PROGRAMMAZIONE AVANZATA

6 Mappe

Sommario

- Illustriamo il contenitore associativo **mappa**
- Usiamo le mappe congiuntamente ad alcuni algoritmi predefiniti nella libreria standard del C++ per realizzare un programma che calcola la **frequenza** delle parole in un testo (a titolo di esempio usiamo questa versione de “I promessi sposi”)

Mappe

Definizione

Una **mappa** è un contenitore i cui elementi sono coppie (k, v) formate da una **chiave** k e da un **valore** v associato a quella chiave con la proprietà che non ci sono due coppie con la stessa chiave.

- La mappa è realizzata in maniera tale che inserimento, cancellazione, **ricerca** per chiave sono operazioni (relativamente) efficienti
- Il tempo per effettuare una di queste operazioni è proporzionale a $\log_2(n)$ dove n è il numero di elementi nella mappa
- **Attenzione**: il tempo per **cercare un valore** all'interno della mappa rimane proporzionale al numero di elementi della mappa.

Esempi di mappe nel mondo reale

Mappa	**Chiave**	**Valore**
Legenda	simbolo/colore	significato
Dizionario italiano-inglese	parola in italiano	parola in inglese
Indice analitico di un testo	voce	pagina in cui occorre

Perché il nome “mappa”?

Una mappa m di elementi (k, v) in cui le chiavi hanno tipo T ed i valori tipo S può essere vista come una **funzione parziale**

$$m : T \hookrightarrow S$$

il cui dominio è un **sottoinsieme finito** di T (insieme totalmente ordinato)

- **Cercare** il valore associato a una chiave k significa applicare m a k

$$m(k) = v \iff (k, v) \text{ è } \textbf{un elemento della mappa}$$

- **Inserire** un elemento in m significa estendere il dominio di m
- **Rimuovere** un elemento da m significa restringere il dominio di m

Esempio

Descrizione del problema

Realizzare un programma che visualizzi le **10 parole più frequenti** che si trovano in un testo, in **ordine decrescente** di frequenza.

Strategia

1. **Creare una mappa** di coppie (p, c) in cui p è una parola del testo e c è il numero di volte che p occorre nel testo. La mappa è inizialmente **vuota**.
2. **Leggere** una ad una le parole del testo. Per ogni parola p trovata nel testo si procede come segue:
   - se nella mappa non c'è alcuna coppia (p, c), **inserire** $(p, 1)$;
   - se nella mappa c'è una coppia (p, c), **modificarla** in $(p, c + 1)$.
3. **Copiare** le coppie dalla mappa in un **vettore**.
4. **Ordinare** il vettore in modo decrescente in base alla seconda componente delle coppie.
5. **Stampare** i primi 10 elementi del vettore.

Soluzione: definizioni di base

```
#include <iostream>   // stampa su terminale
#include <fstream>    // lettura da file
#include <map>        // fasi 1 e 2
#include <vector>     // fasi 3-5
#include <algorithm>  // fase 4
#include <cctype>     // filtro per i caratteri

typedef std::pair<std::string, int> WordCount;
typedef std::map<std::string, int>  WordCountMap;
typedef std::vector<WordCount>      WordCountVec;
```

Note

- `std::pair<T,S>` è il **tipo delle coppie** in cui la **prima componente** ha tipo `T` e la **seconda componente** ha tipo `S`. Tali componenti sono memorizzate in due **campi pubblici** chiamati `first` e `second`.
- `std::map<T,S>` è il **tipo delle mappe** in cui le chiavi hanno tipo `T` ed i valori hanno tipo `S`. Una mappa di tipo `std::map<T,S>` ha elementi di tipo `std::pair<T,S>`.
- Gli elementi della mappa `WordCountMap` hanno tipo `WordCount`.

Lettura e inserimento delle parole

```
WordCountMap read_word_map(std::istream& is) {
  WordCountMap m;
  std::string w;
  while (is >> w)
    if (valid(w)) m[w]++;
  return m;
}
```

Note

- La funzione `valid` controlla che `w` sia composta esclusivamente da caratteri alfabetici (si veda il codice sorgente per la realizzazione).
- `m[w]` è un **riferimento** al valore associato alla chiave `w` nella mappa `m`.
- Se la mappa non contiene la chiave `w`, viene inserita e associata al valore di default del tipo del valore (nel caso dei numeri è 0)
- La notazione `m[w]` è simile a quella usata per accedere a elementi di array e vettori, ma qui l'indice `w` **non è un numero**!
- La notazione `m[w]` richiama quella dell'applicazione funzionale $m(w)$

Creazione e ordinamento del vettore

```
bool Compare(const WordCount& p, const WordCount& q) {
  return p.second > q.second;
}

WordCountVec sort_word_map(const WordCountMap& m) {
  WordCountVec v = WordCountVec(m.begin(), m.end());
  std::sort(v.begin(), v.end(), Compare);
  return v;
}
```

Note

- `WordCountVec(m.begin(), m.end())` crea un vettore copiandone gli elementi dalla mappa `m`.
- `std::sort(v.begin(), v.end(), Compare)` ordina il contenuto del vettore usando `Compare` come relazione d'ordine per gli elementi.
- `Compare` confronta due coppie considerando solo la seconda componente e stabilendo che $(p, c) < (q, d)$ se e solo se $c > d$. In questo modo l'ordinamento ottenuto è decrescente in base alla frequenza delle parole.

Programma principale

```
int main() {
  std::ifstream f("i_promessi_sposi.txt");
  WordCountMap m = read_word_map(f);
  WordCountVec v = sort_word_map(m);
  for (int i = 0; i < 10 && i < v.size(); i++)
    std::cout << v[i].second << ": "
              << v[i].first << std::endl;
}
```

Nota

- Il programma deve essere eseguito nella stessa cartella in cui è presente il file `i_promessi_sposi.txt` che è incluso nell'archivio.

Esercizi

1. Modificare il programma che calcola la frequenza delle parole in un testo per cercare le 10 parole più frequenti che iniziano con una lettera maiuscola. Qual è il personaggio più citato de “I promessi sposi”?
2. Come faccio a sapere quante parole **diverse** compaiono ne “I promessi sposi”?
3. Qual è la parola più lunga che compare ne “I promessi sposi?” e quante volte compare?
4. Definire una classe `MultiSet` per rappresentare **multinsiemi mutabili** di numeri interi. A tale scopo usare una mappa i cui elementi sono coppie (x, n) di numeri in cui x è un elemento del multinsieme ed n la sua **molteplicità**, ovvero il numero di occorrenze di x nel multinsieme. Definire i seguenti metodi:
   - `size` per calcolare la **cardinalità** del multinsieme;
   - `operator[]` per determinare la molteplicità di un elemento nel multinsieme. **Suggerimento**: usare il metodo `find`;
   - `insert(x, n)` per aggiungere n occorrenze di x al multinsieme;
   - `remove(x, n)` per rimuovere n occorrenze di x dal multinsieme (attenzione!);
   - `_union` per calcolare l’unione di due multinsiemi.