ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ (2016-17)

Εργασία 4

Πολύ συχνά, ένα υπολογιστικό πρόβλημα έχει περισσότερες από μία λύση. Για παράδειγμα, αν θέλουμε να βρούμε ένα υποσύνολο ενός συνόλου ακεραίων, που το άθροισμα των στοιχείων του υποσυνόλου να είναι άρτιος αριθμός, είναι προφανές ότι μπορεί να υπάρχουν πολλά τέτοια υποσύνολα. Αν θέλουμε να κατασκευάσουμε ένα ωρολόγιο πρόγραμμα μαθημάτων για ένα σχολείο, το οποίο να ικανοποιεί κάποιους περιορισμούς, αυτό μπορεί να γίνει με πολλούς τρόπους. Σε προβλήματα σαν αυτά, εκείνο που κάνουμε είναι να εξερευνήσουμε το σύνολο (χώρο) των πιθανών λύσεων του προβλήματος, για να βρούμε κάποια (ή κάποιες) που μας ικανοποιεί. Με άλλα λόγια, *αναζητούμε* μία ή περισσότερες λύσεις μέσα από τον χώρο των πιθανών λύσεων, και γι΄ αυτόν τον λόγο, τα προβλήματα αυτά ονομάζονται *προβλήματα αναζήτησης* (search problems). Συνήθως, δεν μας ενδιαφέρει σε ένα πρόβλημα αναζήτησης να βρούμε όλες τις λύσεις του, επειδή, πιθανότατα, είναι πάρα πολλές, αλλά μας αρκεί απλώς μία λύση. Όμως, είναι ενδεχόμενο κάποιες λύσεις να είναι "καλύτερες" από κάποιες άλλες. Οπότε, αν έχουμε ένα σαφές και καλά ορισμένο μέτρο της ποιότητας μίας λύσης, είναι εύλογο να μας ενδιαφέρει να βρούμε κάποια καλή λύση, αν είναι δυνατόν την καλύτερη (*βέλτιστη*). Όταν σε ένα πρόβλημα αναζήτησης μας ενδιαφέρει να βρούμε μία βέλτιστη λύση του, ή έστω κάποια πολύ κοντά στη βέλτιστη, λέμε ότι το πρόβλημα αυτό είναι *πρόβλημα βελτιστοποίησης* (optimization problem).

Υπάρχουν πολλές τεχνικές για την επίλυση προβλημάτων βελτιστοποίησης. Μεταξύ αυτών είναι και εκείνες που ανήκουν στην οικογένεια των *αλγορίθμων που βασίζονται σε πληθυσμούς* (population-based algorithms). Οι αλγόριθμοι αυτοί προσομοιώνουν, κατά κανόνα, λειτουργίες που εμφανίζονται στη φύση και η γενική ιδέα τους είναι ότι συντηρούν ένα σύνολο από πιθανές λύσεις του προβλήματος βελτιστοποίησης (πληθυσμός) και, ανάλογα με τα ιδιαίτερα χαρακτηριστικά και τη φιλοσοφία του κάθε αλγορίθμου, γίνεται προσπάθεια να βελτιώνεται η ποιότητα των λύσεων (ατόμων) του πληθυσμού. Ενδεικτικά αναφέρονται οι αλγόριθμοι:

- Βελτιστοποίηση με αποικίες μυρμηγκιών (Ant colony optimization)
- Βελτιστοποίηση με σμήνος σωματιδίων (Particle swarm optimization)
- Βελτιστοποίηση μέσω αναζήτησης τροφής από βακτήρια (Bacterial foraging optimization)
- Βελτιστοποίηση με αποικίες μελισσών (Bee colony optimization)
- Τεχνητά ανοσοποιητικά συστήματα (Artificial immune systems)
- Βελτιστοποίηση βασισμένη στη βιογεωγραφία (Biogeography-based optimization)
- Βελτιστοποίηση μέσω καταιγισμού ιδεών (Brain storm optimization)

Όμως, η περισσότερο διαδεδομένη και παλαιότερη κατηγορία αλγορίθμων που εντάσσεται στην οικογένεια των αλγορίθμων που βασίζονται σε πληθυσμούς είναι οι *γενετικοί αλγόριθμοι* (genetic algorithms). Οι αλγόριθμοι αυτοί προσομοιώνουν τη διαδικασία της *εξέλιξης* (evolution) στη φύση, ακολουθώντας τους κανόνες της *φυσικής επιλογής* (natural selection). Συντηρούν ένα πληθυσμό από πιθανές λύσεις του προβλήματος ως άτομα ή *χρωμοσώματα* (chromosomes), τον οποίο εξελίσσουν σε συνεχόμενες *γενεές* (generations), εφαρμόζοντας τους γενετικούς τελεστές της *διασταύρωσης* (crossover) και της *μετάλλαξης* (mutation). Ένα χρωμόσωμα αποτελείται από *γονίδια* (genes), τα οποία κωδικοποιούν τα χαρακτηριστικά της πιθανής λύσης που αναπαριστά το χρωμόσωμα. Σε κάθε γενετικό αλγόριθμο, ορίζεται και μία *συνάρτηση ευρωστίας* (fitness function), η οποία ποσοτικοποιεί

την ποιότητα κάθε χρωμοσώματος, άρα και της πιθανής λύσης που αναπαριστά. Περισσότερες λεπτομέρειες μπορείτε να δείτε στον σύνδεσμο `https://en.wikipedia.org/wiki/Genetic_algorithm`, καθώς και στα παραδείγματα προβλημάτων που θα ακολουθήσουν.

Αντικείμενο της εργασίας αυτής είναι η υλοποίηση σε C γενετικών αλγορίθμων που επιλύουν συγκεκριμένα προβλήματα βελτιστοποίησης.

Μαντεύοντας ένα δυαδικό αριθμό (10%)

Έστω ότι επιλέγετε ένα δυαδικό αριθμό και θέλετε να γράψετε ένα πρόγραμμα που να μπορεί να τον μαντέψει. Για να βοηθηθεί το πρόγραμμα, θα πρέπει να μπορεί να γνωρίζει για κάθε "μαντεψιά" που κάνει πόσο κοντά είναι αυτή στον δυαδικό σας αριθμό, δηλαδή πόσα 1 και 0 έχει στις σωστές θέσεις.

