Αλγόριθμοι!

**Κάνε κλικ σε ένα από τα παρακάτω για να μεταφερθείς κατευθείαν στο section που θες!*

Ασυμπτωτικός Συμβολισμός **3**
- Πολυπλοκότητες συναρτήσεων 3
- Διάταξη συναρτήσεων 4
- Κάποια Αθροίσματα 5
- Αναδρομικές σχέσεις 5

Δομές Δεδομένων **7**
- Δέντρα 8
- Σωρός (Heap) 8

Union-Find **9**

Lowest Common Ancestor **9**

Αλγόριθμοι Ταξινόμησης **9**
- Quicksort 9
- QuickSelect 10
- Counting sort 10
- Radix sort 10
- Bucket sort 10

Αλγόριθμοι Αναζήτησης **10**
- Γραμμική Αναζήτηση 10
- Δυαδική αναζήτηση 10
- Αναζήτηση με παρεμβολή 10

Άπληστοι Αλγόριθμοι **11**
- Επιλογή δραστηριοτήτων 11
- Χρωματισμός Διαστημάτων 11

Δυναμικός Προγραμματισμός **11**

Γραφήματα **12**
- BFS - DFS 12

NP Complete Προβλήματα **13**
- List of NP complete problems 13
- Λίστα Αναγωγών 14

Σύνοψη αλγορίθμων **14**
- Sorting Algorithms 14
- Shortest Path 14

Minimum Spanning Tree 15
Flow Network 16
Kosaraju 16

Ερωτήσεις Σωστού-Λάθους **17**
Κανονική 2015-16 17
Κανονική 2017-18 17
Επί πτυχίω 2020 18
Κανονική 2020 19
Κανονική 2021 20

Ασυμπτωτικός Συμβολισμός

Πολυπλοκότητες συναρτήσεων

- $f(n) = \Theta(g(n))$ ~ ασυμπτωτικά $f(n) = g(n)$
  $f(n) = O(g(n))$ ~ ασυμπτωτικά $f(n) \leq g(n)$
  $f(n) = o(g(n))$ ~ ασυμπτωτικά $f(n) < g(n)$
  $f(n) = \Omega(g(n))$ ~ ασυμπτωτικά $f(n) \geq g(n)$
  $f(n) = \omega(g(n))$ ~ ασυμπτωτικά $f(n) > g(n)$

$$O(1) \subset O(\log^* n) \subset O(\log n) \subset O(\text{poly}(\log n)) \subset$$
$$\subset O(\sqrt{n}) \subset O(n) \subset O(n \log n) \subset O(\text{poly}(n)) \subset$$
$$\subset O(n^{\log n}) \subset O(2^n) \subset O(3^n) \subset O(n!) \subset O(n^n) \subset O(A(n))$$

- n^(1/logn) = Θ(1)
- log(n!) = Θ(nlogn)
- n/log2(base n) + n = Θ(nlogn)
- (logn)^logn = Θ(n^loglogn)
- 2^((logn)^3) = Θ(n^(logn)^2)
- (n 4) διωνυμικός συντελεστής = n! / (4! * (n - 4)!) = n*(n-1)*(n-2)*(n-3)/4! = Θ(n^4)

Έχουμε πως:

$$\sum_{k=1}^{n} k2^{-k} = \Theta(1)$$

διότι $\sum_{k=1}^{n} k2^{-k} \leq \sum_{k=1}^{\infty} k2^{-k} = 2$.

Έπειτα, είναι: $n2^{2^{2^{100}}} = \Theta(n)$, όπως και $\log(\binom{2n}{n}) = \Theta(n)$, διότι

$$2^n \leq \binom{2n}{n} \leq (2e)^n$$

Έχουμε $2^{(\log_2 \log_2 n)^4} = \Theta((\log_2 n)^{(\log_2 n)^3})$.

Επίσης, αφού $\log(n!) = \Theta(n \log n)$, είναι: $(\sqrt{n})^{\log_2 \log_2 (n!)} = O((\sqrt{n})^{\log_2 n})$.

Είναι: $n \sum_{k=0}^{n} \binom{n}{k} = \Theta(n2^n) = \sum_{k=1}^{n} k2^k$.

Απομένει η $\frac{\log(n!)}{(\log \log n)^5} = \frac{n \log(n)}{(\log \log n)^5}$

Άρα, για να τα συγκρίνω, λόγω μονοτονίας, αρκεί να ελέγχω τους λογαρίθμους τους.

Διάταξη συναρτήσεων

από σημειώσεις/ασκήσεις

$$n^{1/\log n} = \Theta(1),\ \log\log n,\ \log^4 n,\ (\log n)^{100}\log\log n,$$
$$n^{0.1}\log\log n,\ n^{0.6},$$
$$n\log\log n,\ \log(n!) = \Theta(n\log n),\ \frac{n}{\log_n 2} + n = \Theta(n\log n),$$
$$(\log n)^{\log n} = \Theta(n^{\log\log n}),\ n^{\log n},\ 2^{\log^3 n} = \Theta(n^{\log^2 n}),$$
$$2^n,\ 2^n + n^{2^{100}} = \Theta(2^n),\ 2^{5n}$$

1. $\sum_{k=1}^{n} k2^{-k}$
2. $\frac{\log(n!)}{(\log n)^3}$
3. $\log(\binom{2n}{n})$, $n2^{2^{2^{100}}} = \Theta(n)$
4. $2^{(\log n)^4} = \Theta(\ n^{lon^3 n}\ )$
5. $\sum_{k=1}^{n} k2^k$, $n\sum_{k=1}^{n}\binom{n}{k} = \Theta(\ n2^n\ )$
6. $(\sqrt{n})!$

α) Για τις δοθείσες συναρτήσεις έχουμε:

1. $\sum_{k=1}^{n} k2^{-k} = 2^{-n}(-n + 2^{n+1} - 2) = 2 - \frac{n+2}{2^n} = O(1)$
2. $\frac{\log^2 n}{\log\log n}$
3. $\log^4 n$
4. $\log\binom{n}{\log n}$
5. $\frac{\log(n!)}{\log^3 n}$
6. $\log\binom{2n}{n} = O(n) = n \times 2^{2^{2^{100}}}$
7. n^2
8. $\frac{n^3}{\log^8 n}$
9. $\binom{n}{6} = \frac{n\times(n-1)\times(n-2)\times(n-3)\times(n-4)\times(n-5)}{6!} = O(n^6)$
10. $\log_2 n^{\log_2 n}$
11. $\sqrt{n}^{\log\log n!} = O(n^{\log(n\log n)})$
12. $2^{\log_2^4 n} = O(n^{\log_2^3 n})$
13. $\sum_{k=1}^{n} k2^k = O(n2^n) = n\sum_{k=1}^{n}\binom{n}{k}$
14. $\sqrt{n}!$

