ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ ΠΛΗΡΟΦΟΡΙΑΣ
Computational Methodologies & Operations Research
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
artificial intelligence and data analytics

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (2ο εξάμηνο)

Εργασία: week – 10 (4η Εργασία)
Ακαδημαϊκό Έτος : 2022-2023
Εισηγητής: Ηλίας Σακελλαρίου

Άσκηση 1: Compiling on the Cloud

Πρέπει να μεταγλωττίσετε (compile) ένα αριθμό από αρχεία (c1, c2, c3..κλπ) τα οποία είναι διαφορετικού μεγέθους και έχουν *διαφορετικούς χρόνους μεταγλώττισης* (compilation). Για τις μεταγλωττίσεις αυτές έχετε *ένα αριθμό από servers* και έχετε την ελευθερία να μεταγλωττίσετε το αρχείο *σε οποιονδήποτε server*, αλλά μόνο σε έναν. Κάθε server μπορεί να μεταγλωττίσει ένα μόνο αρχείο κάθε χρονική στιγμή. Τα source αρχεία είναι διαθέσιμα εξαρχής σε όλους τους servers. Υπάρχουν *εξαρτήσεις* μεταξύ των αρχείων, για παράδειγμα το compile του αρχείου c3 απαιτεί να υπάρχει το μεταγλωττισμένο αρχείο του c1 στον server που θα γίνει η μεταγλώττιση του αρχείου c3, δηλαδή στην γενική περίπτωση απαιτείται:

- να έχει προηγηθεί η μεταγλώττιση του c1 στον *ίδιο server,* **ή**
- να έχει να έχει προηγηθεί η μεταγλώττιση του c1 σε *άλλο server* και να έχει μεταφερθεί το αρχείο στον server όπου θα μεταγλωττιστεί το c3.

Τα διάφορα αρχεία έχουν διαφορετικούς χρόνους μεταγλώττισης και μεταφοράς. Για παράδειγμα το αρχείο c1, απαιτεί 10 χρονικές στιγμές για την μεταγλώττιση του και 18 για την μεταφορά του στους άλλους servers. Υποθέτουμε ότι ένα αρχείο που μεταγλωττίζετε σε ένα server μεταφέρεται σε compiled μορφή αυτόματα σε όλους τους άλλους servers.

Τα στοιχεία για κάθε στιγμιότυπο του προβλήματος σας δίνονται με την ακόλουθη μορφή:

```
FILES = {c0, c1, c2, c3, c4, c5};
```

FILES είναι απαριθμητός τύπος με τα ονόματα των αρχείων,

```
servers = 2;
```

servers, ο αριθμός των διαθέσιμων servers,

```
timeTo = [|15,5|10,18|15,35| 13,52|20,52|15,21|];
```

timeTo είναι ένας πίνακας 2 διαστάσεων με κάθε γραμμή να αντιστοιχεί σε ένα αρχείο (c0,c1,...) και η πρώτη στήλη να είναι ο χρόνος μεταγλώττισης (**COMP**) και η δεύτερη στήλη ο χρόνος μεταφοράς (**REPL**). Για παράδειγμα στον παραπάνω πίνακα για το αρχείο c0, ο χρόνος μεταγλώττισης είναι **timeTo[c0,COMP] = 15** και ο χρόνος ολοκλήρωσης **timeTo[c0,REPL] = 5**.

```
dependencies = [{},{},{c0},{c1},{c1,c2},{c2,c3}];
```

dependencies είναι μια λίστα από σύνολα (με ένα στοιχείο για κάθε αρχείο) όπου το κάθε σύνολο δίνει τις εξαρτήσεις του κάθε αρχείου. Για παράδειγμα το αρχείο c2 εξαρτάται από το c0.

```
deadline = [20,20,10,40,30,40];
```

deadline είναι η καταληκτική χρονική στιγμή που θα πρέπει να είναι έτοιμο (μεταγλωττισμένο) το κάθε αρχείο. Προφανώς η καταληκτική ημερομηνία του πρώτου αρχείου (c0) είναι το 20, κοκ.

