Risposta alle domande di Algoritmi

17 Febbraio 2010

Domanda 1:

Il problema dell'ordinamento si propone di trovare algoritmi come soluzione di problemi computazionali, dove in input sono presenti un numero qualsiasi di elelmenti e in output la permutazioni di tali elementi ordinati in ordine crescente.

Radix Sort:
Assumiamo che i valori degli elementi dell'array siano interi rappresentabili con al più d cifre in una certa base b. Per ordinare l'array si usa d volte un algoritmo di ordinamento stabile (come CountingSort) per ordinare l'array rispetto a ciascuna delle d cifre partendo dalla meno significativa.

Pseudocodice:

```
RadixSort(A,n)
    for j=1 to d do
        "usa un algoritmo stabile per ordinare A[1..n] rispetto alla j-esima cifra"
(A[1 .. n] è permutazione ordinata di a1,...,an)
```

Complessità:

Teta(d(n+k))

BucketSort:
Assume che i valori da ordinare siano numeri reali in un intervallo sempiaperto [a,b) che per semplicità di esposizione assumiamo sia l'intervallo [0,1). Per ordinare un array A[1..n] divide l'intervallo in n parti uguali e usa un array B[0..n-1] di liste (i bucket) mettendo in B[k] gli A[i] che cadono nella k-esima parte dell'intervallo. Dopo di che riordina ciascuna lista e infine ricopia in A tutti gli elementi delle liste.

Pseudocodice:

```
BucketSort(A)
    n=length(A)
    for i=1 to n do
        "inserisci A[i] nella lista B[[nA[i]]]"
    (a1,...,an sono stati tutti inseriti nelle liste B[0..n-1] e gli elementi in una lista B[i]
    sono minori degli elementi nella lista successiva B[i+1])
    for i=0 to n-1 do
        "ordina la lista B[i] con insertion sort"
    "concatena ordinatamente le liste B[0],B[1],...,B[n-1]"
    (A[1..n] è una permutazione ordinata di a1,...,an)
```

Complessità:

- Caso peggiore: Teta(n^2)
- Caso migliore: Teta(n)
- Caso medio: Teta(n)

Domanda 2:

1 Strutture dati per insiemi disgiunti:
Servono a mantenere una collezione S = {S1,S2,...,Sk} di insiemi disgiunti. Ogni insieme

della collezione è individuato da un rappresentante che è un particolare elemento dell'insieme.
Operazioni sugli insiemi disgiunti:

1) Make-Set(x): aggiunge alla struttura dati un nuovo insieme contenente solo l'elemento x (x non deve comparire in nessun altro insieme della struttura).
2) Find-Set(x): ritorna il rappresentante dell'insieme che contiene x.
3) Union(x,y): unisce i due insiemi contenenti x ed y in un'unico insieme.

Rappresentazione con foreste:
Ogni insieme è rappresentato con un albero i cui nodi, oltre al campo info hanno soltanto un campo parent che punta al padre.
Euristica dell'unione per rango:
Per ogni nodo x manteniamo un campo rank che è un limite superiore all'altezza del sottoalbero di radice x.
Union mette la radice con rango minore come figlia di quella di rango maggiore.

2 Un sottografo del grafo G = (V,E) è un grafo G' = (V',E') tale che $V' \subseteq V$ e $E' \subseteq E$.
Il sottografo di G = (V,E) indotto da $V' \subseteq V$ è il grafo G' = (V',E') tale che:
$E' = \{uv: uv \in E \text{ e } u,v \in V'\}$

