Μανιάτης Ανδρέας el18070 maniatis.andreas@gmail.com

2η σειρά γραπτών ασκήσεων

Άσκηση 1 : Πολύχρωμος Πεζόδρομος

Θα λύσουμε το πρόβλημα με δυναμικό προγραμματισμό.
Έστω το model array [c1, ..., cn] και έστω ότι ξεκινάμε με το array [0, ..., 0], έστω construction.

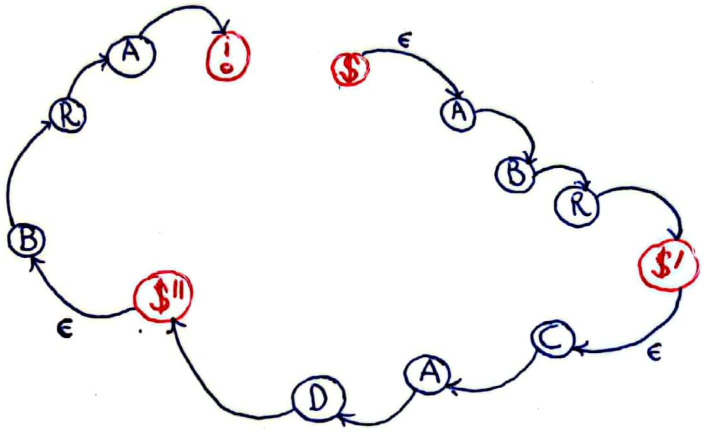
- Για μια ακολουθία n αριθμών η μέγιστη αλλαγή χρώματος είναι και για τους n αριθμούς.
  Αυτό όμως δεν χρειάζεται να δοκιμαστεί για όλα τα χρώματα, <u>αλλά μόνο για αυτά των δύο ακριανών θέσεων</u>.
- Όταν προχωράμε σε μια αλλαγή χρώματος (στο construction array) λοιπόν αυτή γίνεται από τα άκρα και αφού γίνει αυτή κάνουμε "slide" τον αριστερό και το δεξί δείκτη στο construction και model array. Οι θέσεις που αφήνουν πίσω τους οι δείκτες δεν αλλάζουν πάλι χρώμα.
- Οι άλλες επιλογές που έχουμε (πέρα από το να αλλάξουμε χρώμα και στις n θέσεις) είναι να δουλέψουμε ξεχωριστά στα n − 1 κοψίματα στα δύο της ακολουθίας.
- Επομένως κάθε ακολουθία n αριθμών συγκρίνει για το τελικό αποτέλεσμα n + 1 <u>το πολύ</u> αναδρομικές κλήσεις μειούμενου μεγέθους.

Αν λύσουμε αναδρομικά από το construction array $[0,0,\ldots,0]$ δημιουργούνται τελικά $n(n-1)/2$ τιμές για bottom-up υπολογισμό. Λόγω όμως των χρωματισμών έχουμε n τέτοιους πίνακες, δηλαδή $n^2(n-1)/2$ τιμές.

Στην επιστροφή των επιμέρους αναδρομικών κλήσεων έχουμε να ελέγξουμε το πολύ $n+1$ τιμές άρα τελικά η πολυπλοκότητα του αλγορίθμου είναι $O(n^4)$.

Άσκηση 2: String Matching

- Η τροποποίηση του αλγορίθμου θα βασιστεί στην εξής τροποποίηση του αρχικού αυτομάτου, όπως παρουσιάζεται (Jeff Erickson) με μόνο ένα αρχικό σύμβολο '$' και ένα τελικό '!', όπου κάθε φορά πριν συναντήσουμε μπαλαντέρ φτάνουμε σε μια ενδιάμεση κατάσταση :



Παράδειγμα εκφώνησης

- Όταν φτάσουμε σε μια ενδιάμεση κατάσταση δεν μπορούμε να φτάσουμε σε κατάσταση πριν από αυτήν με ακμές αποτυχίας αλλά μέχρι το πολύ αυτήν. Αυτό συμβαίνει γιατί έχουμε δει το πρώτο κομμάτι της συνολικής συμβολοσειράς και όποιο σύμβολο δούμε στη συνέχεια εξαιρουμένων των επόμενων συνεχόμενων συμβόλων που μας πάν στην επόμενη ενδιάμεση κατάσταση θεωρούνται μέρος της αυθαίρετης συμβολοσειράς – μπαλαντέρ που μπορεί να έχει οποιοδήποτε μήκος.
- Ο αλγόριθμος κατασκευής πίνακα fail που ορίζει τις αποτυχημένες ακμές backtracking είναι γραμμικός. Άρα για να φτιάξουμε όλους τους υποπίνακες (όσοι ο αριθμός των μπαλαντέρ συν ένα) είναι $O(m)$ όπου m ο αριθμός των συμβόλων του μοτίβου.

