Progetto #3

Prima parte - Campionati

Per quanto riguarda l'interfaccia grafica del progetto, abbiamo assunto che questa venga usata su un sistema embedded, allo stadio, in modo da permettere agli ospiti di fare analisi su determinati campionati.

Il programma viene eseguito a schermo intero. In alto vi sono dei campi che permettono di selezionare il campionato, una volta selezionato questo, vi è un altro campo che permette di selezionare le giornate, e selezionata la giornata, un altro campo permette di selezionare le squadre del campionato, un altro permette di selezionare le date relative a quella giornata e un altro ancora il numero di squadre che si vogliono visualizzare per particolari interrogazioni.

Tutto questo è stato fatto per facilitare l'utente nella formulazione delle "domande", ma è possibile benissimo inserire tali campi manualmente.

Sulla destra vi sono, invece, bottoni al cui click, il software risponde alle domande richieste dal progetto mostrando a sinistra dei bottoni e sotto i campi in alto una tabella.

Al fine di stampare l'elenco delle squadre dato il campionato, abbiamo optato nell'usare una tabella hash con separate chaining.

Un'istanza di una tabella hash ha principalmente due parametri che ne influenzano le prestazioni: la capacità iniziale e il load factor.

La capacità è il numero di bucket nella tabella hash, e la capacità iniziale è semplicemente lo stesso, ma appena creata la tabella hash.

Il fattore di carico è una misura di quanto è consentito che una tabella hash si riempia prima che la sua capacità è aumentata automaticamente. Quando il numero di inserimenti nella tabella hash supera il prodotto del load factor e la capacità corrente, la tabella hash viene rifatta, similmente alla struttura dati array ne viene raddoppiata la size, e la funzione hash adattata a questa nuova size.

I campionati contenuti nel file excel sono 22, quelli che dovevamo considerare ai fini del progetto sono 11, noi abbiamo supposto di

dimensionare la nostra tabella hash per supportare tutti i campionati contenuti nel file excel.

Nel caso di separate chaining sappiamo da studi sperimentali che dobbiamo garantire almeno $\lambda \leq 0{,}9$ mentre per linear probing $\lambda \leq 0{,}5$, poiché il load factor è definito come $\lambda = \frac{n}{N}$, dove n è il numero di elementi della tabella hash, mentre N è il numero di bucket in separate chaining e in linear probing, N è il numero di elementi della tabella.

Facendo i calcoli sia per separate chaining che per linear probing noi otteniamo: $\frac{n}{0{,}9} \leq N_{sc} => \frac{22}{0{,}9} = 24{,}44 \leq N_{sc}$, dunque per separate chaining la tabella hash deve avere capacità di 25 bucket almeno, ovviamente scegliendo N maggiore di 25 si avrebbero prestazioni ancora migliori, ma questo consumerebbe anche spazio aggiuntivo, non necessario per la nostra applicazione.

Facciamo lo stesso discorso per linear probing: $\frac{n}{0{,}5} \leq N_{lp} => \frac{22}{0{,}5} = 44 \leq N_{lp}$, dobbiamo avere una tabella hash con capacità 44.

Ci chiediamo inoltre, dopo quanto sarà ridimensionata la tabella hash? Nel caso di separate chaining: $\lambda_{sc} * N_{sc} = 23$, mentre nel caso di linear probing: $\lambda_{lp} * N_{lp} = 22$.

In entrambi i casi abbiamo vantaggi e svantaggi, nel caso del linear probing, teoricamente consumiamo meno spazio in quanto non abbiamo bisogno di spazio aggiuntivo per i bucket, ma possiamo inserire meno elementi prima di dover ridimensionare la tabella al fine di poter garantire basse collisioni.

Tipicamente le squadre di calcio sono già note alla lettura del file, poichè si ricavano dalla seguente formula:

$$\frac{N_{SQUADRE}\left(N_{SQUADRE} - 1\right)}{2} * N_{GIRONI} = N_{RIGHE} - 1$$

Questa equazione di secondo grado, chiaramente, risolta ci da due soluzione è necessario scegliere quella fisicamente accettabile, la seguente:

$$N_{SQUADRE} = \frac{1 + \sqrt{1 + 4 * N_{RIGHE}}}{2}$$

È un esempio di soluzione nel caso del campionato italiano.