3 Gli alberi binari di ricerca sono strutture dati che possono eseguire molte operazioni su insiemi dinamici come: SEARCH, MINIMUM, MAXIMUM, PREDECESSOR, SUCCESSOR, INSERT, DELETE. Possono essere utilizzati sia come dizionari che come code di priorità. Il tempo delle operazioni è proporzionale all'altezza dell'albero. Ogni nodo può aver al massimo due figli rappresentati da record. Un albero binario di ricerca è rappresentato da una struttura dati concatenata in cui ogni nodo è un oggetto. Ogni nodo ha i seguenti campi: puntatore al padre, figlio sx e dx, chiave e dati. La mancanza di un figlio è rappresentata con puntatore a NIL. La radice è l'unico nodo ad avere puntatore al padre NIL. Le proprietà da rispettare sono: tutti i valori dei nodi del sottoalbero sinistro sono minori della radice mentre quelli del sottoalbero destro sono maggiori della radice. La visita si può esegue in profondità con uno dei 3 algoritmi di ricorsione: preordine, inordine, postordine.
La cancellazione ha come argomento il puntatore al nodo z da cancellare e ci sono 3 casi:

1) Z non ha figli quindi il puntatore a z del padre si sostituisce con NIL.
2) Z ha un solo figlio. Il puntatore al padre di z punterà al figlio di z.
3) Z ha due figli, si crea un nuovo collegamento per rimuovere il suo successore y che non ha un figlio sinistro, poi si sostituiscono i dati di z con quelli di y.

4 La tecnica divide-et-impera è una tecnica ricorsiva che divide il problema in sotto problemi dipendenti e li risolve ricorsivamente (approccio top-down). Non si utilizza con problemi che non sono indipendenti perchè possono venire risolti più volte.
La tecnica di programmazione dinamica è una tecnica iterativa che si utilizza quando ci sono sottoproblemi in comune. La soluzione viene costruita a partire da più sotto-problemi non indipendenti potenzialmente ripetuti (approccio bottom-up).
Si applica quando:
Sottostruttura ottimale: E' possibile combinare soluzioni di sottoproblemi per trovare la soluzione di un problema più grande.
Sottoproblemi ripetuti: Un sottoproblema può occorrere più volte.
Spazio dei sottoproblemi: Deve essere polinomiale.
Le fasi della programmazione dinamica sono:
Fase 1: Definire la struttura di una soluzione ottima. Mostrare che una soluzione ottima si ottiene da soluzioni ottime di sottoproblemi.
Fase 2: Definire ricorsivamente la soluzione ottima. La soluzione ottima contiene le soluzioni ottime ai sottoproblemi.

Fase 3: Calcolare il valore di una soluzione ottima con strategia bottom up.
Fase 4: Costruzione di una soluzione ottima a partire dalle informazioni calcolate.

6 Una componente fortemente connessa di un grafo G è un insieme U di vertici contenuto in V tale che per ogni u,v appartente a U esiste un cammino da u a v e da v a u.
Dato un grafo orientato G, il grafo delle componenti fortemente connesse di G è H ed ha come vertici le componenti fortemente connesse di G e un arco da un cfc C ad una cfc C' se e solo se in G c'è un arco che connette un vertice di C ad un vertice di C'.
Per calcolare le componenti fortemente connesse di un grafo ci sono tre fasi:

1) Si usa la visita in profondita su G per ordinare i vertici in ordine decrescente di completamento.
2) Si calcola il grafo trasposto di G.
3) Si esegue la visita in profondita sul grafo trasposto usando l'ordine dei vertici calcolato nella prima fase.

Gli alberi della visita in profondità nel grafo trasposto rappresentano le componenti fortemente connesse.

27 Gennaio 2010

Domanda 2:

1 L'ordinamento topologico di un grafo orientato è un ordinamento lineare dei suoi vertci tale che:

1) Per ogni arco uv appartenenti a E il vertice u precede il vertice v.
2) Per transtitività, se u è raggiungibile da v allora u viene prima di v nell'ordinamento.

L'ordinamento topologico è utilizzato per determinare l'ordine di esecuzione di un insieme di attività in presenza di vincoli di precedenza.
La soluzione diretta consiste nelle seguenti fasi:

1) Si cercano tutti i vertici che non hanno archi incidenti in ingresso.
2) Si stampa il verticie e si elimina il vertice con i suoi archi.
3) Si ripete la procedura fino a che non ci sono più vertici.

