HY437 – Αλγόριθμοι CAD

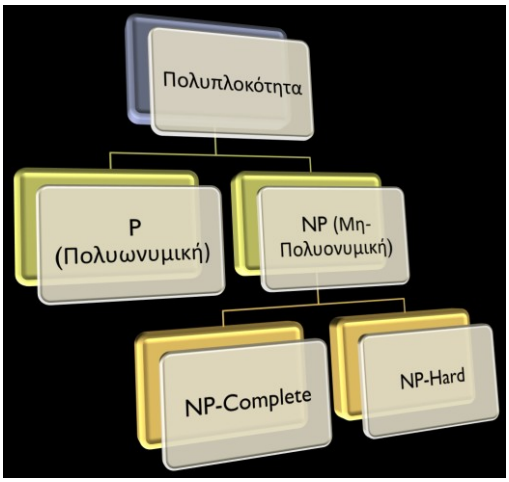
Διδάσκων: Χ. Σωτηρίου

http://inf-server.inf.uth.gr/courses/CE437/

Περιεχόμενα

- Πολυπλοκότητα – Είδη Προβλημάτων
- Τύποι Αλγορίθμων – Κατηγοριοποίηση
- Γραμμικός Προγραμματισμός (Σχεδιασμός)
- Άπληστος Αλγόριθμος (Greedy)
- Αλγόριθμος Επιλογής και Φραγής (Branch and Bound)
- Δυναμικός Προγραμματισμός (Dynamic Programming)
- Αλγόριθμοι Γράφων
  - Βραχύτερο/Μακρύτερο Μονοπάτι
- Ευριστική Αναζήτηση
- Αλγόριθμος Simulated Annealing

Πολυπλοκότητα



Πρακτικό Παράδειγμα

- Σκεφτείτε έναν ακέραιο στο διάστημα [1,B], έστω x
- Ποια είναι η πολυπλοκότητα για να τον βρει κάποιος ερωτώντας σας ως προς την τιμή του;
- Βάση της *φύσης του προβλήματος* πόσες πιθανές ερωτήσεις υπάρχουν; → πολυπλοκότητα του προβλήματος
- Αν εφαρμόσω *συγκεκριμένο αλγόριθμο*, ποια είναι η χείριστη περίπτωση αριθμού ερωτήσεων; → πολυπλοκότητα της λύσης

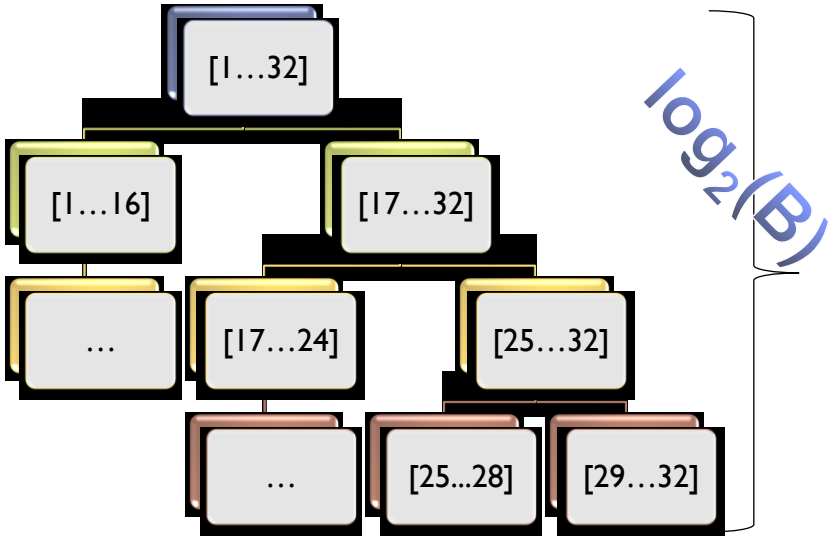
Πρακτικό Παράδειγμα - Συνέχεια

- Εφαρμόζοντας Δυαδική Εύρεση (Binary Search)…
- Για B=32:

Ερώτηση	Απάντηση	Διάστημα Πιθανών Τιμών του x μετά την Απάντηση
-	-	[1…32]
Ισχύει x > 16;	**Όχι**	[1…16]
Ισχύει x > 8;	**Ναι**	[9…16]
Ισχύει x > 12;	**Όχι**	[9…12]
Ισχύει x > 10;	**Ναι**	[11…12]
Ισχύει x > 11;	**Όχι**	11

- Άρα, μετά από $log_2(B)$ ερωτήσεις έχουμε απάντηση!