Πώς θα μπορούσαμε να αντιμετωπίσουμε το πρόβλημα αυτό με ένα γενετικό αλγόριθμο; Ας το δούμε μέσω ενός συγκεκριμένου παραδείγματος. Έστω ότι επιλέξατε ένα δυαδικό αριθμό με 8 ψηφία, που είναι ο

1	1	0	0	0	1	0	0

Θα πρέπει να καθορίσουμε μία κωδικοποίηση των λύσεων του προβλήματος ως χρωμοσώματα, δηλαδή ακολουθίες από γονίδια. Είναι προφανές ότι ένας δυαδικός πίνακας 8 στοιχείων (1 ή 0) είναι κατάλληλος για μία τέτοια κωδικοποίηση. Ο γενετικός αλγόριθμος του προγράμματος θα πρέπει να συντηρεί και να εξελίσσει ένα πληθυσμό από τέτοια χρωμοσώματα. Για παράδειγμα, ένα τέτοιο χρωμόσωμα θα μπορούσε να ήταν το

1	0	0	1	0	1	1	0

το οποίο αναπαριστά μία πιθανή λύση του προβλήματος, δηλαδή μία "μαντεψιά" του προγράμματος, η οποία έχει πέντε σωστά bits σε σχέση με τον επιλεγμένο από εσάς αριθμό.

Για τον γενετικό αλγόριθμο που θα καθορίσουμε, θα πρέπει να ορίσουμε και τη συνάρτηση ευρωστίας για τα άτομα του πληθυσμού. Αν φροντίσουμε το πρόγραμμα να μάθει τον δυαδικό αριθμό N που έχουμε επιλέξει, θα μπορούσε να έχει ορίσει μία τέτοια συνάρτηση f, η οποία να επιστρέφει για κάθε χρωμόσωμα C ένα αριθμό/ευρωστία που θα είναι τόσο πιο μεγάλος όσο πιο πολλά είναι τα κοινά bits των N και C. Αν το πλήθος των κοινών bits είναι k, θα μπορούσαμε να ορίσουμε την f ως $f(C) = k^2$, για να "πριμοδοτηθούν" κάπως περισσότερο τα χρωμοσώματα που έχουν πιο πολλά κοινά bits με τον επιλεγμένο αριθμό.

Ας δούμε όμως πώς θα μπορούσαμε να υλοποιήσουμε ένα γενετικό αλγόριθμο για το εν λόγω πρόβλημα, αλλά και γενικότερα, για οποιοδήποτε πρόβλημα βελτιστοποίησης που η δυαδική κωδικοποίηση στα χρωμοσώματα είναι μία καλή επιλογή. Εδώ θα πρέπει να σημειωθεί ότι οι γενετικοί αλγόριθμοι έχουν στοχαστικό χαρακτήρα, δηλαδή σε διάφορα σημεία τους πρέπει να πάρουν τυχαίες αποφάσεις. Αυτό επιτυγχάνεται μέσω των γεννητριών ψευδοτυχαίων αριθμών που παρέχουν οι γλώσσες προγραμματισμού. Επίσης, ένας γενετικός αλγόριθμος έχει διάφορες παραμέτρους, οι οποίες είναι καλό σε μία υλοποίηση να ορίζονται με τέτοιο τρόπο ώστε να μπορούν να αλλάζουν εύκολα, για τη διευκόλυνση πειραματισμών.

Βήμα 1 (αρχικοποίηση): Έστω P το μέγεθος του πληθυσμού (παράμετρος). Δημιουργούμε P τυχαία χρωμοσώματα, δηλαδή τον αρχικό μας πληθυσμό, που γίνεται ο τρέχων για τον αλγόριθμο. Σημειώνεται ότι δεν είναι απαραίτητο τα άτομα του πληθυσμού να είναι διαφορετικά μεταξύ τους. Λόγω της τυχαιότητας, υπάρχει ενδεχόμενο να γεννηθεί το ίδιο άτομο περισσότερες από μία φορά.

Βήμα 2 (έλεγχος τερματισμού): Αν ο αλγόριθμος έχει ήδη δημιουργήσει G γενεές πληθυσμών (παράμετρος), τερματίζει και επιστρέφει ως λύση το καλύτερο άτομο του τρέχοντος πληθυσμού (ή το καλύτερο άτομο από όλες τις γενεές που έχουν δημιουργηθεί). Αλλιώς, αν στον τρέχοντα πληθυσμό

υπάρχει άτομο με ευρωστία μεγαλύτερη ή ίση κάποιας τιμής-στόχου FT (παράμετρος), ο αλγόριθμος τερματίζει και επιστρέφει το άτομο αυτό ως λύση.

Βήμα 3 (επιλογή): Δημιουργείται, με βάση τον τρέχοντα πληθυσμό, ένας προσωρινός πληθυσμός για την επόμενη γενεά με τον εξής τρόπο. Έστω ότι τα P άτομα a_1, a_2, ..., a_P του πληθυσμού έχουν ευρωστίες F_1, F_2, ..., F_P, αντίστοιχα, και

$$SF = \sum_{i=1}^{P} F_i$$

είναι το άθροισμα όλων των ευρωστιών. Η σχετική ευρωστία του ατόμου a_i $(1 \leq i \leq P)$ ορίζεται ως

$$f_i = \frac{F_i}{SF}$$

Θεωρούμε ότι το διάστημα $[0, 1)$ χωρίζεται στα P υποδιαστήματα, πλάτους f_i το καθένα

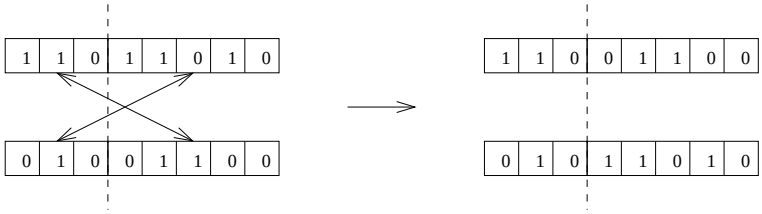
$$[0, f_1), \quad [f_1, f_1 + f_2), \quad [f_1 + f_2, f_1 + f_2 + f_3), \quad \ldots, \quad [f_1 + f_2 + \ldots + f_{P-1}, 1)$$

Δηλαδή, το i-οστό διάστημα είναι το

$$\left[\sum_{j=1}^{i-1} f_j, \sum_{j=1}^{i} f_j\right)$$

Επιλέγουμε ένα τυχαίο αριθμό στο διάστημα $[0, 1)$. Αν ο αριθμός αυτός ανήκει στο i-οστό διάστημα από τα παραπάνω, τότε το άτομο a_i "περνά" στον προσωρινό πληθυσμό. Επαναλαμβάνουμε αυτή τη διαδικασία P φορές, οπότε ο προσωρινός πληθυσμός περιέχει πλέον P άτομα, όχι κατ' ανάγκη διαφορετικά μεταξύ τους, έστω τα b_1, b_2, ..., b_P.

