Συντομότερα Μονοπάτια

- Απόσταση ή κόστος ή βάρος μονοπατιού σε βεβαρημένα γραφήματα:
  - Άθροισμα των βαρών των εμπλεκομένων ακμών
- Ανάγκη υπολογισμού των συντομότερων *απλών* μονοπατιών
- Τριγωνική ανισότητα:

$$w(x,y) \leq w(x,z) + w(z,y),\ (x,y),(x,z),(z,y) \in E$$

- Τεχνική ***χαλαρώσεως ακμών*****:**
  - Δίδονται προσεγγίσεις για τις αποστάσεις, οι οποίες, προϊόντος του χρόνου, βελτιώνονται, δοκιμάζοντας μήπως η χρήση κάποιας ακμής «χαλαρώνει» την αρχική εκτίμηση για την κορυφή-απόληξη
- Τεχνική ***χαλαρώσεως μονοπατιού*****:**
  - η χρήση κάποιας ακμής «χαλαρώνει» την αρχική εκτίμηση για ένα μονοπάτι

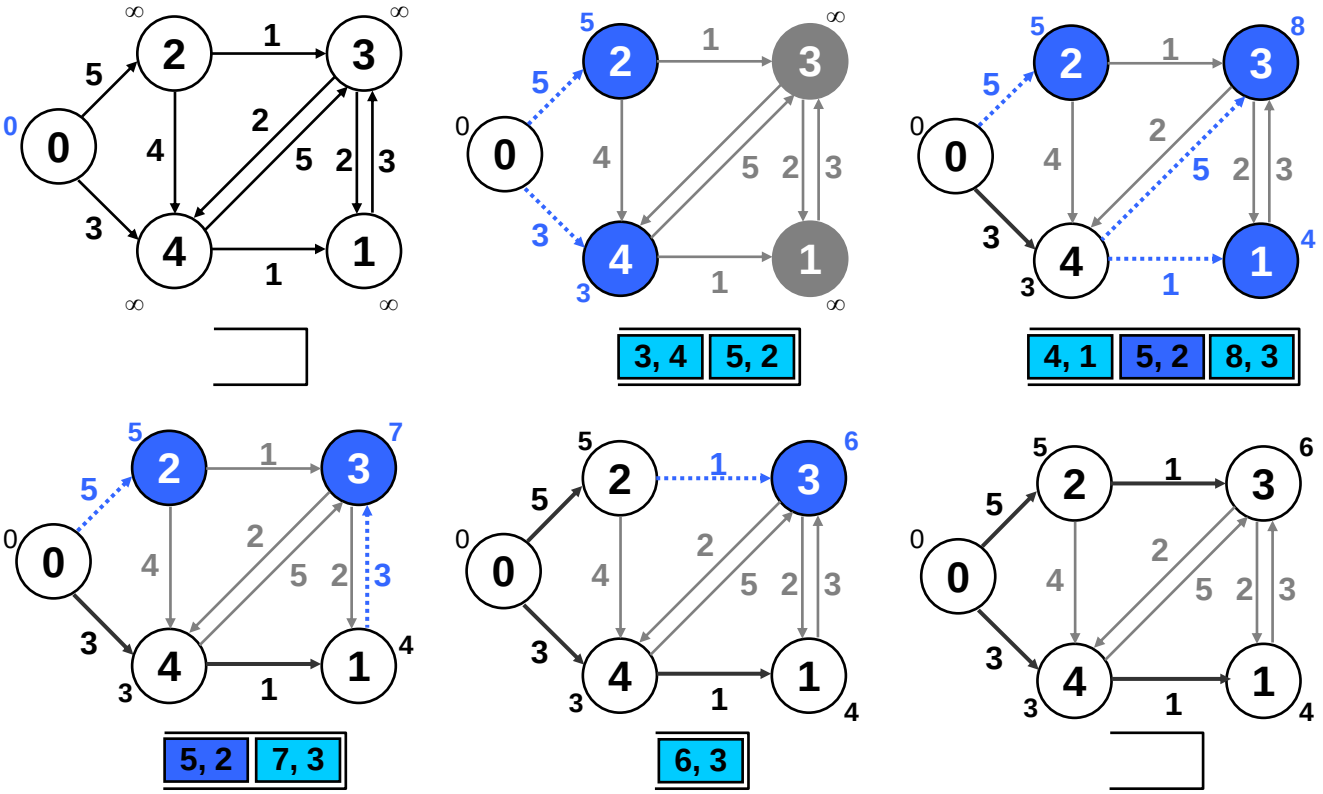
Συντομότερα Μονοπάτια Μοναδικής Πηγής (ΣΜΜΠ)

- **Το Ζητούμενο**
  - Εύρεση όλων των συντομότερων μονοπατιών από μία κορυφή-πηγή (source) *ν* προς τις υπόλοιπες
  - Τα ανωτέρω μονοπάτια σχηματίζουν το ***δένδρο ΣΜΜΠ*** με ρίζα την κορυφή *ν*

