Γενίκευση μεθόδου Yates-Hennell

Σε αυτό το κείμενο παρουσιάζω το πρόβλημα εύρεσης κατάλληλων μονοπατιών για branch testing με βάση την μέθοδο των Yates-Hennell καθώς και το πώς αυτό μπορεί να επεκταθεί βρίσκοντας και συνδυάζοντας όλα τα δυνατά forward και backward trees του DD-graph που υλοποιεί η μέθοδος αυτή. Χρησιμοποιώ απτά και γραφικά παραδείγματα (το γνωστό παράδειγμα με το triangle program – όλα τα γραφήματα γίνανε με το πρόγραμμα dot) προκειμένου να έχω και εγώ μια βάση για το τι θα πρέπει να υλοποιήσω στην συνέχεια.

Το πρόβλημα ξεκινάει ως εξής: ένας *στατικός αναλυτής* μας έχει δώσει το block diagram του προγράμματος στο οποίο θέλουμε να κάνουμε branch testing. Συνήθως αυτό έχει μορφή:

```
BLOCK   1 CONNECTS TO BLOCKS   2              // για ευκολία εμείς θα παίρνουμε
BLOCK   2 CONNECTS TO BLOCKS   3   19         // την είσοδο ως: (19 nodes)
BLOCK   3 CONNECTS TO BLOCKS   4   5                1 2
BLOCK   4 CONNECTS TO BLOCKS   18                   2 3 19
BLOCK   5 CONNECTS TO BLOCKS   6   7                3 4 5 κοκ.
…
```

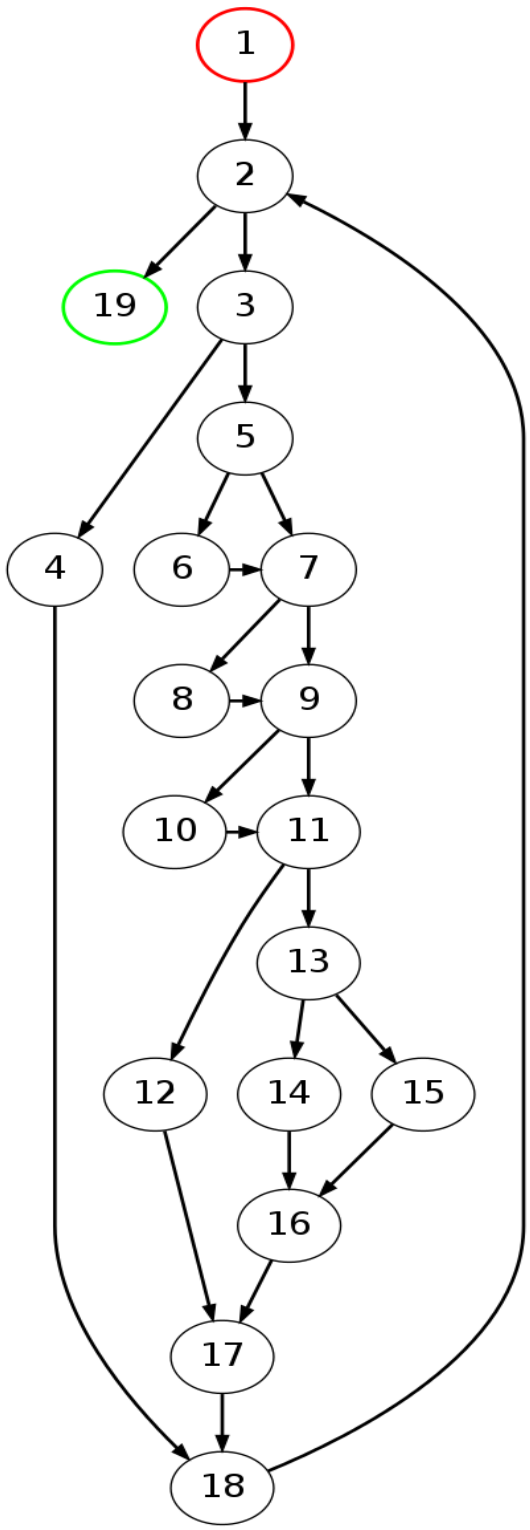
Δηλαδή ο στατικός αναλυτής μας δίνει τα blocks καθώς και το πώς συνδέονται αυτά μεταξύ τους στο πρόγραμμα (τις ακμές ουσιαστικά στο block diagram γράφο – δες Σχήμα 1).

