II° APPELLO DI ALGORITMI E STRUTTURE DATI
7 Gennaio 2014 Prof. Livio COLUSSI

STUDENTE: MATRICOLA:

1. Trovare la soluzione della ricorrenza $T(n) = 2T(n/\sqrt{2}) + n^{9/4} \log n$ usando il metodo dell'esperto.

2. Dire quali sono le operazioni fondamentali su di una coda con priorità e come si possa implementare una tale coda usando un array S. Scrivere lo pseudocodice dell'operazione CHANGE-PRIORITY(S, i, p) che cambia la priorità all'elemento i-esimo $S[i]$ dell'array assegnando il nuovo valore p alla sua priorità $S[i].key$.

3. La seguente versione dell'algoritmo MERGE-SORT divide l'array di dimensione n in tre parti invece che in due parti: una parte centrale di lunghezza $n2 = \lfloor n/5 \rfloor$, una parte iniziale di lunghezza $n1 = \lfloor (n - n2)/2 \rfloor$ ed una parte finale di lunghezza $n3 = \lceil (n - n2)/2 \rceil$.

```
MERGE-SORT-TERNARIO(A, p, r)
 1  n = r − p + 1
 2  if n < 5
 3      // Usa INSERT-SORT per ordinare A[p..r]
 4      INSERT-SORT(A, p, r)
 5  else n2 = ⌊n/5⌋, n1 = ⌊(n − n2)/2⌋, n3 = ⌈(n − n2)/2⌉
 6      MERGE-SORT-TERNARIO(A, p, p + n1 + n2 − 1)
 7      // A[p..p + n1 + n2 − 1] è ordinato.
 8      MERGE-SORT-TERNARIO(A, p + n1, r)
 9      // A[p + n1..r] è ordinato, A[p..p + n1 − 1] è rimasto
          ordinato e gli ultimi n2 elementi dell'array sono maggiori
          o uguali di tutti i precedenti.
10      MERGE(A, p, p + n1 − 1, r − n2 − 1)
```

Valutare la complessità asintotica di tale algoritmo.

4. Scrivere un algoritmo che data una chiave k ed un albero rosso-nero aumentato con il campo *size* calcola il numero di nodi dell'albero con chiave maggiore o uguale di k. Attenzione: nell'albero possono esserci più nodi con la stessa chiave.

5. Usare la programmazione dinamica per risolvere il seguente problema: Date due stringhe X ed Y di lunghezze rispettive m ed n calcolare la distanza di

editing tra le due stringhe, ossia il minimo numero di errori di ricopiatura necessari a trasformare la stringa X nella stringa Y. Gli errori di ricopiatura possibili sono i seguenti:

a) inserimento di un carattere;

b) cancellazione di un carattere;

c) trasformazione di una doppia in singola (Es. aa trasformata in a).

Mostrare che il problema ha sottostruttura ottima e ripetizione dei sottoproblemi e scrivere la soluzione ricorsiva.

6. Una azienda di una grande città ha diverse agenzie. Su richiesta sindacale l'azienda ha istituito un servizio di trasporto per i dipendenti. A tale scopo la ditta utilizza un autubus che al mattino effettua un percorso che passa per tutte le agenzie raccogliendo i dipendenti nei punti del percorso per loro più comodi per portarli alle rispettive agenzie. Per ciascuno degli n dipendenti è noto il punto di partenza $s[i]$ e il punto di arrivo $f[i]$. Scrivere un algoritmo che determina il numero minimo m di posti che deve avere l'autobus per poter trasportare tutti i dipendenti. L'algoritmo deve anche determinare l'assegnazione del posto a ciascun dipendente. Lo stesso posto può essere utilizzato da più dipendenti purché i tragitti non si sovrappongano.

7. In un B-albero T ci possono essere più occorrenze di una stessa chiave k (di solito con informazioni associate diverse). Scrivere una funzione NUMK(T, k) che dato un B-albero T ed una chiave k calcola il numero di occorrenze della chiave k. Si assuma che, al fine di facilitare tale operazione, il B-albero sia stato aumentato aggiungendo ad ogni nodo x i campi $x.size_i$, $i = 1, \ldots, x.n + 1$ che contengono il numero totale di chiavi presenti nei sottoalberi $x.c_i$.

Moretto Alessandro

1 $T(n) = 2T(n/\sqrt{2}) + n^{9/4}\log n$

$a = 2 \quad b = \sqrt{2} \qquad \log_b a = \log_{\sqrt{2}} 2 = 2$

$\lim_{n\to\infty} \frac{n^{9/4}\log n}{n^2} = \infty$ caso 3?

