Αλγόριθμοι και Πολυπλοκότητα

Δυναμικός Προγραμματισμός

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Δυναμικός Προγραμματισμός

Η τεχνική του δυναμικού προγραμματισμού, όπως και η διαίρει και βασίλευε, λύνει ένα πρόβλημα συνδυάζοντας λύσεις σε υποπροβλήματα[1].

[1] Η λέξη προγραμματισμός αναφέρεται σε μέθοδο με πίνακα και όχι σε κώδικα υπολογιστών.

Δυναμικός Προγραμματισμός

Η τεχνική του δυναμικού προγραμματισμού, όπως και η διαίρει και βασίλευε, λύνει ένα πρόβλημα συνδυάζοντας λύσεις σε υποπροβλήματα[1].

Κοινά Υποπροβλήματα. Αντίθετα από την τεχνική ”διαίρει και βασίλευε” η τεχνική του δυναμικού προγραμματισμού εφαρμόζεται όταν τα υποπροβλήματα δεν είναι ανεξάρτητα.

[1] Η λέξη προγραμματισμός αναφέρεται σε μέθοδο με πίνακα και όχι σε κώδικα υπολογιστών.

Δυναμικός Προγραμματισμός

Η τεχνική του δυναμικού προγραμματισμού, όπως και η διαίρει και βασίλευε, λύνει ένα πρόβλημα συνδυάζοντας λύσεις σε υποπροβλήματα[1].

Κοινά Υποπροβλήματα. Αντίθετα από την τεχνική ”διαίρει και βασίλευε” η τεχνική του δυναμικού προγραμματισμού εφαρμόζεται όταν τα υποπροβλήματα δεν είναι ανεξάρτητα.

Αποθήκευση σε πίνακα. Πολλές φορές λύνοντας ένα υποπρόβλημα, συναντάμε το ίδιο και το ίδιο μικρότερο υποπρόβλημα. Αντί να το λύνουμε συνεχώς όπως θα έκανε η τεχνική του ”διαίρει και βασίλευε” αποθηκεύουμε την λύση σε έναν πίνακα και ανατρέχουμε στον πίνακα την επόμενη φορά που θα μας ζητηθεί η λύση.

[1] Η λέξη προγραμματισμός αναφέρεται σε μέθοδο με πίνακα και όχι σε κώδικα υπολογιστών.

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Είσοδος. Μας δίνεται μια σειρά από n πίνακες $\langle A_1, A_2, \ldots, A_n \rangle$ των οποίων θέλουμε να υπολογίσουμε το γινόμενο $A_1 A_2 \cdots A_n$.

Σειρά Πολλαπλασιασμών.

- για τον πολλαπλασιασμό θα χρησιμοποιήσουμε τον κλασικό αλγόριθμο του σχολείου
- αυτό που μας ενδιαφέρει είναι η σειρά με την οποία θα κάνουμε τους πολλαπλασιασμούς

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Παράδειγμα 4 πινάκων.
για $\langle A_1, A_2, A_3, A_4 \rangle$ έχουμε 5 διαφορετικούς τρόπους να βρούμε το ίδιο αποτέλεσμα

1. $(A_1(A_2(A_3A_4)))$
2. $(A_1((A_2A_3)A_4))$
3. $((A_1A_2)(A_3A_4))$
4. $((A_1(A_2A_3))A_4)$
5. $(((A_1A_2)A_3)A_4)$

κάθε διαφορετικός τρόπος έχει και διαφορετικό κόστος

Πολλαπλασιασμός Πινάκων

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{pmatrix}$$

```
for( i = 0; i < n; i++ )
   for( j = 0; j < p; j++ )
      for( k = 0, c[i][j] = 0.0; k < m; k++ )
         c[i][j] += a[i][k] * b[k][j];
```

Πολλαπλασιασμός Πινάκων

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{pmatrix}$$

```
for( i = 0; i < n; i++ )
   for( j = 0; j < p; j++ )
      for( k = 0, c[i][j] = 0.0; k < m; k++ )
         c[i][j] += a[i][k] * b[k][j];
```

- ο πολλαπλασιασμός είναι αυτός που κοστίζει περισσότερο εδώ,
- κάνουμε $n \cdot p \cdot m$ πολλαπλασιασμούς.

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Παράδειγμα 3 πινάκων.
Έστω οι 3 πίνακες $\langle A_1, A_2, A_3 \rangle$ με διαστάσεις 10×100, 100×5 και 5×50 αντίστοιχα.

Μπορούμε να τους πολλαπλασιάσουμε με τους παρακάτω δύο τρόπους

1. $((A_1 A_2) A_3)$
2. $(A_1 (A_2 A_3))$

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Παράδειγμα 3 πινάκων.
Έστω οι 3 πίνακες $\langle A_1, A_2, A_3 \rangle$ με διαστάσεις 10×100, 100×5 και 5×50 αντίστοιχα.

Μπορούμε να τους πολλαπλασιάσουμε με τους παρακάτω δύο τρόπους

1. $((A_1 A_2) A_3)$
2. $(A_1 (A_2 A_3))$

Πρώτη Έκφραση. Ο πρώτος τρόπος κοστίζει $10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$ πολλαπλασιασμούς.

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Παράδειγμα 3 πινάκων.
Έστω οι 3 πίνακες $\langle A_1, A_2, A_3 \rangle$ με διαστάσεις 10×100, 100×5 και 5×50 αντίστοιχα.

Μπορούμε να τους πολλαπλασιάσουμε με τους παρακάτω δύο τρόπους

1. $((A_1 A_2) A_3)$
2. $(A_1 (A_2 A_3))$

Πρώτη Έκφραση. Ο πρώτος τρόπος κοστίζει $10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$ πολλαπλασιασμούς.

Δεύτερη Έκφραση. Ο δεύτερος τρόπος κοστίζει $100 \times 5 \times 50 + 10 \times 100 \times 50 = 75000$ πολλαπλασιασμούς.

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Παράδειγμα 3 πινάκων.
Έστω οι 3 πίνακες $\langle A_1, A_2, A_3 \rangle$ με διαστάσεις 10×100, 100×5 και 5×50 αντίστοιχα.

Μπορούμε να τους πολλαπλασιάσουμε με τους παρακάτω δύο τρόπους

1. $((A_1 A_2) A_3)$
2. $(A_1 (A_2 A_3))$

Πρώτη Έκφραση. Ο πρώτος τρόπος κοστίζει $10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$ πολλαπλασιασμούς.

Δεύτερη Έκφραση. Ο δεύτερος τρόπος κοστίζει $100 \times 5 \times 50 + 10 \times 100 \times 50 = 75000$ πολλαπλασιασμούς.

Ο πρώτος τρόπος είναι 10 φορές πιο γρήγορος!

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Πρόβλημα. Δεδομένου μιας σειράς n πινάκων $\langle A_1, A_2, \ldots, A_n \rangle$, όπου για $i = 1, 2, \ldots, n$ ο πίνακας A_i έχει μέγεθος $p_{i-1} \times p_i$, βρείτε την τοποθέτηση παρενθέσεων έτσι ώστε το γινόμενο $A_1 A_2 \cdots A_n$ να υπολογίζεται με τον μικρότερο αριθμό πολλαπλασιασμών.

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Πρόβλημα. Δεδομένου μιας σειράς n πινάκων $\langle A_1, A_2, \ldots, A_n \rangle$, όπου για $i = 1, 2, \ldots, n$ ο πίνακας A_i έχει μέγεθος $p_{i-1} \times p_i$, βρείτε την τοποθέτηση παρενθέσεων έτσι ώστε το γινόμενο $A_1 A_2 \cdots A_n$ να υπολογίζεται με τον μικρότερο αριθμό πολλαπλασιασμών.

Εκθετικός Αριθμός Λύσεων. Μια λύση ωμής βίας δεν είναι εφικτή αφού μπορούμε να δείξουμε πως ο αριθμός των δυνατών τρόπων να τοποθετήσουμε τις παρενθέσεις είναι εκθετικός όσο αναφορά των αριθμό των πινάκων n.