Αλγόριθμος του Dijkstra

- Κλασσικό παράδειγμα αλγορίθμου που βασίζεται στην *αρχή της απληστίας*
- Μόνο για βεβαρημένα γραφήματα ***θετικού βάρους (κόστους)***
- Διατηρεί μια διαμέριση του συνόλου *V* των κορυφών:
  - σε αυτές που ξέρουμε ακριβώς την απόσταση (*S*), και
  - σε αυτές που η γνώση μας είναι ελλιπής και προσεγγιστική (*V*-*S*)
- Επαναληπτικά,
  - αφαιρεί την «κοντινότερη» κορυφή *v* από το δεύτερο σύνολο,
  - εντάσσει την *v* στο πρώτο σύνολο,
  - διενεργεί χαλαρώσεις επί των εξερχόμενων ακμών της *v* με απολήξεις κορυφές του *V*-*S, ώστε το σύνολο παρυφής να είναι ενημερωμένο*

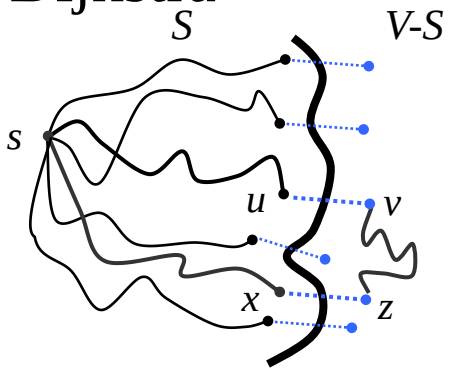
Παράδειγμα Dijkstra



Σημείωση. Για λόγους «οικονομίας» χώρου, στην ουρά δεν εικονίζονται οι αρχικές τιμές-προσέγγιση ∞ προς όλες τις κορυφές

Απόδειξη Ορθότητας

- Βασίζεται στην απουσία αρνητικών ακμών:
  - Όταν μία κορυφή *v* επιλέγεται ως η επόμενη που θα προστεθεί, τότε *πράγματι η απόσταση της από την πηγή έχει συγκλίνει στην τελική τιμή*
  - Έστω $s \in u \rightarrow v$ το μονοπάτι που επιστρέφει ο αλγόριθμος. Ας υποθέσουμε ότι υπήρχε άλλο συντομότερο μονοπάτι για την *v*: $s \in x \rightarrow z \in v$, το οποίο περιλαμβάνει και κορυφές που εξετάζει ο Dijkstra αργότερα.
  - Άτοπο, γιατί:
    - $\text{dis}(s \in u \rightarrow v) \leq \text{dis}(s \in x \rightarrow z)$ (λόγω τρεξίματος)
    - $\text{dis}(z \in v) > 0$ (θετικά βάρη)



Ψευδοκώδικας

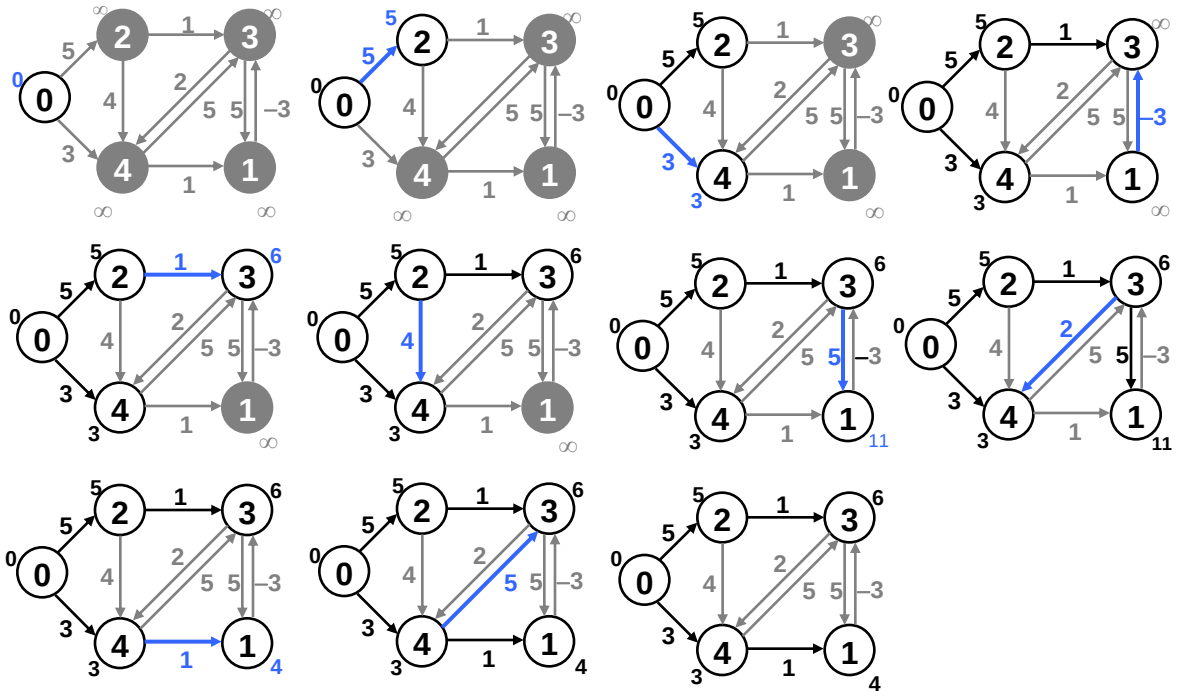
```
Algorithm DijkstraSSSP(graph g, double dis[], vertex s)
1.  pqueue pq = new pqueue();          // η ουρά προτεραιότητας
2.  for (v = 0; v < g.V; v++){         // αρχικοποίηση
3.    dis[v] = INFTY;
4.    pq.insKey(v,dis[v]);
5.  }
6.  dis[s] = 0.0;
7.  pq.decKey(s,dis[s]);
8.  while (!pq.isEmpty()) {            // όσο υπάρχουν κορυφές παρυφής
9.    v = pq.delmin();                 // εκτελείται #V φορές
10.   for (x = g.List[v]; x != null; x = x.getNext())  // έλεγχος των γειτονικών κορυφών της v
11.     if ((dis[v] + x.weight) < dis[w = x.v]){
12.       dis[w] = dis[v] + x.weight;  // χαλάρωση
13.       pq.decKey(w,dis[w]);         // ενημέρωση ουράς: #E φορές, το πολύ
14.       g.T[w] = v;                  // προσθήκη στο δένδρο
15.     }
16. }
```

