- Esercizio 21 - alberi binari di ricerca Red-Black
- Esercizio 22 - sotto-sequenza più lunga
- (★) Esercizio 23 - distanza di editing
- Esercizio 24 - massimo valore espressione
- Esercizio 25 - testo giustificato
- Esercizio 26 - arbitraggio

Progetto di laboratorio (algoritmi di selezione)

Scopo del progetto di laboratorio è verificare che lo studente sia in grado di progettare, implementare e analizzare la complessità di un programma usando gli strumenti illustrati durante le lezioni del corso di algoritmi e strutture dati. I programmi prodotti dovranno risolvere alcuni problemi classici di natura computazionale e dovranno essere formalmente corretti. Ogni progetto di laboratorio consiste nella produzione di uno o più programmi che risolvano un problema computazionale assegnato, nella stima empirica dei tempi di esecuzione di tali programmi al variare della dimensione dell'input e di eventuali altri parametri, e nella redazione di una relazione in cui si discutano puntualmente alcune scelte implementative e si faccia un'analisi della stima di complessità.

Di seguito sono riportate le consegne per il progetto di laboratorio dell'anno accademico 2021/22 (per i progetti degli anni accademici passati fare riferimento alla pagina https://elearning.uniud.it/moodle/course/view.php?id=2278). *I progetti devono essere consegnati necessariamente prima dell'iscrizione all'esame finale di teoria, ed entro il termine dell'anno accademico corrente.*

Il progetto richiede l'implementazione e l'analisi dei tempi medi di esecuzione di *tre algoritmi di selezione* (calcolo del k-esimo elemento più piccolo in un vettore non ordinato di interi). I tre algoritmi di selezione, denominati rispettivamente "quick select", "heap select" e "median-of-medians select", dovranno avere le seguenti caratteristiche:

Quick select

Si tratta di una variante dell'algoritmo di ordinamento "quick sort", in cui ogni chiamata ricorsiva su un intervallo $[i, j]$ del vettore fornito in input termina restituendo il k-esimo elemento più piccolo del vettore qualora $k - 1 \in [i, j]$, oppure un valore indefinito qualora $k \notin [i, j]$. In particolare, nel secondo caso, una chiamata sull'intervallo $[i, j]$ con $k \notin$ \$(i, j]$ può terminare in tempo costante senza dover eseguire la procedura partition. L'algoritmo dovrà avere quindi complessità temporale asintotica $\Theta(n^2)$ nel caso pessimo e $O(n)$ nel caso medio, dove n è il numero di elementi del vettore.

Heap select

Questo algoritmo di selezione utilizza due min-heap, denominate H1 e H2. La prima heap H1 é costruita a partire dal vettore fornito in input in tempo lineare e non viene modificata. La seconda heap H2 contiene inizialmente un solo nodo, corrispondente alla radice di H1. All'i-esima iterazione, per i che va da 1 a $k - 1$, l'algoritmo estrae la radice di H2, che corrisponde a un nodo x_i in H1, e reinserisce in H2 i nodi successori (figli sinistro e destro) di x_i nella heap H1. Dopo $k - 1$ iterazioni, la radice di H2 corrisponderà al k-esimo elemento più piccolo del vettore fornito in input.
L'algoritmo descritto ha complessità temporale $O(n + k\ log\ k)$ sia nel caso pessimo che in quello medio. Per k sufficientemente piccolo, quindi, l'algoritmo "heap select" sarà preferibile, almeno nel caso pessimo, all'algoritmo "quick select". È possibile implementare una variante che utilizzi opportunamente min-heap o max-heap, a seconda del valore di k.

Median-of-medians select

L'algoritmo è basato sulla suddivisione del vettore fornito in input in blocchi di dimensione limitata e sul calcolo della mediana delle mediane. Più precisamente, l'algoritmo esegue le seguenti operazioni:
- divisione dell'array in blocchi di 5 elementi, escluso eventualmente l'ultimo blocco che potrà contenere meno di 5 elementi,
- ordinamento e calcolo della mediana di ciascun blocco,
- calcolo della mediana M delle mediane dei blocchi, attraverso chiamata ricorsiva allo stesso algoritmo
- partizionamento dell'intero array attorno alla mediana M, attraverso una variante della procedura "partition" dell'algoritmo "quick sort"
- chiamata ricorsiva nella parte di array che sta a sinistra o a destra della mediana M, in funzione del valore k fornito in input.