Ο αλγόριθμος των Yates-Hennell δημιουργεί από τον block diagram γράφο, τον Decision to Decision γράφο (DD-graph) αφαιρώντας ουσιαστικά τους κόμβους που έχουν out degree 1.

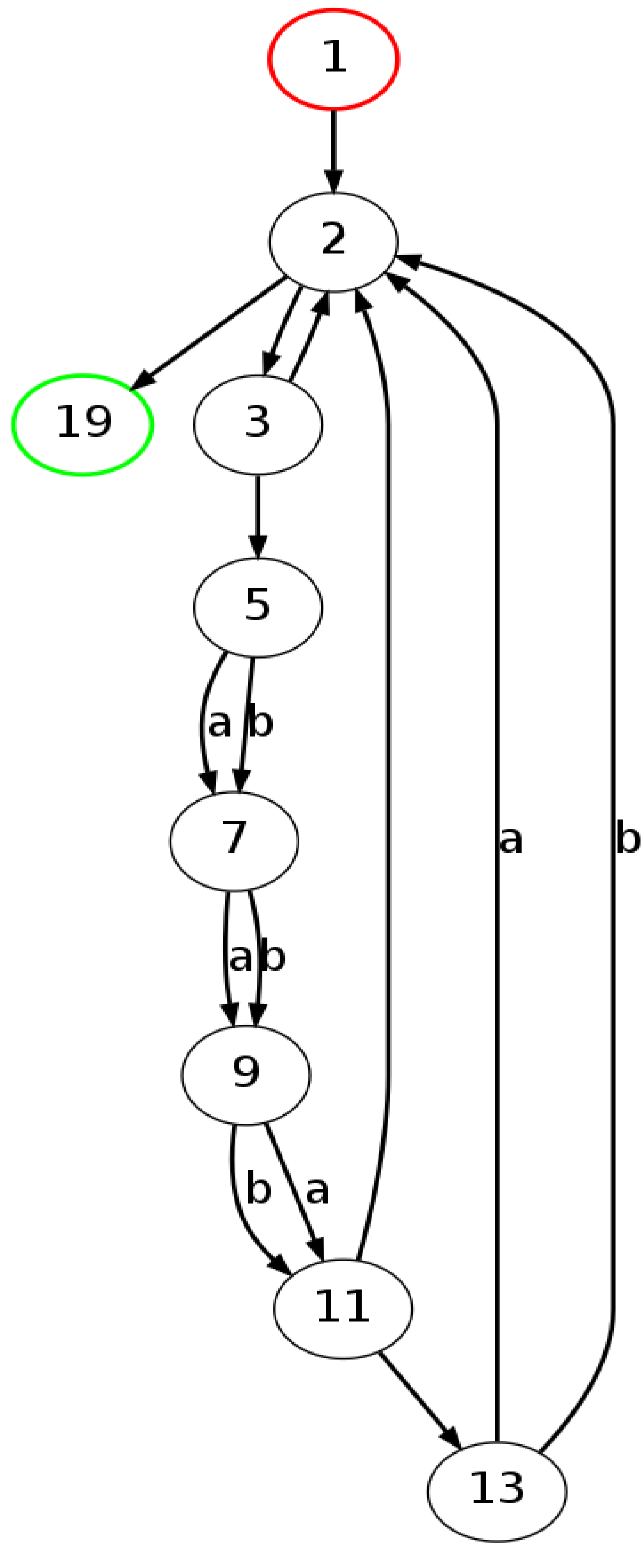
Ο DD-graph συνήθως είναι *multigraph* και όχι ένας απλός γράφος (δες Σχήμα 2). Σε αυτό το στάδιο θα πρέπει να υπάρχει μια δομή που να θυμάται ότι π.χ. (3,2) edge του DD-graph είναι το μονοπάτι (3,4,18,2) στον block-diagram graph, κτλ. για όλες τις ακμές του DD-graph που είναι ουσιαστικά «ευθεία» μονοπάτια στον block-diagram graph.

