Αλέξανδρος Αριστόβουλος

1063199

Εργασία με χρήση αραιών μητρών (Sparsity Technology)

Δημιουργία πίνακα με τα βάρη πτήσεων

Χρησιμοποιούμε τη βιβλιοθήκη numpy για να γεμίσουμε ένα πίνακα με τυχαία νούμερα μεταξύ των (LOWEST_WEIGHT = 50 και HIGHEST_WEIGHT = 100). Έπειτα φροντίζουμε η διαγώνιος του πίνακα να είναι μηδέν καθώς το κόστος για να μετακινηθούμε από ένα αεροδρόμιο στο ίδιο είναι ανύπαρκτο.

```
######################### Initialisation and Printing Functions ##########################
def createWeightsArray(nodeNumber):
    #create a random 2d array for the weights
    arr = np.random.randint(LOWEST_WEIGHT, HIGHEST_WEIGHT, size=(nodeNumber,nodeNumber))
    #The weight to go from one node to the same node is 0 (zero)
    np.fill_diagonal(arr, 0)
    return arr
```

Αποτέλεσμα της συνάρτησης δημιουργίας βαρών για 10 αεροδρόμια

```
-------------The weights array-------------

[[ 0 95 51 66 75 96 82 82 71 91]
 [81  0 88 92 64 63 90 95 62 64]
 [73 53  0 95 91 53 94 67 69 51]
 [75 89 68  0 79 77 54 52 55 82]
 [73 82 78 94  0 82 61 50 73 93]
 [72 78 67 71 62  0 58 94 52 90]
 [71 51 52 51 86 81  0 56 68 92]
 [62 83 60 91 86 56 78  0 73 86]
 [69 58 85 50 93 53 85 67  0 75]
 [87 52 95 57 59 76 61 81 86  0]]
```

Συνάρτηση δημιουργίας κόμβων ταξιδιού και υπολογισμού κόστους

Για κάθε ταξίδι ο μέγιστος αριθμός πτήσεων που θα κάνουμε είναι ίσος με το βάρος που έχουμε ορίσει ότι θα κάνουμε σε κάθε ταξίδι διά το ελάχιστο βάρος ταξιδιού

στρογγυλοποιημένο προς τα πάνω. Καθώς όμως θέλουμε τον μέγιστο αριθμό αεροδρομίων προσθέτουμε ένα (1) στον αριθμό ταξιδιών.
(1 ταξίδι -> 2 αεροδρόμια
2 ταξίδια -> 3 αεροδρόμια κλπ.)
Χρησιμοποιώντας τη συνάρτηση random.choice της βιβλιοθήκης numpy παίρνουμε ένα τυχαίο δείγμα των αεροδρομίων (για τα οποία είμαστε σίγουροι ότι θα είναι αρκετά επειδή χρησιμοποιούμε τον μέγιστο αριθμό αεροδρομίων που βρήκαμε πριν). Έτσι χρησιμοποιώντας αυτό το δείγμα παίρνουμε ένα-ένα τα αεροδρόμια και υπολογίζουμε το κόστος ταξιδιού μέχρι να ξεπεραστεί το νούμερο που θέλουμε. Μόλις ξεπεραστεί κρατάμε τη λίστα με τα αεροδρόμια που χρειαστήκαμε καθώς και το κόστος ταξιδιού και έτσι έχουμε ένα «έγκυρο» ταξίδι.

```
def createTrip(nodeNumber,weightArr):
    #find the maximum length of a trip to get that many nodes
    lengthOfTrip = math.ceil(TRIP_WEIGHT / LOWEST_WEIGHT) + 1

    #temporary trip, (replace=False so that we don't choose the same node twice)
    a = np.random.choice(nodeNumber, lengthOfTrip, replace=False)

    #append the first 2 nodes since we need them to have at least 1 transition between nodes
    trip = [a[0], a[1]]

    #get the current trip weight
    curTripWeight = weightArr[a[0]][a[1]]

    #stop the trip after we exceed the weight
    for i in range(2, lengthOfTrip):
        trip.append(a[i])
        curTripWeight += weightArr[a[i-1]][a[i]]

        if curTripWeight > TRIP_WEIGHT:
            break

    return trip, curTripWeight
```

Δημιουργία αραιάς μήτρας A με τον τρόπο 1 του μαθήματος (Simple Sparce Matrix)

Αρχικά δημιουργούμε τις λίστες a1_jloc (tripStarts), a1_irow (nodes), a1_val (weightSums) στις οποίες αποθηκεύουμε το index των αρχών των ταξιδιών, όλα τα αεροδρόμια με τη σειρά που δημιουργούνται από τη συνάρτηση δημιουργίας ταξιδιών και το κόστος του κάθε ταξιδιού αντίστοιχα. Ο τρόπος με τον οποίο το πετυχαίνουμε είναι ο εξής:
Δημιουργούμε όσα ταξίδια μας ζητούνται και για κάθε ταξίδι επεκτείνουμε τη λίστα των αεροδρομίων με τα αεροδρόμια που μας έδωσε η συνάρτηση και προσθέτουμε το κόστος της διαδρομής σαν τελευταίο στοιχείο της αντίστοιχης λίστας. Όσον αφορά τη λίστα με τις αρχές των ταξιδιών θέτουμε το πρώτο στοιχείο ίσο με το 0 (καθώς το πρώτο ταξίδι ξεκινάει στο πρώτο αεροδρόμιο) και έπειτα το επόμενο στοιχείο της

