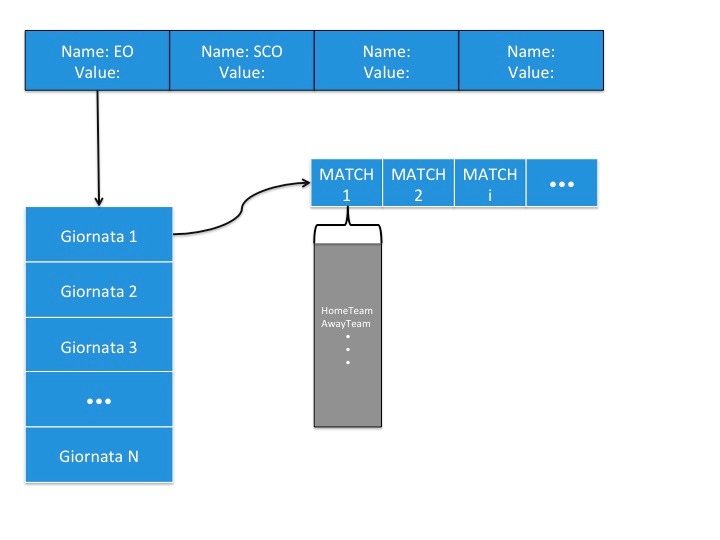
**PROGETTO #3: RELAZIONE**

di Giuseppe Mascolo, Sara Esposito, Alfonso Esposito

**Prima parte**

**Organizzazione della struttura dati**

****

La struttura dati è composta come segue: vi è una lista (DataList) costituita da coppie chiave-valore, in cui la chiave è costituita dal nome del campionato, mentre il valore è una mappa (Championship) che racchiude tutte le informazioni sullo specifico campionato. Championship contiene la lista delle squadre del campionato e una mappa delle giornate di campionato (DayofSeason) e delle giornate che hanno ospitato partite rinviate. Esse a loro volta, contengono la lista dei match della giornata di riferimento. Ogni match racchiude al suo interno tutte le informazioni della riga del foglio Excel utili per quella partita: Date, HomeTeam, AwayTeam, FTHG, FTAG, FTR, HTHG, HTAG, HTR.

Si è scelto di andare a inserire tutti i dati nella struttura al run dell’interfaccia, invece di andare ad aggiungere nuovi elementi ogni volta che viene richiesta una delle opzioni, in modo tale da avere in questo caso solo ritardi dovuti alla ricerca. Per non far annoiare l’utente mentre attende la comparsa della domanda per inserire la propria richiesta, abbiamo inserito il metodo caricamentoDatabase() che utilizza la classe threading: esso istanzia un thread che stampa per 3 volte la scritta “…Caricamento”.

La struttura dati è stata utilizzata per tutte le richieste, cosicché si è potuto risparmiare spazio. E potrebbe essere utilizzata anche per rispondere a quesiti non previsti dal progetto, per esempio se si volesse espandere il quesito 9 tra tutti i campionati: dati una giornata, stampare la squadra con il maggior numero di vittorie, la squadra con il maggior numero di vittorie in casa, la quadra con il maggior numero di vittorie in trasferta tra tutti i campionati.

**Complessità di tempo e di spazio della soluzione proposta**

Poiché la struttura dati, illustrata al paragrafo precedente, è stata utilizzata per rispondere a tutti i quesiti, richiede più tempo per l’inserimento dei dati, ma meno spazio.

Precisamente la complessità temporale per l’inserimento dei dati all’interno della struttura dati è O(k\*n) dove k sono il numero di campionati e , n il numero di righe, colonne presenti in ogni foglio corrispondente al campionato, inoltre durante la lettura del foglio vengono create le classifiche parziali e totali con complessità O(r\*m) dove r sono le righe presenti nel foglio excel e m il numero di squadre.

Questo sistema favorisce però la ricerca.

