APPELLO A.S.D. 13 GIUGNO 2013

① IL COSTO DELLA PROCEDURA RICORSIVA PUÓ ESSERE ESPRESSO DALLA SEGUENTE RELAZIONE DI RICORRENZA:

$$T(n) \begin{cases} d & \text{per } n < 17 \\ 2T\left(\frac{n}{2}\right) + cn & \text{per } n \geq 17 \end{cases}$$

IN QUANTO SE $n \geq 17$ LA PROCEDURA mistery VIENE CHIAMATA RICORSIVAMENTE 2 VOLTE SU SOTTOPROBLEMI DI DIMENSIONE $n/2$ CIASCUNO. INOLTRE, SI HA UN COSTO DI RICOMBINAZIONE LINEARE (cn) DOVUTO AL CICLO FOR CHE VA DA $i \leftarrow n$ A $\lfloor n/2 \rfloor$. PER IL TEO. DELLE RICORRENZE LINEARI CON PARTIZIONE BILANCIATA, POSTI $a = b = 2$, $\beta = 1$ E $\alpha = \frac{\log a}{\log b} = 1$ LA COMPLESSITA' RISULTA $O(n^\alpha \log n) = O(n \log n)$.
SE SI CONSIDERA LA DIMENSIONE DELL'INPUT d, SUPPONENDO PER SEMPLICITA DI ESSERE IN BASE 2 (SI PUÓ IN REALTA' GENERALIZZARE PER OGNI BASE $b > 1$) SI HA CHE d E' UN $O(\log n)$ DA CUI SI RICAVA CHE LA COMPLESSITA' NELLA DIM. DELL'INPUT E' $O(2^d \log 2^d) = O(2^d \cdot d)$ CIOE' ESPONENZIALE.

② UN ALGORITMO EFFICIENTE RISOLVE QUESTO PROBLEMA IN $O(n)$, UTILIZZANDO UNA SUFFICIENTEMENTE GRANDE TABELLA HASH H. L'ALGORITMO DAPPRIMA SCORRE OGNI ELEMENTO L_i DI L PER $i = 1, \dots, n$ MEMORIZZANDO IN H COPPIE CHIAVE-VALORE (L_i, C_i) DOVE C_i E' IL NUMERO DI OCCORRENZE DI L_i IN L. QUINDI, SI SCORRE L UNA SECONDA VOLTA E PER OGNI L_i SI CONTROLLA IL CORRISPONDENTE VALORE C_i IN H: SE $C_i = 1$, L_i VIENE RIMOSSO DA L ED INSERITO IN M; SE $C_i > 1$ PROSEGUO LA SCANSIONE.
SE H E' SUFFICIENTEMENTE GRANDE ($2n$ E' UNA BUONA SCELTA) ALLORA IL COSTO DI RICERCA E' MEDIAMENTE $O(1)$, QUINDI IL COSTO COMPLESSIVO DELL'ALGORITMO E' $O(n)$ NEL CASO MEDIO. SEGUE LO PSEUDO-CODICE:

```
REMOVE_DUPLICATES (LIST L, LIST M)
    POS p ← L.head()
    integer n ← 0
    while not L.finished(p) do
        n ← n+1
        p ← L.next(p)

    HASH H ← Hash(2n)
    p ← L.head()
```

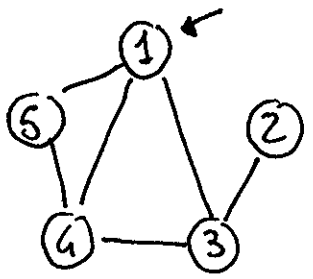
```
while not L.finished(p) do
    integer l ← L.read(p)
    integer c ← H.lookup(p)
    if c = nil then
        H.insert(l, 1)
    else
        H.insert(l, c+1)
    p ← L.next(p)
p ← L.head()
while not L.finished(p) do
    int c ← H.lookup(l)
    if c = 1 then
        M.insert(M.tail(), l)
        L.remove(p)
    else
        p ← L.next(p)
```