Βήμα 4 (διασταύρωση): Τα άτομα του προσωρινού πληθυσμού δημιουργούν τα ζευγάρια (b_1, b_2), (b_3, b_4), (b_5, b_6), ... Αν το P είναι περιττός, το άτομο b_P μένει "αζευγάρωτο". Για κάθε ζευγάρι, αποφασίζουμε αν θα διασταυρωθεί επιλέγοντας ένα τυχαίο αριθμό r στο διάστημα $[0, 1)$ και συγκρίνοντάς τον με την πιθανότητα διασταύρωσης p_c (παράμετρος) που έχουμε καθορίσει για τον αλγόριθμό μας. Αν ισχύει $r < p_c$, το ζευγάρι θα διασταυρωθεί και θα αντικατασταθεί στον προσωρινό πληθυσμό από τους απογόνους του, αλλιώς τα άτομά του θα παραμείνουν ως έχουν στον προσωρινό πληθυσμό (στον προσωρινό πληθυσμό παραμένει και το b_P, αν το P είναι περιττός). Για τη διασταύρωση ενός ζευγαριού, επιλέγεται με τυχαίο τρόπο ένα σημείο κοπής των δύο χρωμοσωμάτων (στην ίδια θέση και για τα δύο) και αυτά διασταυρώνονται χιαστί για να δημιουργήσουν δύο απογόνους. Για την επιλογή του σημείου κοπής, αν ένα χρωμόσωμα αποτελείται από N γονίδια, οπότε υπάρχουν $N - 1$ πιθανά σημεία κοπής, αρκεί να επιλέξουμε ένα τυχαίο ακέραιο αριθμό στο διάστημα $[1, N - 1]$. Στο παράδειγμα του σχήματος που ακολουθεί, δύο χρωμοσώματα διασταυρώνονται στη θέση κοπής 3 και παράγουν δύο απογόνους.



Οπότε, μετά το βήμα της διασταύρωσης, έχουμε ένα νέο προσωρινό πληθυσμό από P άτομα, έστω τα c_1, c_2, ..., c_P.

Βήμα 5 (μετάλλαξη): Για κάθε γονίδιο κάθε χρωμοσώματος c_i του προσωρινού πληθυσμού αποφασίζουμε με πιθανότητα p_m (παράμετρος) αν θα μεταλλαχθεί. Δηλαδή, επιλέγουμε ένα τυχαίο

αριθμό r στο διάστημα $[0, 1)$ και τον συγκρίνουμε με το p_m. Αν $r < p_m$, το γονίδιο μεταλλάσσεται, δηλαδή αν είναι 1, γίνεται 0, ή αν είναι 0, γίνεται 1. Μετά και το βήμα αυτό, ο προσωρινός πληθυσμός γίνεται ο πληθυσμός της τρέχουσας γενιάς, οπότε μεταβαίνουμε στο **βήμα 2**.

Ο αλγόριθμος που παρουσιάστηκε προηγουμένως ακολουθεί τις αρχές των γενετικών αλγορίθμων, αλλά θα μπορούσαμε να σκεφτούμε και πολλές παραλλαγές του, που επίσης να είναι αποδεκτοί γενετικοί αλγόριθμοι. Για παράδειγμα, τα βήματα "επιλογή – διασταύρωση – μετάλλαξη" δεν είναι απαραίτητο να γίνονται με αυτή τη σειρά. Επίσης, δεν είναι απαραίτητο οι απόγονοι της διασταύρωσης να αντικαθιστούν τους γονείς τους. Θα μπορούσαν να συνυπάρχουν με αυτούς. Ή τα μεταλλασσόμενα χρωμοσώματα μπορούν να συνυπάρχουν με αυτά που προκύπτουν από τις μεταλλάξεις. Η διασταύρωση δεν είναι απαραίτητο να γίνεται με ένα σημείο κοπής. Θα μπορούσαμε να είχαμε δύο ή περισσότερα, ιδιαίτερα σε προβλήματα με μεγάλο μέγεθος χρωμοσωμάτων. Το μέγεθος του πληθυσμού δεν είναι απαραίτητο να παραμένει σταθερό από γενεά σε γενεά. Μπορεί να εφαρμοσθεί *ελιτισμός* (elitism), δηλαδή στη διαδικασία της επιλογής, τα K (παράμετρος) καλύτερα άτομα να περνούν απ' ευθείας στην επόμενη γενεά και η τυχαία επιλογή να εφαρμόζεται μόνο για τα υπόλοιπα άτομα του πληθυσμού. Σε προβλήματα που δεν είναι κάθε πιθανό χρωμόσωμα έγκυρη λύση του προβλήματος, επειδή ενδεχομένως παραβιάζει κάποιους περιορισμούς, θα μπορούσαμε είτε να έχουμε κάποια "ποινή" στην ευρωστία αυτών χρωμοσωμάτων, έτσι ώστε να τείνουν να εξαφανίζονται στην εξέλιξη του πληθυσμού, είτε να εφαρμόζουμε επάνω τους κατάλληλο τελεστή *επιδιόρθωσης* (repair) ώστε να αντιστοιχούν σε έγκυρες λύσεις.