Dunque alla lettura del file abbiamo già note il numero di squadre per campionato, ciò ci ha fatto optare per usare una tabella hash con separate chaining.

E' vero che consumiamo più spazio per le strutture dati aggiuntive, ma è pur vero che rispetto a linear probing istanzieremo un numero di elementi della tabella hash minore.

Inoltre, avendo scelto 0.9 come fattore di carico, la probabilità di avere collisioni è abbastanza bassa.

Facciamo un esempio: per il campionato italiano il quale è costituito da 20 squadre, è necessario istanziare la tabella hash con capacità iniziale di 23 elementi, mentre per linear probing ben 40 elementi.

Abbiamo visto che sperimentalmente con separate chaining si verificano su questo caso massimo 3 collisioni, allungando il tempo di accesso di pochissimo.

Abbiamo dunque optato per questa soluzione, in ragione del fatto che il numero di squadre è noto ed è fisso.

Soluzione diversa abbiamo preso in considerazione per la struttura dati che rappresenta i campionati, per questa abbiamo usato una tabella hash con linear probing.

L'assunzione fatta è stata: è vero che con separate chaining consumo meno spazio, ma in linea teorica in un applicazione pratica potrei voler analizzare tutti i campionati del mondo, tipicamente le squadre per campionato sono poche e il tempo che si perde per accedervi con una tabella hash che utilizza separate chaining è ragionevole.

Nel caso dei campionati proprio perchè si può voler aggiungere un numero non noto a priori di campionati, lo spazio aggiuntivo e il tempo di accesso dovuto alle strutture dati linkate non è ragionevole.

Dunque, è vero che la struttura dati verrà raddoppiata prima, ma i vantaggi in termini di tempo di accesso, sono evidenti rispetto allo spazio consumato.

Poiché conosciamo il numero di giornate, e vi vogliamo accedere attraverso un indice, che è il numero di giornata, per le giornate è stata

scelta la struttura dati lista di python, la quale ha complessità $O(1)$ sia per inserimenti che cancellazioni.

Al fine di risolvere il problema proposto dal progetto sono state necessarie delle assunzioni:

Chi calcola il calendario per un girone all'italiana, usa l'algoritmo di Berger, la prima assunzione che è stata fatta è che non è possibile rinviare più di una partita per giornata, questo perché neanche un operatore umano se guardasse i dati disponibili sarebbe in grado di capire quali sono le partite effettivamente rinviate e quali no.

Facciamo un esempio, se alla prima giornata venissero rinviate ben due partite, nelle giornate successive possiamo ritrovare le 4 squadre a giocare una delle seguenti partite, date dalle disposizioni di 4 su 2:

$$N_{partite} = \frac{4!}{4! * (4-2)!} = 12$$

Ben 12 partite, di cui nessuno senza avere il calendario originario potrebbe mai capire quali sono state rimandate e quali no.

Il file viene aperto una sola volta, ma viene letto due volte, la prima per creare l'elenco delle squadre e la seconda per creare le altre strutture dati.

La prima volta non leggiamo tutto il file, ma solo un numero di righe poco più grande del numero di giornate.

Dalla formula presentata precedentemente noi sappiamo il numero di squadre, e quindi leggiamo le righe del file finché non riempiamo la struttura dati col numero di squadre che conosciamo.

La seconda volta invece leggiamo tutto il file poiché dobbiamo riempire le restanti strutture dati.

L'algoritmo di base usato per gestire il problema delle partite rimandate è il seguente:

Si mantiene un contatore per ogni squadra, il quale indica il numero di partite giocate (utile per risolvere anche dei quesiti proposti), si inizia a leggere il file, mantenendo un contatore di giornata, man mano che si legge si incrementano i contatori delle squadre delle partite giocate, ad un certo punto, il contatore di una certa squadra sarà superiore a quello

del contatore di giornata, e allora vorrà dire che una nuova giornata è cominciata.

