Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Programmazione Concorrente e Distribuita
Soluzioni che sfruttano il parallelismo: esercizi

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
luigi.lavazza@uninsubria.it

Esercizio 1: somma righe matrice

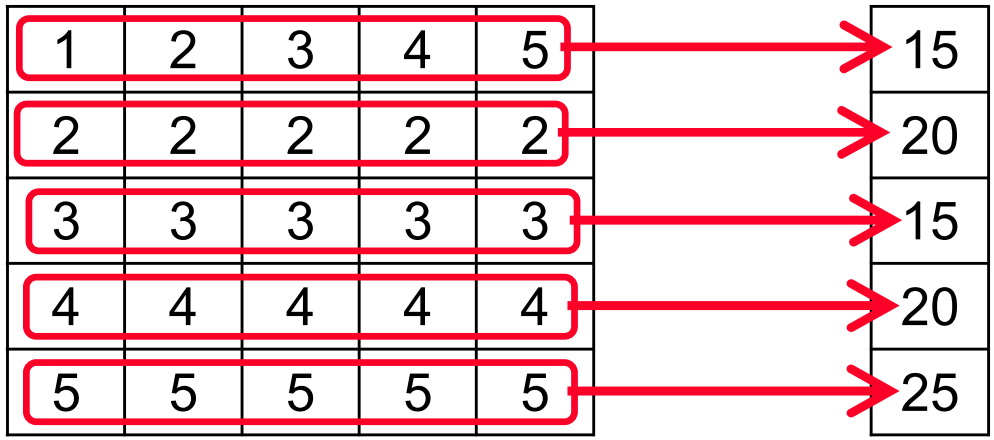
- Sia A una matrice di dimensione n × m.
  - Sia a_{ik} l'elemento della matrice alla riga i, colonna k

- Si vuole ottenere il vettore C i cui elementi sono definiti come segue:

$$c_i = \sum_{k=0}^{m-1} a_{ik}$$

  dove $0 \leq i < n$.

- Ad es.,



Esercizio 1: somma righe matrice

- Implementare un programma per calcolare gli elementi c_i che sfrutti la concorrenza.

Esercizio 1: somma righe matrice

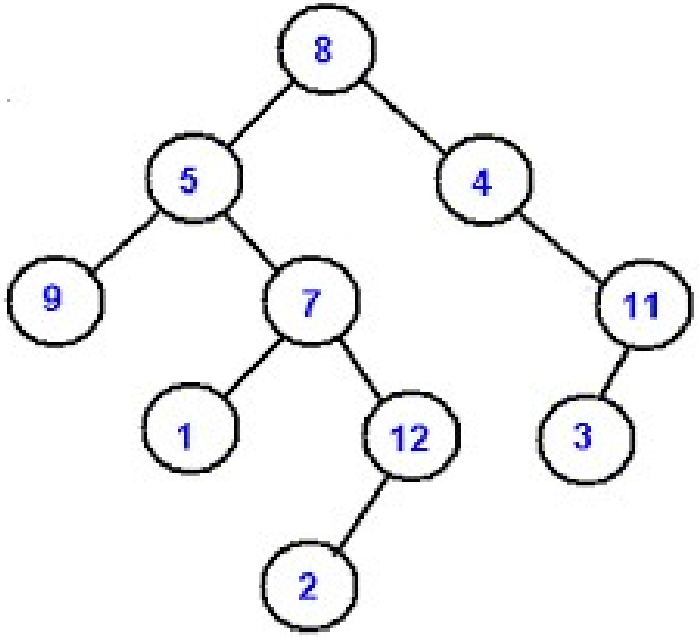
- Dobbiamo calcolare, per ogni i

$$c_i = \sum_{k=0}^{m-1} a_{ik}$$

- Un programma sequenziale fa prima tutti i calcoli per i=0, poi tutti i calcoli per i=1, ecc...
- Un programma multithread può creare un thread che si occupa di c_0, un altro che si occupa di c_1, uno per c_2, ecc.
  - Cioè si può creare un thread per ogni riga della matrice.
  - Ciascun thread procede concorrentemente agli altri