Στην συνέχεια ο αλγόριθμος υπολογίζει το λεγόμενο *forward* και *backward tree* του DD-graph που είναι ουσιαστικά το δέντρο συντομότερων μονοπατιών από τον αρχικό και τελικό κόμβο του DD-graph αντίστοιχα (είναι οι κόμβοι 1-red και 19-green στα παρακάτω σχήματα). Υποθέσεις/απλοποιήσεις που πρέπει να κάνουμε εδώ είναι ότι πάντα θα υπάρχει μονοπάτι από τον αρχικό κόμβο του DD-graph προς οποιονδήποτε άλλο κόμβο (και από οποιονδήποτε προς τον τελικό κόμβο) για να βγαίνουν τα 2 ειδών trees.

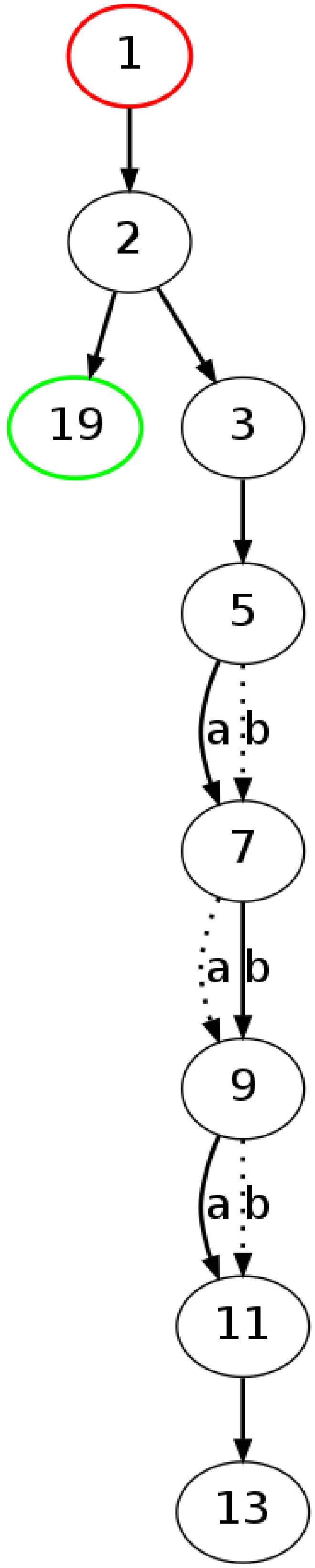
Στα Σχήματα 3,4 όπου και αναπαρίστανται τα forward και backward trees καταλαβαίνουμε επίσης ότι μπορεί να υπάρχουν παραπάνω από 1 forward και backward tree στον γράφο DD-graph. Απλά αντί π.χ. της ακμής (5,7)-a μπορούμε να πάρουμε την (5,7)-b (αρά και διαφορετικό μονοπάτι στον αρχικό block-diagram graph). Εδώ έγκειται και η επέκταση που μπορούμε να κάνουμε στον αλγόριθμο των Yates-Hennell: αντί να πάρουμε ένα τυχαίο forward και ένα τυχαίο backward tree (το οποίο μπορεί να μας οδηγήσει σε πολλά μη-προσπελάσιμα μονοπάτια στον block-diagram graph) απλά παίρνουμε όλους τους δυνατούς συνδυασμούς των δέντρων αυτών! <u>Το πόσες multiple edges</u> έχουμε παίζει μεγάλο ρόλο σε αυτό, π.χ. βλέποντας τα σχήματα 3,4 συμπεραίνουμε ότι ο αριθμός *όλων* των δυνατών forward trees είναι 2*2*2=8, ενώ



