3619366

Execution of Check Time: 638.88 seconds.
Checker returned TRUE!

[st1084651@diogenis bin]$
```

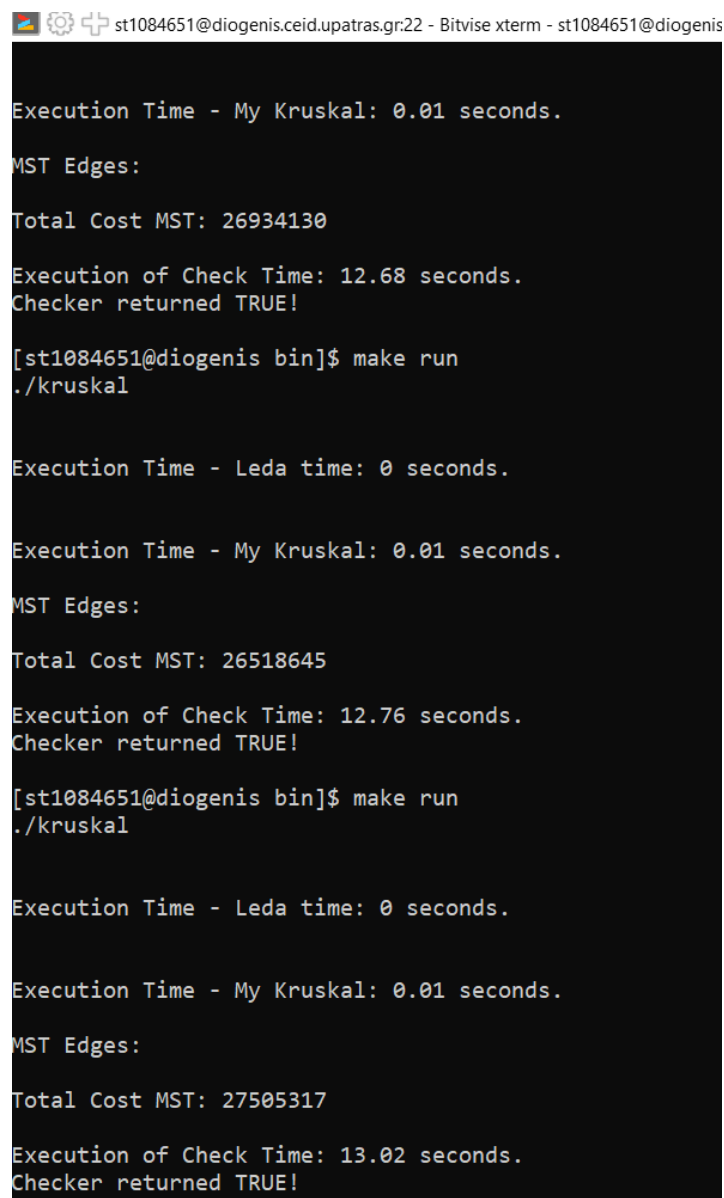
## ΠΑΡΑΤΗΡΗΣΗ:

Ο χρόνος εκτέλεσης της LEDA ήταν 0.05 δευτερόλεπτα, ενώ ο δικός μας Kruskal εκτελέστηκε σε περίπου 0.37–0.38 δευτερόλεπτα. Ο ελεγκτής χρειάστηκε πλέον πάνω από 10 λεπτά (περίπου 640–650 δευτερόλεπτα). Το συνολικό κόστος του MST κυμάνθηκε γύρω στα 3.600.000. Γενικότερα, καθώς αυξάνεται το μέγεθος των γραφημάτων, γίνεται εμφανής η χρησιμότητα και η αξία χρήσης της LEDA.

### 2) *Γραφήματα τύπου πλέγματος (grid) μεγέθους $r \times c$ (γραμμές $\times$ στήλες), όπου $(r, c) \in \{(100,100), (200,200), (300,300)\}$*

Στην παρούσα δοκιμή χρησιμοποιήθηκε γράφημα τύπου grid διαστάσεων 100×100, δηλαδή με 10.000 κορυφές και συνολικά 9.900 ακμές, καθώς κάθε κόμβος συνδέεται με τον δεξιό και κάτω γείτονά του. Λόγω της κατασκευαστικής του δομής, το grid είναι εκ φύσεως συνεκτικό, οπότε δεν απαιτήθηκε η χρήση της συνάρτησης Make\_Connected της LEDA.

#### SCREENSHOT ΓΙΑ 100x100:



```
st1084651@diogenis.ceid.upatras.gr:22 - Bitwise xterm - st1084651@diogenis

Execution Time - My Kruskal: 0.01 seconds.
MST Edges:
Total Cost MST: 26934130
Execution of Check Time: 12.68 seconds.
Checker returned TRUE!

[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0 seconds.

Execution Time - My Kruskal: 0.01 seconds.
MST Edges:
Total Cost MST: 26518645
Execution of Check Time: 12.76 seconds.
Checker returned TRUE!