Κάποια Αθροίσματα

$$\sum_{i=1}^{n} i = \Theta(n^2),\ \sum_{i=1}^{n} i^2 = \Theta(n^3), \ldots, \sum_{i=1}^{n} i^k = \Theta(n^{k+1})$$

$$\sum_{i=1}^{n} 1/i = \Theta(\log n),\ \sum_{i=1}^{n} 2^i = \Theta(2^n)$$

- $\sum_{k=0}^{n-2} \frac{1}{n-k} = \Theta(logn)$

Αναδρομικές σχέσεις

[Master Theorem Solver Online]

- **T(n) = 2T(n/3) + 29n**, T(1) = 7 → Θ(n)

Λύση. (α) Εργαζόμαστε με την μέθοδο της επανάληψης:

$$\begin{aligned} T(n) &= 2\,T(n/3) + 29n \\ &= 2(2\,T(n/3^2) + 29n/3) + 29n \\ &= 2(2(2\,T(n/3^3) + 29n/3^2) + 29n/3) + 29n = \dots \\ &= 2^{i+1}\,T(n/3^{i+1}) + 29n \sum_{k=0}^{i} (2/3)^k \\ &= 2^{i+1}\,T(n/3^{i+1}) + 87n(1 - (2/3)^{i+1}) \end{aligned}$$

Για $i = \log_3 n - 1$, έχουμε ότι $n/3^{i+1} = 1$, και:

$$\begin{aligned} T(n) &= 2^{\log_3 n}\,T(1) + 87n(1 - n^{\log_3 2/3}) \\ &= 7n^{\log_3 2} + 87n(1 - n^{-1+\log_3 2}) \\ &= 7n^{\log_3 2} + 87n - 87n^{\log_3 2} \\ &= 87n - 80n^{\log_3 2} = \Theta(n) \end{aligned}$$

- **T(n) = 5T(n/5) + n/logn(base 5)**, T(1) = Θ(1)
  - Δεν μπορούμε να χρησιμοποιήσουμε Master Theorem γιατί η n/logn είναι ασυμπτωτικά μικρότερη της n αλλά όχι πολυωνυμικά!
  - T(n) = Θ(n*loglogn)
- **T(n) = 2T(n/4) + n^2 * sqrt(n)**, T(1) = Θ(1)
  - Master Theorem → T(n) = Θ(n^(5/2))
- **T(n) = T(n-1) + 1/n**, T(1) = Θ(1)

$$\begin{aligned} T_3(n) &= T_3(n-1) + \frac{1}{n} \\ &= T_3(n-2) + \frac{1}{n-1} + \frac{1}{n} = \dots \\ &= T_3(1) + \sum_{k=0}^{n-2} \frac{1}{n-k} \\ &= \Theta(\log n) \end{aligned}$$

  - T(n) = Θ(logn)
- **T(n) = 3T(n/2) + n^2 * logn**
  - 3η περίπτωση Master Theorem → T(n) = Θ(n^2 * logn)
- **T(n) = 4T(n/2) + n^2 * logn**
  - Δέντρο αναδρομής → T(n) = Θ(n^2 * (logn)^2)
- **T(n) = 5T(n/2) + n^2 * logn**
  - 1η περίπτωση Master Theorem → T(n) = Θ(n^(log5))
- **T(n) = T(n/2) + T(n/3) + n**
  - Έχουμε ½ + ⅓ < 1 → T(n) = Θ(n)
- **T(n) = T(n/2) + T(n/3) + T(n/6) + n**
  - Δέντρο αναδρομής → T(n) = Θ(nlogn)
- **T(n) = T(n/4) + sqrt(n)**

- 3η περίπτωση Master Theorem → T(n) = Θ(sqrt(n))

- **T(n) = 2T(n/3) + nlogn**
  - 3η περίπτωση Master Theorem → T(n) = Θ(nlogn)
- **T(n) = 3T(n/3) + nlogn**
  - Δέντρο αναδρομής → T(n) = Θ(n*(logn)^2)
- **T(n) = 4T(n/3) + nlogn**
  - 1η περίπτωση Master Theorem → T(n) = $\Theta(n^{log_3 4})$
- **T(n) = T (n^(⅝))+ Θ(logn)**
  - Δέντρο αναδρομής → T(n) = Θ(log n)
- **T(n) = 6T(n/3) + n^2 * logn**
  - Master Theorem → Θ(n^2 * logn)
- **T(n) = 2T(n/2) + n * (logn)^3**
  - T(n) = Θ(n * (logn)^4) και Ω(n * (logn)^3)
- **T(n) = 2T(n/8) + n^(⅓) , T(1) = Θ(1)**
  - T(n) = Θ(n^(⅓)*logn)
- **T(n) = 3T(n/3) + n*(logn)^5 , T(1) = Θ(1)**
  - T(n) = Ω(n*(logn)^5)
- **T(n) = 4 T(n/2)+Θ(n), T(1) = Θ(1)**
  - Master Theorem → Θ(n^2)
- **T(n) = T(sqrt(n)) + Θ(1), T(1) = Θ(1)**
  - Δέντρο αναδρομής → Θ(loglogn)
- **T(n) = 2T(n/2) + sqrt(n)**
  - Master Theorem → Θ(n)
- **T(n) = T(sqrt(n))+(logn)^2**

$T(n) = 9T(n/3) + n^2 \log n = \Theta(n^2 \log^2 n)$ (Όχι ΜΤ, όχι πολυωνυμικά διαχωρίσιμες. Επειδή $n^{\log_3 9} = n^2$ και $f(n) = n^2 \log n$, υποπτευόμαστε $T(n) = \Theta(n^2 \log^2 n)$ και το αποδεικνύουμε με επαγωγή ή χρησιμοποιώντας το δέντρο αναδρομής)

$T(n) = 11T(n/3) + n^2 \log n = \Theta(n^{\log_3 11})$ (ΜΤ)

$T(n) = T(n/4) + T(n/2) + n = \Theta(n)$ (Επειδή $n/4 + n/2 < n$, υποπτευόμαστε $T(n) = \Theta(n)$ και το αποδεικνύουμε με επαγωγή ή χρησιμοποιώντας το δέντρο αναδρομής)

$T(n) = 2T(n/4) + T(n/2) + n = \Theta(n \log n)$ (Επειδή $2n/4 + n/2 = n$, υποπτευόμαστε $T(n) = \Theta(n \log n)$ και το αποδεικνύουμε με επαγωγή ή χρησιμοποιώντας το δέντρο αναδρομής)

$T(n) = T(n^{2/3}) + \Theta(\log n)$ (Είναι $T(n) \leq \Theta(\log n) \sum_{k=0}^{+\infty} (2/3)^k = \Theta(\log n)$)

