ASD (FI2 12 CFU - tutti), ASD (5 CFU)

Appello del 14-9-2015 – a.a. 2014-15 – Tempo a disposizione: *120 minuti* – somma punti in gioco: 33

Problema 1

`[Punti: (a) 4/30; (b) 4/30]`

Si considerino i metodi Java di seguito illustrati, in cui si fa uso della ricorsione multipla.

```
static boolean isPDS(int[][] mat, int i, int j) {
  if((i < 1) || (j < 1)) return false;
  if((i > mat.length - 2) || (j > mat[0].length - 2)) return false;
  int p = mat[i][j], nord = mat[i-1][j], est = mat[i][j+1],
      sud = mat[i+1][j], ovest = mat[i][j-1];
  if((p > nord) && (p > sud) && (p < est) && (p < ovest)) return true;
  if((p < nord) && (p < sud) && (p > est) && (p > ovest)) return true;
  return false;
}

static int contaPDS(int[][] mat, int up, int right, int down, int left) {
  if((up > down) || (left > right)) return 0;
  if((up == down) && (left == right))
    if(isPDS(mat, up, left)) return 1; else return 0;
  int o = (up+down)/2, v = (left+right)/2;
  return contaPDS(mat, up, v, o, left) + contaPDS(mat, up, right, o, v+1) +
         contaPDS(mat, o+1, v, down, left) + contaPDS(mat, o+1, right, down, v+1);
}

static int contaPDS(int[][] mat) {
  return contaPDS(mat, 0, mat[0].length, mat.length, 0);
}
```

Sviluppare, *argomentando adeguatamente* (il 50% del punteggio dell'esercizio sarà sulle argomentazioni addotte), quanto segue:

(a) Scrivere e risolvere un'equazione di ricorrenza per determinare il costo computazionale $T(m, n)$ di `contaPDS(int[][])`, in funzione del numero di righe m e del numero di colonne n della matrice in input.

(b) Scrivere e risolvere un'equazione di ricorrenza analoga alla precedente per determinare la funzione $T'(z)$ che esprime lo stesso costo, ma questa volta in funzione di z, dimensione dell'input.

Bozza soluzione caso (a)

$$T(m,n) = \begin{cases} c' & (m \leq 1) \wedge (n \leq 1) \\ 4T(m/2, n/2) + c & (m > 1) \vee (n > 1) \end{cases}$$

$$T(m,n) = 4T(m/2,n/2) + c = 4^2T(m/2^2, n/2^2) + 2c = 4^3T(m/2^3, n/2^2) + 3c = \cdots$$

$$= 4^kT(m/2^k, n/2^k) + kc$$

Per $k = \max\{\log_2 m, \log_2 n\} = \log_2 \max\{m,n\}$ otteniamo $T(m,n) = 4^{\log_2 \max\{m,n\}}c' + c\log_2 \max\{m,n\} = (2^2)^{\log_2 \max\{m,n\}}c' + c\log_2 \max\{m,n\} = 2^{2\log_2 \max\{m,n\}}c' + c\log_2 \max\{m,n\} =$

$2^{\log_2 \max^2\{m,n\}} c' + c \log_2 \max\{m,n\} = c' \max^2\{m,n\} + c \log_2 \max\{m,n\} \in \Theta(\max^2\{m,n\}) = \Theta(m^2 + n^2)$

Problema 2

[Punti: 8/30]

Dato un albero generico τ e due suoi nodi u e v, definiamo *least common ancestor* di u e v, usando la notazione LCA(u, v), quel nodo di τ che sia, fra tutti gli antenati[1] comuni di u e v, il più distante possibile dalla radice. Si noti che LCA(u, v) esiste sempre perché, dati due nodi, l'insieme degli antenati comuni contiene almeno la radice.

Ciò premesso si richiede di scrivere un metodo Java, ovvero una funzione C, che dato un albero generico e due suoi nodi u e v, restituisca LCA(u, v). L'algoritmo usato deve essere il più efficiente possibile e se ne deve valutare (e giustificare) il costo computazionale.

Che tipo di visita è stata eseguita?

Bozza idea (qui solo pseudocodice, ma la soluzione va scritta in Java o C)

Si esegue visita in post-ordine, costo $\Theta(n)$. L'algoritmo ricorsivo, chiamato su un un nodo x, restituisce una coppia di informazioni: un `int` (num: quanti fra i nodi u e v si trovano nel sottoalbero di radice x) e un Nodo (lca: LCA(u, v), se appartenente al sottoalbero di radice x, o `NULL` altrimenti). Pseudocodice (si assume che x, u e v siano tutti diversi da `NULL` e che $u \neq v$, casi banali facilmente gestiti prima di chiamare `getLCA`; notare che successivamente x potrebbe essere `NULL`):

```
(int num, Nodo lca) getLCA(Nodo x, Nodo u, Nodo v)
  if(x == NULL) return (0, NULL)
  cont = 0
  for(Nodo t = x.firstChild; t != NULL; t=t.nextSibling)
    p = getLCA(t, u, v)
    if(p.num == 2) return p // LCA è in sottoalbero di radice t
    cont += p.num
  if(x == u) or (x == v) cont++
  if(cont == 0) return (0, NULL)
  if(cont == 1) return (1, NULL)
  if(cont == 2) return (2, x) // LCA è x
  // non può essere cont > 2 o cont < 0
```

Problema 3

[Punti: 8/30]

Descrivere (solo pseudo-codice[2]) gli algoritmi HeapSort e SelectionSort, discutendone similarità e differenze. Determinare, illustrandone il ragionamento, i costi computazionali dei due algoritmi.

Problema 4

[Punti: 9/30]

All'interno di un grande edificio sono posizionate n workstation (WS) che debbono essere fra loro interconnesse attraverso collegamenti di elevata qualità (basati su fibra ottica). Il collegamento fra due generiche WS può essere *diretto* (attraverso un cavo in fibra dedicato) o *indiretto* (prevedendo il transito fra WS intermedie direttamente interconnesse che inoltreranno le comunicazione in modo opportuno, grazie a uno specifico software di routing).

[1] Gli antenati di un nodo v sono tutti i nodi che si trovano sul percorso dalla radice a v (incluso).

[2] Lo pseudo-codice non deve avvalersi di primitive algoritmicamente significative.

I progettisti debbono collegare con fibra tutte le n WS usando collegamenti diretti e/o indiretti. Dato l'alto costo della fibra se ne desidera minimizzare la lunghezza impiegata.[3] Si assume di conoscere, per ciascuna possibile coppia di WS, la lunghezza del cavo per il loro collegamento diretto.

(a) Formulare il problema dei progettisti come problema su grafo. A tal fine occorre dapprima definire il grafo a cui si farà riferimento, definendone nodi ed archi e chiarendo come esso sia collegato al problema in esame; successivamente va formalizzato il problema come problema su tale grafo, utilizzando una terminologia appropriata.

(b) Definire un algoritmo (pseudo-codice) per la soluzione generale del problema: collegare le WS minimizzando la lunghezza del cavo impiegato. Tale soluzione dovrà specificare quali WS debbano essere fra loro direttamente collegate.

(c) Con riferimento a una specifica modalità di rappresentazione di grafi, determinare i costi computazionali dell'algoritmo (tempo e spazio). La modalità di rappresentazione scelta sarà oggetto di valutazione.

[3] Assumere che un cavo lungo ℓ possa essere frazionato a costo zero in più cavi, la somma delle cui lunghezze è proprio ℓ.