Δομή της Βέλτιστης Λύσης

Το πρώτο βήμα που πρέπει να κάνουμε είναι να χαρακτηρίσουμε την δομή των βέλτιστων λύσεων του προβλήματος.

Ορισμός. Έστω $A_{i..j}$ το γινόμενο $A_i A_{i+1} \cdots A_j$.

Βέλτιστη Λύση. Η βέλτιστη λύση για τους πίνακες $A_1 A_2 \cdots A_n$ σπάει το γινόμενο σε κάποια θέση $1 \leq k < n$ μεταξύ των πινάκων A_k και A_{k+1}.

- υπολογίζουμε δηλαδή πρώτα το γινόμενο $A_{1..k}$
- έπειτα το $A_{k+1..n}$
- τέλος πολλαπλασιάζουμε αυτά μεταξύ τους για το τελικό γινόμενο $A_{1..n}$

Δομή της Βέλτιστης Λύσης

Βέλτιστη Λύση. Η βέλτιστη λύση για τους πίνακες $A_1 A_2 \cdots A_n$ σπάει το γινόμενο σε κάποια θέση $1 \leq k < n$ μεταξύ των πινάκων A_k και A_{k+1}.

- υπολογίζουμε δηλαδή πρώτα το γινόμενο $A_{1..k}$
- έπειτα το $A_{k+1..n}$
- τέλος πολλαπλασιάζουμε αυτά μεταξύ τους για το τελικό γινόμενο $A_{1..n}$

Παρατήρηση Κλειδί. Στην βέλτιστη λύση του προβλήματος $A_1 A_2 \cdots A_n$ η λύση του υποπροβλήματος $A_1 A_2 \cdots A_k$ πρέπει να είναι και αυτή βέλτιστη για το πρόβλημα των πινάκων $A_1 A_2 \cdots A_k$.

- Αλλιώς θα μπορούσαμε να την αλλάξουμε με την βέλτιστη και να πάρουμε μια καλύτερη λύση για το συνολικό πρόβλημα, άτοπο.
- Το ίδιο ισχύει και για το υποπρόβλημα $A_{k+1..n}$.

Δομή της Βέλτιστης Λύσης

Ιδιότητα. Η βέλτιστη λύση σε ένα στιγμιότυπο του προβλήματος περιέχει βέλτιστες λύσεις στα στιγμιότυπα των υποπροβλημάτων.

Η παραπάνω ιδιότητα είναι κάτι που πάντα πρέπει να μας βάζει σε υποψίες για την χρήση της τεχνικής του δυναμικού προγραμματισμού.

Αναδρομικός Ορισμός Βέλτιστης Λύσης

Το δεύτερο βήμα είναι να ορίσουμε την τιμή της βέλτιστης λύσης αναδρομικά με βάση τις βέλτιστες λύσεις των υποπροβλημάτων.

Υποπρόβλημα. Έστω λοιπόν το υποπρόβλημα του υπολογισμού του ελάχιστου αριθμού πράξεων (πολλαπλασιασμών) για τον πολλαπλασιασμό $A_i A_{i+1} \cdots A_j$ για $1 \leq i \leq j \leq n$.

Ορισμός. Έστω $C[i, j]$ ο ελάχιστος αριθμός πράξεων (πολλαπλασιασμών) που χρειάζονται για να υπολογίσουμε το γινόμενο $A_{i..j}$. Ουσιαστικά μας ενδιαφέρει να υπολογίσουμε την τιμή $C[1, n]$.

Αναδρομικός Ορισμός Βέλτιστης Λύσης

Μπορούμε να ορίσουμε το $C[i, j]$ αναδρομικά:

Αναδρομικός Ορισμός Βέλτιστης Λύσης

Μπορούμε να ορίσουμε το $C[i, j]$ αναδρομικά:

- αν $i = j$ τότε το γινόμενο είναι ο πίνακας $A_{i..i} = A_i$ και δεν χρειάζεται κανένας πολλαπλασιασμός.
  Άρα $C[i, i] = 0$ για κάθε $i = 1, 2, \ldots, n$.

Αναδρομικός Ορισμός Βέλτιστης Λύσης

Μπορούμε να ορίσουμε το $C[i, j]$ αναδρομικά:

- αν $i = j$ τότε το γινόμενο είναι ο πίνακας $A_{i..i} = A_i$ και δεν χρειάζεται κανένας πολλαπλασιασμός.
  Άρα $C[i, i] = 0$ για κάθε $i = 1, 2, \ldots, n$.
- αν $i < j$ τότε η βέλτιστη λύση σπάει το πρόβλημα σε κάποιο $i \leq k < j$.

Αναδρομικός Ορισμός Βέλτιστης Λύσης

Μπορούμε να ορίσουμε το $C[i, j]$ αναδρομικά:

- αν $i = j$ τότε το γινόμενο είναι ο πίνακας $A_{i..i} = A_i$ και δεν χρειάζεται κανένας πολλαπλασιασμός.
  Άρα $C[i, i] = 0$ για κάθε $i = 1, 2, \ldots, n$.
- αν $i < j$ τότε η βέλτιστη λύση σπάει το πρόβλημα σε κάποιο $i \leq k < j$.

  Η βέλτιστη λύση τότε είναι ίση
  - ελάχιστο κόστος υπολογισμού του $A_{i..k}$
  - ελάχιστο κόστος υπολογισμού του $A_{k+1..j}$
  - κόστος πολλαπλασιασμού των $A_{i..k}$ και $A_{k+1..j}$ που είναι $p_{i-1}p_kp_j$ πράξεις (πολλαπλασιασμοί).

Αναδρομικός Ορισμός Βέλτιστης Λύσης

Μπορούμε να ορίσουμε το $C[i, j]$ αναδρομικά:

- αν $i = j$ τότε το γινόμενο είναι ο πίνακας $A_{i..i} = A_i$ και δεν χρειάζεται κανένας πολλαπλασιασμός.
  Άρα $C[i, i] = 0$ για κάθε $i = 1, 2, \ldots, n$.
- αν $i < j$ τότε η βέλτιστη λύση σπάει το πρόβλημα σε κάποιο $i \leq k < j$.

  Η βέλτιστη λύση τότε είναι ίση
  - ελάχιστο κόστος υπολογισμού του $A_{i..k}$
  - ελάχιστο κόστος υπολογισμού του $A_{k+1..j}$
  - κόστος πολλαπλασιασμού των $A_{i..k}$ και $A_{k+1..j}$ που είναι $p_{i-1}p_kp_j$ πράξεις (πολλαπλασιασμοί).

  Καταλήγουμε λοιπόν πως

$$C[i, j] = C[i, k] + C[k + 1, j] + p_{i-1}p_kp_j.$$

Αναδρομικός Ορισμός Βέλτιστης Λύσης

Μπορούμε να ορίσουμε το $C[i, j]$ αναδρομικά:

- αν $i = j$ τότε το γινόμενο είναι ο πίνακας $A_{i..i} = A_i$ και δεν χρειάζεται κανένας πολλαπλασιασμός.
  Άρα $C[i, i] = 0$ για κάθε $i = 1, 2, \ldots, n$.
- αν $i < j$ τότε η βέλτιστη λύση σπάει το πρόβλημα σε κάποιο $i \leq k < j$.

  Η βέλτιστη λύση τότε είναι ίση
  - ελάχιστο κόστος υπολογισμού του $A_{i..k}$
  - ελάχιστο κόστος υπολογισμού του $A_{k+1..j}$
  - κόστος πολλαπλασιασμού των $A_{i..k}$ και $A_{k+1..j}$ που είναι $p_{i-1}p_kp_j$ πράξεις (πολλαπλασιασμοί).

  Καταλήγουμε λοιπόν πως

$$C[i, j] = C[i, k] + C[k + 1, j] + p_{i-1}p_kp_j.$$

Μα δεν γνωρίζουμε την τιμή του k!!