$T(n) = T(n/3) + \sqrt{n} = \Theta(\sqrt{n})$ (Είναι $T(n) \leq \sqrt{n} \sum_{k=0}^{+\infty} (1/\sqrt{3})^k = \Theta(\sqrt{n})$)

Δομές Δεδομένων

Υλοποιήσεις Λεξικού

- Μη ταξινομημένη **διασυνδεδεμένη λίστα** (συχνές εισαγωγές, σπάνιες αναζητήσεις)
    - Εισαγωγή: O(1)
    - Αναζήτηση/Τυχαία Διαγραφή: O(n)
- Ταξινομημένος **πίνακας** (συχνές αναζητήσεις, σπάνιες αλλαγές πχ λεξικό)
    - Αναζήτηση (δυαδική): O(logn)
    - Ταξινόμηση: O(nlogn)
- Δυαδικά Δέντρα Αναζήτησης
    - Αναζήτηση/Εισαγωγή/Διαγραφή: O(logn)
    - Μέγιστο/ελάχιστο/k-οστό: O(logn)
- Πίνακας Κατακερματισμού
    - Αναζήτηση/διαγραφή: O(1)
    - Εισαγωγή: O(1), O(logn)

Ουρά Προτεραιότητας
- Υλοποίηση με γραμμική λίστα
    - Διαγραφή μέγιστου ή εισαγωγή: γραμμικός χρόνος O(n)
- Υλοποίηση με σωρό

Δέντρα
- n κορυφές, m=n-1 ακμές

Δυαδικό δέντρο: κάθε κορυφή <= 2 παιδιά
- h+1 <= **#κορυφών** <= 2^(h+1)-1, h: ύψος δέντρου
- log(n+1) base2 - 1 <= **ύψος** <= n-1
- **Πλήρες**: Όλα τα επίπεδα συμπληρωμένα εκτός ίσως από τελευταίο που γεμίζει από αριστερά προς τα δεξιά.
- Αναπαράσταση με πίνακα
    - Ρίζα : Π[1]
    - Π[i] : πατέρας Π[i/2], αριστερό παιδί Π[2i], δεξιό παιδί Π[2i +1]

Σωρός (Heap)
- Δέντρο μέγιστου (ελάχιστου): Τιμή κάθε κορυφής ≥ (≤) τιμές παιδιών της.
    - ∀ i A[i] ≥ A[2i], A[2i+1]
- Ύψος Θ(log n) , #φύλλων = n / 2
- Χώρος: Θ(1), in-place
- Χρόνοι:
    - createHeap : Θ(n)
    - insert, deleteMax : O(log n)
    - max, size, isEmpty : Θ(1)
- Χρονική Πολυπλοκότητα Ταξινόμησης: O(n log n)
- **Εύρεση ελαχίστου από σωρό μεγίστου**: The only guarantee you have, is that each node contains the maximum element of the subtree below it. In other words, the minimum element can be any leaf in the tree. → **O(n)**. In each step you need to traverse

both left and right subtrees in order to search for the minimum element. In effect, this means that you need to traverse all elements to find the minimum.

Union-Find

- Χρόνος χ.π. για m finds και n unions: O(m*n + n)
    - Union : O(1) χρόνος
    - Find : O(ύψος δέντρου)
        - Χειρότερη περίπτωση: ύψος = n – 1

- Βεβαρυμένη Ένωση

A low-cost approach to reducing the height is to be smart about how two trees are joined together. One simple technique, called the weighted union rule, joins the tree with fewer nodes to the tree with more nodes by making the smaller tree's root point to the root of the bigger tree. This will limit the **total depth of the tree to O(logn)**, because the depth of nodes only in the smaller tree will now increase by one, and the depth of the deepest node in the combined tree can only be at most one deeper than the deepest node before the trees were combined. The **total number of nodes in the combined tree is therefore at least twice the number in the smaller subtree**. Thus, the depth of any node can be increased at most logn times when n equivalences are processed (since each addition to the depth must be accompanied by at least doubling the size of the tree).

Χρόνος χ.π. για m finds και n unions: O(m log n + n)

- Σύμπτυξη Μονοπατιών

The weighted union rule helps to minimize the depth of the tree, but we can do better than this. Path compression is a method that tends to create extremely shallow trees.

Lowest Common Ancestor

- Using Sqrt-Decomposition, it is possible to obtain a solution answering each query in O(sqrtN) with preprocessing in O(N) time.
- Using a Segment Tree you can answer each query in O(logN) with preprocessing in O(N) time.
- Since there will almost never be any update to the stored values, a Sparse Table might be a better choice, allowing O(1) query answering with O(NlogN) build time.

Αλγόριθμοι Ταξινόμησης

Quicksort

Γρηγορότερος αλγόριθμος στην πράξη (για n ≥ 30)

Partition

- Χρόνος εκτέλεσης partition για n στοιχεία = Θ(n).

QuickSelect

- Πιθανοτικός αλγόριθμος με γραμμικό χρόνο (μ.π.)
- Ντετερμινιστικός αλγόριθμος με γραμμικό χρόνο (χ.π.)
- Χειρότερη περίπτωση: O(n^2)
    - This occurs for example in searching for the maximum element of a set, using the first element as the pivot, and having sorted data.

Counting sort

- https://www.geeksforgeeks.org/counting-sort/
- Time Complexity: O(n+k) where n is the number of elements in the input array and k is the range of input.

Radix sort

- https://www.geeksforgeeks.org/radix-sort/
- Ταξινομούμε πρώτα με βάση το τελευταίο ψηφίο, μετά με το αμέσως επόμενο, κοκ.
- O(d*(n+b)) time where b is the base for representing numbers, for example, for the decimal system b is 10. What is the value of d? If k is the maximum possible value, then d would be $O(\log_b(k))$.
    - O(n*logM), M = ο μέγιστος αριθμός που έχουμε

Bucket sort

Αλγόριθμοι Αναζήτησης

Γραμμική Αναζήτηση

- Χρόνος χ.π. / αποτυχημένης αναζήτησης: Θ(n) (βέλτιστος)
- Χρόνος καλύτερης περίπτωσης: Θ(1)

Δυαδική αναζήτηση

- Χρόνος O(log n) (βέλτιστος ως προς χ.π.)
-

Αναζήτηση με παρεμβολή

- Χρόνος μέσης περίπτωσης: O(loglogn)
- Χρόνος χειρότερης περίπτωσης: O(n)
- Χρόνος χειρότερης περίπτωσης βελτιώνεται με Δυαδική Αναζήτηση με Παρεμβολή