- Ομοιότητα με Prim, όμως διαφορά στον στόχο (υπολογισμό αποστάσεων και όχι δένδρου)
- **Πολυπλοκότητα:** με δυωνυμική ουρά $O(V\log V+E\log V)=O(E\log V)$, με Fibonacci $O(V\log V+E)$

Αλγόριθμος Bellman-Ford

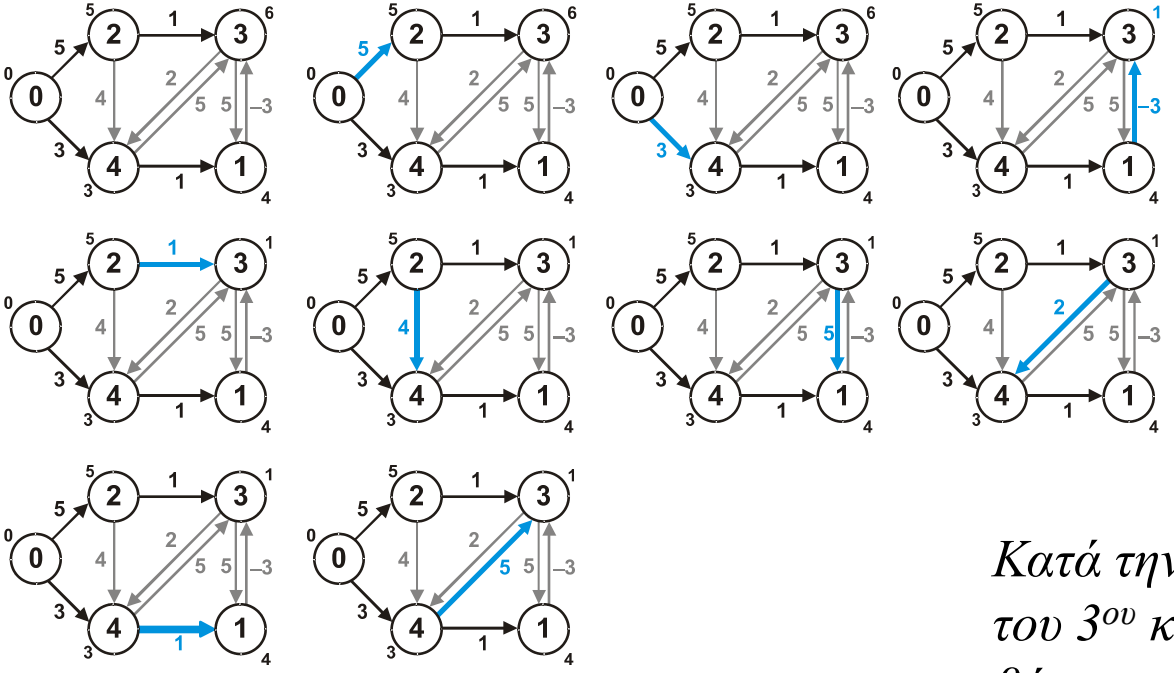
- Δουλεύει ***και*** με αρνητικά βάρη
- Στηρίζεται στην αρχή χαλαρώσεως:
  - Λόγω αρνητικών βαρών, είναι δυνατόν μία ακμή να χαλαρώσει περισσότερες από μία φορές
  - Γι' αυτό και εκτελεί V-1 βήματα χαλάρωσης, εξετάζοντας ***όλες*** τις ακμές
  - Αν κατά το V-στό βήμα σημειωθεί χαλάρωση, τότε ***υπάρχει αρνητικός κύκλος***