$\lim_{n\to\infty} \frac{n^{9/4}\log n}{n^{2+\varepsilon}}$ deve risultare $\neq 0$ per un $\varepsilon > 0$

$2+\varepsilon < 9/4 \qquad \varepsilon < 9/4 - 2 \qquad \varepsilon < \frac{1}{4} \qquad \varepsilon = \frac{1}{8}$

$af(n/b) \leq kf(n)$

$2\cdot\left(\frac{n}{\sqrt{2}}\right)^{9/4}\log\frac{n}{\sqrt{2}} \leq k\, n^{9/4}\log n$

$\frac{2}{(\sqrt{2})^{9/4}}\cdot\log\frac{n}{\sqrt{2}} \leq k\, n^{9/4}\log n$ nei log si verificano le condizioni per tirarli via

$\frac{2}{2^{\frac{1}{2}\cdot\frac{9}{4}}} \leq k \qquad \frac{2}{2\cdot 2^{1/8}} \leq k \qquad k \geq \frac{1}{\sqrt[8]{2}}$

2 Una coda di priorità è facilmente implementabile attraverso un Heap ordinato secondo il campo priorità di ogni elemento. Le operazioni fondamentali sono:

- Insert (S, x) = aggiunge x alla coda S
- Maximum (S) = ritorna $x \in S$ con x.key massima
- ExtractMax (S) = " " " " " e la rimuove dalla coda
- IncreaseKey (S, x, p) = aumenta a p la priorità x.key dix
- ChangePriority (S, i, p) = cambia la priorità dell'i-esimo nodo della coda a p

```
ChangePriority (S, i, p)
  old = S[i].key
  S[x].key = p
  if old < S[x].key
     HeapfyR (S, x)
  else
     Heapfy (S, x)
```

3 $r = n+p-1$

$n_1 = \left\lfloor \frac{n - \lfloor \frac{n}{5} \rfloor}{2} \right\rfloor = \left\lfloor \frac{4}{5} n \cdot \frac{1}{2} \right\rfloor = \left\lfloor \frac{2}{5} n \right\rfloor$

$n_3 = \left\lceil \frac{n - \lfloor \frac{n}{5} \rfloor}{2} \right\rceil = \left\lceil \frac{2}{5} n \right\rceil$

$$T(n) = \begin{cases} c & n < 5 \\ = T(p + n1 + n2 - 1 - p + 1) + T(n + p - 1 - p - n1 + 1) + \\ \quad + T_M(n + p - 1 - n2 - 1 - p + 1) & n \geq 5 \\ = T\left(\left\lfloor \frac{2}{5} n \right\rfloor + \left\lfloor \frac{n}{5} \right\rfloor\right) + T\left(n - \left\lfloor \frac{2}{5} n \right\rfloor\right) + T_M\left(n - \left\lfloor \frac{n}{5} \right\rfloor - 1\right) \\ = T\left(\frac{3}{5} n\right) + T\left(\frac{3}{5} n\right) + T_M\left(\frac{4}{5} n - 1\right) \\ = 2T\left(\frac{3}{5} n\right) + T_M\left(\frac{4}{5} n - 1\right) \end{cases}$$

esperto:

$a = 2 \quad b = \frac{5}{3} \quad \log_{\frac{5}{3}} 2 \cong 1{,}36 \quad f(n) = c_1 n + c_2$

$\lim_{n \to \infty} \frac{c_1 n + c_2}{n^{1{,}36\ldots}} = 0$ caso 1 ?

$\lim_{n \to \infty} \frac{c_1 n + c_2}{n^{1{,}36 - \varepsilon}}$ deve esser finito, lo è per

$1{,}36 - \varepsilon > 1 \qquad \varepsilon < 1{,}36 - 1 \qquad \varepsilon < 0{,}36$

per esempio $\varepsilon = 0{,}10$

$\Rightarrow T(n) = \Theta\left(n^{\log_{\frac{5}{3}} 2}\right)$

4
```
MAGUG(T, k)
  if T=nil
    return 0
  if T.key < k
    return MAGUG(T.right, k)
  return 1 + T.right.size + MAGUG(T.left, k)
```

5 Sottostruttura ottima:

Indico con: I_a l'inserimento del carattere a, D_a la cancellazione del carattere a, $T_{aa,a}$ la trasformazione di una doppia in singola (aa in a), ed infine C_a il ricopiamento corretto del carattere a.

Sia S una soluzione ottima del problema (seq. di operaz.) L'ultima operazione può esser una di quelle citate prima:

- $S = S'I_a \rightarrow S'$ è una sequenza di operazioni che trasforma X in Y_{n-1} (seq Y privata dell'ultimo carattere). S' deve essere ottima. Se esistesse S'' tale che il numero di oper. in S'' è minore al num di op. di S' si otterrebbe una soluzione migliore di S, il che è assurdo.
- $S = S'D_a \rightarrow S'$ è una seq. di operaz. che trasforma X_{m-1} in Y. Anche qui come in precedenza si dimostra per assurdo che S' è soluz. ottima.
- $S = S'T_{aa,a} \rightarrow S'$ è una sequenza di operazioni che trasforma X_{m-2} in Y_{n-1}. Anche qui come in precedenza si dimostra per assurdo che S' è sol. ottima di X_{m-2} e Y_{n-1}.
- $S = S'C_a \rightarrow S'$ è una sequenza di operazioni che trasforma X_{m-1} in Y_{n-1}, e come dimostrato prima è sicuram. ottima.