Per gestire poi, alcune particolarità di certi campionati ci siamo affidati ad euristiche.

L'algoritmo proposto per definire la classifica alle giornate indicate funziona, quindi, se a priori è specificato il numero di volte in cui le squadre si scontreranno all'interno del campionato.

Nel caso in cui vengano sospese 2 o più partite, l'algoritmo continua a funzionare a patto che non si verifichi il caso precedentemente descritto e che una delle partite tra squadre coinvolte nelle partite rinviate non sia la partita inaugurale di una nuova giornata (quindi responsabile di far scattare il contatore delle giornate)

Alla luce delle considerazioni fatte, questi casi particolari sono stati gestiti singolarmente, assicurandoci che in caso di una partita rientrante nella categoria suddetta, venga creata una nuova giornata (per questo occorre disporre del calendario delle partite).

Questa tecnica è stata utilizzata per il campionato italiano e inglese e le partite interessate sono Bologna-Crotone alla 20 giornata e Leicester-Crystal Palace alla 32.

Particolare attenzione meritano i campionati Portoghese, Scozzese e Greco.

Nel campionato portoghese nell'anno 2016/2017 è stata giocata una partita, Porto-Maritimo della 15a giornata prima della 14a. Per assicurare il corretto funzionamento dell'algoritmo non viene fatta "scattare" la nuova giornata quando dopo la partita suddetta, dal dispatcher viene letto il Porto e il Maritimo per le partite della 14a giornata.

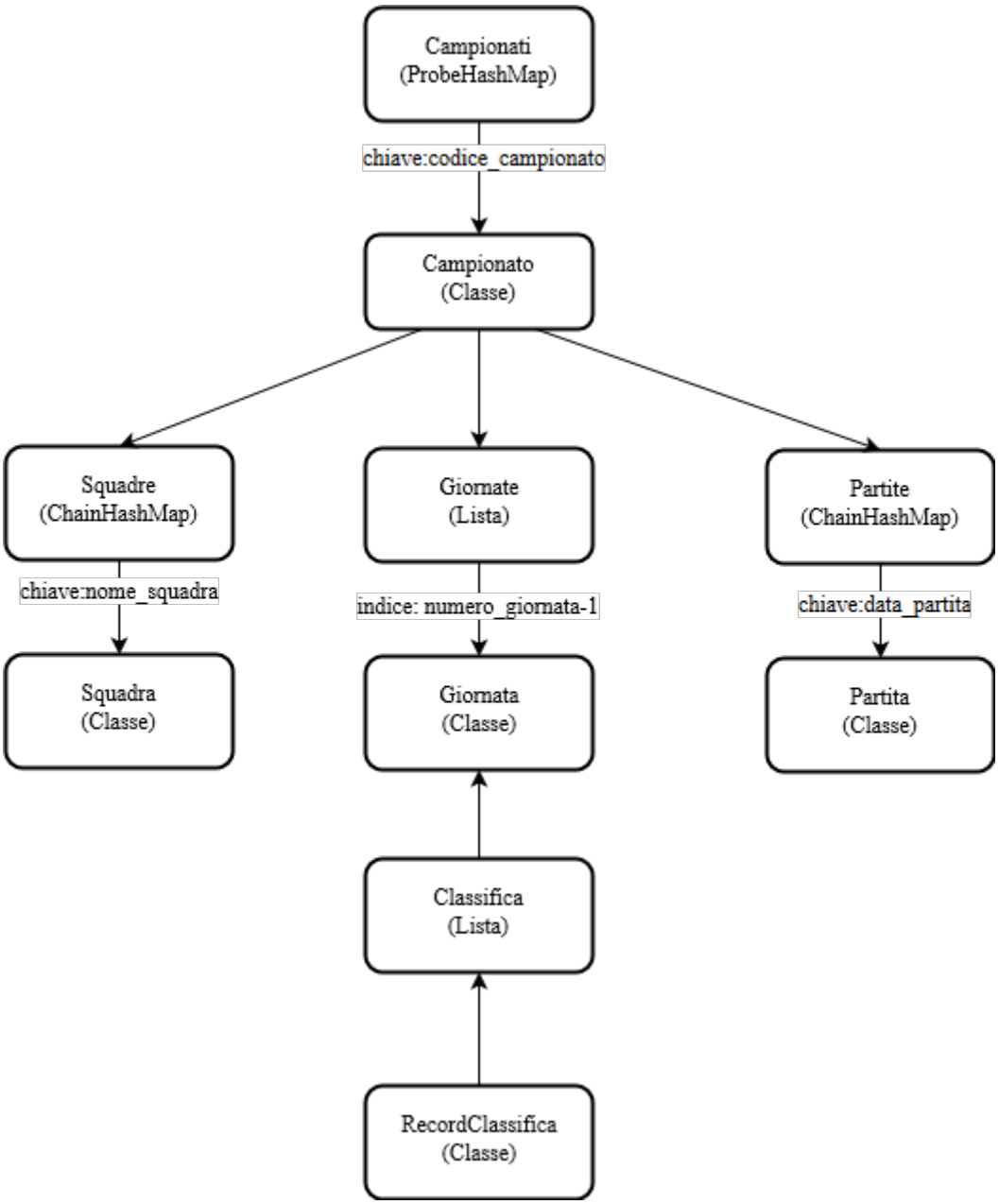
Per quanto riguarda la Scozia, si sono considerate solo le partite del primo "torneo" a 12 squadre. Anche in questo campionato si verifica il caso detto in precedenza e risolto come per gli altri campionati interessati.