Με βάση τα προηγούμενα, το ζητούμενο είναι να υλοποιήσετε σε C ένα γενετικό αλγόριθμο που να κάνει καλές "μαντεψιές" για δυαδικούς αριθμούς που επιλέγουμε. Το πρόγραμμα που θα γράψετε (έστω ότι το εκτελέσιμο ονομάζεται "`guessga`") θα πρέπει να διαβάζει από την πρότυπη είσοδο τον δυαδικό αριθμό που επιλέξαμε και πρέπει αυτό να μαντέψει. Εννοείται ότι το πρόγραμμα δεν θα χρησιμοποιεί τον αριθμό αυτό, παρά μόνο για τον υπολογισμό της ευρωστίας κάθε χρωμοσώματος του πληθυσμού. Τις παραμέτρους του γενετικού αλγορίθμου μπορείτε είτε να τις ορίσετε μέσα στο πρόγραμμά σας ως συμβολικές σταθερές, είτε να τις δίνετε ως ορίσματα στη γραμμή εντολής κατά την εκτέλεση του προγράμματος. Στην τελευταία περίπτωση, έχετε απόλυτη ελευθερία στον τρόπο με τον οποίο θα δίνονται τα ορίσματα, αλλά θα πρέπει να τεκμηριώσετε με σαφήνεια, είτε σε σχόλια στο ίδιο το πρόγραμμα, είτε σε κάποιο συνοδευτικό αρχείο README, τις ακριβείς οδηγίες για την εκτέλεση του προγράμματός σας. Το ιδανικό είναι να ακολουθήσετε συνδυασμό των δύο μεθόδων, δηλαδή να ορίσετε μέσω συμβολικών σταθερών κάποιες default τιμές για τις παραμέτρους, αλλά να υπάρχει η δυνατότητα να δοθούν διαφορετικές επιθυμητές τιμές μέσω ορισμάτων.

Ακολουθεί μία ενδεικτική εκτέλεση ενός γενετικού αλγορίθμου για το πρόβλημα. Σημειώστε ότι είναι αδύνατον να αναπαραγάγετε τα αποτελέσματα αυτά, λόγω της τυχαιότητας του αλγορίθμου και των διαφοροποιήσεων που έχει τη δυνατότητα να κάνει κάποιος στις λεπτομέρειες της υλοποίησης. Πάντως για την παρακάτω εκτέλεση, έχουμε πληθυσμό $P = 10$ ατόμων, μέγιστο πλήθος γενεών $G = 50$, τιμή-στόχο FT για την ευρωστία ίση με L^2, όπου L είναι το μήκος του δυαδικού αριθμού που επιλέξαμε, και πιθανότητες διασταύρωσης $p_c = 0.3$ και μετάλλαξης $p_m = 0.1$.

```
$ ./guessga
10001110010110011100010101010101
Generation 0: Best fitness = 361   Average fitness = 225.400000
Generation 1: Best fitness = 324   Average fitness = 254.600000
Generation 2: Best fitness = 361   Average fitness = 260.800000
Generation 3: Best fitness = 361   Average fitness = 280.000000
Generation 4: Best fitness = 441   Average fitness = 326.800000
Generation 5: Best fitness = 484   Average fitness = 301.100000
.............................................................
```

```
Generation 45: Best fitness = 529   Average fitness = 260.400000
Generation 46: Best fitness = 625   Average fitness = 301.000000
Generation 47: Best fitness = 625   Average fitness = 461.700000
Generation 48: Best fitness = 625   Average fitness = 422.500000
Generation 49: Best fitness = 529   Average fitness = 385.800000
Generation 50: Best fitness = 576   Average fitness = 352.300000
Your number: 10001110010110011000101010101
My guess is: 00001110010110101111010101010101
Found in generation 46
Fitness = 625 (25 matches out of 30)
$
```

Μεγιστοποίηση συνάρτησης πολλών πραγματικών μεταβλητών (20%)

Έστω η πραγματική συνάρτηση f πέντε πραγματικών μεταβλητών x_1, x_2, x_3, x_4, x_5, που παίρνει τιμές από το διάστημα $[-1, 1]$[1]

$$f(x_1, x_2, x_3, x_4, x_5) = 10 - 0.1 \sum_{i=1}^{5} \cos(5\pi x_i) + \sum_{i=1}^{5} x_i^2$$

Υλοποιήστε σε C ένα γενετικό αλγόριθμο που να βρίσκει τις τιμές των x_i για τις οποίες η συνάρτηση μεγιστοποιείται, καθώς και τη μέγιστη τιμή της συνάρτησης.

Υπόδειξη: Μπορείτε να κωδικοποιήσετε μία πιθανή λύση ως ένα χρωμόσωμα πέντε γονιδίων που θα παίρνουν πραγματικές τιμές στο διάστημα $[-1, 1]$. Ως συνάρτηση ευρωστίας για τα χρωμοσώματα μπορείτε να θεωρήσετε την ίδια τη συνάρτηση. Η διασταύρωση μπορεί να γίνει κλασικά με ένα σημείο κοπής, όπως στον αλγόριθμο που περιγράφηκε για το προηγούμενο πρόβλημα. Για τη μετάλλαξη, ένα γονίδιο (τιμή μεταβλητής x_i) μπορεί να αλλάζει σε κάποια τυχαία τιμή στο διάστημα $[-1, 1]$.

Θα ήταν ενδιαφέρον, αν το πρόγραμμά σας ήταν πιο γενικό, δηλαδή αν μπορούσε να βρει το μέγιστο της συνάρτησης N μεταβλητών

$$f(x_1, x_2, \ldots, x_N) = 2N - 0.1 \sum_{i=1}^{N} \cos(5\pi x_i) + \sum_{i=1}^{N} x_i^2$$

Παρατηρήστε ότι η συνάρτηση που δόθηκε αρχικά με τις πέντε μεταβλητές είναι ειδική περίπτωση αυτής για $N = 5$. Αν υλοποιήσετε τη γενικότερη εκδοχή του γενετικού αλγορίθμου, το N να δίνεται στο πρόγραμμά σας από τη γραμμή εντολής.

Το πρόβλημα του πλανόδιου πωλητή (70%)

Στο *πρόβλημα του πλανόδιου πωλητή* (traveling[2] salesman problem), είναι δεδομένο ένα σύνολο από πόλεις και, για κάθε πιθανό ζευγάρι πόλεων c_i και c_j, είναι γνωστό το κόστος μετάβασης $d(c_i, c_j)$ από τη μία προς την άλλη, που είναι το ίδιο για οποιαδήποτε από τις δύο πιθανές κατευθύνσεις, δηλαδή $d(c_i, c_j) = d(c_j, c_i)$. Αυτό το κόστος μπορεί να είναι, για παράδειγμα, η απόσταση των δύο πόλεων. Το ζητούμενο του προβλήματος είναι να βρεθεί με ποια σειρά πρέπει να επισκεφθεί ο πωλητής όλες τις πόλεις, αρχίζοντας από κάποια και καταλήγοντας σε αυτήν από την οποία ξεκίνησε, ώστε το συνολικό κόστος των μεταβάσεών του να είναι το ελάχιστο δυνατό. Η εκδοχή αυτή του προβλήματος αναφέρεται

[1] cos είναι η συνάρτηση του συνημιτόνου και π είναι το γνωστό 3.14159...