[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0 seconds.

Execution Time - My Kruskal: 0.01 seconds.
MST Edges:
Total Cost MST: 27505317
Execution of Check Time: 13.02 seconds.
Checker returned TRUE!
```

#### SCREENSHOT ΓΙΑ 200x200:



```
st1084651@diogenis.ceid.upatras.gr:22 - Bitwise xterm - st1084651@diogenis:~/AI
[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0.03 seconds.

Execution Time - My Kruskal: 0.05 seconds.

MST Edges:

Total Cost MST: 108061694

Execution of Check Time: 236.16 seconds.
Checker returned TRUE!

[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0.03 seconds.

Execution Time - My Kruskal: 0.04 seconds.

MST Edges:

Total Cost MST: 107318553

Execution of Check Time: 221.28 seconds.
Checker returned TRUE!

[st1084651@diogenis bin]$
```

### SCREENSHOT ΓΙΑ 300x300:

```
st1084651@diogenis.ceid.upatras.gr:22 - Bitwise xterm - st1084651@diogenis:~/AI
[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0.11 seconds.

Execution Time - My Kruskal: 0.18 seconds.

MST Edges:

Total Cost MST: 242257385

make: *** [run] CPU time limit exceeded (core dumped)
[st1084651@diogenis bin]$ make run
./kruskal

Execution Time - Leda time: 0.11 seconds.

Execution Time - My Kruskal: 0.18 seconds.

MST Edges:

Total Cost MST: 242331928
```

### ΠΑΡΑΤΗΡΗΣΗ:

Το grid αποτελεί ένα αραιό γράφημα, χαρακτηριστικό που επιτρέπει στον αλγόριθμο Kruskal να εκτελείται ιδιαίτερα αποδοτικά, όπως επιβεβαιώθηκε από τους χρόνους εκτέλεσης. Αυτό όμως δεν ισχύει και για τον checker, ο οποίος στη δικιά μου υλοποίηση (που ο BFS τρέχει για κάθε ακμή εκτός του MST) χρειάζεται πολύ χρόνο για να επιστρέψει το αποτέλεσμα, και κατά συνέπεια στο 300x300 grid βγάζει “CPU time limit exceeded”. Η κανονική διάταξη του grid καθιστά την κατασκευή του MST πιο προβλέψιμη και ομοιόμορφη σε σχέση με τα τυχαία γραφήματα, με αποτέλεσμα ένα συνεκτικό και γεωμετρικά ισορροπημένο ελάχιστο δέντρο.

### 3) Συνθετικά Γραφήματα χειρότερης περίπτωσης:



Ο Kruskal είναι ταχύτερος όταν οι ενώσεις των συνεκτικών συνιστωσών γίνονται γρήγορα και δεν χρειάζεται πολλές ενημερώσεις. Η "χειρότερη" περίπτωση είναι εκείνη που:

- 1) προκαλεί πολλές συγχωνεύσεις λιστών,
- 2) οδηγεί σε μεγάλες λίστες που συγχωνεύουν μικρές και απαιτούν πολλές ενημερώσεις των δεικτών first,
- 3) έχει πάρα πολλές ακμές, άρα ο αλγόριθμος κάνει πολλούς ελέγχους κύκλου,
- 4) έχει ακμές με παρόμοιο ή ίδιο κόστος, οπότε δεν γίνεται γρήγορη αποκοπή.

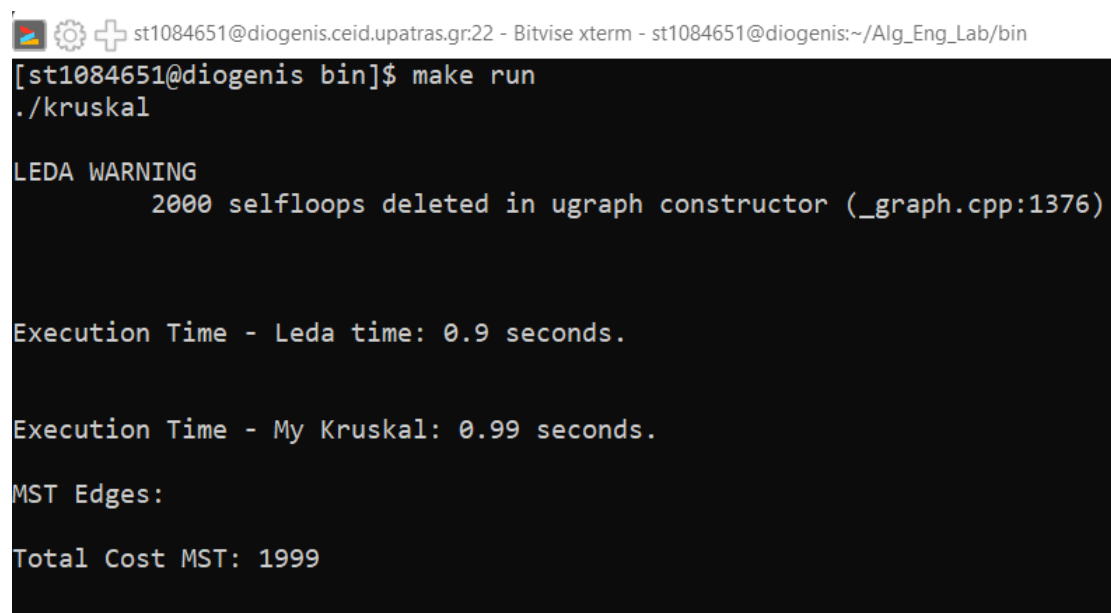
### **Επιλογή Πλήρους Γραφήματος με Ομοιόμορφα Βάρη**

Για να μελετήσουμε τη χειρότερη περίπτωση του αλγορίθμου Kruskal, χρησιμοποιούμε ένα πλήρες γράφημα με 2000 κορυφές. Σε ένα πλήρες γράφημα, κάθε κόμβος συνδέεται με όλους τους άλλους, άρα έχουμε πολλές ακμές. Ο αλγόριθμος Kruskal βασίζεται στη διαδικασία ταξινόμησης των ακμών, και όσο περισσότερες είναι, τόσο μεγαλύτερο είναι το κόστος της ταξινόμησης.

Όλες οι ακμές έχουν βάρος 1, κάτι που σημαίνει ότι ο αλγόριθμος πρέπει να εξετάσει όλες τις ακμές, χωρίς να μπορεί να απορρίψει κάποια νωρίτερα. Αυτό σημαίνει ότι η εκτέλεση του αλγορίθμου εξαρτάται κυρίως από τη διαχείριση των συνιστωσών (ομάδων) κορυφών και τον έλεγχο των κύκλων, κάτι που κάνει τη διαχείριση των δεδομένων πιο δύσκολη.

### **SCREENSHOT ΓΙΑ 2000 NODES**

**(COMPLETE GRAPH WITH THE SAME WEIGHTS ON THE EDGES):**