Τέλος η ποιότητα της λύσης εξαρτάται από το lateness, την διαφορά μεταξύ του ολοκλήρωσης της μεταγλώττισης και του deadline και μπορεί να πάρει και αρνητικές τιμές.

Να γράψετε ένα πρόγραμμα σε MiniZinc το οποίο να βρίσκει πότε και σε ποιον server θα γίνει το compilation του κάθε αρχείου, έτσι ώστε να ελαχιστοποιείται το lateness. Το MiniZinc πρόγραμμά σας θα εμφανίζει:

- Τους συνολικούς πόντους (**points**) της αξιολόγησης της λύσης, που θα αντιστοιχούν στο άθροισμα του lateness των μεταγλωττίσεων των αρχείων. Προφανώς η καλύτερη λύση είναι εκείνη με τους *ελάχιστους πόντους*.
- Το πότε θα ξεκινήσει το compilation του κάθε αρχειου (**start**).
- Σε ποιον server θα γίνει (**in_server**).

Για παράδειγμα:

```
points = 79 ;
start = [0, 0, 15, 30, 43, 63] ;
in_server = [1, 2, 1, 1, 1, 1] ;
----------
```

Η παραπάνω λύση αξιολογείται με 79 πόντους, το compilation του c0 θα αρχίσει την χρονική στιγμή 0 και το c0 θα γίνει compile στον server 1. *Η λύση ΔΕΝ είναι η βέλτιστη.*

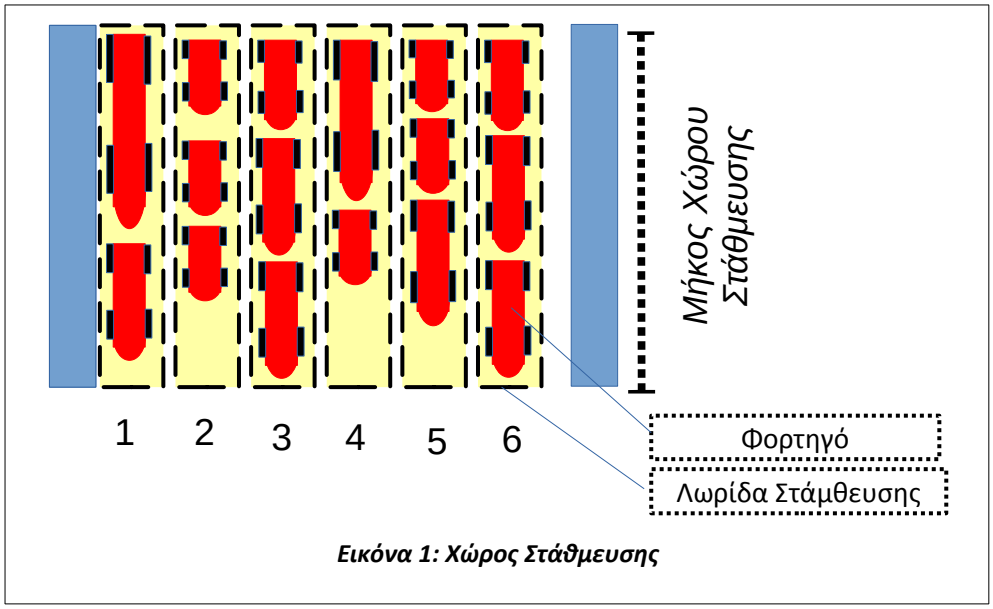
Τα δεδομένα των προβλημάτων και ο σκελετός βρίσκονται στο eclass.

(β) Να δοκιμάσετε διαφορετικούς δύο ευρετικούς αλγορίθμους κατά την αναζήτηση και να παραθέσετε τα αποτελέσματα τους (χρόνοι εκτέλεσης) για τα 4 διαφορετικά problem instances που σας δίνονται. Η σύγκρισή σας να περιλαμβάνει και τους χρόνους για την αναζήτηση χωρίς ευρετικό αλγόριθμο (συνολικά 3 χρόνοι για κάθε αρχείο). Μπορείτε να θέσετε ένα άνω χρονικό όριο, αν για κάποιο instance η συγκεκριμένη στρατηγική αναζήτησης διαρκεί πολύ.