δείχνει στο τέλος του τωρινού ταξιδιού. Έτσι αυτή η λίστα καταλήγει στο τέλος με ένα παραπάνω στοιχείο από ότι ταξίδια και το οποίο χρησιμοποιούμε μόνο σε ελέγχους.

```
def createSparceMatrixSimple(weightArr, tripsNumber, nodeNumber):
    #the index of the list nodes where the trip starts
    #(last element shows the end of the last trip)
    tripStarts = [0]
    #the total weight for the trip
    weightSums = []
    #all the nodes we pass in all the trips
    nodes = []

    for i in range(tripsNumber):
        trip, sum = createTrip(nodeNumber, weightArr)
        nodes.extend(trip)
        weightSums.append(sum)
        tripStarts.append(len(nodes))

    return tripStarts, weightSums, nodes
```

Παράδειγμα αραιάς μήτρας A με τον τρόπο 1 του μαθήματος (με 10 αεροδρόμια, 5 ταξίδια και κόστος ταξιδιού ίσο με 200)

```
Simple Sparce Matrix (A1)

Start of each trip (a1_jloc)
[0, 5, 10, 14, 18, 22]
Weight of each trip (a1_jval)
[287, 249, 207, 244, 214]
All the nodes (a1_irow)
[2, 9, 4, 1, 7, 1, 3, 6, 2, 9, 6, 8, 1, 0, 9, 8, 2, 0, 1, 3, 8, 7]

The trips more clearly are:
Trip 0 [2, 9, 4, 1, 7]
Trip 1 [1, 3, 6, 2, 9]
Trip 2 [6, 8, 1, 0]
Trip 3 [9, 8, 2, 0]
Trip 4 [1, 3, 8, 7]
```

Δημιουργία αραιάς μήτρας A με τον τρόπο 2 του μαθήματος (Improved Sparce Matrix)

Οι λίστες a2_jloc (tripStarts), a2_irow (nodes), a2_val (weightSums) δημιουργούνται ακριβώς όπως με τον τρόπο 1. Η μόνη διαφορά είναι ότι έχουμε και μία λίστα next στην οποία αποθηκεύουμε τη διεύθυνση του επόμενου ταξιδιού. Στην αρχικοποίηση αυτής της μήτρας δίνουμε στη κάθε θέση next το index της θέσης αυτής αυξημένο

κατά ένα (επειδή όλα τα ταξίδια είναι στη σειρά). Μόνη εξαίρεση αποτελεί αν στο index+1 ξεκινάει άλλο ταξίδι που τότε το next[index] παίρνει την τιμή -1 για να ξέρουμε ότι εκεί τελειώνει το ταξίδι.

```python
def createSparceMatrixImproved(weightArr, tripsNumber, nodeNumber):
    #we get the tripStarts,weightSums and nodes the same way
    tripStarts, weightSums, nodes = createSparceMatrixSimple(weightArr, tripsNumber, nodeNumber)

    #counter for which tripStart to use (starts at 1 instead of 0 to get the end of the first trip)
    counter = 1

    #to have a trip we need at least 2 nodes so by definition the first node points to the second
    nextNode = [1]

    #make it so if nextNode[i] == -1 then we have finished this trip
    for i in range(1, len(nodes)):
        if(tripStarts[counter] == i + 1):
            nextNode.append(-1)
            counter+=1
        else:
            nextNode.append(i + 1)

    return tripStarts, weightSums, nodes, nextNode
```