Πρακτικό Παράδειγμα - Συνέχεια



Τύποι Αλγορίθμων - Κατηγοριοποίηση

- Αλγεβρικοί
  - Γραμμικός Προγραμματισμός (Linear Programming)
  - Ακέραιος Γραμμικός Προγραμματισμός (Integer LP)
- Διαίρει και Βασίλευε (Divide and Conquer)
  - Επιλογής και Φραγής (Branch and Bound)
- Άπληστοι (Greedy)
- Δυναμικού Προγραμματισμού (Dynamic Programming)
- …
- *Μονόσημο Πρόβλημα Κάλυψης – (Unate Covering Problem)*
- *Προσομοίωσης Ανόπτησης (Simulated Annealing)*
- *Γενετικοί*

Γραμμικός Προγραμματισμός

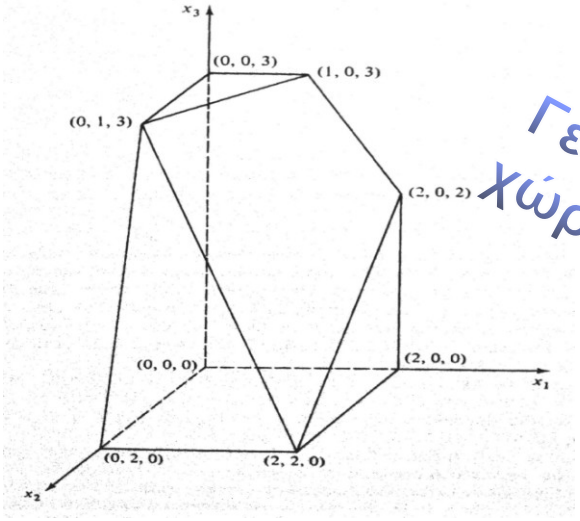
Γραμμικός Προγραμματισμός

- Παράδειγμα:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ \text{and} \quad & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Περιορισμοί (Ax <= b)
x1 + x2 + x3 <= 4
x1 <= 2
x3 <= 3
3x2 + x3 <= 6
x1 >= 0
x2 >= 0
x3 >= 0

Γραμμικός Προγραμματισμός



Γεωμετρικός χώρος λύσης

Άπληστος (Greedy)

Άπληστος Αλγόριθμος Δρομολόγησης

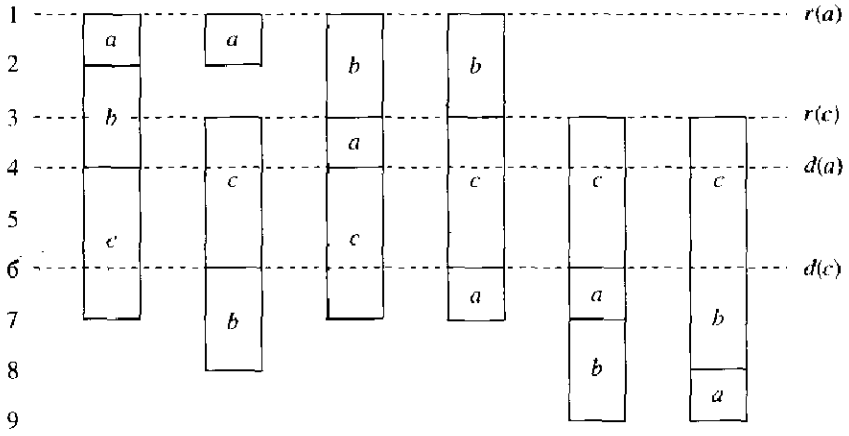
- Σύνολο από Εργασίες (Tasks), **T**
- Για κάθε εργασία t του T:
  - l(t) – διάρκεια της εργασίας (length)
  - r(t) – ελάχιστος χρόνο έναρξης (release time)
  - d(t) – προθεσμία λήξης (deadline)
- Αναζητούμε Δρομολόγηση που να ικανοποιεί τα l, r, d

Αλγόριθμος GREEDY-SCHEDULING (T)

```
GREEDY-SCHEDULING(T) {
   i = 1;
   repeat {
        while ((Q = {unscheduled tasks with release time < i} == 0) do
            i = i + 1;
        if (exists unscheduled task p: i + l(p) > d(p)) return FALSE;
        select q in Q with smallest deadline;
        Schedule q at time i;
        i = i + l(q);
   } until (all tasks scheduled);}
  return TRUE;
```