1. Per il primo quesito è necessaria prima la selezione del campionato giusto e poi la stampa dell’elenco delle squadre. Per la ricerca del campionato, poiché abbiamo una lista di campionati, la complessità è O(n), dove n è il numero dei campionati. Per stampare le squadre la complessità è O(1) poiché esse sono già salvate nella lista dei team. Quindi la complessità totale è O(n) con n numero dei campionati.
2. Per il secondo quesito è necessaria prima la selezione di un campionato, poi quella di una giornata, e infine, la stampa della classifica per la giornata indicata, con le partite giocate da ogni squadra. Per la selezione del campionato è richiesto tempo O(n) con n numero dei campionati poiché devo scorrere tutta la lista nel caso peggiore, ovvero la ricerca del campionato G1. Per la selezione della giornata, la complessità è O(logk) poiché viene utilizzata una ricerca binaria dalla getItem della SortedTableMap, con k numero delle giornate + giornate delle partite rinviate. Per la classifica di quella giornata sarebbe necessario soltanto una complessità O(1), ma il risultato viene ordinato prima di essere stampato da una funzione sortRank che usa un algoritmo di Quick Sort; per questo motivo la complessità diventa O(mlogm) con m il numero delle squadre. Quindi, la complessità totale è di O(n + logk+ mlogm), ma poiché questo numero è molto piccolo, è una complessità che abbiamo ritenuto accettabile.
3. Per il terzo quesito è necessaria prima la selezione di un campionato, poi quella di una giornata, e infine, la stampa della classifica parziale (riguardante il primo tempo) per la giornata indicata, con le partite giocate da ogni squadra. Per la selezione del campionato è richiesto tempo O(n) con n numero dei campionati. Per la selezione della giornata, la complessità è O(logk) poiché viene utilizzata una ricerca binaria dalla getItem della SortedTableMap, con k numero delle giornate + giornate delle partite rinviate. Per la classifica parziale di quella giornata, poiché utilizziamo una funzione sortRank che usa un algoritmo di Quick Sort, la complessità diventa O(mlogm) con m il numero delle squadre. Quindi, la complessità totale è di O(n + logk+ mlogm), come nel quesito precedente.
4. Per il quarto quesito è necessaria prima la selezione di un campionato e di una giornata, e infine, stampare per ogni squadra gli ultimi 5 risultati (sequenza W/D/A). Per la selezione del campionato è richiesto tempo O(n) con n numero dei campionati. Per la selezione della giornata, la complessità è O(logk), con k numero delle giornate + giornate delle partite rinviate. Per accedere alla squadra in questione il tempo richiesto è O(m), m numero dei match, poiché si effettua un ciclo for sui 5 giorni (ma 5 è costante e quindi lascia il risultato invariato) e un ciclo for sui match di quella giornata che richiede tempo O(m). Per accedere ai risultati sono necessarie operazioni costanti. Quindi la complessità totale è O(n + logk + m).
5. Per il quinto quesito è necessaria, previa la scelta di un determinato giorno, la stampa dei risultati di tutte le partite giocate in quel giorno. Poiché le date nella struttura dati vengono salvate in timestamp, avviene prima la conversione della data con tempo costante nella funzione getMatches(self, date). Questa funzione poi chiama la \_searchDay(self, date, start, end) che utilizza una ricerca binaria ricorsiva per trovare la data di interesse e costruisce una lista di match disputati in quel giorno. La complessità è quindi O(n(logk+m)) dove n è il numero di campionati, k le giornate e m, i match disputati in quella specifica giornata.
6. Per il sesto quesito è necessaria prima la selezione di una giornata in tutti i campionati, poi di un intero k per permettere la stampa delle k squadre che tra tutti i campionati hanno segnato più goal. In questo caso viene utilizzata la funzione getTeamMoreGoal(self, k, day). Questa funzione crea una lista con tutte le squadre e i goal di tutti i campionati fino a quella giornata effettuandone l’ordinamento tramite Quick Sort, e alla fine crea una seconda lista in cui ci sono soltanto le k squadre i cui goal rispondono correttamente al quesito. La complessità dell’algoritmo è O(n \*(logx + mlogm + m) + k) con n numero dei campionati, x il numero delle giornate, ed m il numero delle squadre del singolo campionato, k le squadre restituite.
7. Per il settimo quesito è necessaria prima la selezione di una giornata in tutti i campionati, poi di un intero k per permettere la stampa delle k squadre che tra tutti i campionati hanno subito meno goal. In questo caso viene utilizzata la funzione getTeamLessGoal(self, k, day). Questa funzione crea una lista con tutte le squadre e i goal di tutti i campionati fino a quella giornata effettuandone l’ordinamento tramite Quick Sort, e alla fine crea una seconda lista in cui ci sono soltanto le k squadre i cui goal rispondono correttamente al quesito. La complessità dell’algoritmo è, come nel caso precedente, O(n \*(logx + mlogm + m) + k) con n numero dei campionati, x il numero delle giornate, ed m il numero delle squadre del singolo campionato, k le squadre restituite.
8. Per l’ottavo quesito è necessaria prima la selezione di una giornata in tutti i campionati, poi di un intero k per permettere la stampa delle k squadre che tra tutti i campionati hanno la migliore differenza reti. In questo caso viene utilizzata la funzione getTeamDiffGoal(self, k, day). Questa funzione crea una lista con tutte le squadre e la differenza di goal di tutti i campionati fino a quella giornata effettuandone l’ordinamento tramite Quick Sort, e alla fine crea una seconda lista in cui ci sono soltanto le k squadre i cui goal rispondono correttamente al quesito. La complessità dell’algoritmo è, come nel quesito 6 e 7, O(n \*(logx + mlogm + m) + k) con n numero dei campionati, x il numero delle giornate, ed m il numero delle squadre del singolo campionato, k le squadre restituite.
9. Per il nono quesito è necessaria la selezione di un campionato e poi di una giornata, per stampare la squadra con il maggior numero di vittorie, quella con il maggior numero di vittorie in casa, e quella con il maggior numero di vittorie in trasferta. Per la selezione del campionato è richiesto tempo O(n) con n numero dei campionati. Per la selezione della giornata, la complessità è O(logk), con k numero delle giornate + giornate delle partite rinviate, mentre per l’ordinamento della lista di squadre in base alle vittorie, tramite un Quick Sort, richiede O(mlogm) con m il numero delle squadre. Lo stesso procedimento viene adottato per la ricerca delle squadre con il maggior numero di vittorie in casa e in trasferta. Quindi, la complessità totale è O(n + logk + mlogm).