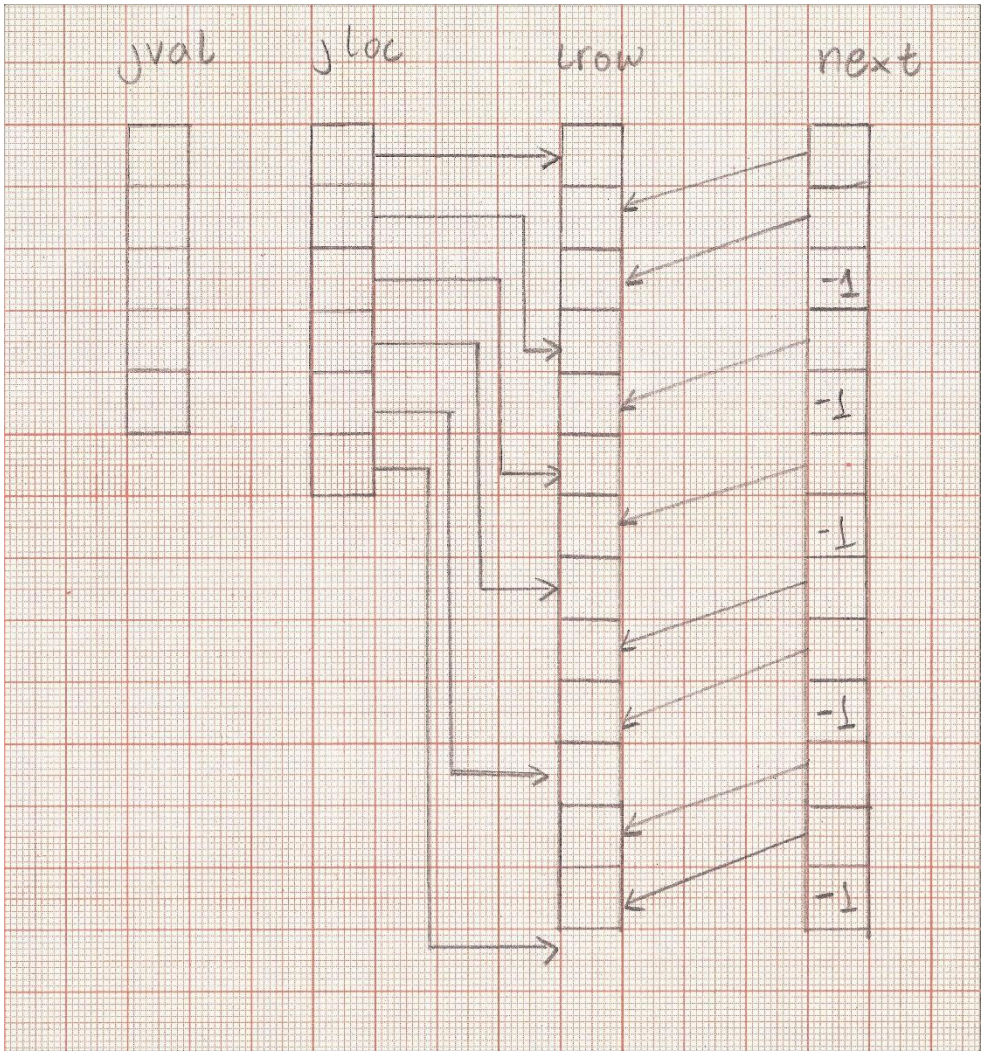
Παράδειγμα αραιάς μήτρας A με τον τρόπο 1 του μαθήματος (με 10 αεροδρόμια, 5 ταξίδια και κόστος ταξιδιού ίσο με 200)

```
Improved Sparce Matrix (A2)

Start of each trip (a2_jloc)
[0, 4, 9, 13, 17, 21]
Weight of each trip (a2_jval)
[219, 276, 222, 249, 228]
All the nodes (a2_irow)
[4, 3, 7, 8, 4, 7, 9, 1, 2, 7, 6, 9, 1, 1, 0, 6, 4, 0, 9, 4, 2]
Next node index (a2_next), (-1 signals the end of the trip)
[1, 2, 3, -1, 5, 6, 7, 8, -1, 10, 11, 12, -1, 14, 15, 16, -1, 18, 19, 20, -1]

The trips more clearly are:
Trip 0 [4, 3, 7, 8]
Trip 1 [4, 7, 9, 1, 2]
Trip 2 [7, 6, 9, 1]
Trip 3 [1, 0, 6, 4]
Trip 4 [0, 9, 4, 2]
```

Σχηματική αναπαράσταση των λιστών jval, jloc, irow που περιέχουν και οι δύο τρόποι καθώς και η λίστα next που περιέχει μόνο ο δεύτερος τρόπος



Συνάρτηση ελέγχου για το αν υπάρχει προορισμός σε ταξίδι

Για να ελέγξουμε αν υπάρχει κάποιο συγκεκριμένο αεροδρόμιο σε ένα ταξίδι του πίνακα A1 αρκεί να ελέγξουμε αν υπάρχει το αεροδρόμιο μεταξύ της αρχής του ταξιδιού αυτού και της αρχής του επόμενου. Για να δούμε αν υπάρχει το αεροδρόμιο σε ταξίδι του πίνακα A2 ξεκινάμε από την αρχή του ταξιδιού και μεταβαίνουμε στο επόμενο αεροδρόμιο με τη χρήση της λίστας nextNodes. Ο έλεγχος σταματάει όταν το αντίστοιχο στοιχείο της next ισούται με -1. Στη συγκεκριμένη συνάρτηση, και σε πολλές άλλες, ελέγχω αν ο χρήστης μας έδωσε στοιχεία του πίνακα A1 ή A2 με τη λίστα nextNodes. Αν ο χρήστης δώσει μια άδεια λίστα [] στη θέση της nextNodes τότε σημαίνει ότι έχω στοιχεία πίνακα A1 αλλιώς έχω στοιχεία πίνακα A2. Η συνάρτηση αυτή επιστρέφει True αν βρει ότι υπάρχει ήδη το αεροδρόμιο στο ταξίδι και False αν δεν υπάρχει.

```python
def checkIfDestinationExistsInTrip(tripNumber, destination, tripStarts, nodes, nextNodes):
    if nextNodes == []:
        for i in range(tripStarts[tripNumber], tripStarts[tripNumber+1]):
            if destination == nodes[i]:
                return True

    else:
        curNode = tripStarts[tripNumber]
        while curNode != -1:
            if destination == nodes[curNode]:
                return True
            curNode = nextNodes[curNode]

    return False
```

Συνάρτηση προσθήκης προορισμού στην αρχή ή στο τέλος του ταξιδιού

Για να προσθέσουμε προορισμό σε ένα ταξίδι πρώτα ελέγχουμε αν αυτός ο προορισμός υπάρχει ήδη στο ταξίδι. Αν δεν υπάρχει τότε ακολουθούμε διαφορετική διαδικασία σε κάθε πίνακα.