Il modo più semplice per implementare quest'algoritmo consiste nell'allocare, ad ogni chiamata ricorsiva, un nuovo vettore per memorizzare le mediane dei blocchi. Esiste tuttavia un approccio più efficiente e "quasi in place" che riutilizza lo spazio allocato per il vettore originariamente fornito in input (l'unico spazio aggiuntivo utilizzato è dato dalla pila dedicata alla gestione delle chiamate ricorsive). La valutazione del progetto terrà conto della variante implementata (quella "quasi in place", essendo più complicata ma anche più efficiente, sarà valutata con un punteggio più alto).
Indipendentemente dalla variante implementata, nel caso pessimo l'algoritmo dovrà avere complessità, sia temporale che spaziale, pari a $\mathcal{O}(n)$.

Modalità di consegna

Si richiede:

1) L'implementazione in un linguaggio a scelta (ad esempio, C, C++, Java) dei tre algoritmi descritti sopra, in modo che siano formalmente corretti (è possibile assumere che gli input siano ben formati, ovvero che i vettori non siano vuoti e che il parametro k sia sempre positivo e minore o uguale alla dimensione n del vettore). Per agevolare la verifica di correttezza da parte del docente sono stati predisposti tre moduli "Virtual Programming Laboratory" (VPL) da utilizzare per caricare il codice degli algoritmi. Una condizione necessaria alla valutazione dell'elaborato è il *superamento di tutti i test previsti*, per tutti e tre gli algoritmi. Nota: l'esecuzione di un programma lato server attraverso un modulo VPL garantisce uno spazio di memoria di almeno 64KB, giudicato ampiamente sufficiente per risolvere il problema assegnato con qualunque algoritmo fra quelli sopra descritti.

2) La stima dei *tempi medi di esecuzione* per tre algoritmi, al variare della dimensione n del vettore ed eventualmente del parametro k (nei casi, ovviamente, in cui si ritenga esista una correlazione fra tempo di esecuzione e parametro k). Si consiglia di generare almeno un centinaio di campioni per la dimensione n dei vettori, con n che varia indicativamente fra 100 e 5000000 e con distribuzione esponenziale (ad esempio, seguendo la relazione $n_i = A \cdot 2^{B \cdot i}$, per un indice $i = 0, \ldots, 99$ e per opportune costanti A e B in modo che $n_0 = 100$ e $n_{99} = 5000000$. In questo modo la densità dei campioni di n tenderà a decrescere con l'aumentare dei valori di n. Per ogni campione n, bisogna poi generare, in modo pseudo-casuale e prestando attenzione al range di valori generati, uno o più vettori di dimensione n e contenenti interi eventualmente anche negativi (utilizzare in ogni caso un range di interi sufficientemente grande, soprattutto in rapporto a n). Il tempo di inizializzazione dei vettori dev'essere opportunamente scomputato dalla stima del tempo di esecuzione.
I codici sorgenti degli algoritmi valutati in questa parte dovranno essere gli stessi di quelli presentati al punto 1), fatta esclusione, ovviamente, per le parti di codice che costruiscono l'input e gestiscono la misurazione dei tempi medi di esecuzione al variare di n. Inoltre, i tempi di esecuzione devono essere misurati con un *errore relativo massimo pari a 0.01 (1%)*. A tal fine si consiglia di procedere nel modo seguente:

- Per tutte le misurazioni di intervalli di tempo è necessario utilizzare un clock di sistema monotono (utilizzare, ad esempio, la procedura *clock_gettime(CLOCK_MONOTONIC, &timeNow)* della libraria *<time.h>* del linguaggio C, oppure il metodo *steady_clock::now()* della libreria *<chrono>* del C++, oppure *System.nanoTime()* del linguaggio Java).

- Il primo passo consiste nello stimare la **risoluzione** del clock di sistema, utilizzando un ciclo while per calcolare l'intervallo minimo di tempo misurabile. A tale scopo è possibile utilizzare uno dei seguenti frammenti di codice in linguaggio C, C++, Java:

```
// linguaggio C
#include <time.h>
...
double duration(struct timespec start, struct timespec end) {
    return end.tv_sec - start.tv_sec
         + ((end.tv_nsec - start.tv_nsec ) / (double) 1000000000.0);
}
double getResolution(){
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);
    do {
        clock_gettime(CLOCK_MONOTONIC, &end);
    } while (duration(start, end) == 0.0);
    return duration(start, end);
}

// linguaggio C++
#include <stdlib.h>
#include <chrono>
...
using namespace std;
using namespace std::chrono;
...
double getResolution() {
    steady_clock::time_point start = steady_clock::now();
    steady_clock::time_point end;
    do {
        end = steady_clock::now();
    } while (start == end);
    typedef duration<double, seconds::period> secs;
    return duration_cast<secs>(end - start).count();
}

// linguaggio Java
import java.util.*;
...
class ... {
    ...
    private static double getResolution() {
       double start = System.nanoTime();
       double end;
       do {
           end = System.nanoTime();
       } while (start == end);
       return end - start;
}
```

- Successivamente, in funzione della risoluzione stimata R e dell'*errore relativo massimo ammissibile* ($E = 0.001$), si calcola il **tempo minimo misurabile**

$T_{\min} = R \cdot (\frac{1}{E} + 1)$.