Παράδειγμα Bellman-Ford

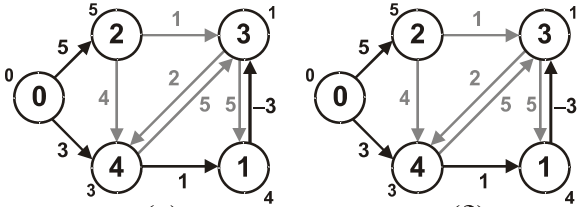


(α)

Παράδειγμα Bellman-Ford (συν.)



(β)



(γ) (δ)

Κατά την διάρκεια του 3ου και 4ου βήματος χαλάρωσης (Σχ. (γ)-(δ)), δεν σημειώνεται καμμία αλλαγή

Απόδειξη Ορθότητας

- Με επαγωγή:
  - Κατά το *i*-στο βήμα, οι αποστάσεις που ανακαλύπτονται, δεν ξεπερνούν τα συντομότερα μονοπάτια το *πολύ i* ακμών.

    Απόδειξη. Έστω ότι το συντομότερο μονοπάτι από την *s* προς την *w*, το πολύ *i* ακμών, περνά από την (*v*,*w*). Κατά το (*i*+1)-στάδιο, δοκιμάζουμε χαλάρωση της (*v*,*w*)
    - *Καμμία αλλαγή*:
      - Τότε, βάσει επαγωγής, η dis(s,w) είναι το πολύ ίση με την συντομότερη απόσταση μονοπατιού το πολύ *i* ακμών, άρα και το πολύ *i*+1ακμών
    - *Επιτυχής χαλάρωση*:
      - Τότε, το συντομότερο μονοπάτι από την *s* στην *v* έχει το πολύ *i* ακμές - βάσει επαγωγής, η dis(*s*,*v*), αφού ολοκληρώθηκε κατά το *i*-στό στάδιο, πήρε τιμή που δεν ξεπερνά το συντομότερο μονοπάτι *i*, το πολύ, ακμών
      - Άρα, η απόσταση dis(*s*,*w*)= dis(*s*,*v*) + βάρος(*v*,*w*) δεν ξεπερνά την τιμή του συντομότερου μονοπατιού *i*+1, το πολύ, ακμών

Απόδειξη Ορθότητας (συν.)

- Επομένως, μετά τα V-1 στάδια, όλες οι αποστάσεις που έχουν υπολογιστεί δεν ξεπερνούν το μήκος των συντομότερων μονοπατιών V-1 ακμών, το πολύ (μέγιστο # ακμών)
- Εάν κατά το V στάδιο σημειωθεί κάποια μείωση, τότε σίγουρα υπάρχει αρνητικός κύκλος, αφού το μονοπάτι θα έχει > V-1 ακμές

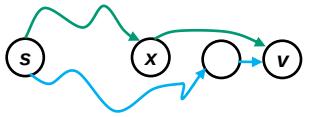
Ψευδοκώδικας

```
Algorithm BellmanFordSSSP(graph g, vertex s)
1.  double[] dis = new double[g.V];
2.  for (v = 0; v < g.V; v++){  // αρχικοποίηση
3.    dis[v] = INFTY;
4.    g.T[v] = -1;
5.  }
6.  dis[s] = 0;
7.  for (i = 0; i < g.V-1; i++)          // Βήμα χαλάρωσης, #V-1 φορές συνολικά
8.    for (x = 0; x < g.E; x++)          // δοκιμή χαλάρωσης κάθε ακμής, #E φορές συνολικά
9.      if ((dis[v = g.Edges[x].v]+g.Edges[x].weight) < dis[w = g.Edges[x].w]){
10.       dis[w] = dis[v]+g.Edges[x].weight;
11.       g.T[w] = v;
12.     }
13. for (x = 0; x < g.E; x++)          // Δοκιμή για αρνητικό κύκλο
14.   if  ((dis[g.Edges[x].v]+g.Edges[x].weight) < dis[g.Edges[x].w])
15.     return "αρνητικός κύκλος";
16. return dis;
```