Για τον πίνακα A1 σπάμε τη λίστα irow σε 3 κομμάτια:

- partStart (Το κομμάτι της λίστας πριν το ταξίδι)
- Το κομμάτι της λίστας του ταξιδιού
- partEnd (Το κομμάτι της λίστας μετά το ταξίδι)

Αν θέλουμε να προσθέσουμε το ταξίδι στην αρχή τότε βάζουμε πρώτα το partStart έπειτα τον καινούργιο προορισμό, το κομμάτι του ταξιδιού και το partEnd. Αν πάλι θέλουμε να το προσθέσουμε στο τέλος τότε ακολουθούμε τη σειρά partStart, ταξίδι, καινούργιος προορισμός και partEnd. Και στις 2 περιπτώσεις αυξάνουμε τις τιμές jloc των επόμενων ταξιδιών κατά ένα για να δείχνουν στη σωστή αρχή και αυξάνουμε καταλλήλως το κόστος του ταξιδιού.

```python
def addDestinationInSimpleMatrix(direction, destination, tripNumber, tripStarts, weightSums, nodes, weightArr):

    #the nodes before and after the start of the trip remain the same
    partStart = nodes[0:tripStarts[tripNumber]]
    partEnd = nodes[tripStarts[tripNumber+1]:]

    if direction == "start":
        #first add the new destination then the rest of the trip
        partStart.append(destination)
        partStart.extend(nodes[tripStarts[tripNumber]:tripStarts[tripNumber+1]])
        #update the weight
        weightSums[tripNumber] += weightArr[destination][nodes[tripStarts[tripNumber]]]
    else:
        #first add the rest of the trip and then the destination
        partStart.extend(nodes[tripStarts[tripNumber]:tripStarts[tripNumber+1]])
        partStart.append(destination)
        #update the weight
        weightSums[tripNumber] += weightArr[nodes[tripStarts[tripNumber+1]-1]][destination]

    #join the lists to get the new nodes list
    nodes = partStart + partEnd

    #update all the following starts
    for i in range(tripNumber+1, len(tripStarts)):
        tripStarts[i] += 1

    return tripStarts, weightSums, nodes
```

Για τον πίνακα A2 προσθέτουμε τον καινούργιο περιορισμό κατευθείαν στο τέλος της λίστας irow. Αν θέλουμε να είναι στην αρχή του ταξιδιού τότε αλλάζουμε την αντίστοιχη τιμή του jloc για να δείχνει στο καινούργιο προορισμό και προσθέτουμε την τιμή της θέσης του προηγούμενου αρχικού σημείου στη λίστα next. Αν θέλουμε ο καινούργιος προορισμός να προστεθεί στο τέλος του ταξιδιού τότε βρίσκουμε το προηγούμενο τέλος του ταξιδιού (περνώντας από όλα τα στοιχεία του διαδοχικά με τη χρήση του πίνακα next). Μόλις το βρούμε, αλλάζουμε την τιμή next του στοιχείου αυτού από -1 στη τιμή της θέσης του καινούργιου προορισμού. Έπειτα προσθέτουμε στο τέλος της λίστας next τη τιμή -1 για να δείξουμε ότι το ταξίδι τελειώνει στο καινούργιο προορισμό. Δεν ξεχνάμε την απαραίτητη πρόσθεση για να ενημερώσουμε το βάρος του ταξιδιού.

```
def addDestinationInImprovedMatrix(direction, destination, tripNumber, tripStarts, weigh
    #add the destination to the end of the nodes list
    nodes.append(destination)

    if direction == "start":
        #update the nextNode with start of the trip
        nextNode.append(tripStarts[tripNumber])

        #update the start of the trip
        tripStarts[tripNumber] = len(nextNode) - 1

        #update the weight
        weightSums[tripNumber] += weightArr[destination][nodes[tripStarts[tripNumber]]]
```

```
    else:
        #we use the variable last to store the last destination of the trip
        last = start = tripStarts[tripNumber]
        next = nextNode[start]

        #when next == -1 it means that we reached the end of the trip
        while(next != -1):
            last = next
            next = nextNode[next]

        #update the last element of the trip to point to the new destination
        nextNode[last] = len(nodes) - 1
        #add the end tag in the nextNode list for this new destination
        nextNode.append(-1)

        #update the weight
        weightSums[tripNumber] += weightArr[nodes[last]][destination]

    #the last tripStart shows the index of the last element
    tripStarts[-1] +=1

    return tripStarts, weightSums, nodes, nextNode
```