5 Tecnica divide et impera: E' una tecnica ricorsiva dove il problema viene diviso in sotto problemi indipendenti che vengono risolti ricorsivamente (strategia top-down). E' utile solo quando i sottoproblemi sono indipendenti altrimenti gli stessi sottoproblemi possono venire risolti più volte.

2 Novembre 2009

Domanda 3:

1 I codici di Huffman vengono utilizzati per comprimere i file. Consentono di risparmiare uno spazio che va dal 20% al 90% in base al tipo di file. Si basano sulla frequenza delle lettere nel file che permettono la costruzione di un codice ottimo. Ogni carattere ha associata una parola codice che può essere di lunghezza fissa o variabile.
Utizza dei codici prefissi che è un codice in cui nessuna parola è prefisso di un altra.
La decodifica avviene in quattro fasi:
Fase 1: Viene trovata la prima parola nel file codificato.
Fase 2: Viene tradotta la parola e scritta nel file di decodifica.
Fase 3: Viene eliminata la parola dal file di codificato.
Fase 4: Si ripete fino a che non è stato decodificato l'intero file.

Dato un file F, un codice C è ottimo per F se non esiste nessun altro codice che permette di risparmiare più spazio. Possono esistere più codici ottimi e sono rappresentato con un albero binario in cui ogni nodo interno ha due figli.
Il principio del codice di Huffman è il seguente: Deve minimizzare la lunghezza del codice delle parole più frequenti e assegnare ai caratteri con meno frequenza i codici corrispondenti ai percosi più lunghi all'interno dell'albero.
Ogni codice è progettato per uno specifico file e le fasi sono le seguenti:
Fase 1: Si ottengono le frequenze dei caratteri.
Fase 2: Si costruisce il codice ottimo.
Fase 3: Si rappresenta il file tramire il codice ottimo.
Fase 4: Si aggiunge al file una rappresentazione del codice ottimo.
L'algoritmo di Huffman è gready perchè ad ogni passo costruisce il nodo interno avente la frequenza minima possibile.

3 Una componente fortemente connessa in un grafo orientato è un insieme massimale di vertici U contenuti in V. Per ogni vertice u,v appartentente a U esiste un cammino che va da u a v e uno che va da v a u.
Si calcola in tre fasi:
Fase 1: Si usa la visita in profondità in G per ordinare i vertici in ordine di tempo di completamento f decrescente.
Fase 2: Si calcola il grafo trasposto G' di G.
Fase 3: Si esegue la visita in profondità su G' usando l'ordine dei vertici calcolato nella prima fase.
Gli alberi della vista in profondità del grafo trasposto rappresentano le componenti fortemente connesse di G.

21 Settembre 2009

Domanda 3:

3 L'ordinamento topologico di un grafo orientato è un ordinamento lineare dei suoi vertci tale che:

1) Per ogni arco uv appartenenti a E il vertice u precede il vertice v.
2) Per transtitività, se u è raggiungibile da v allora u viene prima di v nell'ordinamento.

L'ordinamento topologico è utilizzato per determinare l'ordine di esecuzione di un insieme di attività in presenza di vincoli di precedenza.
Si può utilizzare la visita in profondità e per ordinare i vertici in ordine di tempo decrescente f di completamento oppure la soluzione diretta.
La soluzione diretta consiste nelle seguenti fasi:

1) Si cercano tutti i vertici che non hanno archi incidenti in ingresso.
2) Si stampa il verticie e si elimina il vertice con i suoi archi.
3) Si ripete la procedura fino a che non ci sono più vertici.

20 Luglio 2009

Domanda 3)

3 La visita in ampiezza BFS permette di visitare il grafo a partire da un vertice s detto sorgente e visita sistematicamente tutto il grafo per scoprire tutti i vertici raggiungibili da s. Calcola le distanze minime di ogni vertice da s. Produce anche un albero i cui rami sono cammini di lunghezza minima. La visita espande uniformemente la frontiera tra i vertici scoperti e quelli non ancora scoperti.