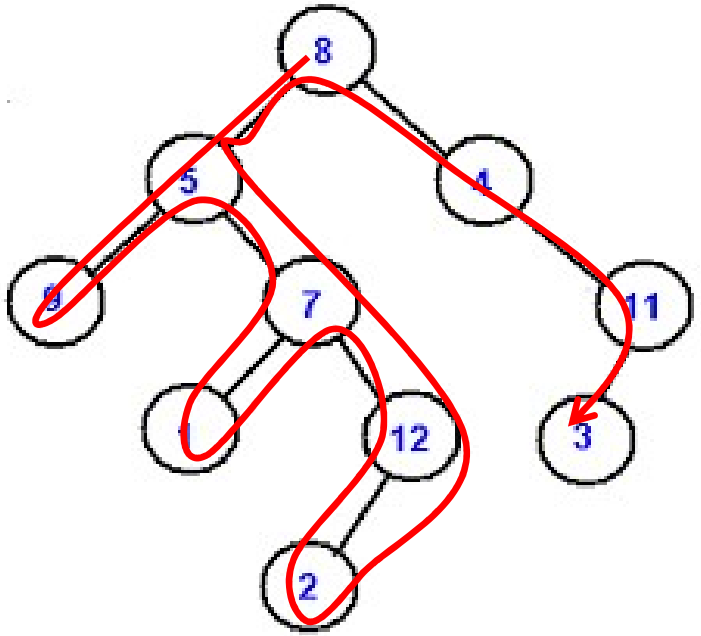
Esercizio 2: ricerca in alberi binari

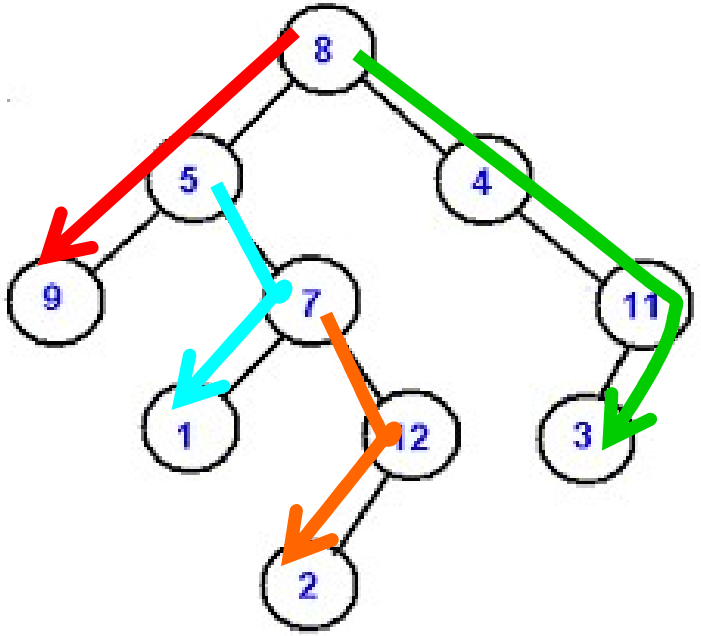
- Scrivere un programma che sfrutta molteplici thread per effettuare una ricerca in un albero binario dato.



Esercizio 2: ricerca in alberi binari

- Suggerimento.
- Un programma sequenziale visita tutto l'albero.





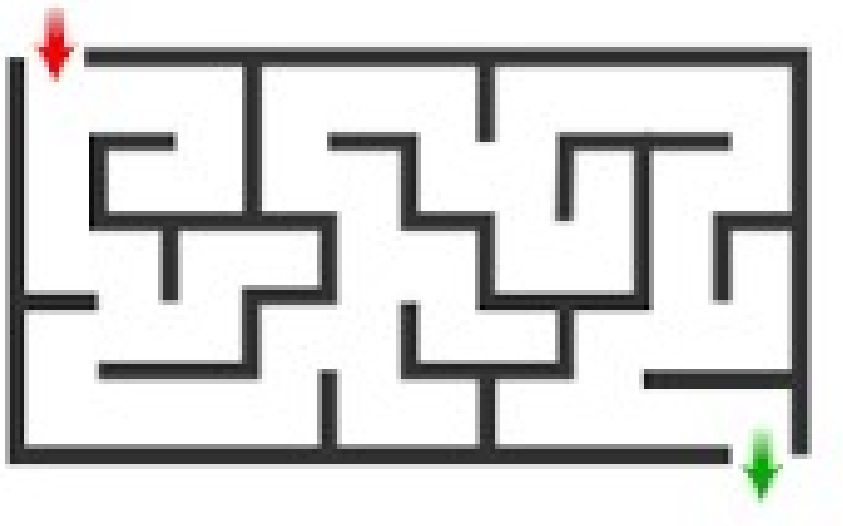
- Un programma multithread può far visitare ciascuna «sezione» dell'albero da un thread dedicato.