• Έχουμε τον αλγόριθμο:

```
fail[1...m] ← ComputeFailure(P)          → O(m)
j ← 1
for i ← 1 to n                            → O(n)
    while j > 0 and T[i] ≠ P[j]
        j ← fail[j]
    if j = m
        return i - m + 1
    j ← j + 1
return None
```

Πίνακας 1

• Το δεύτερο loop έχει πολυπλοκότητα $O(n)$ παρά το εσωτερικό while αφού για όλες συνολικά τις επαναλήψεις του i το j μπορεί να μειωθεί το πολύ $n-1$ φορές λόγω της συνθήκης $T[i] \neq P[j]$. Επομένως, η συνολική πολυπλοκότητα είναι $O(m+n)$.

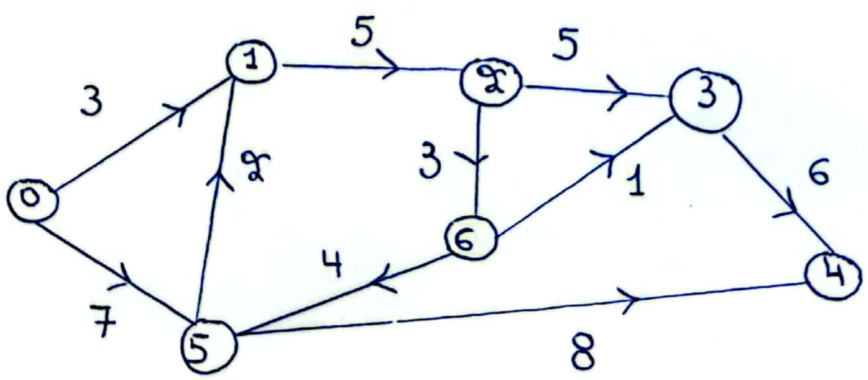
• Στον αλγόριθμο του Πίνακα 1 μπορούμε αντί για ένα μόνο array fail να χρησιμοποιήσουμε ένα dictionary με κλειδιά τον αριθμό των μπαλαντέρ που έχουμε δει και τιμές το εκάστοτε array $fail_i$.

Άσκηση 3: Συντομότερα Μονοπάτια με Συντομεύσεις Ενδιάμεσων Ακμών

1) Για τη λύση του προβλήματος θα χρειαστεί να επεκτείνουμε / τροποποιήσουμε τον αλγόριθμο Dijkstra, ώστε κάθε κορυφή να κρατάει ως πληροφορία το μήκος της μέγιστης ακμής του ελάχιστου μονοπατιού προς αυτήν. Έστω πίνακας max_edge.
Αυτό θα χρειαστεί για την ενημέρωση των αποστάσεων των adjacent κορυφών v η οποία δίνεται στον κλασικό Dijkstra από $D(u) + w(u,v)$
Στην σύγκριση της νέας με την παλιά απόσταση για λόγους ενημέρωσης, η νέα-υποψήφια είναι:

$$D'[v] = \begin{cases} D[u] + max\_edge[u], & \text{αν } w(u,v) > max\_edge[u] \\ D[u] + w(u,v), & \text{αν } w(u,v) \leq max\_edge[u] \end{cases}$$

Στην πρώτη περίπτωση ανανεώνουμε και το max_edge[v] με το w(u,v).
Έστω παράδειγμα:



Θα εκτελέσουμε τον αλγόριθμο για όσο $|S| < |V|$ και με αρχικοποιήσεις ομοίως με Dijkstra.
Οι παρενθέσεις αναπαριστούν την τιμή του max_edge.