Άπληστοι Αλγόριθμοι

Επιλογή δραστηριοτήτων

- n δραστηριότητες: αρχή και τέλος $[s_i, f_i)$
- Επιλογή δραστηριοτήτων χωρίς χρονικές επικαλύψεις και δρομολόγηση σε κοινό πόρο.
- Ζητούμενο: δρομολόγηση μέγιστου #δραστηριοτήτων.
- Κριτήριο άπληστης επιλογής: **Ελάχιστος χρόνος ολοκλήρωσης.**
- **Αλγόριθμος**:
    - Ταξινόμηση σε αύξουσα σειρά χρόνου ολοκλήρωσης.
    - Επόμενη δραστηριότητα:
        - Δρομολογείται αν είναι εφικτό (πόρος είναι ελεύθερος).
        - Αγνοείται αν δρομολόγηση δεν είναι εφικτή.
- Χρόνος O(n log n)

Χρωματισμός Διαστημάτων

- Χρωματισμός όλων ώστε επικαλυπτόμενα διαστήματα να έχουν διαφορετικό χρώμα. †
- Ζητούμενο: χρωματισμός με ελάχιστο #χρωμάτων.
- Άπληστος αλγόριθμος:
    - Ταξινόμηση με χρόνο έναρξης.
    - Κάθε διάστημα που αρχίζει παίρνει πρώτο διαθέσιμο χρώμα.
    - Κάθε διάστημα που τελειώνει «απελευθερώνει» το χρώμα του.

Δυναμικός Προγραμματισμός

Βασική ιδέα:
Υπολογίζουμε κάθε λύση μία φορά και την αποθηκεύουμε σε έναν πίνακα που συμπληρώνεται καθώς επιλύουμε όλο και μεγαλύτερα υπό-στιγμιότυπα. Ο δυναμικός προγραμματισμός εφαρμόζεται bottom-up. Ξεκινά με την επίλυση των στοιχειωδών στιγμιοτύπων και συνεχίζει με όλο και μεγαλύτερα στιγμιότυπα, των οποίων τις λύσεις συνθέτει από τις λύσεις που έχει ήδη υπολογίσει. Η μέθοδος τερματίζει όταν υπολογίσει τη λύση του αρχικού στιγμιότυπου.

- Οι αλγόριθμοι δυναμικού προγραμματισμού επιλύουν μία φορά κάθε υπο-στιγμιότυπο και **αποθηκεύουν τις λύσεις σε έναν πίνακα για μελλοντική χρήση.** → Απαιτητικοί σε χώρο.
- Ο χρόνος εκτέλεσης του αλγόριθμου δυναμικού προγραμματισμού **δεν είναι (και δεν πρέπει) να χαρακτηριστεί πολυωνυμικός στο μέγεθος του στιγμιότυπου εισόδου**.

- Πρόβλημα Περιοδεύοντος/Πλανόδιου Πωλητή (Travelling Salesman Problem) - NP hard
  - Εξαντλητική αναζήτηση: Ο χρόνος εκτέλεσης και ο χώρος αποθήκευσης αυξάνονται εκθετικά με τον αριθμό των σημείων. Αυτός ο αλγόριθμος μπορεί να χρησιμοποιηθεί στην πράξη μόνο για στιγμιότυπα με μικρό αριθμό σημείων (n<=20).
  - Ο αλγόριθμος δυναμικού προγραμματισμού είναι δραματικά ταχύτερος.
- Το Πρόβλημα του Σακιδίου (Knapsack Problem) - NP hard
  - Ψευδο-πολυωνυμικός χρόνος

Γραφήματα

BFS - DFS

Υλοποίηση

- Πίνακας κατάστασης: **m[v]** = { A, Y, E }.
- Πίνακας γονέων: **p[v]** = πατέρας v στο BFS-δάσος.
- Χρόνος εκτέλεσης **Θ(n + m)**.

```
BFS(G(V, E), s)
    addToQueue(s); m[s] ← Y; p[s] ← NULL;
    for all v ∈ V \ {s} do
        m[v] ← A; p[v] ← NULL;
    while not emptyQueue() do
        u ← extractFromQueue(); m[u] ← E;
        for all v ∈ L[u] do
            if m[v] = A then
                addToQueue(v); m[v] ← Y; p[v] ← u;
```

Υλοποίηση

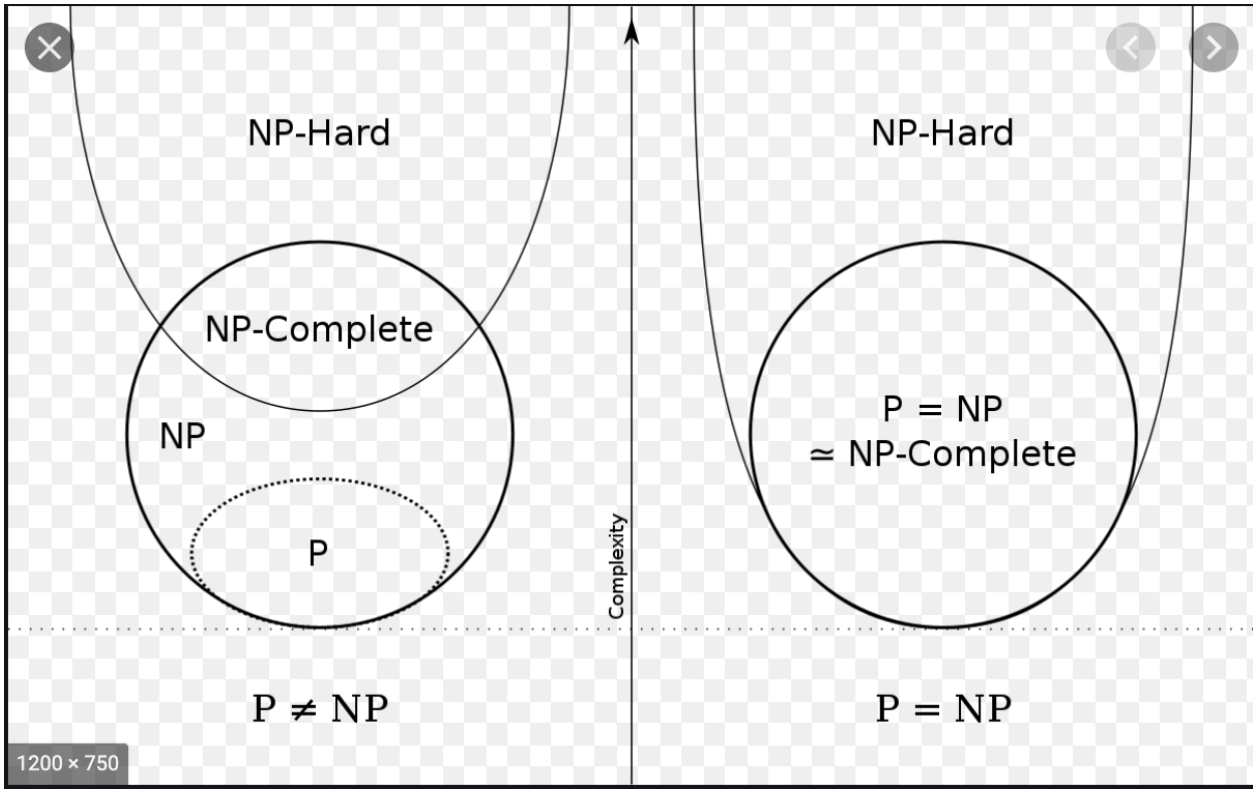
- Πίνακας κατάστασης: **m[v]** = { A, Y, E }.
- Πίνακας γονέων: **p[v]** = πατέρας v στο DFS-δάσος.
- «Χρόνοι» πρώτης επίσκεψης **d[v]** και αναχώρησης **f[v]**.
- Χρόνος εκτέλεσης **Θ(n + m)**.
- DFS σε (α) δέντρο, (β) πλήρες γράφημα, (γ) κύκλο.

```
DFS_Init(G(V, E))
    t ← 0;
    for all v ∈ V do
        m[v] ← A; p[v] ← NULL;
    for all v ∈ V do
        if m[v] = A then DFS(v);
```

```
DFS(v)
    m[v] ← Y; d[v] ← ++t;
    for all u ∈ L[v] do
        if m[u] = A then
            p[u] ← v; DFS(u);
    m[v] ← E; f[v] ← ++t;
```