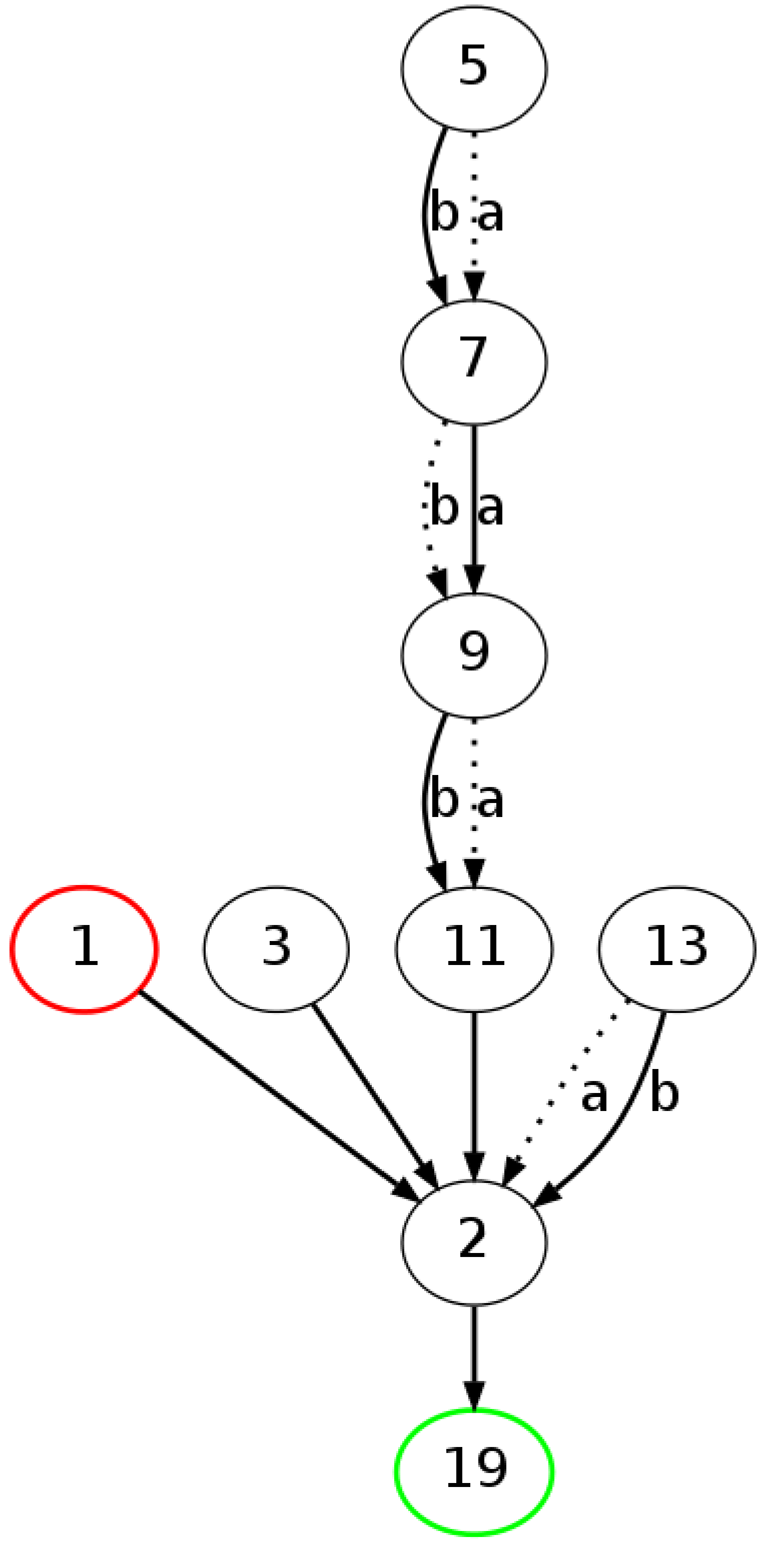
Σχήμα 1: **Block-diagram graph** για το triangle program