[2] "traveling" σε American English ή "travelling" σε British English

και σαν το *συμμετρικό* πρόβλημα του πλανόδιου πωλητή, επειδή τα κόστη μεταβάσεων μεταξύ δύο πόλεων είναι τα ίδια και για τις δύο κατευθύνσεις.

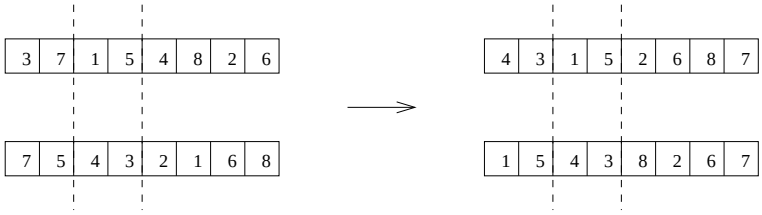
Το πρόβλημα του πλανόδιου πωλητή θα μπορούσε να αντιμετωπισθεί με τη βοήθεια ενός κατάλληλα σχεδιασμένου γενετικού αλγορίθμου. Πρώτα πρέπει να δει κανείς πώς θα γίνει η κωδικοποίηση μίας πιθανής λύσης του προβλήματος. Μία εύλογη επιλογή, αλλά όχι η μοναδική, θα ήταν η εξής. Αν έχουμε N πόλεις, μία λύση θα μπορούσε να κωδικοποιείται σαν ένα χρωμόσωμα από N γονίδια, που είναι κατά σειρά οι αύξοντες αριθμοί των πόλεων που θα επισκεφθεί ο πωλητής. Για παράδειγμα, αν $N = 8$, ένα πιθανό χρωμόσωμα θα μπορούσε να ήταν το

5	3	8	2	6	4	1	7

το οποίο σημαίνει ότι ο πωλητής ξεκινά από την πόλη 5, πηγαίνει μετά στην 3, μετά στην 8 κοκ., μέχρι να φτάσει στην πόλη 7 και να επιστρέψει τελικά πάλι στην 5. Τότε, αν είναι C το συνολικό κόστος μεταβάσεων μεταξύ πόλεων στη διαδρομή του πωλητή, δηλαδή $C = d(c_5, c_3) + d(c_3, c_8) + d(c_8, c_2) + \ldots + d(c_1, c_7) + d(c_7, c_5)$, θα μπορούσαμε να ορίσουμε την ευρωστία του ατόμου αυτού ως $F = 1/C$. Θυμηθείτε ότι καλύτερο άτομο σημαίνει μικρότερο C, δηλαδή μεγαλύτερο F.

Εκτός από τη διαφοροποίηση στην κωδικοποίηση που κάναμε σε σχέση με τα προηγούμενα προβλήματα, θα πρέπει να δούμε και το πώς θα ορίσουμε τους τελεστές διασταύρωσης και μετάλλαξης.

Εύκολα μπορούμε να δούμε ότι αν πάρουμε δύο χρωμοσώματα και τα διασταυρώσουμε με τον τρόπο που περιγράψαμε, δηλαδή με χιαστί συνδυασμό των τμημάτων των γονέων με βάση ένα σημείο κοπής, θα δημιουργήσουμε δύο απογόνους που δεν θα είναι έγκυρες κωδικοποιήσεις πιθανών λύσεων στο πρόβλημα του πλανόδιου πωλητή, αφού, τόσο θα επαναλαμβάνονται πόλεις δύο φορές, όσο και θα λείπουν πόλεις από τα χρωμοσώματα-απογόνους. Για τον λόγο αυτό, συνήθως στο πρόβλημα του πλανόδιου πωλητή ακολουθούμε μία παραλλαγή του κλασικού τελεστή διασταύρωσης με ένα σημείο κοπής, που λέγεται *διασταύρωση σειράς* (order crossover), φυσικά πάλι έχοντας δεδομένη μία πιθανότητα διασταύρωσης p_c και μέσω της επιλογής ενός τυχαίου αριθμού r στο διάστημα $[0, 1)$, να αποφασίζεται αν σε ένα ζευγάρι χρωμοσωμάτων θα γίνει διασταύρωση ή όχι. Με τον τελεστή διασταύρωσης σειράς, επιλέγονται τυχαία δύο σημεία κοπής των γονέων (τα ίδια σημεία και για τους δύο). Δηλαδή κάθε γονέας αποτελείται από τρία τμήματα, αυτό πριν το πρώτο σημείο κοπής, αυτό ανάμεσα στα δύο σημεία κοπής και αυτό μετά το δεύτερο σημείο κοπής. Έστω ότι ο ένα γονέας είναι ο P|Q|R και ο άλλος ο S|T|U, όπου τα | δείχνουν τις θέσεις των σημείων κοπής, δηλαδή τα τμήματα P και S έχουν ίσο πλήθος γονιδίων, έστω K, τα τμήματα Q και T έχουν επίσης ίσο πλήθος γονιδίων, έστω L, και ομοίως τα τμήματα R και U έχουν ίσο πλήθος γονιδίων, έστω M. Τότε οι απόγονοι που προκύπτουν από αυτούς τους δύο γονείς είναι οι A|Q|B και C|T|D, όπου τα τμήματα A και C έχουν K γονίδια και τα τμήματα B και D έχουν M γονίδια. Τα γονίδια των B και A συμπληρώνονται με αυτή τη σειρά από τα γονίδια των U, S και T, με αυτή τη σειρά, που δεν έχουν εμφανισθεί στο Q και τα γονίδια των D και C συμπληρώνονται με αυτή τη σειρά από τα γονίδια των R, P και Q, με αυτή τη σειρά, που δεν έχουν εμφανισθεί στο T. Δείτε το παράδειγμα:



Επίσης, για το πρόβλημα του πλανόδιου πωλητή, θα πρέπει να υιοθετήσουμε κάποια ιδιαίτερη εκδοχή του κλασικού τελεστή της μετάλλαξης, γιατί το να μεταβάλουμε απλώς την τιμή ενός ή περισσότερων γονιδίων σε ένα χρωμόσωμα είναι σχεδόν βέβαιο ότι θα οδηγήσει σε μη έγκυρη αναπαράσταση κάποιας λύσης του προβλήματος. Η παραλλαγή που συνήθως υιοθετούμε είναι η εξής. Δεδομένης μίας πιθανότητας μετάλλαξης p_m, αποφασίζουμε, μέσω της επιλογής ενός τυχαίου αριθμού r στο διάστημα

$[0, 1)$, αν σε ένα χρωμόσωμα θα εφαρμοσθεί ο τελεστής. Αν ναι, επιλέγονται τυχαία δύο γονίδια και ανταλλάσσουν θέσεις. Για παράδειγμα, το χρωμόσωμα

5	**3**	8	2	6	**4**	1	7

θα μπορούσε να υποστεί μετάλλαξη στις θέσεις 2 και 6 και να γίνει το

5	**4**	8	2	6	**3**	1	7

Με βάση τα προαναφερθέντα, υλοποιήστε σε C ένα γενετικό αλγόριθμο για το πρόβλημα του πλανόδιου πωλητή. Το πρόγραμμα που θα γράψετε (έστω ότι το εκτελέσιμο ονομάζεται "`tspga`") θα πρέπει να διαβάζει αρχικά από την πρότυπη είσοδο, στην πρώτη γραμμή, το πλήθος των πόλεων και στη συνέχεια, ως ακεραίους, τις αποστάσεις μεταξύ τους. Πιο συγκεκριμένα, στη δεύτερη γραμμή θα δίνονται οι αποστάσεις της 1ης πόλης από τη 2η, την 3η κλπ., στην επόμενη γραμμή, οι αποστάσεις της 2ης πόλης από όλες τις επόμενές της, κοκ., και, τέλος, η απόσταση της προτελευταίας πόλης από την τελευταία. Για τις παραμέτρους του γενετικού αλγορίθμου θα πρέπει να δώσετε default τιμές μέσα στο πρόγραμμα μέσω συμβολικών σταθερών, αλλά θα πρέπει να υπάρχει και η δυνατότητα να εκτελείται ο γενετικός αλγόριθμος και με διαφορετικές τιμές που θα δίνονται ως ορίσματα στη γραμμή εντολής, με τρόπο που θα επιλέξετε εσείς, αλλά θα τεκμηριώσετε με σαφήνεια σε ένα συνοδευτικό αρχείο README.

Ακολουθεί μία ενδεικτική εκτέλεση ενός γενετικού αλγορίθμου για το πρόβλημα. Και για το πρόγραμμα αυτό είναι προφανές ότι είναι αδύνατον να αναπαραγάγετε με ακρίβεια τα αποτελέσματα. Πάντως για την παρακάτω εκτέλεση, έχουμε πληθυσμό $P = 100$ ατόμων, μέγιστο πλήθος γενεών $G = 10000$ και πιθανότητες διασταύρωσης $p_c = 0.5$ και μετάλλαξης $p_m = 0.1$. Επίσης, στο πρόγραμμα αυτό εφαρμόστηκε ελιτισμός ενός ατόμου, δηλαδή το καλύτερο άτομο κάθε γενεάς περνά στην επόμενη, χωρίς να μπει στη διαδικασία της τυχαίας επιλογής, και μάλιστα φροντίζεται ώστε στην επόμενη γενεά να μην μεταλλαχθεί, ούτε να διασταυρωθεί με άλλο άτομο.

```
$ ./tspga
12
31 12 7 2 17 19 25 21 30 10 32
13 11 8 29 23 14 17 4 21 11
9 14 28 25 11 32 14 10 8
22 11 15 8 13 7 23 25
18 19 3 29 5 18 34
14 30 9 22 17 11
22 11 10 8 19
8 24 42 33
17 6 32
15 28
22
Generation     0: BFit = 0.007246   AvFit = 0.004788   BLength = 138
Generation     1: BFit = 0.007246   AvFit = 0.004744   BLength = 138
Generation     2: BFit = 0.007246   AvFit = 0.004839   BLength = 138
Generation     3: BFit = 0.007246   AvFit = 0.004844   BLength = 138
Generation     4: BFit = 0.007246   AvFit = 0.004753   BLength = 138
Generation     5: BFit = 0.009709   AvFit = 0.004939   BLength = 103
.................................
Generation  9999: BFit = 0.011111   AvFit = 0.008248   BLength = 90
Generation 10000: BFit = 0.011111   AvFit = 0.008238   BLength = 90
Best chromosome is: 5 8 3 12 6 9 11 7 10 2 4 1
```

```
Path length = 90
$
```

Μπορείτε να δοκιμάσετε το πρόγραμμά σας και με άλλα στιγμιότυπα του προβλήματος, που γεννώνται τυχαία από το πρόγραμμα `http://www.di.uoa.gr/~ip/hwfiles/ga/randtsp.c`. Το πρόγραμμα αυτό μπορεί να κληθεί είτε χωρίς ορίσματα, είτε από ένα έως τρία ορίσματα, και παράγει στην έξοδό του τυχαία δεδομένα για το πρόβλημα του πλανόδιου πωλητή στη μορφή που τα περιμένει το πρόγραμμα "`tspga`". Οι διαφορετικές εκδοχές χρήσης του προγράμματος περιγράφονται στη συνέχεια:

- `randtsp`: Δημιουργεί ένα τυχαίο στιγμιότυπο για 10 πόλεις, με αποστάσεις μεταξύ τους από 1 έως 100 και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `randtsp <n>`: Δημιουργεί ένα τυχαίο στιγμιότυπο για `<n>` πόλεις, με αποστάσεις μεταξύ τους από 1 έως 100 και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `randtsp <n> <max>`: Δημιουργεί ένα τυχαίο στιγμιότυπο για `<n>` πόλεις, με αποστάσεις μεταξύ τους από 1 έως `<max>` και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `randtsp <n> <max> <seed>`: Δημιουργεί ένα τυχαίο στιγμιότυπο για `<n>` πόλεις, με αποστάσεις μεταξύ τους από 1 έως `<max>` και με φύτρο της γεννήτριας τυχαίων αριθμών το `<seed>`.