Ο κώδικας που συνδυάζει και τις 2 συναρτήσεις προσθήκης προορισμού για τον κάθε πίνακα καθώς και τη συνάρτηση ελέγχου για το αν μπορεί να προστεθεί ο προορισμός είναι ο εξής:

```
def addDestination(direction, destination, tripNumber, tripStarts, weightSums, nodes, nextNode, weightArr):
    if checkIfDestinationExistsInTrip(tripNumber, destination, tripStarts, nodes, nextNode):
        print("The destination {} already exists in trip {}" .format(destination, tripNumber))

    else:
        if nextNode == []:
            tripStarts, weightSums, nodes = addDestinationInSimpleMatrix(
                direction, destination, tripNumber, tripStarts, weightSums, nodes, weightArr)
            print("Added destination {} at the {} of trip {} in Simple Matrix (A1)" .format(destination, direction, tripNumber))
        else:
            tripStarts, weightSums, nodes, nextNode = addDestinationInImprovedMatrix(
                direction, destination, tripNumber, tripStarts, weightSums, nodes, nextNode, weightArr)
            print("Added destination {} at the {} of trip {} in Improved Matrix (A2)" .format(destination, direction, tripNumber))
    return tripStarts, weightSums, nodes, nextNode
```

Το αποτέλεσμα προσθήκης στοιχείων είναι το εξής:

```
Adding Destinations

Added destination 0 at the start of trip 0 in Simple Matrix (A1)
Added destination 6 at the end of trip 0 in Simple Matrix (A1)
The destination 2 already exists in trip 0
Start of each trip (a1_jloc)
[0, 7, 12, 16, 20, 24]
Weight of each trip (a1_jval)
[416, 249, 207, 244, 214]
All the nodes (a1_irow)
[0, 2, 9, 4, 1, 7, 6, 1, 3, 6, 2, 9, 6, 8, 1, 0, 9, 8, 2, 0, 1, 3, 8, 7]

The trips more clearly are:
Trip 0 [0, 2, 9, 4, 1, 7, 6]
Trip 1 [1, 3, 6, 2, 9]
Trip 2 [6, 8, 1, 0]
Trip 3 [9, 8, 2, 0]
Trip 4 [1, 3, 8, 7]

Added destination 8 at the start of trip 1 in Improved Matrix (A2)
Added destination 3 at the end of trip 1 in Improved Matrix (A2)
The destination 4 already exists in trip 1
Start of each trip (a2_jloc)
[0, 21, 9, 13, 17, 23]
Weight of each trip (a2_jval)
[219, 371, 222, 249, 228]
All the nodes (a2_irow)
[4, 3, 7, 8, 4, 7, 9, 1, 2, 7, 6, 9, 1, 1, 0, 6, 4, 0, 9, 4, 2, 8, 3]
Next node index (a2_next), (-1 signals the end of the trip)
[1, 2, 3, -1, 5, 6, 7, 8, 22, 10, 11, 12, -1, 14, 15, 16, -1, 18, 19, 20, -1, 4, -1]

The trips more clearly are:
Trip 0 [4, 3, 7, 8]
Trip 1 [8, 4, 7, 9, 1, 2, 3]
Trip 2 [7, 6, 9, 1]
Trip 3 [1, 0, 6, 4]
Trip 4 [0, 9, 4, 2]
```

Συνάρτηση εύρεσης αριθμού κοινών στοιχείων μεταξύ 2 ταξιδιών

Για να μπορέσουμε να υπολογίσουμε τον πίνακα Β χρειαζόμαστε μία συνάρτηση η οποία θα υπολογίζει τον αριθμό των κοινών στοιχείων μεταξύ 2 ταξιδιών. Η συνάρτηση αυτή παίρνει ένα-ένα τα στοιχεία του ταξιδιού 1 και τα συγκρίνει με όλα τα στοιχεία του ταξιδιού 2. Συνεπώς η πολυπλοκότητα της είναι O(n^2) όπου n είναι ο αριθμός των σταθμών του κάθε ταξιδιού. Καθώς όμως με βάρος ταξιδιού 200 έχουμε στη χειρότερη περίπτωση 5 στάσεις τότε θα εκτελέσουμε το πολύ 25 επαναλήψεις. Θεωρητικά θα μπορούσαμε να βελτιστοποιήσουμε αλγοριθμικά τη μέθοδο δημιουργώντας ένα προσωρινό πίνακα με μήκος όσοι οι προορισμοί και στον οποίο αρχικοποιούμε όλες τις θέσεις με μηδέν. Έπειτα θα παίρναμε ένα-ένα τα στοιχεία του κάθε ταξιδιού και θα αυξάναμε κατά ένα τη τιμή της θέσης του πίνακα με index το προορισμό αυτό. Στο τέλος θα παίρναμε όλα τα στοιχεία του πίνακα και θα μετράγαμε τον αριθμό των θέσεων του με τιμή ίση με 2. Αυτή η μέθοδος «φαίνεται» πιο αποτελεσματική γιατί είναι πολυπλοκότητας O(n) αλλά είναι πολύ χειρότερη γιατί αυτό το n είναι ο αριθμός των προορισμών που στην άσκηση ισούται με 800. Άρα η μέθοδος που χρησιμοποιείται στο πρόγραμμα είναι τελικά καλύτερη γιατί 25<<800.