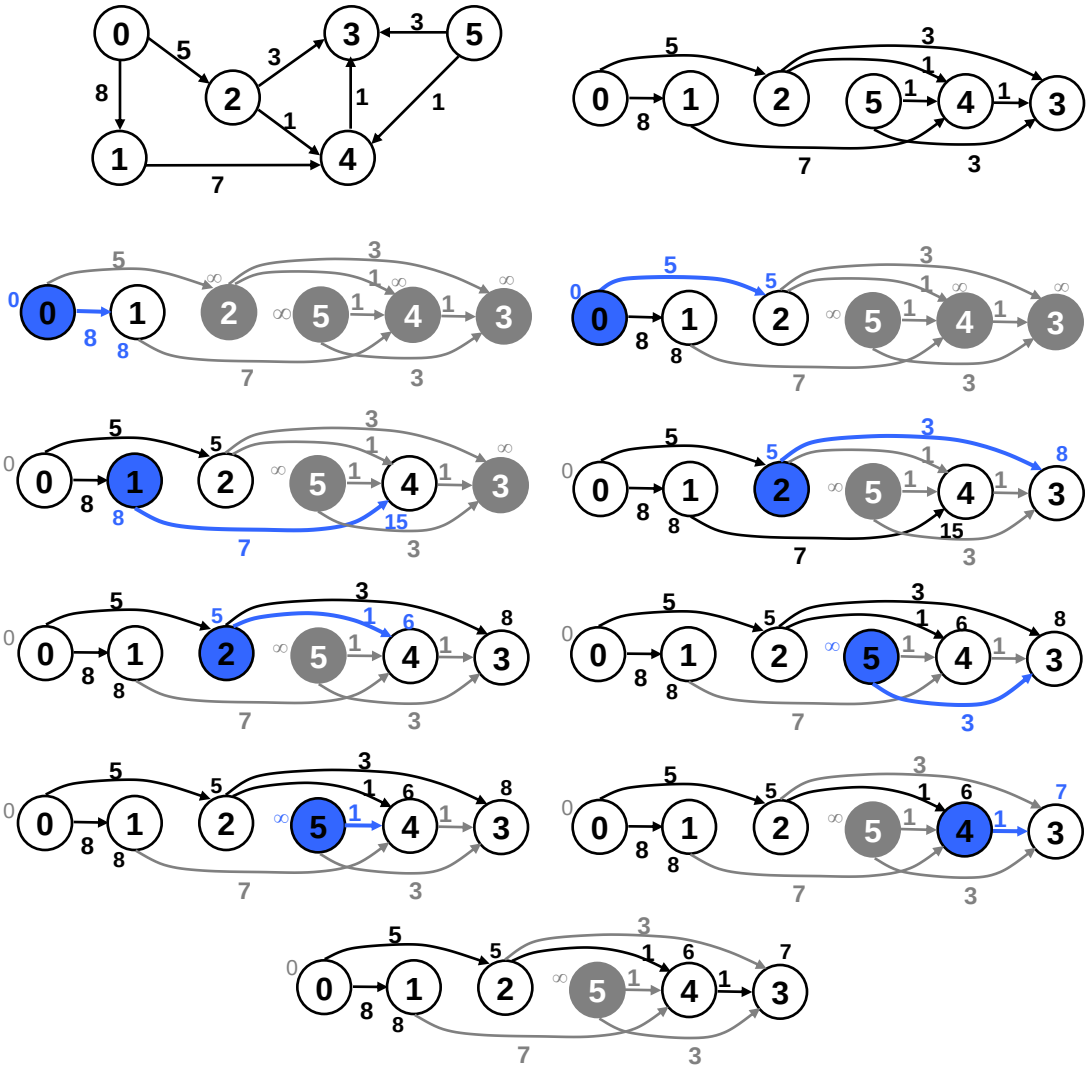
- **Πολυπλοκότητα: *O*(*VE*)**
- **Καλύτερη υλοποίηση με ουρά, γιατί μόνον οι ακμές που καταλήγουν σε κορυφές με ανανεωμένη πληροφορία έχουν πιθανότητα να βελτιώσουν την γνώση**

ΣΜΜΠ ΣΕ ΚΑΓ

- Τα ΚΑΓ επιτρέπουν, σε γραμμικό χρόνο, τον υπολογισμό ΣΜΜΠ με πολύ απλό τρόπο:
  - Οι κορυφές εξετάζονται κατά τοπολογική σειρά, δοκιμάζοντας για χαλάρωση όλες τις ακμές εξόδου τους
- Ορθότητα πηγάζει από την τοπολογική σειρά:
  - Εάν κάποιο μονοπάτι δεν ήταν σωστό, τότε έστω *v* η κορυφή με λανθασμένη απόσταση και τον μικρότερο αριθμό τοπολογικής διατάξεως
  - Τότε, εφ' όσον η απόσταση προς την *v* δύναται να μειωθεί, έστω ότι το αληθινά συντομότερο, για την *v*, μονοπάτι έχει τελευταία ακμή την (*x*,*v*)
  - Η *x* έχει αριθμό τοπολογικής διατάξεως μικρότερο από αυτόν της *v*. Άρα, το μονοπάτι από την *s* στην *x* έχει την σωστή απόσταση και η (*x*,*v*) δεν χρησιμοποιήθηκε για χαλάρωση (άτοπο)