```
st1084651@diogenis.ceid.upatras.gr:22 - Bitwise xterm - st1084651@diogenis:~/Alg_Eng_Lab/bin
[st1084651@diogenis bin]$ make run
./kruskal

LEDA WARNING
    2000 selfloops deleted in ugraph constructor (_graph.cpp:1376)

Execution Time - Leda time: 0.9 seconds.

Execution Time - My Kruskal: 0.99 seconds.

MST Edges:

Total Cost MST: 1999
```

### **ΠΑΡΑΤΗΡΗΣΗ:**

Το συνολικό κόστος του ελάχιστου δέντρου αναζήτησης (MST) είναι 1999, το οποίο είναι ακριβώς αυτό που αναμενόταν για το πλήρες γράφημα με 2000 κόμβους και βάρη 1 για κάθε ακμή. Ωστόσο, η εκτέλεση και ολοκλήρωση του αλγορίθμου τόσο στην περίπτωση της LEDA όσο και στη δικιά μας υλοποίηση πλησίασε το 1 δευτερόλεπτο, σαφώς μεγαλύτερη σε σχέση με προηγούμενα γραφήματα, κάτι που αναδεικνύει την χειρότερη περίπτωση.

### **SCREENSHOT ΓΙΑ 3000 NODES (!)**

**(COMPLETE GRAPH WITH THE SAME WEIGHTS ON THE EDGES):**

```
st1084651@diogenis.ceid.upatras.gr:22 - Bitvise xterm - st1084651@diogenis:~/Alg_Eng_Lab/bin
[st1084651@diogenis bin]$ make run
./kruskal

LEDA WARNING
    3000 selfloops deleted in ugraph constructor (_graph.cpp:1376)

LEDA ERROR HANDLER
    memory_manager::allocate_vector: out of memory (_memory_std.cpp:363)

#00 at 0x0041a420 in
#01 at 0x0041a629 in
#02 at 0x00417a47 in
#03 at 0x0041c3a1 in
#04 at 0x00409bc1 in
#05 at 0xc5a58d20 in __libc_start_main
#06 at 0x00407bb9 in
make: *** [run] Aborted (core dumped)
```

#### ΠΑΡΑΤΗΡΗΣΗ:

Με την πρόσθεση περισσότερων κόμβων, (όπως τους 3000 κόμβους), το πλήρες γράφημα γίνεται πολύ μεγαλύτερο και απαιτεί περισσότερη μνήμη για να αποθηκευτούν οι ακμές και οι δομές δεδομένων του γραφήματος. Αυτός ο μεγάλος αριθμός ακμών απαιτεί πολύ περισσότερη μνήμη για την αποθήκευση της πληροφορίας, τόσο για τις ακμές όσο και για τις δομές δεδομένων που χρησιμοποιούνται στον αλγόριθμο Kruskal (π.χ. για τις ενώσεις και τον έλεγχο κύκλων).