# ALGORITMI TEORIA (TIPOLOGIE DI ESERCIZI)

- FATTI
  - MIN/MAX HEAP
  - RB TREES
  - RICORRENZE
  - HUFFMAN
  - DIJKSTRA
  - COUNTING SORT
- DA FARE
  - TABELLE HASH
  - PROGRAMMAZIONE DINAMICA
  - STRATEGIA GREEDY

# TEORIA ALGORITMI

## FARO

- HEAP
  HEAPSORT
- COUNTING SORT.
  RADIX SORT
- HASH TABLE (CONCATENAZIONE, INDIRIZZAMENTO APERTO) DIM
- RED-BLACK TREES DIM
- LCS
  EDIT DISTANCE
- SELEZIONE DI ATTIVITA'
  PROBLEMA DEL RESTO
  ALL PAIRS SHORTEST PATH (ALG. GENERICO)
- FLOYD-WARSHALL

## CANTONE

INSERTION SORT (COMPLESSITA')
DIVIDE ET IMPERA (MERGE-SORT)
ANALISI ALGORITMI D.E.I
METODO DI SOSTITUZIONE

- NOTAZIONI ASINTOTICHE
  METODO ITERATIVO
- TEOREMA MASTER E COROLLARIO
- MEDIANE E STATISTICHE D'ORDINE
  PROGRAMMAZIONE DINAMICA
- MOLTIPLICAZIONE SEQUENZA DI MATRICI
  ALGORITMI GREEDY
- PROBLEMA SELEZIONE ATTIVITA'
- KNAPSACK PROBLEM (INTERA, FRAZIONARIA)
- CODICI DI HUFFMAN
  LABEL SETTING (GENERIC ALGORITHM)
  DAG-SHORTEST PATH (TOP-SORT)
- DIJKSTRA
  LABEL CORRECTING
- BELLMAN-FORD

# TEOREMA MASTER

- Per ricorrenze della forma

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

con $a \geq 1$, $b > 1$ costanti e $f(n)$ una funzione assegnata

1. se $f(n) = O(n^{\log_b a - \varepsilon})$ per qualche $\varepsilon > 0$,
   $\Rightarrow T(n) = \Theta(n^{\log_b a})$

2. se $f(n) = \Theta(n^{\log_b a})$
   $\Rightarrow T(n) = \Theta(n^{\log_b a} \cdot \log n)$

3. se $f(n) = \Omega(n^{\log_b a + \varepsilon})$ per qualche $\varepsilon > 0$
   e se:
   - $af\left(\frac{n}{b}\right) \leq cf(n)$ per qualche $c < 1$ e per valori di $n$ sufficientemente grandi. } CONDIZIONE DI REGOLARITÀ

   $\Rightarrow T(n) = \Theta(f(n))$

2. se $f(n) = \Theta(n^{\log_b a} \cdot \log^k n)$, $k \geq 0$
   $\Rightarrow T(n) = \Theta(n^{\log_b a} \cdot \log^{k+1} n)$

## COROLLARIO AL TEOREMA MASTER

Siano $a \geq 1$, $b > 1$, $k \geq 0$, $h \geq 0$ costanti:

$$T(n) = aT\left(\frac{n}{b}\right) + n^k (\log n)^h$$

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{se } \log_b a > k \\ \Theta(n^k (\log n)^{h+1}) & \text{se } \log_b a = k \\ \Theta(n^k (\log n)^h) & \text{se } \log_b a < k \end{cases}$$

- $T(n) = O(n) \Rightarrow T(n) \leq O(n)$
- $T(n) = \Omega(n) \Rightarrow T(n) \geq \Omega(n)$
- $T(n) = \Theta(n) \Rightarrow T(n) = \Theta(n)$
- $T(n) = o(n) \Rightarrow T(n) < o(n)$
- $T(n) = \omega(n) \Rightarrow T(n) > \omega(n)$

# NOTAZIONI ASINTOTICHE

- $\Theta(g(n)) = \{ f(n) : \exists c_1, c_2 > 0, n_0 \in \mathbb{N} : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0 \}$
- $O(g(n)) = \{ f(n) : \exists c > 0, n_0 \in \mathbb{N} : 0 \leq f(n) \leq c g(n) \quad \forall n \geq n_0 \}$
- $\Omega(g(n)) = \{ f(n) : \exists c > 0, n_0 \in \mathbb{N} : 0 \leq g(n) \leq f(n) \quad \forall n \geq n_0 \}$
- $o(g(n)) = \{ f(n) : \forall c > 0, \exists n_0 > 0 : 0 \leq f(n) < c g(n) \quad \forall n \geq n_0 \}$
- $\omega(g(n)) = \{ f(n) : \forall c > 0, \exists n_0 > 0 : 0 \leq c g(n) \leq f(n) \quad \forall n \geq n_0 \}$