Άπληστος Αλγόριθμος Δρομολόγησης

- r(a) = r(b) = 1, r(c) = 3, d(a) = 4, d(b) = inf, d(c) = 6, l(a) = 1, l(b) = 2, l(c) = 3;



Επιλογής και Φραγής (Branch and Bound)

Αλγόριθμος Επιλογής και Φραγής

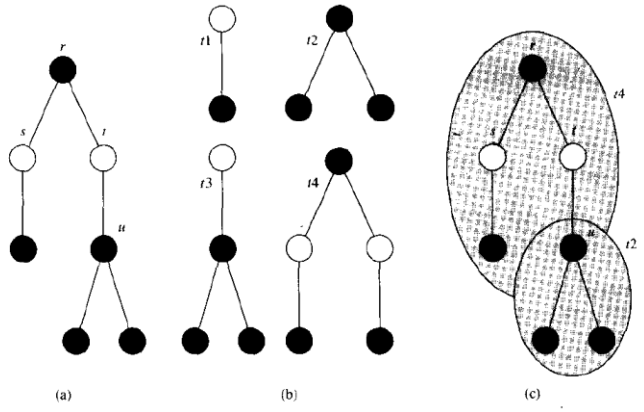
- Ακολουθία επιλογών S - αρχική μηδενική ακολουθία αποφάσεων s0
- Συνήθως το m είναι 2, δηλαδή το δέντρο αποφάσεων είναι δυαδικό!

Αλγόριθμος BRANCH_AND_BOUND

```
BRANCH_AND_BOUND {
   Current_best = anything; Current_cost = inf; S = s0;
   while (S != 0) do {
         Select and element s of S and Remove;
         Branch based on s in sequence {s_i, i = 1, 2, …, m};
         for (i = 1 to m) {
             Compute lower bound b_i of s_i;
             if (b_i >= Current_cost) Remove branch s_i;
             else {
                 if (s_i is complete solution) {
                     Current_best = s_i;
                     Current_cost = cost of s_i;
                 }
                 else add s_i to S;
             }
         }
   }
}
```


Δυναμικός Προγραμματισμός

Αλγόριθμος Κάλυψης Δέντρου



- (a) Υποκείμενος Γράφος, (b) Βασικοί Γράφοι, (c) Βέλτιστη Κάλυψη

Αλγόριθμος Κάλυψης Δέντρου

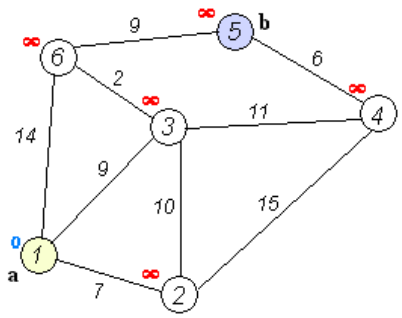
Αλγόριθμος TREE-COVER(Gn(V,E))

```
TREE-COVER(T(V, E)) {
   foreach vertex v in V
        COST(v) = -1;                    // set internal vertex cost to -1 //
   foreach leaf vertex v in V
        if SUCC(v) = NULL, COST(v) = 0; // set leaf vertex cost is 0 //
   while (some vertex has -ve weight) {
         select v in V (bottom-up) s.t. all SUCC(v) have non -ve cost;
         M = set of all matching pattern trees at v;
         cost(v) = min(m in M)(cost(m) + Σ(u in SUCC(m)) cost(u));
         // pick m which minimises cost //
   }
}
```

- Το δέντρο διατρέχεται από τα φύλλα προς την ρίζα (bottom-up)
- ***Σε κάθε κόμβο επιλύουμε βέλτιστα την επιλογή κάλυψης για ελάχιστο κόστος από το σημείο εκείνο και κάτω***

Αλγόριθμος Dijkstra
Βραχύτερο Μονοπάτι (Shortest Path)