Αναδρομικός Ορισμός Βέλτιστης Λύσης

Αφού δεν γνωρίζουμε την τιμή του k, θα δοκιμάσουμε όλες τις δυνατές τιμές που μπορεί να πάρει το k οι οποίες είναι $k = i, i+1, \ldots, j-1$ καταλήγοντας στην αναδρομή

$$C[i,j] = \begin{cases} 0 & \text{αν } i = j, \\ \min\limits_{i \le k < j} \{C[i,k] + C[k+1,j] + p_{i-1} p_k p_j\} & \text{αν } i < j. \end{cases}$$

Αν γράψουμε ένα αναδρομικό πρόγραμμα που να υπολογίζει την παραπάνω συνάρτηση, θα εκτελεστεί σε εκθετικό χρόνο.

Αριθμός Υποπροβλημάτων

Πολυωνυμικός Αριθμός Υποπροβλημάτων. Η σημαντική παρατήρηση εδώ είναι πως ο αριθμός των υποπροβλημάτων είναι πολυωνυμικός. Είναι όλες οι τιμές $C[i, j]$ για $1 \leq i \leq j \leq n$.

$$C[i, j] = \begin{cases} 0 & \text{αν } i = j, \\ \min\limits_{i \leq k < j} \{C[i, k] + C[k + 1, j] + p_{i-1} p_k p_j\} & \text{αν } i < j. \end{cases}$$

Χρήση Πίνακα. Δεν είναι ανάγκη να υπολογίζουμε κάθε τιμή ξανά και ξανά, μπορούμε να τις αποθηκεύσουμε σε έναν $n \times n$ πίνακα.

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Δυναμικός Αλγόριθμος

Input: $1 \leq i \leq j \leq n$
Input: Πίνακες $\langle A_1, A_2, \ldots, A_n \rangle$ όπου ο A_i έχει διαστάσεις $p_{i-1} \times p_i$
for $i = 1$ *έως* n **do**
  $C(i, i) = 0$
end
for $l = 2$ *έως* n **do**
  for $i = 1$ *έως* $n - l + 1$ **do**
    $j \leftarrow i + l - 1$
    $C(i, j) \leftarrow \infty$
    for $k = i$ *έως* $j - 1$ **do**
      $T \leftarrow C(i, k) + C(k + 1, j) + p_{i-1} p_k p_j$
      if $T < C(i, j)$ **then**
        $C(i, j) \leftarrow T$
      end
    end
  end
end
return $C(1, n)$

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Δυναμικός Αλγόριθμος

Input: $1 \leq i \leq j \leq n$
Input: Πίνακες $\langle A_1, A_2, \ldots, A_n \rangle$ όπου ο A_i έχει διαστάσεις $p_{i-1} \times p_i$
for $i = 1$ *έως* n **do**
  $C(i, i) = 0$
end
for $l = 2$ *έως* n **do**
  for $i = 1$ *έως* $n - l + 1$ **do**
    $j \leftarrow i + l - 1$
    $C(i, j) \leftarrow \infty$
    for $k = i$ *έως* $j - 1$ **do**
      $T \leftarrow C(i, k) + C(k + 1, j) + p_{i-1} p_k p_j$
      if $T < C(i, j)$ **then**
        $C(i, j) \leftarrow T$
      end
    end
  end
end
return $C(1, n)$

Αρχικά ο αλγόριθμος υπολογίζει τις τιμές $C(i, i)$ για $i = 1, \ldots, n$.

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Δυναμικός Αλγόριθμος

Input: $1 \leq i \leq j \leq n$
Input: Πίνακες $\langle A_1, A_2, \ldots, A_n \rangle$ όπου ο A_i έχει διαστάσεις $p_{i-1} \times p_i$
for $i = 1$ *έως* n **do**
  $C(i, i) = 0$
end
for $l = 2$ *έως* n **do**
  for $i = 1$ *έως* $n - l + 1$ **do**
    $j \leftarrow i + l - 1$
    $C(i, j) \leftarrow \infty$
    for $k = i$ *έως* $j - 1$ **do**
      $T \leftarrow C(i, k) + C(k + 1, j) + p_{i-1} p_k p_j$
      if $T < C(i, j)$ **then**
        $C(i, j) \leftarrow T$
      end
    end
  end
end
return $C(1, n)$

Μετά υπολογίζει τις τιμές $C(i, i + 1)$ για $i = 1, \ldots, n - 1$.

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Δυναμικός Αλγόριθμος

Input: $1 \leq i \leq j \leq n$
Input: Πίνακες $\langle A_1, A_2, \ldots, A_n \rangle$ όπου ο A_i έχει διαστάσεις $p_{i-1} \times p_i$
for $i = 1$ *έως* n **do**
  $C(i, i) = 0$
end
for $l = 2$ *έως* n **do**
  for $i = 1$ *έως* $n - l + 1$ **do**
    $j \leftarrow i + l - 1$
    $C(i, j) \leftarrow \infty$
    for $k = i$ *έως* $j - 1$ **do**
      $T \leftarrow C(i, k) + C(k + 1, j) + p_{i-1} p_k p_j$
      if $T < C(i, j)$ **then**
        $C(i, j) \leftarrow T$
      end
    end
  end
end
return $C(1, n)$

Μετά υπολογίζει τις τιμές $C(i, i + 2)$ για $i = 1, \ldots, n - 2$.

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Δυναμικός Αλγόριθμος

Input: $1 \leq i \leq j \leq n$
Input: Πίνακες $\langle A_1, A_2, \ldots, A_n \rangle$ όπου ο A_i έχει διαστάσεις $p_{i-1} \times p_i$
for $i = 1$ *έως* n **do**
  $C(i, i) = 0$
end
for $l = 2$ *έως* n **do**
  for $i = 1$ *έως* $n - l + 1$ **do**
    $j \leftarrow i + l - 1$
    $C(i, j) \leftarrow \infty$
    for $k = i$ *έως* $j - 1$ **do**
      $T \leftarrow C(i, k) + C(k + 1, j) + p_{i-1} p_k p_j$
      if $T < C(i, j)$ **then**
        $C(i, j) \leftarrow T$
      end
    end
  end
end
return $C(1, n)$

Για κάθε τιμή που υπολογίζεται οι προηγούμενες απαραίτητες τιμές έχουν ήδη υπολογιστεί.

Αλυσιδωτός Πολλαπλασιασμός Πινάκων

Δυναμικός Αλγόριθμος

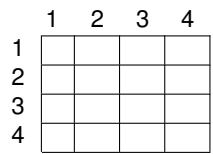
Input: $1 \leq i \leq j \leq n$
Input: Πίνακες $\langle A_1, A_2, \ldots, A_n \rangle$ όπου ο A_i έχει διαστάσεις $p_{i-1} \times p_i$
for $i = 1$ *έως* n **do**
  $C(i, i) = 0$
end
for $l = 2$ *έως* n **do**
  for $i = 1$ *έως* $n - l + 1$ **do**
    $j \leftarrow i + l - 1$
    $C(i, j) \leftarrow \infty$
    for $k = i$ *έως* $j - 1$ **do**
      $T \leftarrow C(i, k) + C(k + 1, j) + p_{i-1} p_k p_j$
      if $T < C(i, j)$ **then**
        $C(i, j) \leftarrow T$
      end
    end
  end
end
return $C(1, n)$

Ο χρόνος εκτέλεσης του αλγορίθμου είναι $\mathcal{O}(n^3)$.

Παράδειγμα

Έστω οι πίνακες

- $\langle A_1, A_2, A_3, A_4 \rangle$, με διαστάσεις
- $2 \times 20, 20 \times 1, 1 \times 10, 10 \times 1$.



Παράδειγμα

Έστω οι πίνακες

- $\langle A_1, A_2, A_3, A_4 \rangle$, με διαστάσεις
- $2 \times 20, 20 \times 1, 1 \times 10, 10 \times 1$.

