Esame I.A. 19 giugno 2020

Soluzioni

1 Domande di riscaldamento

Quali delle seguenti strutture dati può essere definita "lineare"?

- una coda
- una pila
- un array
- ✓ tutte le risposte sono corrette

Un algoritmo ricorsivo tende ad utilizzare più memoria del corrispondente algoritmo iterativo perché:

- Richiede più passi computazionali per trovare la soluzione
- ✓ Le chiamate ricorsive devono essere memorizzate
- Gli algoritmi iterativi possono sfruttare il "divide et impera"
- Gli algoritmi ricorsivi possono sfruttare la "programmazione dinamica"

Il tempo di ricerca in un albero binario di ricerca può degradare fino a:

- $O(n^2)$
- $O(n \log n)$
- ✓ $O(n)$
- $O(1)$

La complessità della ricerca binaria nel caso peggiore corrisponde a quella di:

- radix sort
- ricerca lineare
- merge sort
- ✓ nessuna delle altre risposte

Quanti grafi non diretti (non necessariamente connessi) possono essere costruiti da un dato insieme $V = \{V_1, V_2, \ldots, V_n\}$ di n vertici?

- $\frac{n(n-1)}{2}$
- 2^n
- $n!$
- ✓ $2^{\frac{n(n-1)}{2}}$

2 Risoluzione di problemi

Mentre preparavo questo esame “particolare", mi è venuto in mente di fare una domanda sulla somma massimale. Però il tempo per preparare l'esame, soprattutto a causa delle svariate prove tecnologiche che abbiamo fatto prima di scegliere la modalità, era particolarmente scarso.

Sono andato quindi anche io a cercare un po' di domande da farvi su Internet, e ho trovato proprio un esercizio uguale a quello che volevo darvi. Ho copiato e incollato una lista di numeri da questo sito che ho trovato ma, stupidamente, ho chiuso la tab prima di verificare cosa avevo copiato. Purtroppo, gli spazi che separavano i numeri del vettore sono scomparsi e questo è il risultato: “941538723547796".

Non ho più trovato il sito che avevo trovato in precedenza, ma mi ricordavo soltanto che la somma era pari a "377". Mi sono quindi chiesto: in quanti modi è possibile suddividere questa stringa (“941538723547796") in input in maniera tale da trovare un certo insieme di numeri che, se sommati, danno proprio la somma che mi ricordo?

Aiutatemi scrivendo un algoritmo (in python!) che prenda in input una stringa di numeri di lunghezza n, rappresentata come vettore di interi, in cui ciascun elemento è compreso tra 1 e 9 (escludiamo lo 0!) e un intero $k \leq 100n$, e restituisca il numero di modi in cui si può rappresentare questa stringa come sommatoria di interi positivi il cui valore è pari a k.

Supponete di avere a disposizione una funzione Valore(V, i, j) che restituisce il valore intero delle cifre nel vettore V tra le posizioni i e j—ad esempio, Valore(“12345", 2, 4) restituisce 234.

Incredibilmente, ci sono 34 modi diveri per ottenere come somma 377 suddividendo le cifre di 941538723547796.

Indentate il codice con gli spazi, almeno nel form si leggerà bene.

Soluzione Per identificare una soluzione efficace—più efficace del brute force o del backtracking, che sono comunque possibili—ragioniamo su come possiamo suddividere le cifre di un numero per arrivare ad una certa somma. In particolare i casi banali da prendere in considerazione sono i seguenti:

- se la somma a cui devo arrivare è zero e non ho nessun termine da sommare, ho una sola combinazione;
- se la somma a cui devo arrivare è negativa, non ho alcuna possibilità (effettuiamo solo somme tra valori positivi ed escludiamo lo zero);
- se la somma a cui devo arrivare è positiva, ma non ho alcun termine da sommare, non ho alcuna possibilità.

In generale, se voglio arrivare ad una somma r utilizzando soltanto le prime i cifre del mio numero in input, posso spezzare le i cifre in due parti, diciamo alla cifra s: $V[1 \ldots s-1]$ e $V[s \ldots i]$. Possiamo interpretare le cifre da s ad i come un singolo addendo. Per vagliare le altre possibilità, devo fare tutte le prove spezzando (ricorsivamente) le cifre tra 1 ed $s-1$, applicando lo stesso schema.