Άσκηση 2: Parking

Ο χώρος στάθμευσης φορτηγών μεταφορών (trucks), οργανώνεται σε λωρίδες στάθμευσης, στις οποίες μπορούν να σταθμεύσουν περισσότερα του ενός φορτηγά, αρκεί το συνολικό τους μήκος να μην ξεπερνά το

μήκος της λωρίδας, όπως φαίνεται στο ακόλουθο σχήμα. Κάθε λωρίδα έχει ένα τύπο (laneA|laneB) και κάθε φορτηγό απαιτεί συγκεκριμένο τύπο λωρίδας για την στάθμευσή του.



Εικόνα 1: Χώρος Στάθμευσης

Το πρόβλημα είναι να βρείτε σε ποια λωρίδα θα πάει κάθε φορτηγό ώστε να ικανοποιούνται οι περιορισμοί και ο αριθμός των λωρίδων που χρησιμοποιούνται να είναι ο ελάχιστος δυνατός. Οι σχετικές πληροφορίες δίνονται σε πίνακες στα αντίστοιχα αρχεία δεδομένων (MiniZinc data files), όπως για παράδειγμα:

```
LANETYPES = {laneA, laneB};
```

είναι οι διαφορετικοί τύποι λωρίδων στάθμευσης,

```
number_of_parking_lanes = 5;
```

ο αριθμός των λωρίδων στάθμευσης,

```
parking_lane_length = 8;
```

το μήκος της λωρίδας (ίδιο για όλες),

```
number_of_trucks = 5;
```

ο συνολικός αριθμός των φορτηγών,

```
truck_length = [4,3,2,4,5];
```

Το μήκος του κάθε φορτηγού,

```
truck_type = [laneA,laneB,laneA,laneA,laneA];
```

ο τύπος της λωρίδας που απαιτεί,

```
parking_lane_type = [laneA,laneA,laneB,laneB,laneA];
```

ο τύπος της κάθε λωρίδας.

Να υλοποιήσετε ένα πρόγραμμα σε MiniZinc το οποίο υπολογίζει σε ποια λωρίδα θα σταθμεύσει κάθε φορτηγό (at_parking_lane), έτσι ώστε ο συνολικός αριθμός των λωρίδων (used_lanes) που θα χρησιμοποιηθούν να είναι ο ελάχιστος δυνατός. Παράδειγμα εκτέλεσης:

```
at_parking_lane = [1, 3, 1, 2, 5];
used_lanes = 4;
----------
```

όπου το παραπάνω, δηλώνει ότι το φορτηγό 2 θα σταθμεύσει στην λωρίδα 3, και θα χρησιμοποιηθούν συνολικά 4 λωρίδες.

(η παραπάνω λύση δεν είναι η βέλτιστη)

Στο ECLASS θα βρείτε όλα τα απαραίτητα αρχεία.

Άσκηση 3 - ServerUpdates

Σε ένα cloud πρέπει να γίνουν updates σε ένα σύνολο από servers. Έχετε στην διάθεσή σας ένα σύνολο εντολών οι οποίες είναι:

- **initialise**: Ένας κλειστός server (power off) ξεκινά και μπορεί να εκτελέσει εργασίες
- **stay_off:** Ο server παραμένει στην κατάσταση power off.
- **runserv**: Ο server εκτελεί κάποια εργασία.
- **download**: Ο server "κατεβάζει" τα αρχεία για το update.
- **update**: Γίνεται το update.
- **shutdown**: Ο server κλείνει (power-off).

Οι κανόνες είναι απλοί:

- Θα πρέπει να έχουν κατέβει τα αρχεία (download) **πριν** εκτελεστεί ένα update. Μετά από μια εντολή download, ο server είτε μπορεί να συνεχίσει να εκτελεί εργασίες (runserv) ή να εκτελέσει ένα update. Δεν μπορεί να κάνει shutdown.
- Μετά από update ο server πρέπει να κλείσει (shutdown).
- Εφόσον κάνει initialize μπορεί να εκτελέσει οποιοδήποτε αριθμό εργασιών (runserv), ή να κατεβάσει τα αρχεία (donwload), ή να κάνει shutdown.
- Ο server μπορεί να μείνει κλειστός (stay_off) ή σε λειτουργία (δηλαδή να δέχεται εντολές runserv) για πάντα.