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

Παράδειγμα

Έστω οι πίνακες

- $\langle A_1, A_2, A_3, A_4 \rangle$, με διαστάσεις
- $2 \times 20, 20 \times 1, 1 \times 10, 10 \times 1$.

	1	2	3	4
1	0	40		
2		0		
3			0	
4				0

Παράδειγμα

Έστω οι πίνακες

- $\langle A_1, A_2, A_3, A_4 \rangle$, με διαστάσεις
- $2 \times 20, 20 \times 1, 1 \times 10, 10 \times 1$.

	1	2	3	4
1	0	40		
2		0	200	
3			0	
4				0

Παράδειγμα

Έστω οι πίνακες

- $\langle A_1, A_2, A_3, A_4 \rangle$, με διαστάσεις
- $2 \times 20, 20 \times 1, 1 \times 10, 10 \times 1$.

	1	2	3	4
1	0	40		
2		0	200	
3			0	10
4				0

Παράδειγμα

Έστω οι πίνακες

- $\langle A_1, A_2, A_3, A_4 \rangle$, με διαστάσεις
- $2 \times 20, 20 \times 1, 1 \times 10, 10 \times 1$.

	1	2	3	4
1	0	40	60	
2		0	200	
3			0	10
4				0

Παράδειγμα

Έστω οι πίνακες

- $\langle A_1, A_2, A_3, A_4 \rangle$, με διαστάσεις
- $2 \times 20, 20 \times 1, 1 \times 10, 10 \times 1$.

	1	2	3	4
1	0	40	60	
2		0	200	30
3			0	10
4				0

Παράδειγμα

Έστω οι πίνακες

- $\langle A_1, A_2, A_3, A_4 \rangle$, με διαστάσεις
- $2 \times 20, 20 \times 1, 1 \times 10, 10 \times 1$.

	1	2	3	4
1	0	40	60	52
2		0	200	30
3			0	10
4				0

Σακίδιο

Knapsack

Πρόβλημα. Κατά την διάρκεια μιας κλοπής, ένας διαρρήκτης βρίσκει περισσότερα λάφυρα από όσα περίμενε και πρέπει να αποφασίσει τι θα πάρει μαζί του. Το σακίδιο του (knapsack) μπορεί να μεταφέρει συνολικό βάρος μέχρι W κιλά. Υπάρχουν n είδη από τα οποία μπορεί να επιλέξει, με βάρη $w_1, \ldots, w_n$ και χρηματική αξία $v_1, \ldots, v_n$ αντίστοιχα. Ποιός είναι ο καλύτερος συνδυασμός από πλευράς αξίας που μπορεί να μεταφέρει;

Σακίδιο
Knapsack

Πρόβλημα. Κατά την διάρκεια μιας κλοπής, ένας διαρρήκτης βρίσκει περισσότερα λάφυρα από όσα περίμενε και πρέπει να αποφασίσει τι θα πάρει μαζί του. Το σακίδιο του (knapsack) μπορεί να μεταφέρει συνολικό βάρος μέχρι W κιλά. Υπάρχουν n είδη από τα οποία μπορεί να επιλέξει, με βάρη $w_1, \ldots, w_n$ και χρηματική αξία $v_1, \ldots, v_n$ αντίστοιχα. Ποιός είναι ο καλύτερος συνδυασμός από πλευράς αξίας που μπορεί να μεταφέρει;

Το παραπάνω πρόβλημα μπορεί να μοντελοποιήσει μια μεγάλη ποικιλία εργασιών επιλογής πόρων με περιορισμούς. π.χ ανάθεση διεργασιών σε υπερ-υπολογιστή, κ.τ.λ.

Σακίδιο
Knapsack

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

Σακίδιο
Knapsack

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

Υπάρχουν δύο εκδοχές του προβλήματος, ανάλογα με το αν υπάρχει απεριόριστη ποσότητα κάθε είδους ή μόνο ένα.

1. εαν υπάρχουν απεριόριστες πρέπει να διαλέξουμε το είδος 1 και δύο τεμάχια από το είδος 4 (σύνολο: 48€)
2. εαν υπάρχει μόνο ένα από κάθε είδος, τότε το βέλτιστο σακίδιο περιέχει τα είδη 1 και 3 (σύνολο: 46€)

Σακίδιο

Ως σήμερα δεν υπάρχει πολυωνυμικός αλγόριθμος επίλυσης του προβλήματος του σακιδίου, και είναι το πιο πιθανό να μην υπάρχει καθόλου.

Θα περιγράψουμε μια μέθοδο δυναμικού προγραμματισμού που λύνει το πρόβλημα σε χρόνο $\mathcal{O}(nW)$.

Σακίδιο

Ως σήμερα δεν υπάρχει πολυωνυμικός αλγόριθμος επίλυσης του προβλήματος του σακιδίου, και είναι το πιο πιθανό να μην υπάρχει καθόλου.

Θα περιγράψουμε μια μέθοδο δυναμικού προγραμματισμού που λύνει το πρόβλημα σε χρόνο $\mathcal{O}(nW)$.

Ψευδοπολυωνυμικός Χρόνος. Προσοχή το όριο $\mathcal{O}(nW)$ δεν είναι πολυωνυμικό αφού το μέγεθος της εισόδου πρέπει να είναι ανάλογο του $\log W$ και όχι του W. Τέτοιου είδους όρια ονομάζονται ψευδοπολυωνυμικά (pseudo-polynomial).

Σακίδιο με Επανάληψη

Έστω λοιπόν το πρόβλημα σακιδίου όπου μπορούμε να πάρουμε έναν απεριόριστο αριθμό από κάθε είδος αντικειμένου.

Υποπροβλήματα. Το πρώτο πράγμα που πρέπει να κάνουμε είναι να απαντήσουμε στο ερώτημα ”ποια είναι τα υποπροβλήματα”. Στην συνέχεια πρέπει να εκφράσουμε την βέλτιστη λύση του προβλήματος συναρτήσει βέλτιστων λύσεων υποπροβλημάτων.

Σακίδιο με Επανάληψη

Έστω λοιπόν το πρόβλημα σακιδίου όπου μπορούμε να πάρουμε έναν απεριόριστο αριθμό από κάθε είδος αντικειμένου.

Υποπροβλήματα. Το πρώτο πράγμα που πρέπει να κάνουμε είναι να απαντήσουμε στο ερώτημα "ποια είναι τα υποπροβλήματα". Στην συνέχεια πρέπει να εκφράσουμε την βέλτιστη λύση του προβλήματος συναρτήσει βέλτιστων λύσεων υποπροβλημάτων.

Μπορούμε να φτιάξουμε υποπροβλήματα με δύο τρόπους:

1. είτε εξετάζοντας την περίπτωση σακιδίων μικρότερης χωρητικότητας $w \leq W$
2. είτε την περίπτωση λιγότερων ειδών (π.χ τα είδη $1, 2, \ldots, j$ για $j \leq n$).

Σακίδιο με Επανάληψη

Ορισμός. Έστω λοιπόν $K(w)$ η μέγιστη τιμή η οποία επιτυγχάνεται με ένα σακίδιο χωρητικότητας w.

Θέλουμε να εκφράσουμε την τιμή $K(w)$ συναρτήσει μικρότερων υποπροβλημάτων.

Σακίδιο με Επανάληψη

Ορισμός. Έστω λοιπόν $K(w)$ η μέγιστη τιμή η οποία επιτυγχάνεται με ένα σακίδιο χωρητικότητας w.

Θέλουμε να εκφράσουμε την τιμή $K(w)$ συναρτήσει μικρότερων υποπροβλημάτων.

- Εαν η βέλτιστη λύση περιέχει ένα αντικείμενο είδους i τότε η λύση έχει αξία $v_i + K(w - w_i)$.