③ L'ALGORITMO SCENDE RICORSIVAMENTE L'ALBERO DALLA RADICE FINO AL LIVELLO K, EFFETTUANDO GLI OPPORTUNI CONTROLLI:

```
REMOVE_LEAVES (TREE t, integer k)
    if k<0 or t = nil or (t.parent() = nil and k = 0)
        return
    if k=0 and t.left() = nil and t.right() = nil and t.parent().read() = t.read()
        then
            if t = t.parent().left() then
                t.parent().deleteLeft()
            else
                t.parent().deleteRight()
    REMOVE_LEAVES(t.right(), k-1)
    REMOVE-LEAVES(t.left(), k-1)
```

NEL CASO PEGGIORE, IL COSTO E' $O(n)$ (DEVO VISITARE TUTTI I NODI DI t).

④ PER I DETTAGLI DELLA DFS, SI VEDA LA SEZ. 9.5.4 PAG. 167 DEL LIBRO



ORDINE DI VISITA DI ARCHI E NODI:

① [1,3] ③ [3,1] [3,2] ② [2,3] [3,4] ④ [4,1], [4,3], [4,5]
⑤ [5,1] [5,4] [1,4] [1,5]

⑤ PER I DETTAGLI DELL'ALGORITMO, SI VEDA LA SEZ. 14.3.1 PAG. 268.
L'ESECUZIONE DELL'ALGORITMO È LA SEGUENTE:

PASSO 0. $A=[[1,3],[4,5],[1,4],[1,5],[3,4],[2,3]]$ $n=5$, $m=6$
$M=\{1\}\ \{2\}\ \{3\}\ \{4\}\ \{5\}$ $T=\emptyset$ $c=0$, $i=1$

PASSO 1. $M=\{1,3\}\ \{2\}\ \{4\}\ \{5\}$ $T=\{[1,3]\}$ $c=0$, $i=1$
" 2. $M=\{1,3\}\ \{4,5\}\ \{2\}$ $T=\{[1,3],[4,5]\}$ $c=1$, $i=2$
" 3. $M=\{1,3,4,5\}\ \{2\}$ $T=\{[1,3],[4,5],[1,4]\}$ $c=3$, $i=4$
" 4. $M=$ " " , $T=$ " " " " , $i=5$
" 5. " " " " " " " " , $i=6$
" 6. $M=\{1,..,5\}$ $T=\{[1,3],[4,5],[1,4],[2,3]\}$ $c=4$, $i=7$

L'ALBERO DI COPERTURA RISULTANTE È:

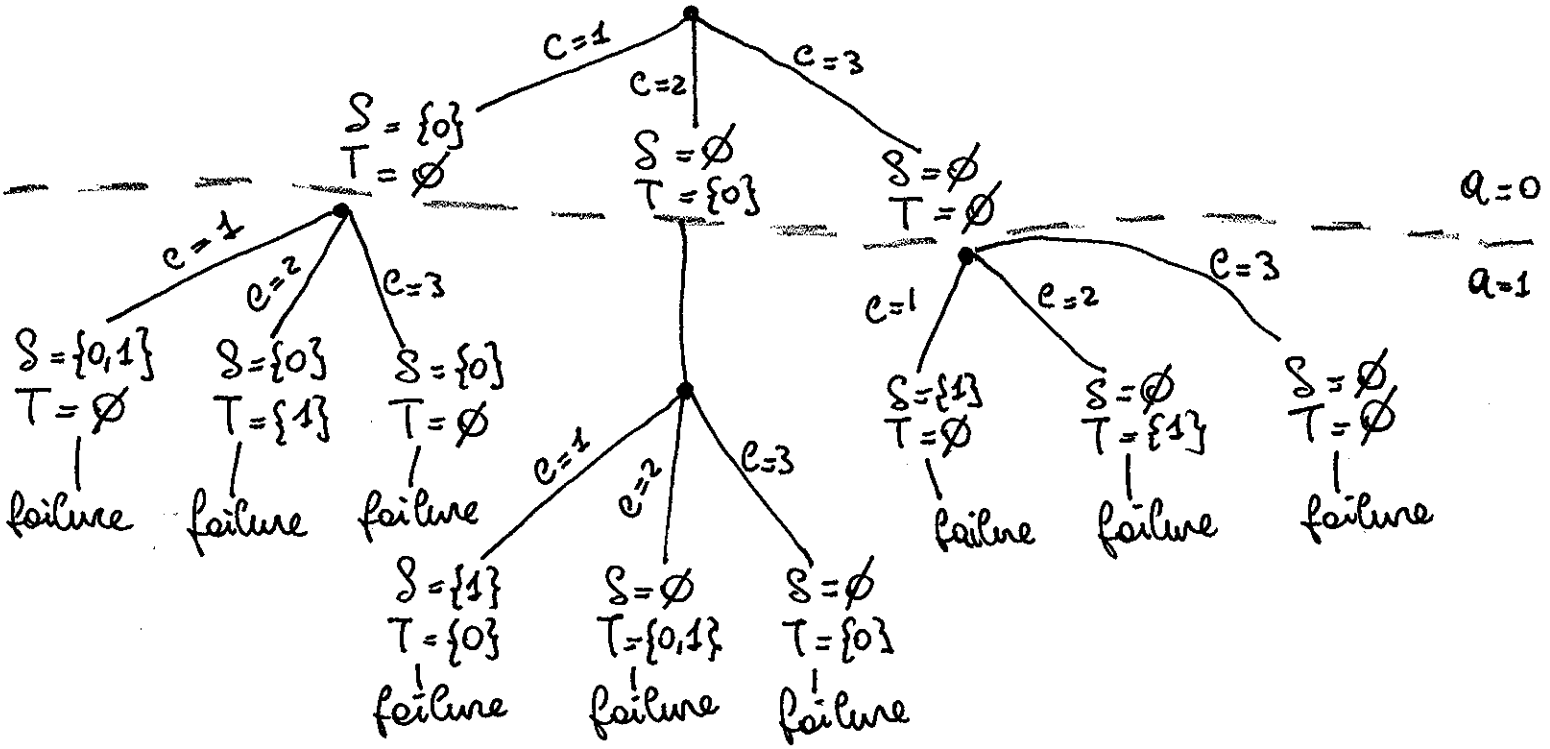
IL CUI PESO COMPLESSIVO È $W(T)=12$.

⑥ L'ALGORITMO NON-DETERMINISTICO AD OGNI PASSO USA L'ISTRUZIONE $c=$ CHOICE $(\{1,2,3\})$ PER SCEGLIERE ARBITRARIAMENTE LA STRADA DA SEGUIRE;
PER OGNI ELEMENTO $a \in A$ INFATTI:
- SE $c=1$, AGGIUNGO a IN S (E AGGIORNO IL PRODOTTO DEI SUOI ELEMENTI)
- SE $c=2$, " " " T (" " LA SOMMA " " ")
- SE $c=3$, NON FACCIO NULLA

SI NOTI INFATTI CHE S E T DEVONO ESSERE NON VUOTI E DISGIUNTI, MA CIÓ NON IMPLICA CHE DEBBANO ESSERE UNA PARTIZIONE DI A (CIOÈ, $S \cup T = A$).
TALE ALGORITMO, DI CUI SEGUE LO PSEUDO-CODICE, È EVIDENTEMENTE POLINOMIALE: LA SCANSIONE DI A COSTA $O(n)$ (DOVE $n=|A|$) MENTRE LA VERIFICA COSTA $O(1)$.

```
FIND_SUBSETS(SET A)
    integer n ← A.size()
    SET S ← Set(n)
    SET T ← Set(n)
    integer sumT ← 0
    integer prodS ← 1
    foreach a∈A do
        c = CHOICE({1,2,3})
        if c=1 then
            S.insert(a)
            prodS ← prodS * a
        else
            if c=2 then
                T.insert(a)
                sumT ← sumT + a
    if S.size()≠0 and T.size()≠0 and (intersection(S,T)).size()≠0
    and 2*prodS = 3*sumT then
        success
    else
        failure
```

AD ES. SUPPONIAMO $A=\{0,1\}$, L'ESECUZIONE E' LA SEGUENTE:



IN QUESTO CASO, FIND_SUBSETS(A) FORNISCE RISPOSTA "NO" (TUTTE LE COPIE ESEGUONO UN'ISTRUZIONE failure).