**Casi particolari**

Abbiamo dovuto adattare il nostro algoritmo con dei controlli per comprendere questi casi particolari:

Bastia-Lyon 🡪 33°giornata campionato Francese (F1) – partita interrotta, non sono presenti risultati parziali

Campionato Inglese 🡪 28° giornata, vengono rinviate 5 partite

Campionato Scozzese 🡪 12 Squadre, 38 giornate in totale.

Campionato Greco 🡪 Le prime due giornate vengono completamente rinviate, ma abbiamo supposto semplicemente che la terza giornata fosse la prima e proseguito come se fosse tutto normale.

**Seconda parte**

Per fare in modo che l’algoritmo di pattern-matching di Knuth-Morris-Pratt contasse il numero di occorrenze non sovrapposte del pattern P, nella stringa di testo T si è fatto in modo che dopo aver trovato la prima occorrenza di P, l’algoritmo non ritornasse il valore dell’indice della prima occorrenza. In particolare, a quel punto del codice viene incrementato un count che tiene conto del numero di occorrenze, dopodiché j viene incrementato di 1 per cercare occorrenze dal carattere successivo, e k viene settato a 0 dato che i confronti dovranno di nuovo partire dal primo carattere di P. Vengono eseguite N esecuzioni della funzione, dopodiché il tempo totale impiegato viene diviso per N, in modo da ottenere il tempo medio impiegato dalla funzione. Si mette a confronto il tempo di esecuzione dell’algoritmo proposto con i tempi esibiti dalla funzione count di Python. Il rapporto delle prestazioni è espresso in percentuali. Per testare l’efficienza della soluzione proposta è stato considerato un testo in italiano, e sono stati cercati pattern che vanno da una frase, a poche parole, alla parola singola (sempre sullo stesso testo) a pattern non presenti nel testo. Inoltre è stato considerato un testo composto da tutte “a” ed una “b” finale. In un caso viene cercata una stringa tutta a con una b finale, in un altro viene cercato solo il carattere b. Ogni caso analizzato mostra i tempi di esecuzione e le performance della soluzione proposta rispetto ai tempi della funzione count di Python.