Σακίδιο με Επανάληψη

Ορισμός. Έστω λοιπόν $K(w)$ η μέγιστη τιμή η οποία επιτυγχάνεται με ένα σακίδιο χωρητικότητας w.

Θέλουμε να εκφράσουμε την τιμή $K(w)$ συναρτήσει μικρότερων υποπροβλημάτων.

- Εαν η βέλτιστη λύση περιέχει ένα αντικείμενο είδους i τότε η λύση έχει αξία $v_i + K(w - w_i)$.

Επειδή δεν γνωρίζουμε ποιο αντικείμενο i περιέχει η βέλτιστη λύση, θα δοκιμάσουμε όλες τις δυνατότητες.

$$K(w) = \max_{i:w_i<w} \{K(w - w_i) + v_i\}.$$

Σε περίπτωση που το σύνολο είναι κενό χρησιμοποιούμε την σύμβαση πως η μέγιστη τιμή του είναι το 0.

Σακίδιο με Επανάληψη

Δυναμικός Αλγόριθμος για Σακίδιο με Επανάληψη

```
K(0) = 0
for w = 1 έως W do
    max ← 0
    for i = 1 εώς n do
        if w_i ≤ w then
            T ← K(w − w_i) + v_i
            if T > max then
                max ← T
            end
        end
    end
    K(w) ← max
end
```

Σακίδιο με Επανάληψη

Δυναμικός Αλγόριθμος για Σακίδιο με Επανάληψη

```
K(0) = 0
for w = 1 έως W do
    max ← 0
    for i = 1 εώς n do
        if w_i ≤ w then
            T ← K(w − w_i) + v_i
            if T > max then
                max ← T
            end
        end
    end
    K(w) ← max
end
```

- Ο αλγόριθμος συμπληρώνει ένα μονοδιάστατο πίνακα μήκους $W + 1$, από τα αριστερά προς τα δεξιά.
- Για κάθε καταχώρηση χρειάζεται το πολύ χρόνος $\mathcal{O}(n)$, οπότε ο συνολικός χρόνος εκτέλεσης είναι $\mathcal{O}(nW)$.

Σακίδιο με Επανάληψη

Παράδειγμα

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

Σακίδιο με Επανάληψη

Παράδειγμα

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

0	0								

Σακίδιο με Επανάληψη

Παράδειγμα

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

0	0	9							

Σακίδιο με Επανάληψη

Παράδειγμα

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

0	0	9	14							

Σακίδιο με Επανάληψη

Παράδειγμα

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

0	0	9	14	18						

Σακίδιο με Επανάληψη

Παράδειγμα

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

0	0	9	14	18	23					

Σακίδιο με Επανάληψη

Παράδειγμα

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

0	0	9	14	18	23	30				

Σακίδιο με Επανάληψη

Παράδειγμα

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

0	0	9	14	18	23	30	32			

Σακίδιο με Επανάληψη

Παράδειγμα

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

0	0	9	14	18	23	30	32	39		

Σακίδιο με Επανάληψη

Παράδειγμα

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

0	0	9	14	18	23	30	32	39	44	

Σακίδιο με Επανάληψη

Παράδειγμα

Έστω $W = 10$ και τα παρακάτω αντικείμενα

είδος	βάρος	τιμή
1	6	30€
2	3	14€
3	4	16€
4	2	9€

0	0	9	14	18	23	30	32	39	44	48

Σακίδιο χωρίς Επανάληψη

Έστω τώρα το πρόβλημα σακιδίου όπου μπορούμε να πάρουμε μόνο μια φορά κάθε αντικείμενο.

Υποπροβλήματα. Τα υποπροβλήματα της προηγούμενης λύσης δεν μας βοηθούν γιατί δεν εκφράζουν εαν ένα αντικείμενο έχει χρησιμοποιηθεί σε μια λύση και άρα αν είναι διαθέσιμο. Πρέπει λοιπόν να προσθέσουμε και μια δεύτερη παράμετρο.

Ορισμός. Έστω λοιπόν $K(w, j)$ η μέγιστη τιμή η οποία επιτυγχάνεται με ένα σακίδιο χωρητικότητας w με τα είδη $1, \ldots, j$.

Σακίδιο χωρίς Επανάληψη

Ορισμός. Έστω λοιπόν $K(w, j)$ η μέγιστη τιμή η οποία επιτυγχάνεται με ένα σακίδιο χωρητικότητας w με τα είδη $1, \ldots, j$.

Σακίδιο χωρίς Επανάληψη

Ορισμός. Έστω λοιπόν $K(w, j)$ η μέγιστη τιμή η οποία επιτυγχάνεται με ένα σακίδιο χωρητικότητας w με τα είδη $1, \ldots, j$.

Βέλτιστη Λύση. Για να εκφράσουμε ένα υποπρόβλημα $K(w, j)$ συναρτήσει μικρότερων υποπροβλημάτων, παρατηρούμε πως η βέλτιστη λύση είτε θα έχει το αντικείμενο j είτε όχι

$$K(w, j) = max\{K(w - w_j, j - 1) + v_j, K(w, j - 1)\}$$

όπου η πρώτη περίπτωση καλείται μόνο αν $w_j \leq w$.

Αρχικές Συνθήκες. Για αρχικές συνθήκες έχουμε $K(0, j) = 0$ για $1 \leq j \leq n$ και $K(w, 0) = 0$ για $0 \leq w \leq W$.

Σακίδιο χωρίς Επανάληψη

Δυναμικός Αλγόριθμος για Σακίδιο χωρίς Επανάληψη

Φτιάξε έναν πίνακα K διαστάσεων $W+1 \times n+1$
for $w = 0$ *έως* W **do**
  $K(w, 0) = 0$
end
for $j = 0$ *έως* n **do**
  $K(0, j) = 0$
end
for $j = 1$ *έως* n **do**
  for $w = 1$ *έως* W **do**
    if $w_j > w$ **then**
      $K(w, j) = K(w, j-1)$
    else
      $K(w, j) = max\{K(w, j-1), K(w - w_j, j-1) + v_j\}$
    end
  end
end
return $K(W, n)$

Σακίδιο χωρίς Επανάληψη

Δυναμικός Αλγόριθμος για Σακίδιο χωρίς Επανάληψη

Φτιάξε έναν πίνακα K διαστάσεων $W+1 \times n+1$
for $w = 0$ *έως* W **do**
  $K(w, 0) = 0$
end
for $j = 0$ *έως* n **do**
  $K(0, j) = 0$
end
for $j = 1$ *έως* n **do**
  for $w = 1$ *έως* W **do**
    if $w_j > w$ **then**
      $K(w, j) = K(w, j-1)$
    else
      $K(w, j) = max\{K(w, j-1), K(w - w_j, j-1) + v_j\}$
    end
  end
end
return $K(W, n)$

- Ο αλγόριθμος συμπληρώνει ένα διδιάστατο πίνακα με $W+1$ γραμμές και $n+1$ στήλες.
- Κάθε καταχώρηση χρειάζεται σταθερό χρόνο και άρα ο αλγόριθμος πέρνει χρόνο $\mathcal{O}(nW)$.

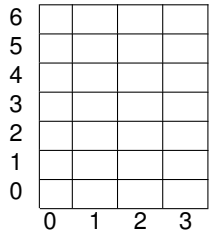
Έστω ένα σακίδιο με $W = 6$ και τρια αντικείμενα με $v_1 = w_1 = 2$, $v_2 = w_2 = 2$, $v_3 = w_3 = 3$.

Η περίπτωση όπου $v_i = w_i$ για κάθε i ονομάζεται πρόβλημα *Αθροίσματος Υποσυνόλων* (subset sum).

Σακίδιο Χωρίς Επανάληψη

Παράδειγμα

Έστω ένα σακίδιο με $W = 6$ και τρια αντικείμενα με $v_1 = w_1 = 2$, $v_2 = w_2 = 2$, $v_3 = w_3 = 3$.