Παράδειγμα ΣΜΜΠ σε ΚΑΓ



Ψευδοκώδικας

Algorithm dagSSSP(graph g, vertex s);

1. **int**[] topsort = topSortList(g, **new int**[g.V]); // $O(V+E)$
2. **double**[] dis = **new double[g.V]**;
3. **for** (v = 0; v < g.V; v++){ *// # V φορές*
4. dis[v] = **INFTY**;
5. g.T[v] = -1;
6. }
7. dis[s] = 0.0;
8. **for** (i = 0; i < g.V; i++) *// χαλάρωση κατά τοπολογική διάταξη*
9. **for** (x = g.List[v = topsort[i]]; x != null; x = x.getNext()){
10. **if** ((dis[v]+x.weight) < dis[w = x.v]){ *// # E φορές*
11. dis[w] = dis[v]+x.weight;
12. g.T[w] = v;
13. }

- **Πολυπλοκότητα: $O(V+E)$**

Συντομότερα Μονοπάτια Όλων των Ζευγών

- Αιτεί:
  - για κάθε ζεύγος κορυφών v,w, το συντομότερο μονοπάτι που τα συνδέει, εάν υπάρχει
- Πρώτη Λύση:
  - Εκτέλεση V φορές του αλγορίθμου Dijkstra, με κόστος $O(VE\log V)$
  - Καλή, εφ' όσον:
    - δεν υπάρχουν αρνητικά βάρη, και
    - το γράφημα είναι αραιό

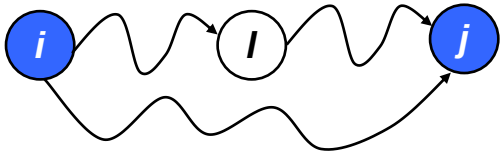
Αλγόριθμος Floyd

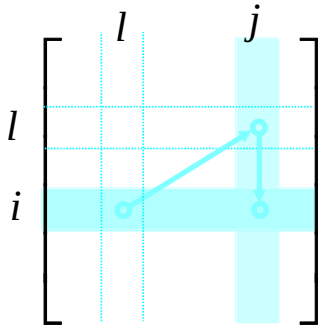
- Βασίζεται στην τεχνική του Δυναμικού Προγραμματισμού
- Ασφαλής επιλογή για πυκνά γραφήματα
- Ομοιάζει με τον Αλγόριθμο του Warshall για τον υπολογισμό της μεταβατικής κλειστότητας, ώστε αναφέρεται και ***ως Αλγόριθμος Floyd-Warshall***

Γενική Αρχή

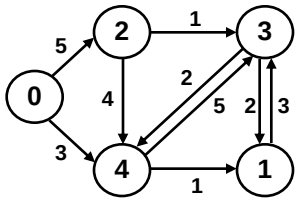
- Επιβάλλεται μία ***τυχαία*** διάταξη στις κορυφές και ακολουθούν V βήματα
- Κατά το l-στό βήμα, υπολογίζονται όλα τα συντομότερα μονοπάτια $\pi_{i,j}$ που χρησιμοποιούν, ως ενδιάμεσες κορυφές, ***μόνο*** τις l πρώτες, μέσω της χαλαρώσεως

$$d^{l}_{i,j} = \min\{d^{l-1}_{i,j}, d^{l-1}_{i,l} + d^{l-1}_{l,j}\}$$



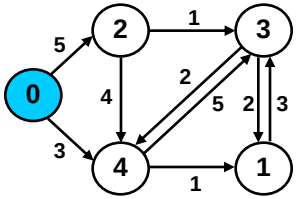


Παράδειγμα Τρεξίματος



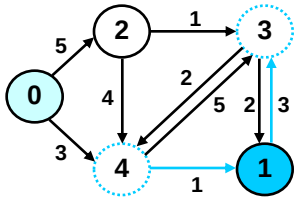
	0	1	2	3	4
0	0	∞	5	∞	3
1	∞	0	∞	3	∞
2	∞	∞	0	1	4
3	∞	2	∞	0	2
4	∞	1	∞	5	0

0	5	2	5	4
5	1	5	3	5
5	5	2	3	4
5	1	5	3	4
5	1	5	3	4



	0	1	2	3	4
0	0	∞	5	∞	3
1	∞	0	∞	3	∞
2	∞	∞	0	1	4
3	∞	2	∞	0	2
4	∞	1	∞	5	0

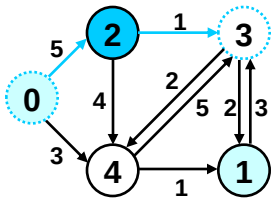
0	5	2	5	4
5	1	5	3	5
5	5	2	3	4
5	1	5	3	4
5	1	5	3	4