NP Complete Προβλήματα

[List of NP complete problems]



Λίστα Αναγωγών

1. MAXSAT γενίκευση SAT
2. Hamilton Path γενίκευση Hamilton Cycle
3. Minimum Leaf Spanning Tree και Longest Path γενίκευση Hamilton Cycle
4. Dense Subgraph γενίκευση Clique
5. Feedback Vertex Set γενίκευση Vertex Cover
6. Vertex Cover αναγωγή σε Dominating Set
7. Vertex Cover αναγωγή σε Δέντρο Steiner
8. Dominating Set αναγωγή σε facility location (χωροθέτηση υπηρεσιών)

Σύνοψη αλγορίθμων

Sorting Algorithms

Αλγόριθμος	Σχόλια	Χειρότερη Πολυπλοκότητα	Καλύτερη Πολυπλοκότητα	Χωρική Πολυπλοκότητα
QuickSort	Pivot, Partition, Sorting, τυχαιότητα	O(n^2)	Ω(nlogn)	O(logn)
MergeSort	Διαίρεση στη μέση, ταξινόμηση, συγχώνευση	O(nlogn)	Ω(nlogn)	O(n)
HeapSort		O(nlogn)	Ω(nlogn)	O(1)
BubbleSort		O(n^2)	Ω(n^2)	O(1)
(binary)Insertion Sort		O(n^2)	Ω(n^2)	O(1)
SelectionSortF		O(n^2)	Ω(n^2)	O(1)
CountingSort	Δεν είναι συγκριτικός αλγόριθμος, είσοδος nlogk	O(n+k)	Ω(n+k)	O(k)

Shortest Path

Dijkstra	Shortest path from S to the other nodes.	List: Θ(V^2logV)		

	Θετικά μήκη, βάζουμε σε όλα άπειρο αρχικά και παίρνουμε κάθε φορά το μικρότερο από τους κόμβους που δεν έχουμε επισκεφτεί και ανανεώνουμε τις διαδρομές.	Binary Heap: Θ(ElogV)		
		Fibonacci Heap: Θ(E+VlogV)		
Bellman Ford	Shortest Path από μία κορυφή. Ενδεχομένως αρνητικά μήκη. **V-1 iterations at most** Τσεκάρουμε έναν-έναν κόμβο και ανανεώνουμε με βάση τις ακμές που ξεκινούν από αυτούς.	O(VE)	Ω(E)	Θ(V)
Johnson Dijkstra	Shortest Path μεταξύ όλων, αρνητικά μήκη	O(EV + V^2logV)		
Floyd Warshall	Shortest Path μεταξύ όλων, αρνητικά μήκη	Θ(V^3)		O(V^2)
Ν Φορές Dijkstra	Shortest Path μεταξύ όλων, θετικά μήκη	Θ(VE + V^2logV)		
Johnson	Μετατροπή μήκους σε θετικά	O(V^2log V + VE)		

Minimum Spanning Tree

	Κρατάμε λίστα με visited nodes, προσθέτουμε έναν κόμβο στη λίστα και εξετάζουμε τις ακμές του, διαλέγουμε τη μικρότερη. Κάθε φορά κοιτάμε τις ακμές από όλους τους κόμβους που έχουμε στη visited list.			
	adjacency matrix, searching	O(|V|^{2})		

Prim	binary heap and adjacency list	O((|V|+|E|)\log |V|)=O(|E|\log |V|)		
	Fibonacci heap and adjacency list	O(|E|+|V|\log |V|)		
Kruskal	Διαλέγουμε κάθε φορά την μικρότερη ακμή που δεν δημιουργεί κύκλο.	O(E log V)		Θ(E+V)
Boruvka		O(E log V)		

Flow Network

Ford Fulkerson	Maximum Flow Διαλέγουμε ένα path, υπολογίζουμε την μέγιστη χωρητικότητα που μπορεί να περάσει από όλες τις ακμές του μονοπατιού. Κάθε φορά προσθέτουμε στο total flow.	O(max_flow * E)		
Edmonds Karp	Όταν όλα τα βάρη είναι ίδια	O(VE^2)		

Kosaraju

- is a linear time algorithm to find the strongly connected components of a directed graph.
- Provided the graph is described using an **adjacency list,** Kosaraju's algorithm performs two complete traversals of the graph and so runs in **Θ(V+E)** (linear) time.

Ερωτήσεις Σωστού-Λάθους

**Οι παρακάτω προτάσεις είναι αληθείς εκτός κι αν διευκρινίζεται το αντίθετο.*

Κανονική 2015-16

- Το max flow (ή αλλιώς, πρόβλημα μέγιστης ροής) διατυπωμένο ως πρόβλημα απόφασης ανήκει στην κλάση NP, επειδή επαληθεύεται σε γραμμικο χρονο.
- Στο max flow αν αυξήσω τη χωρητικότητα 2 ακμών της ελάχιστης τομής κατά k την καθεμία **δεν** αυξάνεται η μέγιστη ροή κατά 2k.
    - Αυτό συμβαίνει επειδή η ελάχιστη τομή αλλάζει με τις νέες χωρητικότητες και θα έχω μια νέα ελάχιστη τομή (κάπου αλλού στο γράφο) η οποία δεν είναι αναγκαστικό πως θα είναι 2k μεγαλύτερη.
- Ο γρηγορότερος συγκριτικός αλγόριθμος για κατασκευή σωρού είναι O(n).
- Το DFS με n κορυφές και n^3/2 ακμές θέλει O(n^2) αν αναπαρίσταται με πίνακα γειτνίασης.
- Στο DFS αν έχω λίστα γειτνίασης θα ήθελα O(κορυφες + ακμες) = O(n + n^3/2) = O(n ^ 3/2)
- Κατά την εκτέλεση του αλγορίθμου των Ford-Fulkerson είναι πιθανό σε κάποιο βήμα η ροή κάποιας ακμής να μειωθεί. Αυτό συμβαίνει επειδή υπάρχουν πίσω ακμές.
- Στον Kruskal αν έχω ταξινομημένες τις ακμές σε φθίνουσα σειρά, βρίσκω το μέγιστο συνδετικό δέντρο.
- Στον Dijkstra αν διαλέγω κάθε φορά αντί για το μικρότερο το μεγαλύτερο **ΔΕΝ** βρίσκω longest paths.