S	1	2	3	4	5	6
$\{0\}$	$0^{(3)}$	∞	∞	∞	$0^{(7)}$	∞
$\{0,1\}$	$0^{(3)}$	$3^{(5)}$	∞	∞	$0^{(7)}$	∞
$\{0,1,5\}$	$0^{(3)}$	$3^{(5)}$	∞	$7^{(8)}$	$0^{(7)}$	∞
$\{0,1,2,5\}$	$0^{(3)}$	$3^{(5)}$	$8^{(5)}$	$7^{(8)}$	$0^{(7)}$	$6^{(5)}$
$\{0,1,2,5,6\}$	$0^{(3)}$	$3^{(5)}$	$7^{(5)}$	$7^{(8)}$	$0^{(7)}$	$6^{(5)}$
$\{0,1,2,3,5,6\}$	$0^{(3)}$	$3^{(5)}$	$7^{(5)}$	$7^{(8)}$	$0^{(7)}$	$6^{(5)}$

Σημαντική παρατήρηση: Όπως και στον κλασικό Dijkstra κρατάμε και τον πίνακα των προγόνων ώστε να ενημερώνουμε τα max_edge που αντιστοιχούν στο μονοπάτι που επεκτείνουμε.

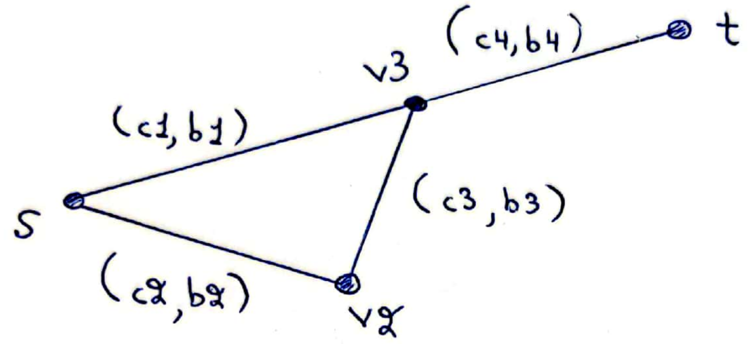
2) Η μόνη διαφορά για K είναι ότι αντί για τη μέγιστη ακμή του ελάχιστου μονοπατιού κρατάμε τις K μεγαλύτερες. Έστω για τον γράφο του προηγούμενου ερωτήματος και K=2

S	1	2	3	4	5	6
$\{0\}$	$0^{(3)}$	∞	∞	∞	$0^{(7)}$	∞
$\{0,1\}$	$0^{(3)}$	$0^{(5,3)}$	∞	∞	$0^{(7)}$	∞
$\{0,1,2\}$	$0^{(3)}$	$0^{(5,3)}$	$3^{(5,5)}$	∞	$0^{(7)}$	$3^{(5,3)}$
$\{0,1,2,5\}$	$0^{(3)}$	$0^{(5,3)}$	$0^{(5,5)}$	$0^{(8,7)}$	$0^{(7)}$	$3^{(5,3)}$
$\{0,1,2,3,5\}$	»	»	»	»	»	»
$\{0,1,2,3,4,5\}$	0	0	0	0	0	3

Η ενημέρωση των K μεγαλύτερων ακμών γίνεται σε $O(1)$ οπότε ο αλγόριθμός μας έχει τελικά ίδια πολυπλοκότητα με τον αλγόριθμο του Dijkstra.

'Άσκηση 4 : Σύντομα Μονοπάτια με Επαρκή Μεταφορική Ικανότητα

1) Έστω το απλούστερο δείγμα γράφου για την εύρεση αντιπαραδείγματος:



και
$$Opt(s-t) = \{ s \rightarrow v2 \rightarrow v3 \rightarrow t \}$$ αλλά
$$Opt(s-v3) = \{ s \rightarrow v3 \}.$$

Γράφοντας τις σχέσεις βρίσκουμε κατάλληλα (c_i, b_i)

$c1 = 4$ $b1 = 5$

$c2 = 1$ $b2 = 2$

$c3 = 2$ $b3 = 3$

$c4 = 2$ $b4 = 1$

τα οποία επαληθεύουν το αντιπαράδειγμα.