Σακίδιο Χωρίς Επανάληψη

Παράδειγμα

Έστω ένα σακίδιο με $W = 6$ και τρια αντικείμενα με $v_1 = w_1 = 2$, $v_2 = w_2 = 2$, $v_3 = w_3 = 3$.

6	0			
5	0			
4	0			
3	0			
2	0			
1	0			
0	0	0	0	0
	0	1	2	3

Σακίδιο Χωρίς Επανάληψη

Παράδειγμα

Έστω ένα σακίδιο με $W = 6$ και τρια αντικείμενα με $v_1 = w_1 = 2$, $v_2 = w_2 = 2$, $v_3 = w_3 = 3$.

6	0	2		
5	0	2		
4	0	2		
3	0	2		
2	0	2		
1	0	0		
0	0	0	0	0
	0	1	2	3

Σακίδιο Χωρίς Επανάληψη

Παράδειγμα

Έστω ένα σακίδιο με $W = 6$ και τρια αντικείμενα με $v_1 = w_1 = 2$, $v_2 = w_2 = 2$, $v_3 = w_3 = 3$.

6	0	2	4	
5	0	2	4	
4	0	2	4	
3	0	2	2	
2	0	2	2	
1	0	0	0	
0	0	0	0	0
	0	1	2	3

Σακίδιο Χωρίς Επανάληψη

Παράδειγμα

Έστω ένα σακίδιο με $W = 6$ και τρια αντικείμενα με $v_1 = w_1 = 2$, $v_2 = w_2 = 2$, $v_3 = w_3 = 3$.

6	0	2	4	5
5	0	2	4	5
4	0	2	4	4
3	0	2	2	3
2	0	2	2	2
1	0	0	0	0
0	0	0	0	0
	0	1	2	3

Συντομότερες Διαδρομές σε Γραφήματα

Το πρόβλημα.
Μας δίνεται ένα κατευθυνόμενο γράφημα $G(V, E)$ όπου

- έχουμε ένα καθορισμένο κόμβο εκκίνησης $s \in V$,
- κάθε ακμή $e \in E$ έχει μήκος, το οποίο δηλώνει το χρόνο (ή την απόσταση ή το κόστος) που απαιτείται για τη διάσχιση της ακμής e,
- για μια διαδρομή P, το μήκος της P, είναι το άθροισμα των μηκών όλων των ακμών της και συμβολίζεται ως $l(P)$.

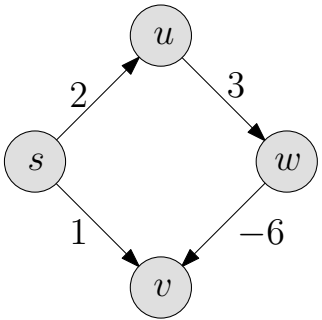
Σκοπός είναι να προσδιορίσουμε τη συντομότερη διαδρομή από τον s προς οποιονδήποτε άλλο κόμβο του γραφήματος.

Συντομότερες Διαδρομές σε Γραφήματα

Αλγόριθμος του Dijkstra. Είδαμε έναν άπληστο αλγόριθμο ο οποίος είναι πολύ αποδοτικός. Γιατί δεν χρησιμοποιούμε αυτόν τον αλγόριθμο;

Αρνητικά Μήκη. Ο αλγόριθμος του Dijkstra υποθέτει πως τα μήκη των ακμών είναι μη-αρνητικά.

Το πρόβλημα είναι πως ο αλγόριθμος του Dijkstra διαλέγει να ξεκινήσει ένα μονοπάτι χρησιμοποιώντας την φτηνότερη δυνατή λύση χωρίς να λαμβάνει υπόψη του ότι λόγω ύπαρξης αρνητικών μηκών, η καλύτερη διαδρομή μπορεί να ξεκινά με ακριβότερη ακμή και να αντισταθμίζει το κόστος αργότερα.



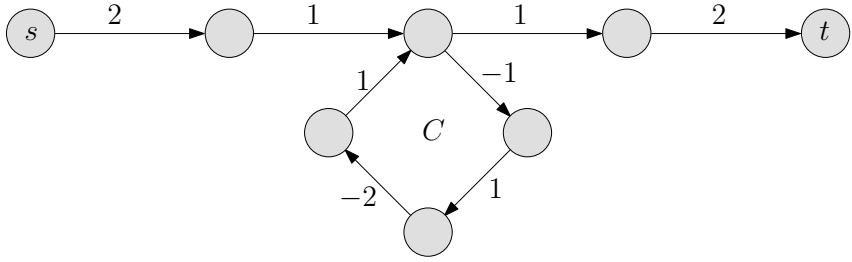
Αρνητικά Μήκη Ακμών

Θέλουμε να λύσουμε το γενικότερο πρόβλημα με αρνητικά μήκη ακμών.

Αρνητικά Μήκη; Το μήκος μιας ακμής μπορεί να υποδηλώνει και μεγέθη που πέρνουν αρνητικές τιμές. Για παράδειγμα οι κόμβοι ενός γραφήματος μπορεί να είναι ενδιάμεσοι έμποροι και το μήκος μιας ακμής από τον έμπορο i στον έμπορο j να αντιπροσωπεύει το κόστος μιας συναλλαγής στην οποία αγοράζουμε από τον έμπορο i και αμέσως μετά πουλάμε στον έμπορο j.

Αρνητικοί Κύκλοι

Για να έχει νόημα όμως το πρόβλημα **δεν** επιτρέπουμε την ύπαρξη αρνητικών κύκλων.



Αλλιώς μπορούμε να βρούμε διαδρομές με αυθαίρετα μεγάλο αρνητικό κόστος.

Αλγόριθμος Bellman-Ford

Θα αναπτύξουμε μια λύση δυναμικού προγραμματισμού η οποία βασίζεται στην παρακάτω παρατήρηση.

Λήμμα

Αν το G δεν έχει αρνητικούς κύκλους, τότε υπάρχει μια συντομότερη διαδρομή από το s μέχρι το t που είναι απλή (δηλαδή δεν επαναλαμβάνει κόμβους), και έτσι έχει το πολύ $n - 1$ ακμές.

Αλγόριθμος Bellman-Ford

Θα αναπτύξουμε μια λύση δυναμικού προγραμματισμού η οποία βασίζεται στην παρακάτω παρατήρηση.

Λήμμα

Αν το G δεν έχει αρνητικούς κύκλους, τότε υπάρχει μια συντομότερη διαδρομή από το s μέχρι το t που είναι απλή (δηλαδή δεν επαναλαμβάνει κόμβους), και έτσι έχει το πολύ $n - 1$ ακμές.

Απόδειξη

Αφού όλοι οι κύκλοι έχουν μη-αρνητικό κόστος, η συντομότερη διαδρομή P από το s στο t που έχει το μικρότερο αριθμό ακμών δεν επαναλαμβάνει καμία κορυφή v. Αν επαναλάμβανε μια κορυφή v, θα μπορούσαμε να αφαιρέσουμε το τμήμα της P μεταξύ των διαδοχικών επισκέψεων στη v καταλήγοντας σε διαδρομή που δεν έχει μεγαλύτερο κόστος και έχει λιγότερες ακμές. □

Αλγόριθμος Bellman-Ford

Ο αλγόριθμος κρατάει για κάθε κόμβο ένα άνω όριο της ελάχιστης απόστασης του από τον αρχικό κόμβο s.

Ιδέα. Θέλουμε να εκτελέσουμε n φάσεις. Στο τέλος της $1 \leq i \leq n$ φάσης να έχουμε βρει σωστές αποστάσεις για κάθε κόμβο $v \in V$, ο οποίος έχει μια συντομότερη $s - t$ διαδρομή που χρησιμοποιεί το πολύ i ακμές. Επειδή κάθε συντομότερη διαδρομή έχει το πολύ $n - 1$ ακμές στην τελευταία φάση θα έχουμε βρει όλες τις διαδρομές.