Κανονική 2017-18

- Αν P≠NP τότε κανένα πρόβλημα στην κλάση NP δεν λύνεται σε πολυωνυμικό χρόνο → **ΛΑΘΟΣ**
    - An P≠NP τότε τα NP-hard προβλήματα δεν μπορούν να λυθούν σε πολυωνυμικό χρόνο.
- Το πρόβλημα της εύρεσης κύκλου Euler σε γράφο ανήκει στην κλάση NP. Πιο συγκεκριμένα ανήκει στο P και το P είναι υποσύνολο του NP.
    - Εδω: https://math.stackexchange.com/questions/3152801/is-hamiltoncycle-and-euler-cycle-np-complete-or-not λέει ότι: “It is even possible to find an Eulerian path in linear time (in the number of edges). Note that there are at most $n2$ edges in any simple undirected graph (where n is the number of nodes). You can try to find a polynomial algorithm yourself or just look it up on wikipedia. Hint to find the algorithm: Be Greedy.”
      Αρα είναι P και σύμφωνα και με το σχήμα που έχουμε και πιο πάνω P υποσύνολο του NP , άρα σωστό.
- Η βέλτιστη λύση στην διακριτή εκδοχή του προβλήματος του σακιδίου (knapsack) **δεν** περιέχει πάντα το αντικείμενο με το μέγιστο λόγο αξίας προς μέγεθος. Στη λύση του προβλήματος με άπληστο αλγόριθμο περιέχεται, ωστόσο η βέλτιστη λύση είναι με δυναμικό προγραμματισμό.

- Δεν μπορούμε να βρούμε συντομότερες διαδρομές σε γράφο με αρνητικά βάρη προσθέτοντας το ίδιο βάρος σε κάθε ακμή ώστε να γίνουν θετικά τα βάρη όλων των ακμών και στη συνέχεια να τρέξουμε τον αλγόριθμο Dijkstra. Μπορώ να κάνω Bellman Ford ή Johnson - Dijkstra προκειμένου να βρω Shortest Path σε γράφο με αρνητικά μήκη.
- Έστω A μη ταξινομημένος πίνακας με n στοιχεία. Υπάρχει αλγόριθμος κατασκευής σωρού από τα στοιχεία του A που απαιτεί χρόνο Θ(n) τόσο στην καλύτερη όσο και στη χειρότερη περίπτωση.
- Αν η ακμή ελάχιστου βάρους σε έναν γράφο είναι μοναδική τότε αυτή ανήκει σε κάθε ελάχιστο συνδετικό δέντρο του γράφου.
- Μπορούμε να εντοπίσουμε εάν ένα μη κατευθυνόμενο γράφημα έχει κύκλο περιττού μήκους σε γραμμικό χρόνο. Ουσιαστικά ελέγχουμε αν ο γράφος είναι διμερής, το οποίο γίνεται σε γραμμικό χρόνο.

Επί πτυχίω 2020

- Δεδομένου ενός μη κατευθυνόμενου γραφήματος G(V, E), είναι NP-πλήρες να αποφανθούμε αν το G έχει ανεξάρτητο σύνολο με τουλάχιστον 2020 κορυφές. Independent set as NP complete
- Στην υλοποίηση της δομής δεδομένων Union Find με **βεβαρημένη ένωση**, κάθε λειτουργία έχει χρόνο εκτέλεσης χειρότερης περίπτωσης O(log(n)).
    - Η χρονική πολυπλοκότητα της βεβαρυμένης Union Find για m finds και n unions είναι O(m * logn + n).
    - O(m * logn + n) είναι ο χρόνος για m finds και n unions, logN ο χρόνος για 1 union, logN ο χρόνος για 1 find.
- Ο αλγόριθμος δυναμικού προγραμματισμού για το πρόβλημα του Πλανόδιου Πωλητή (travelling salesman) **δεν** έχει ψευδο πολυωνιμικό χρόνο.
    - Έχει πολυπλοκότητα n^2 * 2^n.
- Αν P≠NP, τότε κάθε πρόβλημα που δεν ανήκει στο P **δεν** είναι απαραίτητα NP πλήρες γιατί υπάρχουν και προβλήματα που είναι κάτι ενδιάμεσο από P και NP.
- Ένα κατευθυνόμενο ακυκλικό γράφημα έχει μοναδική τοπολογική διάταξη αν και μόνο αν υπάρχει μοναδική κατευθυνόμενη ακμή μεταξύ κάθε ζεύγους κορυφών (χωρίς να δημιουργείται κύκλος). **ΛΑΘΟΣ**
    - Υπάρχει το εξής θεώρημα: Η τοπολογική διάταξη είναι μοναδική αν και μόνο αν υπάρχει hamiltonian path.
- Δίνεται ένα μη κατευθυνόμενο συνεκτικό γράφημα G με θετικά βάρη στις ακμές. Υπάρχει αλγόριθμος γραμμικού χρόνου για τον υπολογισμό ενός Συνδετικού Δέντρου του G στο οποίο η βαρύτερη ακμή έχει το ελάχιστο δυνατό βάρος. Ο αλγόριθμος αυτός είναι ο Minimum bottleneck spanning tree και γίνεται σε γραμμικό χρόνο O(E).
- Δεδομένων ενός μη κατευθυνόμενου γραφήματος G(V,E) και ενός φυσικού k >= 3, είναι NP-πλήρες να αποφανθούμε αν το G έχει απλό κύκλο μήκους τουλάχιστον k. Βγαίνει από αναγωγή στο Hamilton Circle.
- Ο χρόνος εκτέλεσης του αλγορίθμου ταξινόμησης Counting Sort (O(n+k)) δεν είναι πολυωνυμικός στο μέγεθος της εισόδου (n*logk), είναι εκθετικός.
    - Ουσιαστικά αν αυξήσεις τον αριθμό κατά 1 bit διπλασιάζεις τον χρόνο εκτέλεσης