Σχήμα 2: **DD-graph**



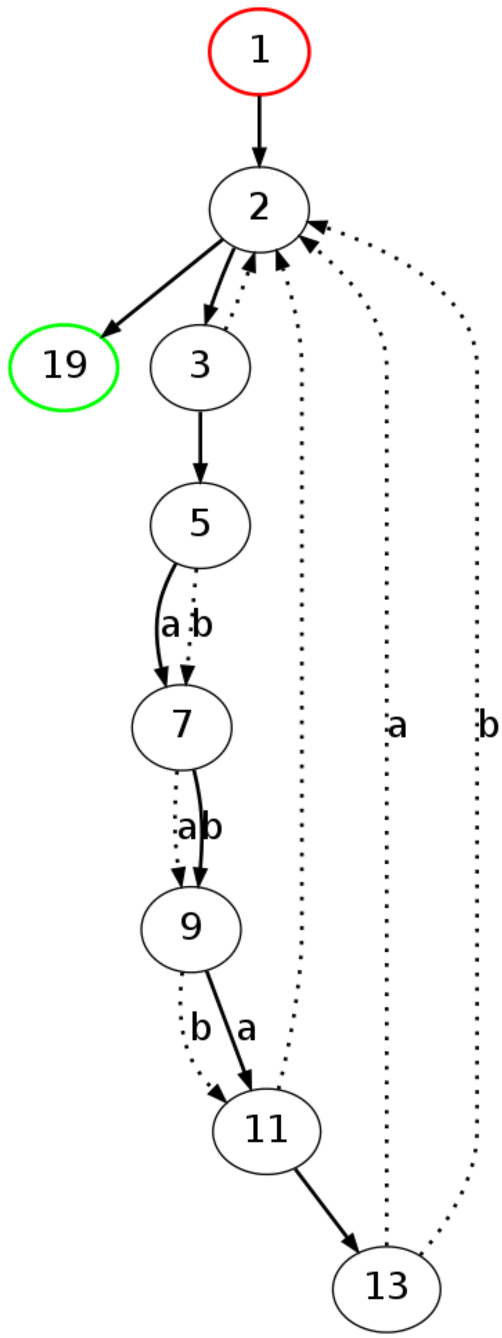
Σχήμα 3: **forward tree**



Σχήμα 4: **backward tree**

των backward trees είναι: 2*2*2*2=16. Άρα αντί να πάρουμε τυχαία 1+1 forward και backward tree όπως λέει ο αρχικός αλγόριθμος των Yates-Hennell, μπορούμε να πάρουμε 8*16=128 διαφορετικούς τέτοιους συνδυασμούς!

Στην συνέχεια ο αλγόριθμος βρίσκει τα μονοπάτια για branch testing επιλέγοντας (σύμφωνα με τον γνωστό τύπο στο paper τους) τις ακμές που ανήκουν στον DD-graph αλλά όχι στο (εκάστοτε για εμάς) forward tree (δες παρακάτω Σχήμα 5).



Σχήμα 5: Το forward tree όπου οι διακεκομμένες ακμές αντιπροσωπεύουν τις ακμές που δεν ανήκουν στο forward tree (αλλά είναι ακμές του DD-graph!).

Στο Σχήμα 5 φαίνεται ξεκάθαρα ότι αυτές οι ακμές (διακεκομμένες) είναι συνολικά 7 και άρα τα μονοπάτια που θα βγάλει ο αλγόριθμος για το πρόγραμμα αυτό είναι 8 (υπολογίζοντας και το συντομότερο μονοπάτι από τον αρχικό (1) μέχρι τον τελικό κόμβο(19)). Αυτό θα ισχύει για κάθε συνδυασμό forward με backward tree, καθώς συμφωνεί και με τον τύπο: numberOfPaths=e-n+2, όπου e,n είναι ο αριθμός των ακμών και των κόμβων αντίστοιχα του DD-graph.

Συνοψίζοντας, πρώτα θα προσπαθήσω να υλοποιήσω το αρχικό αλγόριθμο με ένα forward και ένα backward tree. Το τελικό πρόγραμμα, θέλω/ευελπιστώ να βγάζει ως **output** (σε αρχείο .txt) τα εξής:

Brethikan X forward kai Y backward trees.
Synolikoi sindiasmoi Z monopation: X*Y.

1) Ta Z monopatia:
   1) 1,3,19
   2) 1,4,5,6,19

   Κτλ.
2) Ta Z monopatia:
   a. 1,5,6,19
   b. 1,2,4,17,19
3) Κτλ. Μέχρι και τον τελευταίο συνδυασμό forward και backward tree.