Commento all'es. 1

Dati: si devono trattare i dati relativi agli skilift e agli sciatori, creando opportuni ADT di I classe.

- **ADT di I classe skilift:** la struttura dati wrapper `skilift` contiene la stringa di caratteri che identifica lo skilift e il tempo minimo tra 2 utilizzi dello stesso (intero). Le funzioni permettono di creare la struttura per un nuovo skilift (`SKILIFTnew`), distruggerla (`SKILIFTfree`), leggerne l'identificatore (`SKILIFTid`) e leggerne il tempo (`SKILIFTinterval`).
- **ADT di I classe sciatore:** la struttura dati wrapper `skier` contiene l'intero che identifica lo sciatore (`id`) e un vettore di interi `skiersSkilifts` per gli skilift utilizzati da quello sciatore. Gli skilift sono identificati da interi. Le funzioni permettono di creare la struttura per un nuovo sciatore (`SKIERnew`), distruggerla (`SKIERfree`), leggerne l'identificatore (`SKIERid`), stamparne l'identificatore (`SKIERprint`), assegnare in quale momento uno skilift è stato usato (`SKIERsetTime`), leggere in quale momento uno skilift è stato usato per l'ultima volta (`SKIERgetTime`) ed elencare gli skilift usati dallo sciatore (`SKIERlistSkilifts`).

ADT di I classe per collezioni di dati:

- si utilizza un ADT `ST` per realizzare una tabella di simboli implementata come vettore non ordinato. Le funzioni che vi operano sono quelle standard, con l'aggiunta della funzione `STsearchORInsert` che ricerca se una chiave è già presente e se la trova ne ritorna l'indice oppure se non la trova la inserisce e ne ritorna l'indice. Gli skilift saranno memorizzati nella tabella di simboli `skilifts` di tipo `ST`, poiché le specifiche non impongono nulla in termini di complessità delle operazioni
- si utilizza un ADT B`ST` per realizzare una tabella di simboli basata sun un albero binario di ricerca. Le funzioni che vi operano sono quelle standard. Gli sciatori saranno memorizzati nella tabella di simboli `skiers` di tipo `BST`, poiché le specifiche impongono complessità logaritmica delle operazioni.

Algoritmo: il main crea i 2 ADT per le collezioni di dati `skilifts` e `skiers`, poi legge da file la lista degli skilift e popola la corrispondente tabella di simboli (`readSkiliftData`). Invece di avvenire da tastiera, l'acquisizione dei dati dei passaggi agli skilift avviene da file. La funzione `authorize`:

- tramite ricerca in `ST` ricava l'indice dello skilift per il quale si richiede l'autorizzazione e ne recupera le informazioni
- tramite ricerca in `BST` appura se lo sciatore che chiede l'autorizzazione è nuovo o no: se è nuovo, lo sciatore viene inserito in `BST` aggiungendo lo skilift richiesto alla lista di skilift usati con il tempo appropriato. Se non è nuovo, si recupera l'informazione sull'ultimo tempo di uso dello skilift richiesto, si verifica se è soddisfatta la condizione e se è il caso si autorizza, aggiornando il nuovo tempo.

Anche se il testo richiedeva solo di mantenere, in memoria centrale, l'elenco di tutti gli skilift utilizzati da ciascuno sciatore e per ciascuno skilift l'ora dell'ultimo utilizzo (abilitazione), il main visualizza questi elenchi per ogni sciatore.

Commento all'es. 2

Strutture dati: si definisce una struttura dati wrapper di tipo `oggetti` che al suo interno contiene il numero `nO` di elementi di tipo `oggetto` memorizzati in un vettore `equip`. Il tipo `oggetto` è una struttura con nome e categoria (stringhe) , righe e colonne occupate (interi) e utilità (intero). La funzione `leggiFile` legge i dati da file e li memorizza in una struttura dati wrapper denominata `equip`.

Algoritmo: si tratta di un problema di ottimizzazione del tipo zaino discreto. La funzione `zaino` si basa sul powerset con disposizioni ripetute e pruning. La funzione obiettivo mira a massimizzare l'utilità degli oggetti selezionati. Il pruning consiste nel non scegliere l'oggetto corrente (`pos`) se la sua area eccede quella dell'inventario (`A`) o se le sue dimensioni in righe (`R`) e colonne (`C`) eccedono quelle dell'inventario. La funzione `zaino` è chiamata da una funzione wrapper `risolvi` che si occupa delle allocazioni del vettore della soluzione `sol`, del vettore della soluzione migliore `best_sol` e della matrice dell'inventario `inv` nonché della stampa della soluzione. La verifica di ottimalità di una soluzione valida è effettuata nella funzione `zaino`. La verifica di validità di una soluzione è effettuata dalla funzione `check` che opera:

- controllando che sia soddisfatta la condizione di avere almeno un oggetto per ognuna delle 3 categorie specificate. Si utilizza un vettore di occorrenze `mark`
- se è soddisfatto il controllo precedente, chiama una funzione `posiziona` che determina se gli oggetti presenti nella soluzione possono essere memorizzati nell'inventario e in quali posizioni.

Funzione `posiziona`: è un wrapper che crea un vettore `mark` la cui funzione è di marcare gli oggetti che sono stati posizionati e una matrice `tmp_inv` la cui funzione è di contenere il tentativo corrente di posizionamento ed infine chiama la funzione ricorsiva `posizionaR`.
Funzione `posizionaR`: esamina le caselle dell'inventario scandendole attraverso i loro indici di riga e colonna ed identificando in corrispondenza l'indice della ricorsione `pos`. La condizione di terminazione è raggiunta nei seguenti casi:

- il numero di oggetti piazzati correttamente è pari a quello degli oggetti scelti, cioè alla cardinalità della soluzione corrente. Si tratta di una condizione di terminazione con successo
- sono state esaurite le caselle da considerare (`pos` uguaglia o supera il numero totale di caselle). Si tratta di una condizione di terminazione con insuccesso.

Per ciascuno degli oggetti che fanno parte delle soluzione e che non sono ancora stati posizionati, data la posizione corrente `r, c,` si applica il modello delle disposizioni ripetute con 2 scelte (n=2, 1 oggetto posizionato, 0 non posizionato) e k (numero di oggetti parte della soluzione ma non ancora posizionati) :

- se l'oggetto ha un altezza (`equip->o[i].r`) non compatibile con il numero di righe restanti (`R-r`) in quanto maggiore, lo si abbandona e si ritorna fallimento
- se l'altezza è compatibile con la posizione corrente, ma non la larghezza, l'oggetto potrebbe stare nell'inventario partendo da una riga sotto e da una casella più a sinistra. Si ricorre aggiornando `pos` come `pos+C-c` per saltare le colonne della riga attuale inutili
- se sia l'altezza che la larghezza sono compatibili con la posizione corrente, la funzione occupa marca le caselle dell'inventario che l'oggetto occuperebbe. Se si incontrano caselle già occupate da altri oggetti, si ritorna con fallimento, altrimenti con successo. In

caso di successo si ricorre lavorando su `pos` aggiornato come `pos+equip->o[i].c` e a partire dalla colonna `c+equip->o[i].c` di quella riga. Segue la consueta fase di backtrack. In caso di fallimento si ricorre su `pos+1` e `c+1`.

La funzione occupa cerca di marcare le caselle a partire da quella corrente in funzione delle dimensioni dell'oggetto. Se fallisce, in quanto c'è sovrapposizione con un oggetto già posizionato, ripristina lo stato delle caselle che aveva tentato di marcare.