    - Διάλεξη - 44:45
    - What is the precise definition of pseudo-polynomial time (feat. Counting Sort)
- Έστω κατευθυνόμενο γράφημα G(V, E, w) με θετικά μήκη w στις ακμές, και έστω p ένα συντομότερο u – v μονοπάτι στο G. Το p **δεν** παραμένει συντομότερο u – v μονοπάτι αν υψώσουμε τα μήκη όλων των ακμών στο τετράγωνο.
- Δίνονται 4 πίνακες, ο καθένας από τους οποίους έχει n στοιχεία και είναι ταξινομημένος σε αύξουσα σειρά. Θέλουμε να ταξινομήσουμε τα 4n στοιχείων που προκύπτουν από την ένωση των 4 πινάκων. Ο καλύτερος συγκριτικός αλγόριθμος για αυτό το πρόβλημα χρειάζεται χρόνο Θ(n log(n)).
    - Λύνεται με merge σε Θ(n).
- Μπορούμε να υπολογίσουμε τα n/100 μεγαλύτερα στοιχεία ενός (μη ταξινομημένου) πίνακα με n στοιχεία σε χρόνο O(n) στη χειρότερη περίπτωση.
- Δίνονται μη κατευθυνόμενο γράφημα G(V, E) και θετικός φυσικός k. Είναι NP-πλήρες να αποφανθούμε αν το G έχει συνδετικό δέντρο με μέγιστο βαθμό κορυφής μεγαλύτερο ή ίσο του k. → Degree-constrained spanning tree
- Αν στον αλγόριθμο του Dijkstra, επιλέγουμε τη διαθέσιμη κορυφή με τη μέγιστη πεπερασμένη ετικέτα σε κάθε επανάληψη, τότε **δεν** υπολογίζουμε πάντα τα μονοπάτια μέγιστου μήκους από την αρχική κορυφή s προς κάθε άλλη κορυφή.
- Ο αλγόριθμος του Dijkstra και ο αλγόριθμος του Prim έχουν (ασυμπτωτικά) τον ίδιο χρόνο εκτέλεσης χειρότερης περίπτωσης.
- Δίνονται συνεκτικό μη κατευθυνόμενο γράφημα G(V, E, w), με θετικό βάρος w(e) σε κάθε ακμή e, αύξουσα συνάρτηση f : N -> N και θετικός φυσικός B. Είναι NP-πλήρες να αποφανθούμε αν υπάρχει συνδετικό δέντρο T του G για το οποίο το άθροισμα των τιμών f(w(e)), για τις ακμές e που ανήκουν στο T, είναι μικρότερο ή ίσο του B.
    - Σωστό από αναγωγή σε Minimum Leaf Spanning Tree.
- Το DFS γράφημα με n κορυφές χρειάζεται χρόνο Θ(n^2) αν το γράφημα αναπαρίσταται με πίνακα γειτνίασης.
    - [When an adjacency list is used then, DFS and BFS have complexity O(V+E), and if an adjacency matrix is used, the complexity is O(V^2)]

Κανονική 2020

- Σε έναν αλγόριθμο δυναμικού προγραμματισμού, είναι (ασυμπτωτικά) πιο αποδοτικό να χρησιμοποιήσουμε αναδρομή με απομνημόνευση (memoization) από το να υπολογίσουμε την τιμή της λύσης για κάθε υποπρόβλημα του χώρου καταστάσεων.
  https://centralntua.webex.com/recordingservice/sites/centralntua/recording/6fe3b3f7f3b348fc80e2327ab97bbaaf/playback
  Στο 11:20 ο φωτακης αναφερει οτι λογω της top down φυσης του memorization μπορει να γλυτώσουμε καποιες αχρηστες ενδιαμεσες καταστασεις. Οποτε ασυμπτωτικά σε ορισμενα προβληματα ειναι ποιο αποδοτικο
    - Αν έχουμε bottom-up προσέγγιση, τότε το memoization ασυμπτωτικά είναι το ίδιο.
- Ο χρόνος εκτέλεσης χειρότερης περίπτωσης ενός αλγόριθμου δυναμικού προγραμματισμού με μέγεθος χώρου καταστάσεων S είναι αναγκαστικά Ω(S).

- Έστω μη κατευθυνόμενο γράφημα G(V,E) στο οποίο εφαρμόζουμε BFS με αρχική κορυφή την s. Αν μια κορυφή u ανήκει στο k επίπεδο του BFS-δέντρου, κάθε s-u μονοπάτι έχει μήκος ίσο ή μεγαλύτερο του k.
- Σε ένα s-t δίκτυο, αν μειώσουμε τη χωρητικότητα μιας ακμής της μέγιστης τομής κατά k, τότε η μέγιστη ροή **δεν** μειώνεται κατά k.
    - Μέγιστη ροή = Ελάχιστη τομή, άρα αν αυξήσουμε τη μέγιστη τομή η ελάχιστη τομή δεν επηρεάζεται.
- Αν οι κλάσεις P και NP ταυτίζονται, τότε κάθε πρόβλημα στο NP είναι αναγκαστικά NP-πλήρες.
- Αν οι κλάσεις P και NP είναι διαφορετικές, τότε υπάρχουν προβλήματα που δεν ανήκουν στο NP και δεν είναι NP-πλήρη.

Κανονική 2021

- Θεωρούμε το δέντρο αναδρομής που περιγράφει την εκτέλεση της quicksort, όταν εφαρμόζεται σε μη ταξινομημένο πίνακα. Ο χρόνος εκτέλεσης που δαπανάται σε κάθε επίπεδο του δέντρου αναδρομής, με εξαίρεση ίσως το τελευταίο, είναι της ίδιας τάξης μεγέθους.
    - Notice that every level of the tree has cost n, until a boundary condition is reached at depth log10 n = (lg n), and then the levels have cost at most n.
- Σε έναν αλγόριθμο δυναμικού προγραμματισμού, ο υπολογισμός της λύσης με bottom-up τρόπο μπορεί να είναι ασυμπτωτικά ταχύτερος από τον υπολογισμό της λύσης με top-down τρόπο και απομνημόνευση (memoization). **ΛΑΘΟΣ**
- Μπορούμε να χρησιμοποιήσουμε δυαδική αναζήτηση για να ανάγουμε πολυωνυμικά ένα πρόβλημα συνδυαστικής βελτιστοποίησης στο αντίστοιχο πρόβλημα απόφασης.
- Ένα κατευθυνόμενο ακυκλικό γράφημα G έχει μοναδική τοπολογική διάταξη αν και μόνο αν το G έχει μονοπάτι Hamilton, δηλαδή ένα απλό κατευθυνόμενο μονοπάτι που περνάει από όλες τις κορυφές.
- Ο χρόνος εκτέλεσης ενός αλγορίθμου δυναμικού προγραμματισμού είναι Θ(K*t), όπου K το πλήθος των υποπροβλημάτων και t ο χρόνος που χρειάζεται το κάθε υποπρόβλημα.
- Κάθε πρόβλημα μεγιστοποίησης κέρδους που λύνεται με δυναμικό προγραμματισμό μπορεί να αναχθεί σε ένα πρόβλημα εύρεσης μακρύτερου μονοπατιού σε ένα DAG.
- Σε μη κατευθυνόμενο γράφημα με n κορυφές και m=Θ(n^3/2) ακμές που αναπαρίστανται με **πίνακα γειτνίασης**, το DFS χρειάζεται χρόνο **O(n^2)**.
- Μπορούμε να αποφανθούμε σε γραμμικό χρόνο, αν σε ένα μη κατευθυνόμενο γράφημα με n κορυφές και m>n ακμές, όλοι οι κύκλοι έχουν άρτιο μήκος. ?
- Σε ένα μη κατευθυνόμενο συνεκτικό γράφημα με βάρη στις ακμές, κάθε ακμή ελάχιστου βάρους ανήκει σε κάποιο ΕΣΔ.
- Σε ένα μη κατευθυνόμενο συνεκτικό γράφημα με n κορυφές και n+30 ακμές, μπορούμε να υπολογίσουμε ένα ΕΣΔ σε γραμμικό χρόνο.
- Η βέλτιστη λύση για τη διακριτή εκδοχή του προβλήματος του Σακιδίου **ΔΕΝ** περιέχει πάντα το αντικείμενο με τον μέγιστο λόγο αξίας προς μέγεθος.
- Πρόβλημα επιλογής ενός μέγιστου πλήθους μη επικαλυπτόμενων διαστημάτων