## HEAP

```
INSERT(A,K)                                   O(log n)
    heapsize ++
    i = heapSize          ← A[heapsize]=K
    while(i>1 && A[parent(i)] < A[i])
        swap(A, i, parent(i))
        i = parent(i)
```

```
HEAPIFY(A,i)                                  O(log n)
    l = left(i)
    r = right(i)
    max = i
    if(l ≤ heapsize && A[l] > A[max])
        max = l
    if(r ≤ heapsize && A[r] > A[max])
        max = r
    if(max ≠ i)
        swap(A, i, max)
        HEAPIFY(A, max)
```

```
EXTRACT-MAX(A)
    swap(A, 1, heapsize)      } O(1)
    heapsize --
    HEAPIFY(A,1)              } O(log n)
    return A[heapsize+1]
```

```
INCREASE-KEY(A, i, K)                         O(log n)
    A[i] = K
    while(i>1 && A[parent(i)] < A[i])
        SWAP(A, i, parent(i))
        i = parent(i)
```

```
HEAPSORT(A,n)                                 O(n log n)
    BUILD-HEAP(A,n)
    For(i=1 to n-1)
        EXTRACT-MAX(A)
```

```
BUILD-HEAP(A,n)                               O(n log n)
    heapsize = n                  // in realtà heapify non viene
    For(i = ⌊n/2⌋ down to 1)      // sempre chiamata => O(n)
        HEAPIFY(A,i)
```

# ORDINAMENTO

```
COUNTING SORT (A, m)  O(K+m)
    K = max(A)
    C = new_array(K+1)
    for(i=0 to K)
        C[i]=0
    for(i=0 to m-1)
        C[A[i]]++
    B = new_array(m)
    J=0
    for(i=0 to K+1)
      for(h=1 to C[i])
          B[J] = i
          J++
```

```
RADIX-SORT (A, d)  O(m)
    for(i=1 to d)
      COUNTING SORT (A,i)
```

# STATISTICHE D'ORDINE

## MIN-MAX (A) $\Theta(\lceil \frac{3m}{2} \rceil - 2)$

- Sia A un insieme di m numeri
- si suddividano gli m elementi in $m/2$ coppie
- si confrontino gli elementi di ciascuna coppia mettendo i vincenti in B e i perdenti in C (se c'è un elemento spaiato si aggiunge a B e C)
- Avremo che: $MAX(B) = MAX(A)$
  $MIN(C) = MIN(A)$

## SELECT (A, i) $\Theta(m)$

- si divida A in $\lfloor \frac{m}{5} \rfloor$ gruppi di 5 elementi
- si determinino le mediane degli $\lceil \frac{m}{5} \rceil$ gruppi e sia M la sequenza di tale mediane
- si effettui la chiamata ricorsiva $m = SELECT(M, \lceil \frac{|M|}{2} \rceil)$ (la mediana delle mediane)
- Partizioniamo A in $A_1$, $A_2$, $A_3$ dove:
  - $A_1 = [x \in A : x < m]$
  - $A_2 = [x \in A : x = m]$
  - $A_3 = [x \in A : x > m]$

- L'obiettivo è dunque cercare l'i-esimo elemento, quindi
  if $(|A_1| \geq i)$ → SELECT$(A_1, i)$
  elif $(|A_1| + |A_2| \geq i \geq |A_1|)$ → return m
  else $(|A_1| + |A_2| < i \leq |A|)$ → SELECT$(A_3, i - (|A_1| + |A_2|)$

## TABELLE HASH

- INSERT $(T,K)$ $O(1)$
  $T[h(K)] = K$
- CANC$(T,K)$ $O(1)$
  $T[h(K)] = NIL$
- SEARCH$(T,K)$ $O(n)$
  return $(T[h(K)] != NIL)$