	0	1	2	3	4
0	0	∞	5	∞	3
1	∞	0	∞	3	∞
2	∞	∞	0	1	4
3	∞	2	∞	0	2
4	∞	1	∞	(4)	0

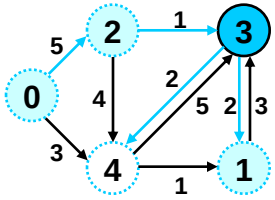
0	5	2	5	4
5	1	5	3	5
5	5	2	3	4
5	1	5	3	4
5	1	5	(1)	4

Παράδειγμα Τρεξίματος (συν.)



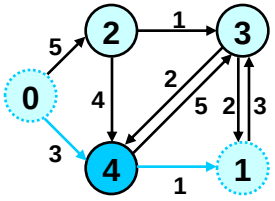
	0	1	2	3	4
0	0	∞	5	**6**	3
1	∞	0	∞	3	∞
2	∞	∞	0	1	4
3	∞	2	∞	0	2
4	∞	1	∞	4	0

0	5	2	**2**	4
5	1	5	3	5
5	5	2	3	4
5	1	5	3	4
5	1	5	1	4



	0	1	2	3	4
0	0	**8**	5	6	3
1	∞	0	∞	3	**5**
2	∞	**3**	0	1	**3**
3	∞	2	∞	0	2
4	∞	1	∞	4	0

0	**2**	2	2	4
5	1	5	3	**3**
5	**3**	2	3	**3**
5	1	5	3	4
5	1	5	1	4



	0	1	2	3	4
0	0	**4**	5	6	3
1	∞	0	∞	3	5
2	∞	3	0	1	3
3	∞	2	∞	0	2
4	∞	1	∞	4	0

0	**4**	2	2	4
5	1	5	3	3
5	3	2	3	3
5	1	5	3	4
5	1	5	1	4

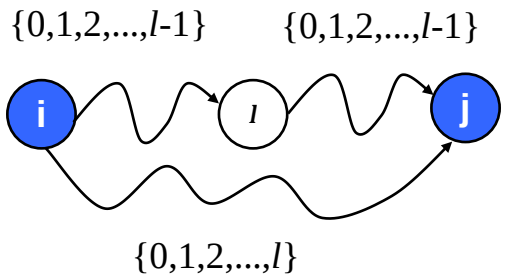
Ψευδοκώδικας

```
Algorithm FloydAPSP(graph g, double[][] d, int[][] path)
1.  for (i = 0; i < g.V; i++)
2.    for (j = 0; j < g.V; j++){
3.       d[i][j] = INFTY;
4.       path[i][j] = g.V;
5.    }
6.  for (i = 0; s < g.V; i++)
7.    for (j = 0; t < g.V; j++)
8.      if ((d[i][j] = g.A[i][j]) < INFTY)
9.         path[i][j] = j;
10. for (l = 0; l < g.V; l++)              // l-στο βήμα, #V φορές
11.   for (i = 0; i < g.V; i++)            // για κάθε i,  #V φορές
12.     if (d[i][l] < INFTY)
13.       for (j = 0; j < g.V; j++)        // και j, #V φορές
14.         if (d[i][j] > d[i][l]+d[l][j]){
15.              d[i][j] = d[i][l]+d[l][j];
16.              path[i][j] = path[i][l];  // καταγραφή της πρώτης κορυφής επί του π_{i,j}
17.          }
```

- **Πολυπλοκότητα: $O(V^3)$**

Ορθότητα

- Προκύπτει πολύ εύκολα από την αρχή της βέλτιστης υποδομής:
  - Κατά το *l*-στό βήμα, υπολογίζονται όλα τα συντομότερα μονοπάτια, αποτελούμενα ***αποκλειστικά*** από ενδιάμεσες κορυφές του συνόλου {0,1,2,...,*l*}



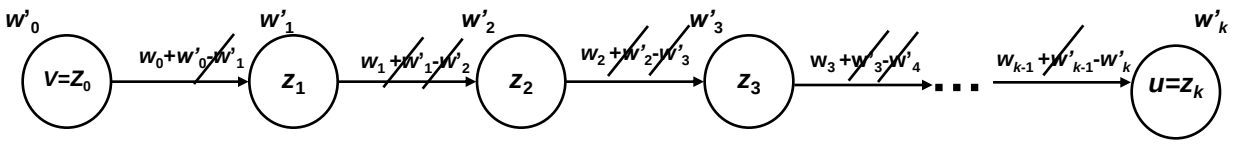
Συμπεράσματα

- Θετικά:
  - χειρίζεται και αρνητικά βάρη
  - εντοπίζει αρνητικούς κύκλους (στην διαγώνιο εμφανίζονται αρνητικές τιμές),
  - ασφαλή επιλογή για πυκνά γραφήματα:
    - εξοικονομείται ένας λογαριθμικός παράγοντας εν σχέσει με την επαναληπτική εφαρμογή Dijkstra
- Διαφορές με Warshall:
  - Άλλος ο στόχος (1 αντί για βάρος, $\vee$ αντί για min και $\wedge$ αντί για +)