Κάποιες επιπλέον εκτελέσεις του προγράμματος "`tspga`", με τυχαίες εισόδους που παράγονται από το πρόγραμμα "`randtsp`", φαίνονται στη συνέχεια.

```
$ ./randtsp 200 500 2016 | ./tspga
Generation     0: BFit = 0.000022   AvFit = 0.000020   BLength = 45385
Generation     1: BFit = 0.000022   AvFit = 0.000020   BLength = 45385
Generation     2: BFit = 0.000022   AvFit = 0.000020   BLength = 45129
Generation     3: BFit = 0.000022   AvFit = 0.000020   BLength = 44827
Generation     4: BFit = 0.000022   AvFit = 0.000020   BLength = 44646
Generation     5: BFit = 0.000023   AvFit = 0.000020   BLength = 43539
.................................
Generation  9999: BFit = 0.000100   AvFit = 0.000074   BLength = 9982
Generation 10000: BFit = 0.000100   AvFit = 0.000074   BLength = 9982
Best chromosome is: .................................
Path length = 9982
```

Το πρόγραμμά σας θα πρέπει να είναι σε θέση να ελέγχει, για δεδομένο στιγμιότυπο του προβλήματος, αν ένα συγκεκριμένο μονοπάτι είναι έγκυρη λύση, δηλαδή αν περιλαμβάνει όλες τις πόλεις ακριβώς μία φορά, και στην περίπτωση αυτή θα πρέπει να εκτυπώνει το μήκος του μονοπατιού. Το μονοπάτι προς έλεγχο θα πρέπει να δίνεται στο πρόγραμμα από την πρότυπη είσοδο αμέσως μετά τα δεδομένα των πόλεων. Για τη λειτουργία αυτή, το πρόγραμμά σας θα πρέπει να καλείται με την επιλογή "`-c`". Για παράδειγμα:

```
$ ./tspga -c
12
31 12 7 2 17 19 25 21 30 10 32
13 11 8 29 23 14 17 4 21 11
9 14 28 25 11 32 14 10 8
22 11 15 8 13 7 23 25
```

```
18 19 3 29 5 18 34
14 30 9 22 17 11
22 11 10 8 19
8 24 42 33
17 6 32
15 28
22
1 4 2 10 7 11 9 6 12 3 8 5
Path length: 90
$
```

Σε διάφορες ιστοσελίδες στο διαδίκτυο υπάρχουν διαθέσιμα δεδομένα για το πρόβλημα του πλανόδιου πωλητή. Μία από αυτές τις ιστοσελίδες είναι η

`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`

από την οποία μπορεί κανείς να κατεβάσει αρχεία δεδομένων για το πρόβλημα του πλανόδιου πωλητή, αλλά και για άλλα συναφή προβλήματα. Συγκεκριμένα, στον σύνδεσμο

`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/`

τα αρχεία με επέκταση "`.tsp.gz`" είναι συμπιεσμένα αρχεία στιγμιοτύπων προβλημάτων πλανόδιου πωλητή. Η μορφή του περιεχομένου των αρχείων αυτών περιγράφεται στο έγγραφο

`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf`

Το πρόγραμμά σας θα πρέπει να είναι σε θέση να διαβάζει αρχεία με επέκταση "`.tsp`" και να λύνει τα προβλήματα που περιγράφουν. Σημειώστε ότι στα αρχεία αυτά ακολουθούνται αρκετά διαφορετικοί τρόποι περιγραφής των δεδομένων, οι οποίοι όμως τεκμηριώνονται λεπτομερώς στο παραπάνω έγγραφο. Όταν το πρόγραμμά σας καλείται με την επιλογή "`-i <tspfile>`", δεν θα πρέπει να διαβάζει τα δεδομένα του από την πρότυπη είσοδο, αλλά από το αρχείο `<tspfile>`. Παράδειγμα:

```
$ ./tspga -i swiss42.tsp
Generation     0: BFit = 0.000243   AvFit = 0.000208   BLength = 4119
Generation     1: BFit = 0.000243   AvFit = 0.000206   BLength = 4119
Generation     2: BFit = 0.000249   AvFit = 0.000207   BLength = 4010
Generation     3: BFit = 0.000249   AvFit = 0.000208   BLength = 4010
Generation     4: BFit = 0.000249   AvFit = 0.000210   BLength = 4010
.................................
Generation  9999: BFit = 0.000786   AvFit = 0.000555   BLength = 1273
Generation 10000: BFit = 0.000786   AvFit = 0.000562   BLength = 1273
Best chromosome is: 7 2 1 33 35 34 21 36 37 32 18 8 38 16 17 15 20 14 6 27 19 13
12 26 11 9 10 42 24 41 25 22 40 23 39 31 30 29 28 3 4 5
Path length = 1273
$
```

Εννοείται ότι οι επιλογές "`-c`" και "`-i`" πρέπει να μπορούν να χρησιμοποιηθούν ταυτόχρονα, καθώς και με οποιεσδήποτε άλλες επιλογές υιοθετήσετε για να δίνετε παραμέτρους του γενετικού αλγορίθμου στη γραμμή εντολής. Φυσικά, η επιλογή "`-c`" δεν είναι συμβατή με επιλογές που καθορίζουν παραμέτρους του γενετικού αλγορίθμου, αφού δεν εκτελείται ο γενετικός αλγόριθμος, αλλά απλώς γίνεται έλεγχος εγκυρότητας μίας λύσης και εύρεση του μήκους του μονοπατιού της. Παραδείγματα:

```
$ ./tspga -c -i swiss42.tsp
7 2 1 33 35 34 21 36 37 32 18 8 38 16 17 15 20 14 6 27 19 13
12 26 11 9 10 42 24 41 25 22 40 23 39 31 30 29 28 3 4 5
Path length = 1273
```

```
$
$ ./tspga -i swiss42.tsp -p 100 -g 1000 -pc 0.85 -pm 0.6
Generation     0: BFit = 0.000237   AvFit = 0.000209   BLength = 4215
Generation     1: BFit = 0.000237   AvFit = 0.000209   BLength = 4215
Generation     2: BFit = 0.000237   AvFit = 0.000209   BLength = 4215
Generation     3: BFit = 0.000237   AvFit = 0.000209   BLength = 4215
Generation     4: BFit = 0.000249   AvFit = 0.000210   BLength = 4011
Generation     5: BFit = 0.000252   AvFit = 0.000211   BLength = 3964
Generation     6: BFit = 0.000255   AvFit = 0.000213   BLength = 3922
.................................
Generation   999: BFit = 0.000551   AvFit = 0.000304   BLength = 1815
Generation  1000: BFit = 0.000551   AvFit = 0.000301   BLength = 1815
Best chromosome is: 26 42 28 33 21 35 34 39 23 31 36 37 18 32 8 20 14 6 11 10 9
24 22 41 25 40 30 29 3 5 4 2 1 15 17 16 38 7 27 19 13 12
Path length = 1815
$
```