## FUNZIONE HASH

CONC. $O(1+\alpha)$

- METODO DELLA DIVISIONE $[\ h(K) = K \bmod m\ ]$
- METODO DI MOLTIPLICAZIONE $[\ h(K) = \lfloor m\,(KA \bmod 1)\rfloor\ ]$ $(A = \frac{S}{2^W}$, $W = \#$ bit di una word $\Rightarrow$ prendo i bit più significativi di $K \cdot S)$

INDIRIZZ. APERTO

- HASHING A SCANSIONE LINEARE $[\ h(K,i) = (h'(K)+i) \bmod m\ ]$ (soffre di agglomerazione primaria)
- HASHING QUADRATICO $[\ h(K,i) = (h'(K) + c \cdot i^2) \bmod m\ ]$ (non soffre sempre)
- HASHING DOPPIO $[\ h(K,i) = [h'(K) + i\,h''(K)] \bmod m\ ]$ (non soffre

RICERCA CONCAT. $O(1+\alpha)$

RICERCA IND. APERTO $= \begin{cases} \frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right) & \text{con successo} \\ \frac{1}{1-\alpha} & \text{senza successo (inserimento)} \end{cases}$

# PROGRAMMAZIONE DINAMICA

```
MATRIX-MULTIPLY(A,B)   O(p·q·r)
    p = rows[A]
    q = colums[A]
    r = colums[B]
    for(i=1 to p)
      for(j=1 to r)
        C[i,j] = 0
        for(k=1 to q)
            C[i,j] = C[i,j] + A[i,j] · B[k,j]
    return C
```

DINAMICO

```
MATRIX-CHAIN-ORDER(p)   Θ(m³)
  for(i=1 to m)
     m[i,i] = 0
  for(Δ=1 to m-1)
    for(i=1 to m-Δ)
       j = Δ + i
       m[i,j] = +∞
       for(k=i to j-1)
          q = m[i,k] + m[k+1,j] + P_{i-1} · P_k · P_j
          if(q < m[i,j]
             m[i,j] = q
             s[i,j] = k
  return m,s

MATRIX-CHAIN-MULTIPLY(A,s,i,j)
   if(i=j) return A
   else
      X = M-C-M(A,s,i,s[i,j])
      Y = M-C-M(A,s,s[i,j]+1,j)
   return(MATRIX_MULTIPLY(X,Y))
```

m[i,j] costo
s[i,j] punto ottimo

A: sequenza di matrici

# SELEZIONE DI ATTIVITÀ

```
RECURSIVE-ACTIVITY-SELECTOR (s,f,i,s)
  For (k = i+1 to j-1)
    if( f_i ≤ s_k < f_k ≤ s_j)
      RETURN {k} ∪ RECURSIVE-ACTIVITY-SELECTOR (s,f,k,s)
  RETURN ∅
```

```
GREEDY-ACTIVITY-SELECTOR (s,f)   O(n) a meno dell'ordinamento di S.
  n = |s|
  A = {1}
  i = 1
  for( m = 2 to n)
    if (s_m ≥ f_i)
      A = A ∪ {m}
      i = m
  return A
```

## CODICI DI HUFFMAN

```
HUFFMAN (C, F)
  n = |C|
  Q = make_queue (C, F)
  for( i = 1 to n-1)
    alloco nodo z
    left[z] = x = EXTRACT_MIN (Q)
    right[z] = y = EXTRACT_MIN(Q)
    f[z] = F[x] + F[y]
    INSERT (Q, z, F)
  return EXTRACT-MIN (Q)
```

- MAKE_QUEUE = $O(n)$
- $(n-1)$ INSERT = $O(n \log n)$
- $(2n-1)$ EX_MIN = $O(n \log n)$

$\Downarrow$

$O(n \log n)$

## LONGEST COMMON SUBSEQUENZE

```
LCS(X_i, Y_s)
  M = new_matrix (m+1, n+1)
  for( s = 1 to n)   M[0, s] = 0      O(n)
  for( i = 1 To m)   M[i, 0] = 0      O(m)      O(m x n)
  for( i = 1 to m)
    for(s = 1 to n)                   O(m, n)
      if(X[i] == Y[s])                O(1)
        M[i, s] = M[i-1, s-1] + 1
      else
        M[i, s] = max( M[i, s-1], M[i-1, s])
  return M[m, n]
```