Ένα επιπλέον «κόλπο» για να μειώσουμε το χρόνο εκτέλεσης της συνάρτησης είναι να επιστρέφεται κατευθείαν σαν απάντηση ο αριθμός των προορισμών του ταξιδιού αν το συγκρίνουμε με τον εαυτό του.

```
def findNumberOfMatches(tripA, tripB, tripStarts, nodes, nextNode):
    count = 0

    #this is for the simple sparce Matrix
    if nextNode == []:
        #if we are comparing a trip with itself then the number of mat
        if tripA == tripB:
            return tripStarts[tripA+1] - tripStarts[tripA]
                #compare each node of trip A to all the nodes of trip
        for i in range(tripStarts[tripA], tripStarts[tripA+1]):
            for j in range (tripStarts[tripB], tripStarts[tripB+1]):
                if nodes[i] == nodes[j]:
                    count+=1
```

```
#this is for the improved sparce Matrix
else:
    #if we are comparing a trip with itself then the number of mat
    if tripA == tripB:
        next = tripStarts[tripA]
        while next != -1:
            next = nextNode[next]
            count+=1
        return count

    #set the current node index as the start of trip
    curNodeA = tripStarts[tripA]

    #move node index until we reach -1 meaning that the trip has e
    while curNodeA != -1:
        curNodeB = tripStarts[tripB]
        while curNodeB != -1:
            if nodes[curNodeA] == nodes[curNodeB]:
                count+=1
            #set current node index the next node index
            curNodeB = nextNode[curNodeB]

        #set current node index the next node index
        curNodeA = nextNode[curNodeA]

return count
```

Συνάρτηση υπολογισμού πίνακα B

Για τον υπολογισμό του πίνακα B δεν χρειάζεται να βρούμε όλα τα στοιχεία του καθώς είναι συμμετρικός. Για αυτό το λόγο βρίσκουμε μόνο το πάνω κομμάτι και την διαγώνιο και αποθηκεύουμε τα δεδομένα σε αραιά μήτρα. Για τον υπολογισμό των στοιχείων του πίνακα χρησιμοποιούμε τη βοηθητική συνάρτηση που προαναφέρθηκε. Η πολυπλοκότητα της συνάρτησης υπολογισμού του πίνακα είναι πολυπλοκότητας O(n^2) καθώς έχουμε τη μία επανάληψη μέσα στην άλλη. Έχει προστεθεί μεταβλητή που μετράει τον αριθμό των στοιχείων με τιμή μηδέν τα οποία και δεν αποθηκεύουμε.

```python
def calculateB(tripStarts, nodes, nextNode):
    #the index of this list is the row
    #the value of it's row shows where the columns of this row start
    bRowStarts = [0]
    #this is where we store the columns which have a non zero value
    bcol = []
    #this where we store the actual value
    bval = []
    #this is just for testing purposes to see how much space we save
    numOfZeroValues = 0

    for i in range (len(tripStarts)-1):
        #the full 2d array is symmetrical we only need to check for the upper ha
        for j in range(i, len(tripStarts)-1):
            result = findNumberOfMatches(i, j, tripStarts, nodes, nextNode)
            if result != 0:
                bcol.append(j)
                bval.append(result)
            else:
                numOfZeroValues+=1

        #the next row will point to the end of the last column from the previous
        bRowStarts.append(len(bcol))

    return bRowStarts, bcol, bval, numOfZeroValues
```

```
Calculating B (A1 Matrix)

b_iloc [0, 5, 9, 12, 14, 15]
b_jcol [0, 1, 2, 3, 4, 1, 2, 3, 4, 2, 3, 4, 3, 4, 4]
b_val [7, 4, 3, 3, 2, 5, 2, 2, 2, 4, 2, 2, 4, 1, 4]
Number of zero elements we didn't store 0

Calculating B (A2 Matrix)

b_iloc [0, 5, 9, 12, 14, 15]
b_jcol [0, 1, 2, 3, 4, 1, 2, 3, 4, 2, 3, 4, 3, 4, 4]
b_val [4, 4, 1, 1, 1, 7, 3, 2, 3, 4, 2, 1, 4, 2, 4]
Number of zero elements we didn't store 0
```