    - Ο άπληστος αλγόριθμος που επιλέγει το διαθέσιμο διάστημα με τη μικρότερη διάρκεια μεταξύ όλων των διαθέσιμων διαστημάτων υπολογίζει πάντα τη βέλτιστη λύση.
    - Αν όλα τα διαστήματα έχουν το ίδιο μήκος, ο άπληστος αλγόριθμος που επιλέγει το διαθέσιμο διάστημα που ξεκινά πρώτο υπολογίζει πάντα τη βέλτιστη λύση.
- Ο χρόνος εκτέλεσης του αλγόριθμου δυναμικού προγραμματισμού για τη διακριτή εκδοχή του προβλήματος Σακιδίου είναι **ψευδοπολυωνυμικός**.
- Μπορούμε να υπολογίσουμε όλες τις γέφυρες και τα σημεία κοπής ενός μη κατευθυνόμενου συνεκτικού γραφήματος σε γραμμικό χρόνο με BFS.
    - Είναι O(V+E)
- Μπορούμε να υπολογίσουμε μια τοπολογική διάταξη ενός κατευθυνόμενου ακυκλικού γραφήματος σε γραμμικό χρόνο με DFS.
    - Είναι O(V+E)
- Δίνεται μη κατευθυνόμενο συνεκτικό γράφημα με θετικά βάρη. Υπάρχει αλγόριθμος γραμμικού χρόνου για τον υπολογισμό ενός συνδετικού δέντρου στο οποίο η βαρύτερη ακμή έχει το ελάχιστο δυνατό βάρος.
- Αν στον αλγόριθμο Dijkstra επιλέγουμε σε κάθε επανάληψη τη διαθέσιμη κορυφή με τη μέγιστη πεπερασμένη ετικέτα, τότε **ΔΕΝ** υπολογίζουμε τα μονοπάτια μέγιστου μήκους.
    - Επειδή οι κύκλοι προσθέτουν μήκος.
- Έστω κατευθυνόμενο γράφημα G(V, E, w) με θετικά μήκη w στις ακμές, και έστω p ένα συντομότερο u – v μονοπάτι στο G. Το p **ΔΕΝ** παραμένει συντομότερο u – v μονοπάτι αν αντικαταστήσουμε το μήκος κάθε ακμής με την τετραγωγική του ρίζα.
- Για ένα δίκτυο ροής με n κορυφές, m>=n ακμές, και ακέραιες χωρητικότητες στις ακμές του, ο αλγόριθμος Ford-Fulkerson έχει χρόνο εκτέλεσης χειρότερης περίπτωσης O(m*|f|) όπου f είναι το μέγεθος της μέγιστης ροής.
- Αν σε ένα δίκτυο ροής αυξήσουμε τη χωρητικότητα όλων των ακμών της ελάχιστης τομής κατά 1, τότε η μέγιστη ροή **ΔΕΝ** αυξάνεται αναγκαστικά τουλάχιστον κατά 1.
- Αν ένα πρόβλημα Π ανάγεται πολυωνυμικά σε ένα γνωστό NP-hard πρόβλημα, τότε το Π είναι αναγκαστικά NP-hard.
- Αν P = NP, τότε όλα τα προβλήματα στην κλάση NP είναι NP-πλήρη.
- Έστω κατευθυνόμενο γράφημα G με n κορυφές, m=n^(3/2) ακμές και (ενδεχομένως αρνητικά) βάρη. Ο αλγόριθμος Johnson στο G, για τον υπολογισμό των συντομότερων μονοπατιών μεταξύ όλων των ζευγών κορυφών, είναι ασυμπτωτικά ταχύτερος από τον αλγόριθμο Floyd-Warshall.
- Το παρακάτω πρόβλημα ανήκει στην κλάση NP: δεδομένου ενός σύνθετου αριθμού n, που γράφεται ως γινόμενο δύο πρώτων αριθμών, να υπολογίσουμε τους δύο πρώτους παράγοντες του n.
- Κάθε πρόβλημα στο NP μπορεί να λυθεί σε εκθετικό χρόνο.
- Δίνεται κατευθυνόμενο ισχυρά συνεκτικό γράφημα G με θετικά μήκη στις ακμές και θετικός ακέραιος B. Υπάρχει αλγόριθμος γραμμικού χρόνου που υπολογίζει μονοπάτι p μεταξύ δύο συγκεκριμένων κορυφών με τέτοιο τρόπο ώστε κάθε ακμή του p να έχει μήκος το πολύ B.
- Αλγόριθμος Dijkstra σε κατευθυνόμενο G ισχυρά συνεκτικό
    - Κάθε ακμή εξετάζεται μια φορά για αναπροσαρμογή.

- Έστω κατευθυνόμενο γράφημα G με (ενδεχομένως αρνητικά) βάρη.
    - Αν το G δεν περιέχει κύκλους μηδενικού ή θετικού μήκους, τότε ο αλγόριθμος Bellman-Ford μπορεί να υπολογίσει ένα μακρύτερο μονοπάτι. **ΛΑΘΟΣ**
    - Αν το G περιέχει έναν κύκλο αρνητικού μήκους, ο αλγόριθμος Bellman-Ford υπολογίζει μια συντομότερη απλή διαδρομή.