Esercitazione di laboratorio n. 9

Esercizio n. 1: coda a priorità I

In un centro sviluppo di una grande azienda di elettronica si fa uso di un insieme consistente di pc, interconnessi tra loro, per eseguire numerose simulazioni sui progetti sviluppati dagli ingegneri lì impiegati. Più precisamente, quando un progettista vuole attivare un determinato test, lancia un comando (*job*) che viene "raccolto" da un sistema di gestione comune che poi provvede a eseguirlo su uno qualunque dei calcolatori dedicati. La strategia seguita dal gestore è la seguente:

- se, nel momento in cui viene lanciato un job, è presente almeno un computer libero, il job viene immediatamente eseguito su di esso
- se non è disponibile alcun calcolatore per eseguire un job, il job stesso viene temporaneamente "salvato" in una lista d'attesa
- nel momento in cui un job termina la sua esecuzione su un pc, il gestore viene "avvertito" e un altro job (se presente nella lista d'attesa) viene selezionato e lanciato su di esso. La selezione del nuovo job da eseguire avviene secondo un criterio di massima priorità.

Si desidera implementare un semplice programma che emuli il comportamento del gestore dei job. In particolare, il sistema deve essere realizzato su tre moduli:

- un modulo per la gestione dei singoli job, implementati come ADT d prima categoria: ciascun job è in realtà caratterizzato da numerose informazioni, ma per semplicità si supponga che, oltre al livello di priorità (intero positivo), ad esso sia associata solo una stringa univoca di lunghezza massima pari a 30 caratteri
- un modulo per la gestione della lista d'attesa: tale struttura dati, di fatto corrispondente a una coda prioritaria, sia implementata (come ADT di I categoria) mediante:
  - una lista ordinata
  - uno heap, nell'ipotesi in cui il numero di job che possono essere memorizzati nella coda sia al più pari a 100
- un programma principale (client) che si occupa dell'interfaccia del gestore verso il mondo esterno, al cui interno sono realizzate le due funzioni con cui il gestore può essere richiamato (ovvero, la richiesta di esecuzione di un nuovo job da parte di un progettista e la segnalazione di terminazione di un job che occupava uno dei computer). Per semplicità, si includa in tale modulo anche un main che richiami tali funzioni a scelta dell'utente, richiedendo all'utente stesso le altre informazioni che sono necessarie per attivare la funzione desiderata (per esempio, se si richiede di eseguire un nuovo job, il suo "nome" e la sua priorità).

Il programma, infine, visualizzi sul video opportune informazioni a fronte di ogni richiesta ricevuta (per esempio, che la richiesta di esecuzione un nuovo job è stata acquisita ma il job è stato salvato nella lista d'attesa, oppure che un determinato job è stato estratto dalla lista d'attesa e lanciato su un calcolatore che si è reso disponibile, etc.). Il numero totale di computer dedicati che sono disponibili sia una costante predefinita, dichiarata nel modulo principale.
Si noti che, per semplicità, non è richiesta la gestione dell'associazione tra ogni job da eseguire e il calcolatore che materialmente esegue quel job: quel che si vuol testare è solo il comportamento del gestore (ovvero, quando uno dei job presi in carico viene mandato in esecuzione).

Esercizio n. 2: gestione di strutture BST

Si realizzi un programma C che, attraverso un'apposita interfaccia utente, permetta di agire su un BST.

Le operazioni permesse devono essere quelle di:
- creazione di un nuovo BST (vuoto)
- inserimento di un nuovo dato (specificato dall'utente) nel BST
- ricerca di una chiave (specificata dall'utente) nel BST
- ricerca del predecessore e del successore di un elemento (specificato dall'utente) nel BST
- rimozione di un elemento (specificato dall'utente) dal BST
- visualizzazione (a video) di tutti gli elementi presenti nel BST (**ordinati per chiave**)
- salvataggio del BST su file, con formato a scelta
- caricamento di un nuovo BST da file.

Il programma deve essere realizzato su tre moduli distinti:
- l'interfaccia utente (il main)
- un modulo per la gestione del BST
- un modulo per la gestione dei dati (acquisizione, stampa, …).