- Per stimare il tempo medio di esecuzione di un algoritmo su una singola istanza dell'input di dimensione n, si utilizza un ciclo *while*, iterando l'esecuzione dell'algoritmo sullo stesso input (che andrà eventualmente rigenerato) e misurando un intervallo di tempo superiore a $T_{\min}$. La misurazione deve essere effettuata senza interrompere il clock, ovvero calcolando l'intero intervallo di tempo trascorso dall'inizio dell'iterazione fino al momento il cui il tempo misurato risulti superiore a $T_{\min}$. Il tempo medio di esecuzione per la singola istanza di input sarà quindi ottenuto calcolando il rapporto fra il tempo totale misurato e il numero di iterazioni dell'algoritmo eseguite.

  Nel caso si utilizzi il linguaggio di programmazione Java, occorre prestare attenzione a non allocare ripetutamente grandi strutture dati (esempio, array o stringhe) in modo dinamico (ad esempio, con l'istruzione *new*). Tale pratica potrebbe esaurire in breve tempo la memoria RAM disponibile e attivare il *garbage collector*, creando picchi nei tempi di esecuzione misurati. Una considerazione simile si applica ai linguaggi C e C++, che tuttavia permettono di gestire in modo esplicito l'allocazione e liberazione della memoria.

- Opzionalmente, è possibile stimare anche la deviazione standard dei tempi di esecuzione rispetto al tempo medio, in funzione del parametro n. In tal caso si procederà effettuando un certo numero di misurazioni (ad esempio, un centinaio) sui tempi di esecuzione per una lunghezza n fissata, facendo questa volta variare l'input generato in modo pseudo-casuale (durante questa fase la dimensione n dell'input non cambia). Ogni misurazione su un input generato sarà effettuata seguendo il metodo descritto al punto precedente, e il risultato di tale misurazione dovrà essere memorizzato in un opportuno array. Una volta popolato l'array con le misurazioni relative ad una particolare scelta di n, si procederà calcolando media e scarto quadratico medio.

- Per effettuare tutte le misurazioni con precisione ragionevole, **non è necessario lasciare il computer in esecuzione per ore!**: un'ora di esecuzione è ampiamente sufficiente a generare tutte le misurazioni richieste. Nella fase di implementazione si consiglia di diminuire opportunamente i vari parametri (es. numero di campioni, ripetizioni per la stima della varianza, etc.) in modo da poter testare velocemente il codice in qualche minuto di esecuzione.

I dati raccolti devono essere presentati e discussi in una *relazione* in formato PDF da caricare sul server. La valutazione della relazione e del codice sorgente contribuirà in modo significativo al voto finale del progetto di laboratorio. Non è necessario inviare una relazione con molte pagine: qualche decina di pagine è largamente sufficiente a discutere gli aspetti importanti dell'implementazione e dell'analisi dei tempi di esecuzione. *Si consiglia l'uso di grafici comparativi, sia in scale lineari - n vs $t(n)$ - che doppiamente logaritmiche - $\log(n)$ vs $\log(t(n))$.*

NOTA BENE: Durante l'elaborazione del progetto, sarà possibile caricare diverse versioni dei programmi sui vari moduli VPL (sostituendo ovviamente i vecchi file). Il progetto viene considerato come consegnato una volta che la relazione in formato PDF viene caricata online. A partire da quel momento il docente potrà prendere visione della relazione e dei codici sorgente forniti.

Sono ammessi gruppi di *massimo 4 persone* per lo svolgimento del progetto. Il sito e-learning dell'università non permette la creazione autonoma di gruppi; si consiglia quindi di procedere scegliendo un "rappresentante" per ogni gruppo, che avrà la responsabilità di caricare tutti i file sul sito, inclusa la relazione finale. **È importante che la relazione riporti in chiaro i nomi, i cognomi, le email, e i numeri di matricola di ciascun componente del gruppo.**

Non è fissata una data precisa per la consegna degli elaborati, ma si chiede di consegnare il progetto entro la fine dell'anno accademico. I progetti consegnati verranno valutati dal docente a intervalli regolari (approssimativamente ogni 2-3 settimane). Qualora fosse necessaria una valutazione in tempi brevi, si prega di contattare il docente per e-mail (gabriele.puppis@uniud.it).

Algoritmo di selezione "quick select"
Algoritmo di selezione "heap select"
Algoritmo di selezione "median-of-medians select"
Programma/i per la stima dei tempi di esecuzione
Relazione in formato PDF