Αρχικές Προσεγγίσεις. Κρατάμε ένα πίνακα n θέσεων π, όπου $\pi(v)$ είναι ένα άνω όριο για την απόσταση του v από το s. Αρχικά

- $\pi(s) = 0$
- $\pi(v) = \infty$ για κάθε $v \in V \setminus \{s\}$.

Χαλάρωση Ακμής

Δεδομένου μιας ακμής $e = (u, v) \in E$ με μήκος c_e μπορούμε να ανανεώσουμε το άνω όριο μας για τον κόμβο v με τον εξής τρόπο.

Χαλάρωση ακμής $e = (u, v)$

Input: Πίνακας π με άνω όριο συντομότερης διαδρομής για κάθε κόμβο
$\pi(v) = min\{\pi(v), \pi(u) + c_e\}$

Προσέξτε πως η καινούρια τιμή του $\pi(v)$ είναι πάλι ένα σωστό άνω όριο για την συντομότερη $s - t$ διαδρομή. Η λειτουργία χαλάρωσης είναι λοιπόν "ασφαλής" με την έννοια πως όσες φορές και να την εκτελέσουμε για μια ακμή, τα άνω όρια παραμένουν σωστά.

Χαλάρωση Ακμών Διαδρομής

Έστω η συντομότερη διαδρομή μεταξύ του κόμβου s και ενός κόμβου t. Η διαδρομή αυτή περιέχει το πολύ $n-1$ ακμές.

Αν έχουμε $\pi(s) = 0$ και $\pi(v) = \infty$ για κάθε $v \in V \setminus \{s\}$ και εκτελέσουμε τις λειτουργίες χαλάρωσης με την σειρά

$$(s, u_1), (s, u_2), (s, u_3), \ldots, (u_k, t)$$

τότε στο τέλος το άνω όριο του κόμβου t, $\pi(t)$ θα είναι η σωστή απόσταση μέσω της συντομότερης αυτής διαδρομής.

Χαλάρωση Ακμών Διαδρομής

Έστω η συντομότερη διαδρομή μεταξύ του κόμβου s και ενός κόμβου t. Η διαδρομή αυτή περιέχει το πολύ $n-1$ ακμές.

Αν έχουμε $\pi(s) = 0$ και $\pi(v) = \infty$ για κάθε $v \in V \setminus \{s\}$ και εκτελέσουμε τις λειτουργίες χαλάρωσης με την σειρά

$$(s, u_1), (s, u_2), (s, u_3), \ldots, (u_k, t)$$

τότε στο τέλος το άνω όριο του κόμβου t, $\pi(t)$ θα είναι η σωστή απόσταση μέσω της συντομότερης αυτής διαδρομής.

Ασφαλής. Επειδή η διαδικασία χαλάρωσης είναι "ασφαλής" και δεν χαλάει το άνω όριο (το πολύ να το κάνει καλύτερο), δεν παίζει ρόλο τι άλλες χαλαρώσεις γίνονται σε αυτές τις ακμές ή στο υπόλοιπο γράφημα.

Χαλάρωση Ακμών Διαδρομής

Έστω η συντομότερη διαδρομή μεταξύ του κόμβου s και ενός κόμβου t. Η διαδρομή αυτή περιέχει το πολύ $n-1$ ακμές.

Αν έχουμε $\pi(s) = 0$ και $\pi(v) = \infty$ για κάθε $v \in V \setminus \{s\}$ και εκτελέσουμε τις λειτουργίες χαλάρωσης με την σειρά

$$(s, u_1), (s, u_2), (s, u_3), \ldots, (u_k, t)$$

τότε στο τέλος το άνω όριο του κόμβου t, $\pi(t)$ θα είναι η σωστή απόσταση μέσω της συντομότερης αυτής διαδρομής.

Ασφαλής. Επειδή η διαδικασία χαλάρωσης είναι "ασφαλής" και δεν χαλάει το άνω όριο (το πολύ να το κάνει καλύτερο), δεν παίζει ρόλο τι άλλες χαλαρώσεις γίνονται σε αυτές τις ακμές ή στο υπόλοιπο γράφημα.

Σωστή σειρά; Επειδή όμως δεν γνωρίζουμε εκ των προτέρων τις συντομότερες διαδρομές, πως μπορούμε να είμαστε βέβαιοι ότι ενημερώνουμε τις σωστές ακμές με τη σωστή σειρά;

Χαλάρωση Ακμών Διαδρομής

Έστω η συντομότερη διαδρομή μεταξύ του κόμβου s και ενός κόμβου t. Η διαδρομή αυτή περιέχει το πολύ $n-1$ ακμές.

Αν έχουμε $\pi(s) = 0$ και $\pi(v) = \infty$ για κάθε $v \in V \setminus \{s\}$ και εκτελέσουμε τις λειτουργίες χαλάρωσης με την σειρά

$$(s, u_1), (s, u_2), (s, u_3), \ldots, (u_k, t)$$

τότε στο τέλος το άνω όριο του κόμβου t, $\pi(t)$ θα είναι η σωστή απόσταση μέσω της συντομότερης αυτής διαδρομής.

Ασφαλής. Επειδή η διαδικασία χαλάρωσης είναι "ασφαλής" και δεν χαλάει το άνω όριο (το πολύ να το κάνει καλύτερο), δεν παίζει ρόλο τι άλλες χαλαρώσεις γίνονται σε αυτές τις ακμές ή στο υπόλοιπο γράφημα.

Σωστή σειρά; Επειδή όμως δεν γνωρίζουμε εκ των προτέρων τις συντομότερες διαδρομές, πως μπορούμε να είμαστε βέβαιοι ότι ενημερώνουμε τις σωστές ακμές με τη σωστή σειρά;

Εύκολη λύση. Θα χαλαρώσουμε όλες τις ακμές τουλάχιστον $n-1$ φορές.

Αλγόριθμος Bellman-Ford

Αλγόριθμος Bellman-Ford

Input: Κατευθυνόμενο γράφημα $G(V, E)$ με μήκη ακμών $\{c_e : e \in C\}$ χωρίς αρνητικούς κύκλους και κορυφή $s \in V$.
Output: Για κάθε κορυφή $v \in V$ προσπελάσιμη από την s, η $\pi(v)$ είναι ίση με την απόσταση από την s στη v.

for *κάθε* $v \in V$ **do**
  $\pi(v) = \infty$
end
$\pi(s) = 0$
for $i = 1$ *έως* $n - 1$ **do**
  for *κάθε ακμή* $e = (u, v) \in E$ **do**
    $\pi(v) = min\{\pi(v), \pi(u) + c_e\}$
  end
end

Αλγόριθμος Bellman-Ford

Αλγόριθμος Bellman-Ford

Input: Κατευθυνόμενο γράφημα $G(V, E)$ με μήκη ακμών $\{c_e : e \in C\}$ χωρίς αρνητικούς κύκλους και κορυφή $s \in V$.
Output: Για κάθε κορυφή $v \in V$ προσπελάσιμη από την s, η $\pi(v)$ είναι ίση με την απόσταση από την s στη v.

for *κάθε* $v \in V$ **do**
  $\pi(v) = \infty$
end
$\pi(s) = 0$
for $i = 1$ *έως* $n - 1$ **do**
  for *κάθε ακμή* $e = (u, v) \in E$ **do**
    $\pi(v) = min\{\pi(v), \pi(u) + c_e\}$
  end
end

Ο αλγόριθμος υπολογίζει για κάθε $1 \leq i \leq n$ τις συντομότερες διαδρομές χρησιμοποιώντας το πολύ i ακμές.