Παραδοτέο

Για να παραδώσετε τη δουλειά σας, θα πρέπει να κάνετε τα εξής:

- Κατ΄ αρχήν, θεωρείται δεδομένο ότι θα έχετε διαχωρίσει τις εφαρμογές σας σε αρκετά πηγαία αρχεία και αρχεία επικεφαλίδας.
- Τοποθετήστε όλα τα αρχεία, για το πρόγραμμα που μαντεύει δυαδικό αριθμό, το πρόγραμμα μεγιστοποίησης συνάρτησης, αλλά και για αυτό του πλανόδιου πωλητή, μέσα σε ένα κατάλογο που θα δημιουργήσετε, έστω με όνομα `ga` (διαφορετικοί υποκατάλογοι για το κάθε πρόβλημα), σε κάποιο σύστημα Unix. Επίσης, τοποθετήστε στους υποκαταλόγους κατάλληλα αρχεία με ονόματα `README`, στα οποία να δίνετε οδηγίες για τη μεταγλώττιση των προγραμμάτων, την περιγραφή των ορισμάτων γραμμής εντολής, καθώς και ό,τι άλλο κρίνετε σκόπιμο να επισημάνετε. Προαιρετικά, μπορείτε να παραδώσετε και ένα αρχείο `Makefile` που να αναλαμβάνει όλη τη διαδικασία της κατασκευής των τελικών εκτελέσιμων μέσω της εντολής "`make`" (δώστε "`man make`" για περισσότερες λεπτομέρειες).
- Όντας στον κατάλογο που περιέχει τον κατάλογο `ga`, δημιουργήστε ένα επιπεδοποιημένο `tar` αρχείο (έστω με όνομα `ga.tar`) που περιέχει τον κατάλογο `ga` και όλα του τα περιεχόμενα. Αυτό γίνεται με την εντολή "`tar cvf ga.tar ga`".[3]
- Συμπιέστε το αρχείο `ga.tar`, ώστε να δημιουργηθεί το αρχείο `ga.tar.gz`. Αυτό γίνεται με την εντολή "`gzip ga.tar`".[4]
- Το αρχείο `ga.tar.gz` είναι που θα πρέπει να υποβάλετε μέσω του eclass.[5]

[3] Αν θέλετε να ανακτήσετε την δενδρική δομή που έχει φυλαχθεί σε ένα επιπεδοποιημένο `tar` αρχείο `file.tar`, αυτό μπορεί να γίνει με την εντολή "`tar xvf file.tar`".

[4] Αν θέλετε να αποσυμπιέσετε ένα αρχείο `file.gz` που έχει συμπιεσθεί με την εντολή `gzip`, αυτό μπορεί να γίνει με την εντολή "`gzip -d file.gz`".

[5] Μην υποβάλετε ασυμπίεστα αρχεία ή αρχεία που είναι συμπιεσμένα σε άλλη μορφή εκτός από `tar.gz` (π.χ. `rar`, `7z`, `zip`, κλπ.), γιατί δεν θα γίνονται δεκτά για αξιολόγηση.

Ομαδική εργασία

Η εργασία αυτή μπορεί να παραδοθεί και από **ομάδες των δύο ατόμων**. Στην περίπτωση αυτή, θα παραδοθεί μόνο από το ένα μέλος της ομάδας, αλλά μέσα σε ένα αρχείο `README` του καταλόγου `ga` θα αναφέρονται σαφώς τα στοιχεία των δύο μελών. Ο στόχος της διαδικασίας αυτής είναι να ενισχυθεί η ιδέα της **ισότιμης** συνεργασίας σε μία ομάδα για την επίτευξη ενός στόχου.

Διαγωνισμός καλύτερων προγραμμάτων για το πρόβλημα του πλανόδιου πωλητή

Βελτιωμένες και κατάλληλα προσαρμοσμένες εκδοχές των προγραμμάτων που θα υποβληθούν για το πρόβλημα του πλανόδιου πωλητή θα μπορούν να συμμετάσχουν, εφ΄ όσον το δηλώσουν οι συγγραφείς τους, σε διαγωνισμό καλύτερων προγραμμάτων που θα διεξαχθεί την τελευταία ημέρα της προφορικής εξέτασης των εργασιών, μετά το τέλος της. Για τα τρία καλύτερα προγράμματα, θα υπάρξει επιβράβευση στη βαθμολογία τους κατά 100%, 70% και 40%, κατά σειρά. Αν υπάρξουν και άλλα προγράμματα που θα έχουν μικρές διαφορές στην ποιότητά τους ως προς το τρίτο κατά σειρά πρόγραμμα, θα έχουν και αυτά μία επιβράβευση 20%. Περισσότερες λεπτομέρειες για τον διαγωνισμό θα ανακοινωθούν εν καιρώ στο φόρουμ του μαθήματος. Λάβετε υπόψη σας ότι η εστίαση του διαγωνισμού θα είναι η εύρεση λύσεων στο πρόβλημα του πλανόδιου πωλητή, σε περιορισμένο χρόνο, που να είναι όσο το δυνατόν πιο κοντά στη βέλτιστη. Τα δεδομένα για τα προγράμματα που θα συμμετάσχουν στον διαγωνισμό θα διαβάζονται μόνο από την πρότυπη είσοδο (πλήθος πόλεων και τριγωνικός πίνακας αποστάσεων), όχι από αρχεία `.tsp`.