In particolare, il modulo per il BST deve corrispondere a un ADT di I categoria. Si supponga che ogni elemento della base dati sia una struttura composta da due campi:
- un nome (stringa di lunghezza massima pari a 1000 caratteri).
- un codice (stringa alfanumerica di lunghezza esattamente pari a 8 caratteri).

Si assuma, inoltre, che il BST utilizzi come chiave il campo "codice" dei dati. Si ipotizzino chiavi distinte. Internamente al BST le stringhe utilizzate per il nome vanno allocate dinamicamente internamente al modulo dell'ADT per la sola lunghezza effettivamente richiesta.

Dopo ogni modifica della base dati (inserzione o rimozione), devono essere visualizzate le seguenti informazioni sul BST:
- numero di nodi
- distanza minima di una foglia dalla radice
- distanza massima di una foglia dalla radice.

Esercizio n. 3 (opzionale): coda a priorità II

Si modifichi il codice sviluppato nell'esercizio 1 in modo tale che:
- ogni job sia caratterizzato, oltre che dal nome e dalla priorità, anche da un tempo di arrivo (ovvero, l'orario in cui ne è stata richiesta l'esecuzione) e da una durata. Entrambi siano espressi nel formato hh:mm:ss
- l'insieme di tutti i job da eseguire, in numero ignoto, sia memorizzato in un file di testo, in ragione di un job per riga del file, con il formato seguente:

  `<nome> <priorità> <arrivo> <durata>`

  Si assuma che i job contenuti in tale file siano ordinati per tempi di arrivo crescenti.
- l'operazione che il gestore dei job deve eseguire *non* sia di volta in volta richiesta all'utente: piuttosto, a partire dal contenuto del file suddetto, il client dell'applicazione *deduca automaticamente* quando i computer sono disponibili e l'istante di tempo in cui ogni job viene effettivamente lanciato su uno dei calcolatori.
- il programma visualizzi sul video un insieme opportuno di informazioni su ciascun avvenimento in ordine cronologico.

SUGGERIMENTO: Il client dell'applicazione, a partire dal contenuto del file dei job, *simuli* il comportamento del gestore seguendo un ordine cronologico: ogni qual volta un nuovo job viene lanciato su un computer, infatti, l'applicazione *conosce* il momento in cui quel calcolatore sarà di nuovo disponibile. In ogni istante, pertanto, il gestore sa quanti sono i calcolatori liberi e quando ogni job che è stato lanciato sarà terminato. Processando un job per volta dal file di ingresso, quindi, è possibile capire qual è l'avvenimento che accade per primo tra la presa in carico del nuovo job e un mutamento nello stato dei calcolatori e/o dei job in lista d'attesa…

Esempio: Si assuma che il numero di calcolatori a disposizione sia pari a 3, e il contenuto del file di ingresso il seguente.

```
j1 25 14:00:00 01:10:00
j2 50 14:10:00 00:30:00
j3 40 14:20:00 01:20:00
j4 60 14:30:00 00:40:00
j5 25 14:50:00 00:30:00
j6 75 14:55:00 00:20:00
j7 50 15:00:00 00:50:00
```

Allora il programma dovrebbe produrre un output del tipo:

```
14:00:00 > Job j1 (p=25) acquisito. Job lanciato su pc.
14:10:00 > Job j2 (p=50) acquisito. Job lanciato su pc.
14:20:00 > Job j3 (p=40) acquisito. Job lanciato su pc.
14:30:00 > Job j4 (p=60) acquisito. Job messo in attesa.
14:40:00 > Job j2 (p=50) terminato. Job j4 (p=60) lanciato su pc.
14:50:00 > Job j5 (p=25) acquisito. Job messo in attesa.
14:55:00 > Job j6 (p=75) acquisito. Job messo in attesa.
15:00:00 > Job j7 (p=50) acquisito. Job messo in attesa.
15:10:00 > Job j1 (p=25) terminato. Job j6 (p=75) lanciato su pc.
15:20:00 > Job j4 (p=60) terminato. Job j7 (p=50) lanciato su pc.
15:30:00 > Job j6 (p=75) terminato. Job j5 (p=25) lanciato su pc.
15:40:00 > Job j3 (p=40) terminato.
16:00:00 > Job j5 (p=25) terminato.
16:10:00 > Job j7 (p=50) terminato.
```