L'algoritmo funziona nel seguente modo: assume che il grafo sia rappresentato con liste di adiacenze. Tutti i nodi adiacenti ad un nodo, sono aggiunti ad una coda Q di tipo FIFO che memorizza la frontiera.
Descrizione:

1) All'inizio dell'algoritmo tutti i vertici sono bianchi tranne la sorgente da cui parte l'esecuzione dell'algoritmo.
2) Con l'andare avanti nella visita i vertici del grafo vengono colorati di grigio, ed inseriti nella coda, che in ogni istante conterrà tutti i nodi colorati di grigio, ma dei quali ancora non sono stati visitati gli adiacenti.
3) Quando tutti i nodi adiacenti ad un nodo grigio sono stati visitati e quindi colorati di grigio, il nodo verrà colorato di nero e rimosso da Q.

Il tempo di esecuzione di questo algoritmo è O(V+E).

4 Tecnica divide et impera: E' una tecnica ricorsiva dove il problema viene diviso in sotto problemi indipendenti che vengono risolti ricorsivamente (strategia top-down). E' utile solo quando i sottoproblemi sono indipendenti altrimenti gli stessi sottoproblemi possono venire risolti più volte. Un algoritmo che ne fà utilizzo è il merge-sort.

17 Giugno 2009

Domanda 2:
Ci sono due tecniche per visitare i grafi la DFS (visita in profondità) e la BFS (visita in ampiezza).

La visita in ampiezza BFS permette di visitare il grafo a partire da un vertice s detto sorgente e visita sistematicamente tutto il grafo per scoprire tutti i vertici raggiungibili da s. Calcola le distanze minime di ogni vertice da s. Produce anche un albero i cui rami sono cammini di lunghezza minima. La visita espande uniformemente la frontiera tra i vertici scoperti e quelli non ancora scoperti.
L'algoritmo funziona nel seguente modo: assume che il grafo sia rappresentato con liste di adiacenze. Tutti i nodi adiacenti ad un nodo, sono aggiunti ad una coda Q di tipo FIFO che memorizza la frontiera.
Descrizione:

1) All'inizio dell'algoritmo tutti i vertici sono bianchi tranne la sorgente da cui parte l'esecuzione dell'algoritmo.
2) Con l'andare avanti nella visita i vertici del grafo vengono colorati di grigio, ed inseriti nella coda, che in ogni istante conterrà tutti i nodi colorati di grigio, ma dei quali ancora non sono stati visitati gli adiacenti.
3) Quando tutti i nodi adiacenti ad un nodo grigio sono stati visitati e quindi colorati di grigio, il nodo verrà colorato di nero e rimosso da Q.

Il tempo di esecuzione di questo algoritmo è O(V+E).

La visita in profondità DFS esplora il grafo partendo da un vertice s detto sorgente. Si esplorano gli archi uscenti dal vertice u raggiunto per ultimo. Se viene scoperto un nuovo vertice ci si sposta su tale vertice. Se tutti gli archi uscenti da u portano a vertici già scoperti si torna indietro e si riprende esplorando archi uscenti dal vertice cui u è stato scoperto. Il procedimento continua fino a che sono stati scoperti tutti i vertici raggiungibili dal vertice s iniziale scelto. Se non sono stati tutti scoperti si ripete il procedimento partendo da un vertice non ancora raggiunto (si sceglie una nuova sorgente).
Si utilizzano due marcatempi:

1) d[u] che registra quando è stato scoperto un vertice e colorato di grigio.
2) f[u] che registra quando il vertice è stato completamente visitato e viene colorato di nero.

Pseudocodice DFS:

```
DFS(G)
   for ogni vertice u in V[G]
     do color[u] ← WHITE
       π[u] ← NIL
   time ← 0
   for ogni vertice u in V[G]
     do if color[u] = WHITE
       then DFS-Visit(u)

DFS-Visit(u)
   color[u] ← GREY
     for ogni v in Adj[u]
       do if color[v] = WHITE
         then π[v] ← u
           DFS-Visit(v)
   color[u] BLACK
   f[u] ← time ← time + 1
```

Il tempo di esecuzione di questo algoritmo è O(|V|+|E|)