Βλέπουμε στο παραπάνω παράδειγμα ότι όλες οι θέσεις έχουν στοιχεία γιατί έχουμε βάλει πολύ λίγα αεροδρόμια και μεγάλο κόστος ταξιδιού. Αν αυξήσουμε τα αεροδρόμια σε 20, τα ταξίδια σε 8 και μειώσουμε το κόστος ταξιδιού σε 100 τότε έχουμε τα εξής:

```
------------The weights array-------------

[[ 0 95 51 66 75 96 82 82 71 91 81 57 88 92 64 63 90 95 62 64]
 [73  0 76 95 91 53 94 67 69 51 75 89 68 57 79 77 54 52 55 82]
 [73 82  0 94 81 82 61 50 73 93 72 78 67 71 62 52 58 94 52 90]
 [71 51 52  0 86 81 76 56 68 92 62 83 60 91 86 56 78 94 73 86]
 [69 58 85 50  0 53 85 67 55 75 87 52 95 57 59 76 61 81 86 65]
 [56 53 79 61 50  0 71 78 63 96 96 66 55 62 90 84 74 54 59 87]
 [89 84 69 98 76 86  0 65 59 62 84 80 56 79 97 88 82 84 66 97]
 [96 66 88 82 80 71 83  0 70 62 93 55 98 71 81 87 51 90 82 93]
 [60 71 57 73 60 69 58 61  0 61 87 89 73 96 62 90 59 93 75 65]
 [89 96 55 97 61 60 85 76 77  0 76 93 84 68 76 70 83 61 84 86]
 [53 87 88 51 63 92 99 84 68 56  0 81 95 55 91 52 63 79 77 97]
 [66 64 50 64 51 56 80 78 70 98 71  0 53 98 83 80 95 76 67 98]
 [60 72 63 51 96 71 77 69 73 93 87 69  0 71 70 73 52 94 99 89]
 [59 92 73 51 77 91 59 53 71 59 91 54 85  0 89 55 62 83 81 80]
 [94 83 59 81 85 83 72 55 67 51 77 52 65 84  0 65 87 56 59 97]
 [52 89 71 69 99 81 86 53 62 78 87 67 95 51 88  0 87 98 66 85]
 [85 76 82 50 56 86 85 70 89 74 74 90 81 89 56 72  0 76 86 84]
 [78 78 82 52 73 74 83 81 75 78 60 52 54 82 60 84 80  0 99 57]
 [80 84 65 58 84 96 74 92 54 58 80 53 59 70 96 92 55 68  0 52]
 [62 74 88 76 78 99 94 54 94 64 84 79 97 52 94 52 97 99 61  0]]
```

```
Simple Sparce Matrix (A1)

Start of each trip (a1_jloc)
[0, 3, 6, 9, 12, 15, 18, 21, 24]
Weight of each trip (a1_jval)
[121, 172, 169, 110, 168, 140, 161, 170]
All the nodes (a1_irow)
[18, 13, 3, 2, 17, 9, 5, 2, 19, 15, 13, 0, 0, 17, 4, 8, 16, 12, 12, 15, 14, 19, 5, 6]

The trips more clearly are:
Trip 0 [18, 13, 3]
Trip 1 [2, 17, 9]
Trip 2 [5, 2, 19]
Trip 3 [15, 13, 0]
Trip 4 [0, 17, 4]
Trip 5 [8, 16, 12]
Trip 6 [12, 15, 14]
Trip 7 [19, 5, 6]
```

```
Improved Sparce Matrix (A2)

Start of each trip (a2_jloc)
[0, 3, 6, 9, 12, 15, 18, 21, 24]
Weight of each trip (a2_jval)
[176, 153, 176, 123, 130, 104, 133, 148]
All the nodes (a2_irow)
[15, 16, 8, 7, 3, 0, 11, 7, 12, 12, 0, 15, 4, 9, 2, 5, 1, 9, 15, 0, 10, 19, 9, 12]
Next node index (a2_next), (-1 signals the end of the trip)
[1, 2, -1, 4, 5, -1, 7, 8, -1, 10, 11, -1, 13, 14, -1, 16, 17, -1, 19, 20, -1, 22, 23, -1]

The trips more clearly are:
Trip 0 [15, 16, 8]
Trip 1 [7, 3, 0]
Trip 2 [11, 7, 12]
Trip 3 [12, 0, 15]
Trip 4 [4, 9, 2]
Trip 5 [5, 1, 9]
Trip 6 [15, 0, 10]
Trip 7 [19, 9, 12]
```

```
Calculating B (A1 Matrix)

b_iloc [0, 2, 5, 7, 10, 11, 13, 14, 15]
b_jcol [0, 3, 1, 2, 4, 2, 7, 3, 4, 6, 4, 5, 6, 6, 7]
b_val [3, 1, 3, 1, 1, 3, 2, 3, 1, 1, 3, 3, 1, 3, 3]
Number of zero elements we didn't store 21

Calculating B (A2 Matrix)

b_iloc [0, 3, 7, 10, 13, 16, 18, 19, 20]
b_jcol [0, 3, 6, 1, 2, 3, 6, 2, 3, 7, 3, 6, 7, 4, 5, 7, 5, 7, 6, 7]
b_val [3, 1, 1, 3, 1, 1, 1, 3, 1, 1, 3, 2, 1, 3, 1, 1, 3, 1, 3, 3]
Number of zero elements we didn't store 16
```

Εύρεση κόμβου με τις περισσότερες/λιγότερες επισκέψεις

Για να βρούμε το κόμβο με τις περισσότερες ή λιγότερες επισκέψεις δημιουργούμε μία λίστα με μήκος το πλήθος των διαφορετικών αεροδρομίων και τιμή μηδέν για κάθε στοιχείο. Έπειτα παίρνουμε τους κόμβους με τη σειρά και αυξάνουμε κατά ένα την τιμή της λίστας στο σημείο με index το αεροδρόμιο. Όταν τελειώσουμε, βρίσκουμε με τη χρήση μίας for loop τα αεροδρόμια με τις περισσότερες ή λιγότερες επισκέψεις καθώς και τον αριθμό των επισκέψεων που έχουν.

```python
def findLeastMostVisitedNode(nodes, nodeNumber):
    #set all node visits to 0
    visits = [0] * nodeNumber

    #update the visit for each node
    for i in range(len(nodes)):
        visits[nodes[i]] +=1

    #(placeholder values) set the minimum and maximum as the position 0
    min = visits[0]
    minPositions = []
    max = visits[0]
    maxPositions = []

    #find the actual minimum value and the position(s)
    for i in range(nodeNumber):
        if visits[i] < min:
            min = visits[i]
            minPositions.clear()
            minPositions = [i]
        elif visits[i] == min:
            minPositions.append(i)

        if visits[i] > max:
            max = visits[i]
            maxPositions.clear()
            maxPositions = [i]
        elif visits[i] == max:
            maxPositions.append(i)


    return minPositions, min, maxPositions, max
```