Αλγόριθμος Johnson

- Η επιλογή Dijkstra συμφέρει για αραιά γραφήματα, όμως *δεν δουλεύει για αρνητικά βάρη*
- Johnson:
  - Παρατήρησε πως υπάρχει τρόπος εξαλείψεως αρνητικών βαρών, *εφ' όσον δεν υπάρχουν αρνητικοί κύκλοι*

Μετασχηματισμός Johnson



- Τα βάρη όλων των μονοπατιών μεταξύ δύο κορυφών *v*, *u* μεταβάλλονται κατά την ίδια ποσότητα $w'_0 - w'_k$:

$$w'_0 + \Sigma(w_i + w'_i - w'_{i+1}) - w'_k$$

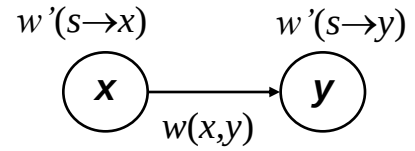
- Άρα, τα συντομότερα, παραμένουν συντομότερα…

Μετασχηματισμός (συν.)

- Για την εξάλειψη των αρνητικών βαρών, σε κάθε κορυφή *v*, ορίζεται -ως βάρος- η συντομότερη απόσταση από μία κορυφή-πηγή.
- Διακρίνονται δύο περιπτώσεις:
  - Η (x,y) ανήκει στο συντομότερο μονοπάτι $s \to y$.

    Άρα, η (x,y) είναι η τελευταία ακμή και $s \to x \to y = s \to y$. Συνεπώς,

    $$w(x,y) + w'(s \to x) - w'(s \to y) = w(x,y) - w(x,y) = 0$$

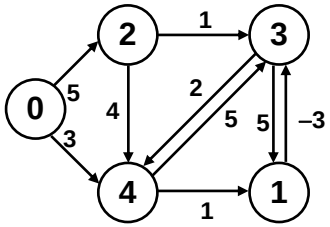
    

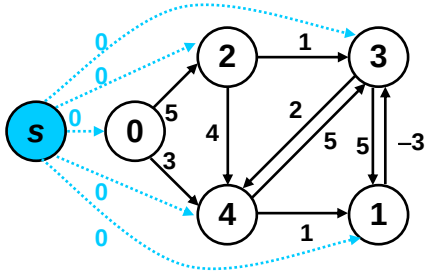
  - Η (x,y) δεν ανήκει στο συντομότερο μονοπάτι $s \to y$.

    Άρα, $s \to x \to y \neq s \to y$. Επομένως,

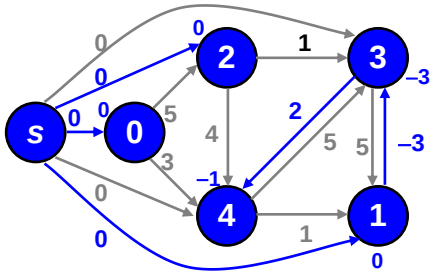
    $$w'(s \to x) + w(x,y) \geq w'(s \to y) \Rightarrow w(x,y) + w'(s \to x) - w'(s \to y) \geq 0$$

Παράδειγμα

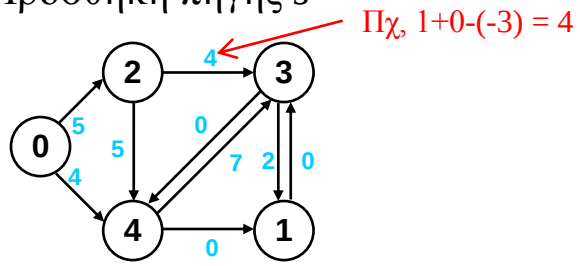




(α) Προσθήκη πηγής *s*



(β) ΣΜΜΠ για την *s*



(γ) Μετασχηματισμός των βαρών

Αλγόριθμος Johnson

- Bellman-Ford για υπολογισμό ΣΜΜΠ ή εντοπισμό κύκλου- κόστος $O(VE)$
- Ο Bellman-Ford εφαρμόζεται από μία πηγή s που συνδέεται με μηδενικές ακμές.
  - Έτσι εξασφαλίζουμε ότι τα βάρη-αποστάσεις θα είναι πεπερασμένα
- V φορές Dijkstra- κόστος $O(VE\log V)=O(V^2\log V)$, εφ'όσον είναι αραιό