Ripetizione dei sottoproblemi:

Tuttavia, invece di dimostrare la cosa in modo rigoroso, mostro con un esempio (ma ve ne sono molti altri) per mostrare che i sottoproblemi effettivamente si ripetono. Suppongo di confrontare le stringhe X_i e Y_J (i e J lunghezze). Se l'ultima operazione della soluzione ottima è I_a allora devo confrontare X_i e Y_{J-1}. Se poi la penultima operazione è D_a allora devo confrontare X_{i-1}, Y_{J-1} ($S = S''D_aI_a$). Tuttavia, anche se le ultime due operazioni fossero permutate dobbiamo sempre confrontare X_{i-1} e Y_{J-1} ($S = S''I_aD_a$) ed il problema da risolvere sarebbe lo stesso di prima. Dunque due soluzioni diverse portano ad uno stesso sotto-problema (di esempi ce ne sono moltissimi altri) $\Rightarrow$ dimostrata la proprietà

Soluzione Ricorsiva:

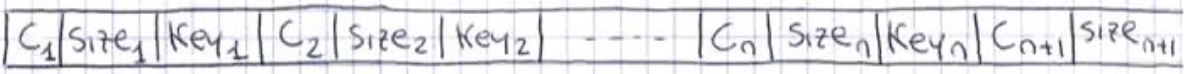
Indico con $e_{i,j}$ il minimo numero di errori necessario per trasformare la stringa X_i nella stringa Y_j.

- Se $i=0$ allora X_i è vuoto e per passare da X_i a Y_j sono necessari j inserimenti $\Rightarrow e_{i,j} = j$
- Se $j=0$ allora Y_j è vuoto e sono necessarie i cancellazioni $\Rightarrow e_{i,j} = i$
- Se $X_i = Y_j$ $i,j > 0$ (attenzione $Y_j \neq X_{i-1}$) allora l'ultima operazione potrebbe essere C_a, I_a o D_a. Se fosse C_a $\rightarrow e_{i,j} = e_{i-1,j-1}$; se fosse $I_a \rightarrow e_{i,j} = e_{i,j-1} + 1$ mentre se fosse $D_a \rightarrow e_{i,j} = e_{i-1,j} + 1$.
  Quindi la soluzione ottima sarà $\rightarrow$
  $e_{i,j} = \min(e_{i-1,j-1}, e_{i-1,j} + 1, e_{i,j-1} + 1)$
- Se $X_i = Y_j$ e $i > 1$ e $j > 0$ e $X_{i-1} = Y_j$ allora l'ultima operazione potrebbe essere stata C_a, I_a, D_a o $T_{aa,a}$ se fosse $T_{aa,a} \rightarrow e_{i,j} = e_{i-2,j-1} + 1$ e la soluz. sarebbe:
  $e_{i,j} = \min(e_{i-1,j-1}, e_{i-1,j} + 1, e_{i,j-1} + 1, e_{i-2,j-1} + 1)$
- Se $X_i \neq Y_j$ e $i,j > 0$ allora potrei aver avuto solamente I_a o D_a e quindi $e_{i,j} = \min(e_{i,j-1} + 1, e_{i-1,j} + 1)$

6

```
PostiMinimi (n, s, f)        // s1 ≤ s2 ≤ --- ≤ sn
    m = 0                    // posti iniziali
    for i = 1 to n           // Dipendenti
        h = m + 1            // ipotizzo mi serva un posto in più
        for j = 1 to m       // posti
            if s_i ≥ t_j
                h = j
        J[i] = h             // indica il posto nel quale si siede il dip.i
        t_h = f_i
        if h == m + 1
            m = m + 1        // se mi è servito aggiungo il posto
    return m, J
```

7




```
NUMK(T,K)
    if T.root = nil
        return 0
    return NUMKEYRIC(T,K)
```

```
NUMKEYRIC(T,K)
    m=0
    i,j=1
    while j≤T.n and T.key_j ≤ K
        if T.key_j = K
            m = m+1+T.size_j
        else
            if T.key_j < K
                i++
        j++
    if i != T.n+1
        if not T.leaf
            DISKREAD(T.C_i)
            m=m+NUMKEYRIC(T.C_i)
            DISKREAD(T.C_j+1)
            m=m+NUMKEYRIC(T.C_j+1)
        return m
    if not T.leaf
        DISKREAD(T.C_i)
        m=NUMKEYRIC(T.C_i)
        return m
    return 0
```