# SINGLE SOURCE SHORTEST PATH (PROCEDURE BASE)

- GENERIC-ALGORITHM $(G, s, w)$
  - INITIALIZE-SINGLE-SOURCE $(G, s)$
  - $S = \emptyset$
  - while $(\exists\, v \in V[G] \setminus S$ tale che $d[v] = \delta_G(s, v))$
    - sia $v \in V[G] \setminus S : d[v] = \delta_G(s, v)$
    - SCAN $(v, G, w)$
    - $S = S \cup \{v\}$

- SCAN $(u, G, w)$
  - for $(v \in Adj[u])$
    - RELAX $(u, v, w)$

- RELAX $(u, v, w)$
  - if $(d[v] > d[u] + w(u, v))$
    - $d[v] = d[u] + w(u, v)$
    - $pred[v] = u$

- INITIALIZE-SINGLE-SOURCE $(G, s)$
  - for $(v \in V[G])$
    - $d[v] = +\infty$
    - $pred[v] = NIL$
  - $d[s] = 0$

## SSSP GRAFI ACICLICI

```
DAG-SHORTEST-PATHS(G, s, w)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  while (V[G] \ S ≠ ∅)
      sia v ∈ V[G] \ S i cui predecessori siano in S
      SCAN(v, G, w)
      S ∪ S ∪ {w}
```

### VARIANTE TOP-SORT

```
DAG-SHORTEST-PATH(G, s, w)
  INITIALIZE-SINGLE-SOURCE(G, s)
  sia τ un top-sorto di G
  for(v ∈ V[G] seguendo τ)
      SCAN(v, G, w)
```

$O(V)$
$O(V+E)$
$O(V+E)$

$\Theta(V+E)$

## SSSP GRAFI CON $w : E \to \mathbb{R}_0^+$

```
DIJKSTRA(G, s, w)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  while(V[G] \ S ≠ ∅)
    sia v ∈ V[G] \ S : d[v] = min{d[u]  u ∈ V \ S}
    SCAN(v, G, w)
    S = S ∪ {v}
```

### VARIANTE CON HEAP

```
DIJKSTRA(G, s, w)
  INITIALIZE-SINGLE-SOURCE(G, s)
  Q = BUILD-HEAP(V[G], d)
  while(Q ≠ ∅)
      v = EXTRACT-MIN(Q, d)
      SCAN(v, G, w)
```

$O(V)$
$O(V)$

$O(V \log V)$
$O(E \log V)$

$O((V+E)\log V)$

# SSSP LABEL CORRECTING

```
BELLMAN-FORD(G, w, s)
    INITIALIZE-SINGLE SORCE(G, s)
    For(i = 1 To |V|-1)
        For(u, v ∈ E)
            RELAx(u, v, w)
    for(u, v ∈ E)
        IF(d[v] > d[u] + w(u, w)   //∃ ciclo⁻
            return false
    return true
```

O(V)

O(V·E)

O(V+E)

O(VE)  $O(V^3)$ se G è molto denso

# ALL-PAIR-SHORTEST-PATH

## ALGORITMO DINAMICO

```
EXTEND_SP(D^{m-1}, W)
    D^m = new_matrix(n x n)
    For (i=0 to n-1)                      ] n^2
      for(j=0 to n-1)                     ]
        d^m_ij = d^{m-1}_ij               ] n
        for(k=0 to n-1)                   ]
          if(d^{m-1}_ik + w_kj < d^m_ij)
            d^m_ij = d^{m-1}_ik + w_kj
    return D^m                                  } n^3

ALL_PAIRS_SP(W)
    D^1 = W
    For (m=2 to n-1)                      ] n
      D^m = EXTEND_SP(D^{m-1}, W)         ] n^3
    return D^{n-1}
                                          O(n^4)

FAST_ALL_PAIR_SP(W)
    D^1 = W
    m = 1
    while(m ≤ n-1)                        ] log n
      D^{2m} = EXTEND_SP(D^m, D^m)        ] n^3
      m = 2m
    return D^m
                                          O(n^3 log n)

FLOYD_WARSHALL(W)
    D^0 = W
    For(k=1 to n)                         ] n
      D^k = new_matrix[n x n]
      For(i=1 to n)                       ] n^2
        For(j=1 To n)                     ]
          d^k_ij = d^{k-1}_ij             ] O(1)
          if(d^k_ij > (d^{k-1}_ik + d^{k-1}_kj))
            d^k_ij = d^{k-1}_ik + d^{k-1}_kj
    return D^n
                                          O(V^3)
```