Αλγόριθμος Bellman-Ford

Αλγόριθμος Bellman-Ford

Input: Κατευθυνόμενο γράφημα $G(V, E)$ με μήκη ακμών $\{c_e : e \in C\}$ χωρίς αρνητικούς κύκλους και κορυφή $s \in V$.
Output: Για κάθε κορυφή $v \in V$ προσπελάσιμη από την s, η $\pi(v)$ είναι ίση με την απόσταση από την s στη v.

for *κάθε* $v \in V$ **do**
  $\pi(v) = \infty$
end
$\pi(s) = 0$
for $i = 1$ *έως* $n - 1$ **do**
  for *κάθε ακμή* $e = (u, v) \in E$ **do**
    $\pi(v) = min\{\pi(v), \pi(u) + c_e\}$
  end
end

Μπορούμε να καταγράψουμε και το δέντρο συντομότερων διαδρομών αν κρατήσουμε ένα πίνακα $prev(\cdot)$ όπου για κάθε κόμβο $v \in V$ κρατάμε την τελευταία ακμή $e = (u, v)$ που χρησιμοποιήσαμε για να μειώσουμε το άνω όριο $\pi(v)$.

Αλγόριθμος Bellman-Ford

Αλγόριθμος Bellman-Ford

Input: Κατευθυνόμενο γράφημα $G(V, E)$ με μήκη ακμών $\{c_e : e \in C\}$ χωρίς αρνητικούς κύκλους και κορυφή $s \in V$.
Output: Για κάθε κορυφή $v \in V$ προσπελάσιμη από την s, η $\pi(v)$ είναι ίση με την απόσταση από την s στη v.

for *κάθε* $v \in V$ **do**
  $\pi(v) = \infty$
end
$\pi(s) = 0$
for $i = 1$ *έως* $n - 1$ **do**
  for *κάθε ακμή* $e = (u, v) \in E$ **do**
    $\pi(v) = min\{\pi(v), \pi(u) + c_e\}$
  end
end

Μπορούμε επίσης να ανιχνεύσουμε αρνητικούς κύκλους αν εκτελέσουμε τον βρόγχο μια ακόμη φορά και γίνει κάποια χαλάρωση.

Ο αλγόριθμος τρέχει σε χρόνο $\mathcal{O}(nm)$.

Συντομότερες Διαδρομές

Μεταξύ Όλων των Ζευγών Κόμβων

Πρόβλημα. Θέλουμε να βρούμε τις συντομότερες διαδρομές μεταξύ κάθε ζεύγους κορυφών ενός γραφήματος (all pairs shortest paths).

Συντομότερες Διαδρομές

Μεταξύ Όλων των Ζευγών Κόμβων

Πρόβλημα. Θέλουμε να βρούμε τις συντομότερες διαδρομές μεταξύ κάθε ζεύγους κορυφών ενός γραφήματος (all pairs shortest paths).

Απλή Λύση. Μπορούμε να εκτελέσουμε τον αλγόριθμο Bellman-Ford (αφού μπορεί να υπάρχουν αρνητικές ακμές) από μια φορά για κάθε κόμβο. Αυτό οδηγεί σε έναν $\mathcal{O}(n^2 m)$ αλγόριθμο.

Συντομότερες Διαδρομές

Μεταξύ Όλων των Ζευγών Κόμβων

Πρόβλημα. Θέλουμε να βρούμε τις συντομότερες διαδρομές μεταξύ κάθε ζεύγους κορυφών ενός γραφήματος (all pairs shortest paths).

Απλή Λύση. Μπορούμε να εκτελέσουμε τον αλγόριθμο Bellman-Ford (αφού μπορεί να υπάρχουν αρνητικές ακμές) από μια φορά για κάθε κόμβο. Αυτό οδηγεί σε έναν $\mathcal{O}(n^2 m)$ αλγόριθμο.

Μπορούμε καλύτερα;

Συντομότερες Διαδρομές

Μεταξύ Όλων των Ζευγών Κόμβων

Πρόβλημα. Θέλουμε να βρούμε τις συντομότερες διαδρομές μεταξύ κάθε ζεύγους κορυφών ενός γραφήματος (all pairs shortest paths).

Απλή Λύση. Μπορούμε να εκτελέσουμε τον αλγόριθμο Bellman-Ford (αφού μπορεί να υπάρχουν αρνητικές ακμές) από μια φορά για κάθε κόμβο. Αυτό οδηγεί σε έναν $\mathcal{O}(n^2 m)$ αλγόριθμο.

Μπορούμε καλύτερα;

θα δείξουμε έναν αλγόριθμο δυναμικού προγραμματισμού που πετυχαίνει χρόνο $\mathcal{O}(n^3)$.

Αλγόριθμος Floyd-Warshall

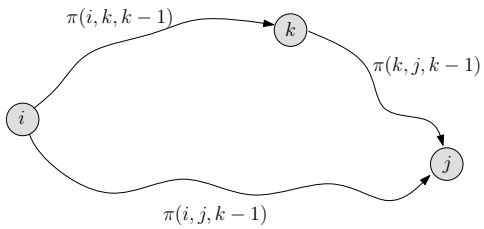
Θα υπολογίσουμε τις συντομότερες διαδρομές με βάση τον αριθμό των ενδιάμεσων κόμβων που χρησιμοποιούν.

- Ξεκινάμε με όλες τις διαδρομές που δεν χρησιμοποιούν κανένα ενδιάμεσο κόμβο.
- Διευρύνουμε σιγά σιγά κατά ένα κόμβο τη φορά, ενημερώνοντας τα μήκη των ενδιάμεσων διαδρομών σε κάθε στάδιο.

Αλγόριθμος Floyd-Warshall

Υποπροβλήματα. Έστω $V = 1, 2, \ldots, n$ και έστω $\pi(i, j, k)$ το μήκος της συντομότερης διαδρομής από τον κόμβο i στον j χρησιμοποιώντας ως ενδιάμεσους κόμβους μόνο τους $\{1, 2, \ldots, k\}$.

Αρχικές Συνθήκες. $\pi(i, j, 0)$ είναι το μήκος της ακμής (i, j), αν υπάρχει, αλλιώς είναι ∞.



Επέκταση Ενδιάμεσου Συνόλου.

$$\pi(i, j, k) = min\{\pi(i, k, k-1) + \pi(k, j, k-1), \pi(i, j, k-1)\}$$

Αλγόριθμος Floyd-Warshall

Input: Κατευθυνόμενο γράφημα $G(V, E)$ με μήκη ακμών $\{c_e : e \in C\}$ χωρίς αρνητικούς κύκλους.
Output: Για κάθε ζεύγος κορυφών $v, u \in V$ η $\pi(v, u, n)$ είναι ίση με την απόσταση από την κορυφή v στην u.

for $i = 1$ *έως* n **do**
  for $j = 1$ *έως* n **do**
    $\pi(i, j, 0) = \infty$
  end
end
for *κάθε ακμή* $e = (i, j) \in E$ **do**
  $\pi(i, j, 0) = c_e$
end
for $k = 1$ *έως* n **do**
  for $i = 1$ *έως* n **do**
    for $j = 1$ *έως* n **do**
      $\pi(i, j, k) = min\{\pi(i, k, k-1) + \pi(k, j, k-1), \pi(i, j, k-1)\}$
    end
  end
end

Αλγόριθμος Floyd-Warshall

Input: Κατευθυνόμενο γράφημα $G(V, E)$ με μήκη ακμών $\{c_e : e \in C\}$ χωρίς αρνητικούς κύκλους.
Output: Για κάθε ζεύγος κορυφών $v, u \in V$ η $\pi(v, u, n)$ είναι ίση με την απόσταση από την κορυφή v στην u.

for $i = 1$ *έως* n **do**
  for $j = 1$ *έως* n **do**
    $\pi(i, j, 0) = \infty$
  end
end
for *κάθε ακμή* $e = (i, j) \in E$ **do**
  $\pi(i, j, 0) = c_e$
end
for $k = 1$ *έως* n **do**
  for $i = 1$ *έως* n **do**
    for $j = 1$ *έως* n **do**
      $\pi(i, j, k) = min\{\pi(i, k, k-1) + \pi(k, j, k-1), \pi(i, j, k-1)\}$
    end
  end
end

Χρόνος εκτέλεσης $\mathcal{O}(n^3)$.