Αλγόριθμος Shortest Path



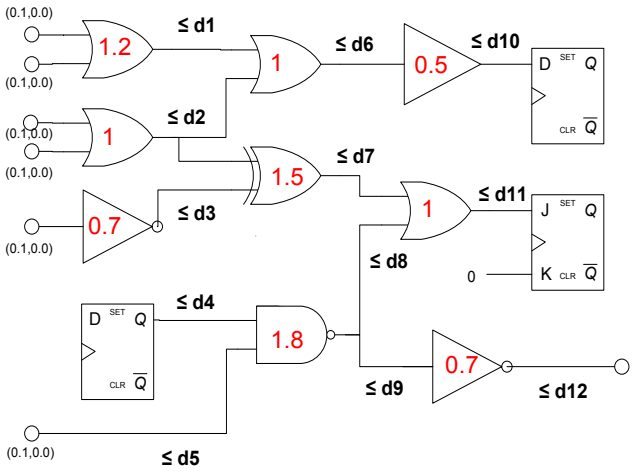
Αλγόριθμος Dijkstra – Shortest Path

Αλγόριθμος DIJKSTRA (Graph(V, E), source)

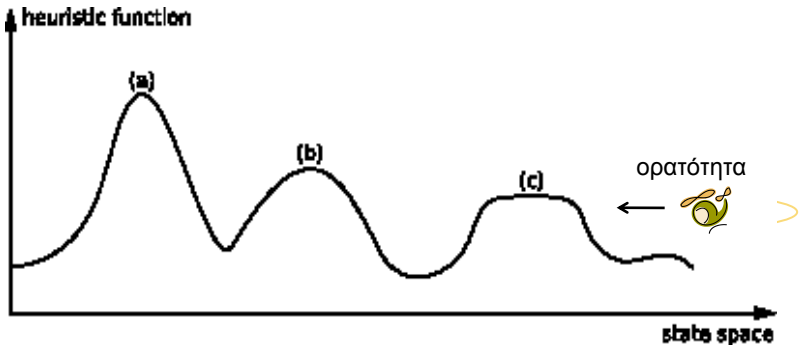
```
for each vertex v in Graph:              // Initializations
  dist[v] := infinity ;                  // Unknown distance function from source to v
  previous[v] := undefined ;             // Previous node in optimal path
end for                                  // from source
dist[source] := 0 ;                      // Distance from source to source
Q := the set of all nodes in Graph ;     // All nodes in the graph are unoptimized
                                         // thus are in Q
while Q is not empty: // the main loop
   u := vertex in Q with smallest distance in dist[] ; // Source node in first case
   remove u from Q ;
   if dist[u] = infinity:
     break ;                             // all remaining vertices are
   end if                                // inaccessible from source

   for each neighbor v of u:             // where v has not yet been removed from Q.
      alt := dist[u] + dist_between(u, v) ;
      if alt < dist[v]:                  // Relax (u,v,a)
         dist[v] := alt ;
         previous[v] := u ;              // Store Shortest Path
         decrease-key v in Q;            // Reorder v in the Queue
      end if
   end for
end while
return dist;
```


Αλγόριθμος Longest Path - STA

- Ποιο είναι το Κρίσιμο Μονοπάτι;



Ευριστική Αναζήτηση



Αλγόριθμος Simulated Annealing

Αλγόριθμος Simulated Annealing

Αλγόριθμος Simulated Annealing(s0, kmax, cmax)

```
s = s0; c = c(s);                          // Initial state/solution, cost
sbest = s; cbest = c;                      // Initial "best" solution
k = 0;                                     // Cost iteration count - may be time-based

while (k < kmax and c > cmax)              // While time left & not good enough:
   T = temperature(t, k, kmax);            // Temperature calculation/schedule
   snew = neighbour(s);                    // Traverse state space to a neighbouring solution
   cnew ← C(snew)                          // Compute new cost

   if (P(c, cnew, T) > random())           // Decision based on T, (cnew - c) and random()
      s = snew; c = cnew                   // if Yes, update current solution

   if (cnew < cbest)                       // Check if this is the best solution found
      sbest = snew; cbest = cnew;          // if Yes, save it

   k = k + 1;                              // Increment iterations counter

return sbest;                              // Return the best solution found
```

- Δρομολόγηση θερμοκρασίας από συνάρτηση `temperature(T, k)`
  - λ.χ. `temperature(k/kmax)`
- Τυπικά, η συνάρτηση Αποδοχής **`P(c, cnew, T)`** *επιστρέφει 1, αν (cnew < c)*
  - **Όταν (cnew – c) > 0 και όσο μεγαλώνει, τυπικά μειώνεται η πιθανότητα αποδοχής**
- Η συνάρτηση `random()` επιστρέφει τιμές στο διάστημα: [0, 1]