Il campionato Greco invece, per la stagione 2016/2017 alle ha visto il campionato iniziare alla 3a giornata e un rinvio di un’intera giornata (11a). Le tre giornate sono state poi recuperate in sequenza a cavallo tra la 17a e la 18a giornata. Per questo campionato si è scelto di inizializzare la classifica alla prima e alla seconda giornata considerando i record

classifica di tutte le squadre con tutti i parametri inizializzati a 0 (punti, partite giocate ecc..). Quindi, si procede con l'applicazione dell'algoritmo descritto in precedenza facendo in modo che in occasione della partita "AEK – Platanias" si passi dalla 10 alla 12 giornata. Una volta poi individuata la partita "A. Tripolis – Olympiacos", che sancisce l'inizio dell'andamento standard del campionato (recuperati i 3 turni non disputati), l'algoritmo procederà la sua esecuzione normalmente (partite giocate e giornate sono "allineate").

NOTA: Campionato Francese, Ligue ONE: gestita la penalizzazione inflitta al Metz dopo la 16 giornata (-2 punti) e l'annullamento di tale sansione a partite dalla 26 (+2 punti).

Segue un'immagine descrittiva delle strutture dati che abbiamo utilizzato:



Per quanto riguarda il primo quesito ci basta semplicemente, dato il codice del campionato da GUI, usare questo come chiave per accedere allo specifico campionato, qui possiamo scorrere la tabella hash che contiene tutte le squadre e realizzare l'operazione di stampa delle squadre con complessità $O(n)$, con n numero di squadre.

Nel secondo e terzo quesito, ci basta accedere alla lista delle giornate, visto che l'utente specifica da interfaccia grafica il numero di giornata, ogni giornata ha un attributo classifica che è una lista di record classifica.

Fatto l'accesso alla classifica, che è una lista la ordiniamo secondo il punteggio del primo tempo oppure quello del secondo tempo, e la mostriamo a schermo.

Chiaramente un'altra soluzione possibile sarebbe stata quella di creare due liste diverse una per la classifica primo tempo e una per la classifica secondo tempo, ma questa soluzione avrebbe consumato troppo spazio, e avremmo avuto ridondanza di dati.

Per poter stampare la classifica, poiché dobbiamo ordinarla prima, questo complessivamente ci prende tempo $O(nlogn)$, con n numero di squadre.