Για τους ακόλουθους πίνακες έχουμε τα εξής αποτελέσματα

```
Simple Sparce Matrix (A1)

Start of each trip (a1_jloc)
[0, 5, 10, 14, 18, 22]
Weight of each trip (a1_jval)
[287, 249, 207, 244, 214]
All the nodes (a1_irow)
[2, 9, 4, 1, 7, 1, 3, 6, 2, 9, 6, 8, 1, 0, 9, 8, 2, 0, 1, 3, 8, 7]

The trips more clearly are:
Trip 0 [2, 9, 4, 1, 7]
Trip 1 [1, 3, 6, 2, 9]
Trip 2 [6, 8, 1, 0]
Trip 3 [9, 8, 2, 0]
Trip 4 [1, 3, 8, 7]
```

```
Improved Sparce Matrix (A2)

Start of each trip (a2_jloc)
[0, 4, 9, 13, 17, 21]
Weight of each trip (a2_jval)
[219, 276, 222, 249, 228]
All the nodes (a2_irow)
[4, 3, 7, 8, 4, 7, 9, 1, 2, 7, 6, 9, 1, 1, 0, 6, 4, 0, 9, 4, 2]
Next node index (a2_next), (-1 signals the end of the trip)
[1, 2, 3, -1, 5, 6, 7, 8, -1, 10, 11, 12, -1, 14, 15, 16, -1, 18, 19, 20, -1]

The trips more clearly are:
Trip 0 [4, 3, 7, 8]
Trip 1 [4, 7, 9, 1, 2]
Trip 2 [7, 6, 9, 1]
Trip 3 [1, 0, 6, 4]
Trip 4 [0, 9, 4, 2]
```

```
Finding most and least visited nodes (A1 Matrix)

The node with the least visits is [5] with 0 visit
The node with the most visits is [1] with 4 visits

Finding most and least visited nodes (A2 Matrix)

The node with the least visits is [5] with 0 visit
The node with the most visits is [4] with 4 visits
```

Χρόνος εκτέλεσης ερωτημάτων 2 και 3

Επειδή ο χρόνος εκτέλεσης για 10.000 και 20.000 είναι πολύ μεγάλος στο σύστημα μου, έτρεξα τον κώδικα για 1.000 και 2.000 ταξίδια (με 800 προορισμούς και βάρος ταξιδιού 200). Παρατηρούμε ότι για τον υπολογισμό του B διπλασιάζοντας τα ταξίδια ο χρόνος τετραπλασιάζεται καθώς έχει πολυπλοκότητα O(n^2). Αντίθετα για την

εύρεση του προορισμού με τις περισσότερες ή λιγότερες επισκέψεις ο χρόνος απλά διπλασιάζεται καθώς η συνάρτηση αυτή είναι πολυπλοκότητας O(n).

```
Calculating B (A1 Matrix 1000 elements)
Total time to calculate B (A1 Matrix 1000 elements): 11.723660945892334 seconds
Calculating B (A2 Matrix 1000 elements)
Total time to calculate B (A2 Matrix 1000 elements): 14.995635986328125 seconds

Finding most and least visited nodes (A1 Matrix 1000 elements)
Total time to find most and least visited nodes (A1 Matrix 1000 elements): 0.003995180130004883 seconds
The nodes with the least visits are [31, 105, 483] with 0 visit
The nodes with the most visits are [594, 717] with 13 visits

Finding most and least visited nodes (A2 Matrix 1000 elements)
Total time to find most and least visited nodes (A2 Matrix 1000 elements): 0.004003763198852539 seconds
The nodes with the least visits are [122, 354] with 0 visit
The nodes with the most visits are [264, 661, 673] with 13 visits
```

```
Calculating B (A1 Matrix 2000 elements)
Total time to calculate B (A1 Matrix 2000 elements): 46.987735748291016 seconds
Calculating B (A2 Matrix 2000 elements)
Total time to calculate B (A2 Matrix 2000 elements): 60.11766695976257 seconds

Finding most and least visited nodes (A1 Matrix 2000 elements)
Total time to find most and least visited nodes (A1 Matrix 2000 elements): 0.006998300552368164 seconds
The nodes with the least visits are [22, 222, 249, 355, 792] with 2 visits
The nodes with the most visits are [114, 276] with 22 visits

Finding most and least visited nodes (A2 Matrix 2000 elements)
Total time to find most and least visited nodes (A2 Matrix 2000 elements): 0.00700688362121582 seconds
The node with the least visits is [317] with 1 visit
The nodes with the most visits are [88, 767] with 21 visits
```