2) Έστω $D(v,i) = \dfrac{N_o(v,i)}{D_e(v,i)}$ ο βέλτιστος λόγος κόστους προς μεταφορική ικανότητα του μονοπατιού s − v χρησιμοποιώντας μέχρι i ακμές. Ισχύει:

$$D(u,i+1) = \min \left\{ D(u,i), \min_{v:(v,u)\in E} \left\{ \frac{N_o(v,i) + w(v,u)}{\min\{D_e(v,i), c(v,u)\}} \right\} \right.$$

Θα υπολογίσουμε τις τιμές $D(u, i)$ με δυναμικό προγραμματισμό. Ουσιαστικά ο αλγόριθμός μας ακολουθεί τον Bellman-Ford, οπότε και αποδεικνύεται επαγωγικά μέσω της προηγούμενης επαγωγικής σχέσης. Η πολυπλοκότητά του είναι $O(mn)$.

Τρέχουμε τον αλγόριθμο για το σχήμα του παραδείγματος:

i | D	D(s,i)	D(v1,i)	D(v2,i)	D(v3,i)	D(t,i)
0	0	∞	∞	∞	∞
1	0	10/4	9/3	∞	∞
2	0	10/4	9/3	15/3	18/2
3	0	10/4	9/3	15/3	24/3

Στο παραπάνω σχήμα δεν έχουμε κάνει απλοποιήσεις ώστε να φαίνονται οι ενημερώσεις των N_0 και D_e $(D(t,2) \rightarrow D(t,3))$.

Άσκηση 5: Αγορές Προϊόντων από Συγκεκριμένα Καταστήματα

Από τα δύο σύνολα μπορούμε να κατασκευάσουμε ένα διμερές γράφημα όπου οι ακμές (s_i, p_i) σημαίνουν ύπαρξη του προϊόντος p_i στο κατάστημα s_i. Έστω λοιπόν γράφημα G.

1) Το πρόβλημα ανάγεται στο Maximum Bipartite Matching αφού θέλουμε:

- κανένα προϊόν να μην συνδέεται με δύο καταστήματα (αγοράζουμε κάθε προϊόν μία φορά)
- κανένα κατάστημα να μην συνδέεται με δύο προϊόντα (ακριβώς ένα προϊόν από κάθε κατάστημα)
- μέγιστο αριθμό ακμών

Αν ο μέγιστος αριθμός ακμών δεν ισούται με τον αριθμό των προϊόντων ο αλγόριθμος διαπιστώνει ότι δεν υπάρχει λύση.

Ο αλγόριθμος είναι ο εξής:

- Προσθέτουμε στο διμερές γράφημα κόμβους s και t
- Προσθέτουμε ακμές από το s προς όλα τα προϊόνται
- και από όλα τα καταστήματα στο t
- Θέτουμε όλες τις χωρητικότητες ίσες με 1
- Λύνουμε το maximum flow πρόβλημα στο νέο γράφο
- Ελέγχουμε αν οι ακμές που χρησιμοποιούνται διέρχονται από όλα τα προϊόντα. Αν ναι αποτελούν την έξοδό μας.

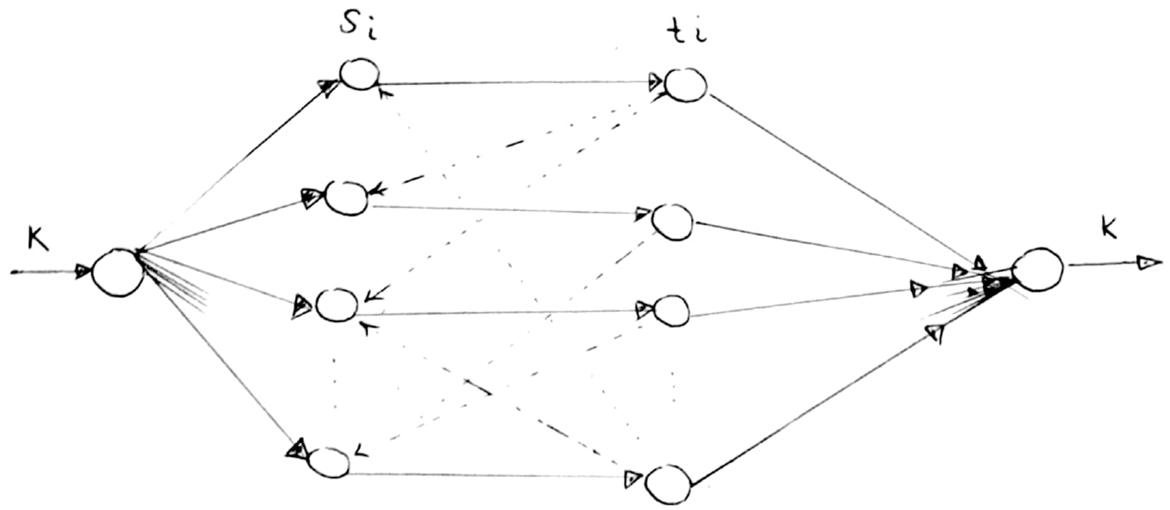
Η πολυπλοκότητα του Max Flow προβλήματος και άρα και του δικού μας είναι $O(nm^2)$ αν χρησιμοποιήσουμε τον Edmonds-Karp αλγόριθμο και $O(mn^2)$ αν χρησιμοποιήσουμε τον αλγόριθμο Dinic