Il quarto quesito, ci basta accedere alla giornata e al suo risultato in classifica, e stessa cosa fare per le quattro giornate precedenti.

Questo ha complessità O(n), si sarebbe potuta ridurre questa complessità, implementando la classifica attraverso una tabella hash anziché una lista, ma questo nonostante consumi più spazio, non avrebbe permesso di rispondere in modo efficiente alle domande in cui l'utente specifica una giornata (e queste domande sono la maggior parte delle richieste del progetto).

Per i quesiti 6,7 e 8, ancora una volta è bastato accedere a tutti i campionati, alla giornata specificata prendere le k prime squadre ordinate secondo la richiesta, questo ha avuto complessità $O(cnlogn)$

Per quanto riguarda il quinto quesito, data una data, semplicemente accediamo a tutti i campionati, dopodiché attraverso la data accediamo alla tabella hash delle partite e le stampiamo. Ricordiamo che una volta acceduto con la data alla tabella hash delle partite, qui è presente una lista di partite che sono state giocate lo stesso giorno. La complessità temporale è quindi $O(nlogn)$, n è il numero di squadre, dopo che queste sono state prelevate dai singoli campionati, nlogn poiché queste si devono ordinare e prendere le prime k.

Dobbiamo accedere a tutti i campionati, quindi dobbiamo effettuare c accessi, qui dobbiamo prelevare k squadre, queste vengono messe in una

lista, di dimensione n, dove n è al più c*k, qui dobbiamo ordinare una lista con n elementi.

Per quanto riguarda il quesito 9, è bastato accedere alla specifica giornata e campionato, qui ordinare la lista classifica secondo il maggior numero di vittorie, vittorie in casa e vittorie in trasferta e prendere le prime squadre con stesso punteggio. Questo ha complessità O(nlogn) che è la complessità dell'ordinamento.

Sotto è riportata una tabella riepilogativa delle varie complessità delle soluzioni proposte a seconda del quesito, sia temporali sia spaziali.

n. quesito	Comp. Temporale	Comp. Spaziale
1	O(n)	O(n)
2	O(nlogn)	O(gn)
3	O(nlogn)	O(gn)
4	O(n)	O(gn)
5	$O(n_p)$	O(p)
6	O(cnlogn+k)	O(cgn)
7	O(cnlogn+k)	O(cgn)
8	O(cnlogn+k)	O(cgn)
9	O(nlogn)	O(n)

Le strutture dati possono essere utilizzate per rispondere a molte richieste non previste dal progetto, ad esempio scegliendo una diversa dimensione iniziale per la tabella hash che contiene i campionati potrebbe essere usata addirittura per rispondere domande su tutti i campionati del mondo.

Nella classe record classifica, associati a una squadra sono presenti un sacco di informazioni, come numero di partite giocate fino a quel punto, vittorie, pareggi, sconfitte, goal fatti, goal subiti, punti, vittorie primo tempo, pareggi primo tempo … vittorie in casa, vittorie in trasferta, … punti in casa, punti in trasferta.

Ordinando dunque i record secondo l'attributo voluto, si possono quindi rispondere a tante altre domande come, chi è la squadra col maggior numero sconfitte in trasferta? O cercare la squadra con un certo numero di vittorie in casa. Nel primo caso avremmo complessità $O(nlogn)$, mentre nel secondo $O(n)$.

Seconda parte - Stringhe

Per implementare la seconda parte del progetto, essendo che find_kmp restituisce l'indice dell'inizio del primo pattern trovato in una data stringa, è bastato riapplicare find_kmp nuovamente nella stringa a partire dall'indice dopo il pattern trovato.

È stato necessario find_kmp al fine di poter migliorare l'efficienza temporale del metodo count_kmp.

In count_kmp calcoliamo i fail una sola volta, che passiamo a find_kmp.

Applicando k volte, find_kmp si riescono a contare tutte le occorenze del pattern in una data stringa, visto che Knuth-Morris-Pratt ha complessità $O(m+n)$, il nostro metodo per contare il numero di occorrenze avrà complessità $O(m+n)$.