Questo ragionamento può essere formalizzato con la seguente relazione:

$$DP[i][r] = \begin{cases} 1 & i = 0 \wedge r = 0 \\ 0 & r < 0 \\ 0 & i = 0 \wedge r > 0 \\ \sum_{s=1}^{i} DP[s-1][r - \textsc{Value}(V, s, i)] & \text{altrimenti} \end{cases}$$

Il risultato al problema sarà nella posizione $DP[n][k]$. La relazione ricorsiva qui sopra può immediatamente essere tradotta nel seguente algoritmo ricorsivo—viene fornita anche una possibile implementazione di VALUE().

```
def value(V, s, e):
    return sum(el * 10 ** (e - s - i) for i, el in enumerate(V[s - 1:e]))


def countRec(V, i, r, DP):
    if i == 0 and r == 0:
        return 1
    if (r < 0) or (i == 0 and r > 0):
        return 0

    if DP[i][r] < 0:
        DP[i][r] = 0
        for s in range(0, i):
            DP[i][r] = DP[i][r] + countRec(V, s, r - value(V, s + 1, i), DP)
    return DP[i][r]


def countSum(V, k):
    DP = [[-1 for i in range(k + 1)] for j in range(len(V) + 1)]
    return countRec(V, len(V), k, DP)
```

Per risolvere lo specifico problema dato nel testo, è possibile invocare il solutore come segue:

```
countSum([9, 4, 1, 5, 3, 8, 7, 2, 3, 5, 4, 7, 7, 9, 6], 377)
```

La funzione countSum() inizializza la matrice a valori "fittizi" -1, che vengono utilizzati poi nell'algoritmo ricorsivo per controllare se stiamo incontrando un valore già calcolato nella tabella ed evitare quindi di ricalcolarlo—questa variante della programmazione dinamica viene tipicamente chiamata "memoization".

Il costo, sia spaziale sia temporale, è piuttosto elevato. La tabella ha dimensione $n \cdot k$, dove $k = O(n)$. Per riempire ogni cella della tabella, sono necessari $O(n^2)$ passi, per un costo complessivo pari a $O(n^4)$.

In quanti modi è possibile suddividere i numeri seguenti, date le somme indicate?

- 8447591929869774 con somma 1062: 73
- 1381371339084125 con somma 1319: 38
- 6853768915211813 con somma 524: 36
- 2265996613987196 con somma 413: 69
- 351345522454547198 con somma 914: 294
- 416667478976616 con somma 1299: 23
- 97172559551283352 con somma 1267: 110
- 3476169396539419711 con somma 1651: 281

3 Domande di defaticamento

Qual è il costo computazionale dei seguenti algoritmi nel caso pessimo?

- Bubble sort: n^2
- Merge sort: $n \log n$
- Ricerca k-simo elemento in un array: $O(1)$
- Eliminazione k-simo elemento da una lista: n
- Estrazione di un elemento da una calendar queue: $O(1)$
- Inserimento di un elemento in un albero binario auto-bilanciato: $\log n$
- Soluzione della Torre di Hanoi: 2^n

Confronto di algoritmi Si consideri la seguente equazione di ricorrenza, parametrizzata rispetto a k:

$$T_k(n) = \begin{cases} k^2 T_k(n/k) + n^{k/2} & n > 1 \\ 1 & n \leq 1 \end{cases}$$

Si supponga che esistano tre algoritmi, con complessità $T_2(n)$, $T_3(n)$ e $T_4(n)$. Quali algoritmi scartereste?

Soluzione. Assumiamo $k > 0$. Applicando il Master Theorem all'equazione in forma parametrica, otteniamo che $\beta = \frac{k}{2}$ e $\alpha = \log_k k^2 = 2$. Poiché α è costante e β è monotono crescente, avremo che tutti gli algoritmi hanno costo minimo fintantoché $\alpha > \beta$, ovvero nel primo caso del Master Theorem. Quindi otteniamo:

$$\alpha > \beta \Rightarrow k < 4.$$

Avremo costo minimo, pertanto, fino a $k = 3$, da cui la risposta:

- $T_2(n)$
- $T_3(n)$
- ✓ $T_4(n)$

Qual è la complessità della seguente funzione?

```
int fun(int n)
{
    int count = 0;
    for(int i = n; i > 0; i /= 2)
        for(int j = 0; j < i; j++)
            count += 1;
    return count;
}
```

L'operazione dominante è indubbiamente l'incremento della variabile `count`. Questa operazione viene eseguita un numero di volte pari a $n + \frac{n}{2} + \frac{n}{4} + \ldots + 1$. La complessità è pari quindi ad $O(n)$.