δ) Η εύρεση του μέγιστου (ως προς τον πληθικό του αριθμό) συνόλου προϊόντων και καταστημάτων $S' \cup P'$, $S' \subseteq S$, $P' \subseteq P$, τέτοιο ώστε κανένα από τα προϊόντα του P' να μην πωλείται από τα καταστήματα του S' ανάγεται στην εύρεση του Maximum Independent Set (MIS) στο γράφημα G.
Το πρόβλημα αυτό μπορεί να αναχθεί στην εύρεση του Minimum Vertex Cover (MVC) αφού το ένα είναι συμπλήρωμα του άλλου. Και τα δύο όμως προβλήματα είναι ακόμα NP-hard.

Σύμφωνα με το Θεώρημα του Königο αριθμός των κόμβων στο MVC είναι ίσος με τον αριθμό των ακμών στο Maximum Matching που συζητήσαμε στο προηγούμενο ερώτημα.
Η εύρεση ακριβώς των κόμβων αυτών βασίζεται στην εύρεση της ελάχιστης τομής του δικτύου-ροής G'_∞ το οποίο σχηματίζεται από το γράφημα G με την διαφορά ότι τώρα οι χωρητικότητες των ακμών μεταξύ των δύο πλευρών του διμερούς γραφήματος θέτονται στο άπειρο. Από αυτήν προκύπτει το $|MVC|$ το οποίο είναι τελικά το μέγιστο που αναζητάμε.
Η ελάχιστη τομή υπολογίζεται και αυτή με τον αλγόριθμο Ford-Fulkerson σε πολυωνυμικό χρόνο.

'Ασκηση 6 : Ενοικίαση Αυτοκινήτων

Θα λύσουμε το πρόβλημα μοντελοποιώντας το ως ένα Minimum-cost flow problem:



Το required flow d είναι ίσο με K γιατί δεν έχει νόημα να υπάρχουν αχρησιμοποίητα αυτοκίνητα (υποθέτουμε $n \geq K$)

Προσθέτουμε πίσω ακμή $e = (t_i, s_j)$ αν $t_i \leq s_j$.

Αν δηλαδή ένα αυτοκίνητο αποδεσμευτεί τη χρονική στιγμή t_i μπορεί να εξυπηρετήσει όλα τα time-slots που αρχίζουν μετά από αυτήν ή σε αυτήν (θέλουμε να υπάρχει μεγάλη κάλυψη.

Επειδή όμως θέλουμε να λύσουμε πρόβλημα ελαχιστοποίησης αλλά έχουμε θετικές ανταμοιβές απλά θέτουμε ως κόστος των ακμών (s_i, t_i) το $-p_i$. Όλες οι υπόλοιπες ακμές έχουν μηδενικό κόστος.

Οι χωρητικότητες όλων των ακμών είναι 1 αφού κάθε time-slot (κράτηση) μπορεί να καλυφθεί το πολύ 1 φορά.

Για την λύση του προβλήματος μπορούμε να χρησιμοποιή- σουμε μόνο αλγόριθμο Bellman-Ford (λόγω αρνητικών βαρών στα συντομότερα μονοπάτια) και η πολυπλοκότητα του αλγορίθμου θα είναι $O(mnd)$ με γενίκευση του αλγορίθμου Ford-Fulkerson. Οι ακμές m είναι της τάξης n^2 λόγω των πίσω-ακμών άρα έχουμε τελικά

$O(Kn^2)$.