Αν θα πρέπει:

- όλοι οι servers πρέπει να κάνουν update ΜΟΝΟ μια φορά,
- κάθε χρονική στιγμή μπορεί να γίνει ΜΟΝΟ ένα download λόγω περιορισμών του δικτύου,
- για κάθε χρονική στιγμή πλην της πρώτης, θα πρέπει να υπάρχει τουλάχιστον ένας server που εξυπηρετεί μια εργασία (runserv) ,

να υλοποιήσετε ένα πρόγραμμα σε MiniZinc που βρίσκει την ακολουθία εντολών στους servers, ώστε να κάνουν όλοι update, και να μεγιστοποιείται ο αριθμός των εντολών runserv που εκτελεί το σύνολο των servers. Παράδειγμα εκτέλεσης:

```
server_commands=array2d(1..4,1..7,[stay_off, stay_off, stay_off,
initilize, download, update, shutdown, initilize, runserv, runserv,
download, runserv, update, shutdown, initilize, runserv, download,
runserv, runserv, update, shutdown, initilize, download, update,
shutdown, initilize, runserv, runserv]);
max=8;
```

όπου **max** είναι ο αριθμός των εντολών runserv και το **server_commands** είναι η ακολουθία των εντολών για κάθε server για το χρονικό διάστημα υπό εξέταση (κανονικά πίνακας 2 διαστάσεων με κάθε γραμμή για ένα server), όπως φαίνεται παρακάτω:

```
stay_off, stay_off, stay_off, initilize, download, update, shutdown,
initilize, runserv, runserv, download, runserv, update, shutdown,
initilize, runserv, download, runserv, runserv, update, shutdown,
initilize, download, update, shutdown, initilize, runserv, runserv
```

Η παραπάνω λύση δεν είναι η βέλτιστη.

Στο ECLASS θα βρείτε το αρχείο με τα δεδομένα του προβλήματος.

Σημείωση: Η λύση που φαίνεται ΔΕΝ είναι η βέλτιστη.

Σημείωσεις

- Τα απαραίτητα αρχεία για την υλοποίηση των τριών ασκήσεων θα τα βρείτε στο ECLASS (Έγγραφα-> Constraints->Assignments)**.**
  - Άσκηση 1:φάκελος CompilingMultipleServers_2023 , αρχείο project CompilingOnMultiServers.mzp
  - Άσκηση 2: φάκελος ParkingTrucks, αρχείο parking_project.mzp
  - Άσκηση 3: φάκελος ServerCommands , αρχείο serverStates.mzn

ΠΑΡΑΔΟΣΗ

Θα παραδώσετε εντός της ημερομηνίας που αναφέρεται στο ECLASS τα ακόλουθα:

- Τα παραπάνω αρχεία με όνομα και τη δομή καταλόγων που έχουν στο ECLASS **σε ένα αρχείο code.zip.**
- Ένα αρχείο **report.pdf** (***σε μορφή pdf***) το οποίο θα περιέχει:
  - **Στην πρώτη σελίδα το Όνομά σας, τον Αριθμό μητρώου σας και το email σας.**
  - Για κάθε μια από τις ασκήσεις:
    - τον **κώδικα** σας (ασχέτως αν βρίσκεται και στο αντίστοιχο αρχείο) και σχολιασμό σχετικά με αυτόν.
    - Παραδείγματα εκτέλεσης (1-2 όπου είναι εφικτό)
    - Bugs και προβλήματα που έχει ο κώδικάς σας.

ΠΡΟΣΟΧΗ: ΝΑ ΑΚΟΛΟΥΘΗΣΕΤΕ ΑΥΣΤΗΡΑ ΤΑ ΟΝΟΜΑΤΑ ΤΩΝ ΜΕΤΑΒΛΗΤΩΝ ΠΟΥ ΔΙΝΟΝΤΑΙ ΠΑΡΑΠΑΝΩ (ΑΥΤΟΜΑΤΟΣ ΕΛΕΓΧΟΣ ΚΩΔΙΚΑ)

Καλή επιτυχία (*και have fun with MiniZinc!*)