Perché usare la concorrenza

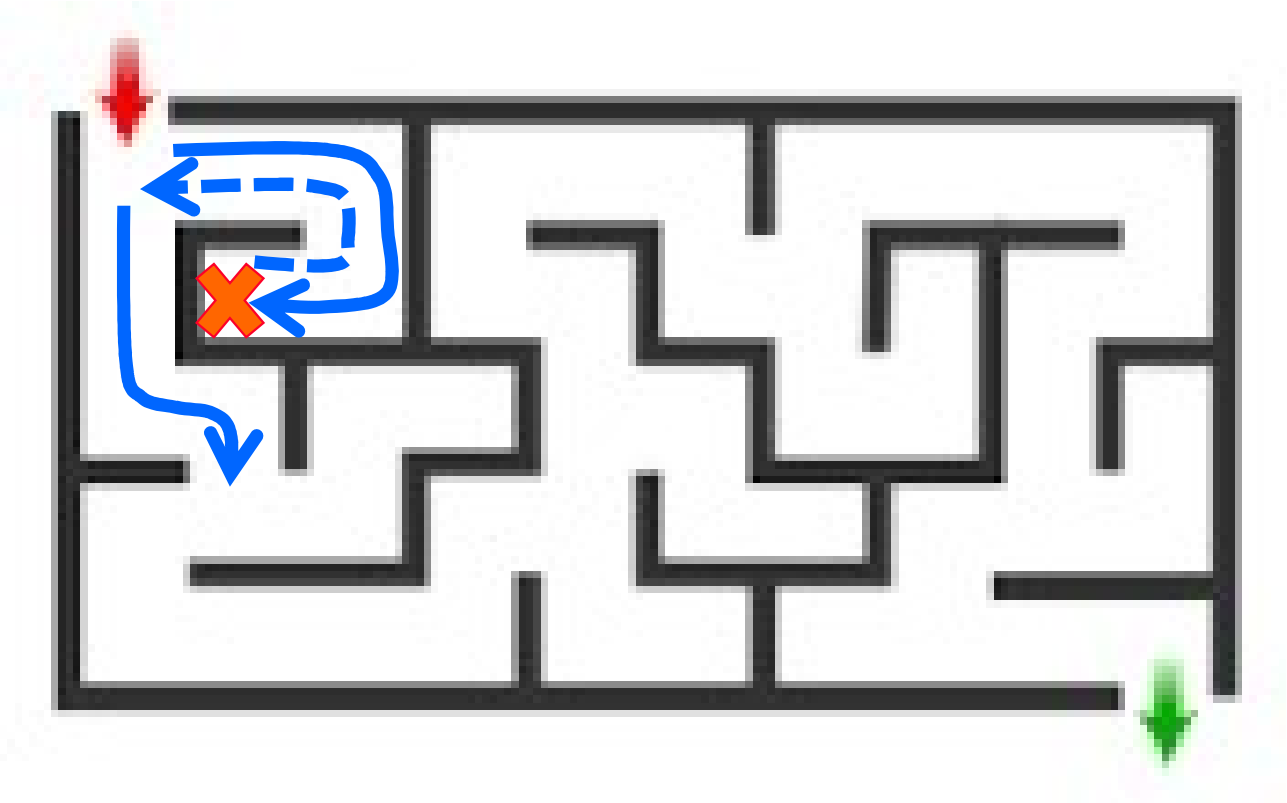
- Nei casi visti, l'unico vantaggio deriva dalla possibilità di abbreviare i tempi complessivi di calcolo.
- Ad es., la visita sequenziale dell'albero visto all'esempio precedente richiede di visitare 12 nodi
- Lo stesso albero può essere vistato da thread concorrenti, ciascuno dei quali visita al massimo 3 nodi. Se disponiamo di un processore per ogni thread il tempo di elaborazione è più breve.
  - NB: notate però che i thread non iniziano tutti insieme, quindi ci può essere un po' di ritardo

- Nel caso successivo, si aggiunge il vantaggio della semplicità di programmazione.

Esercizio 3: trovare la strada in un labirinto



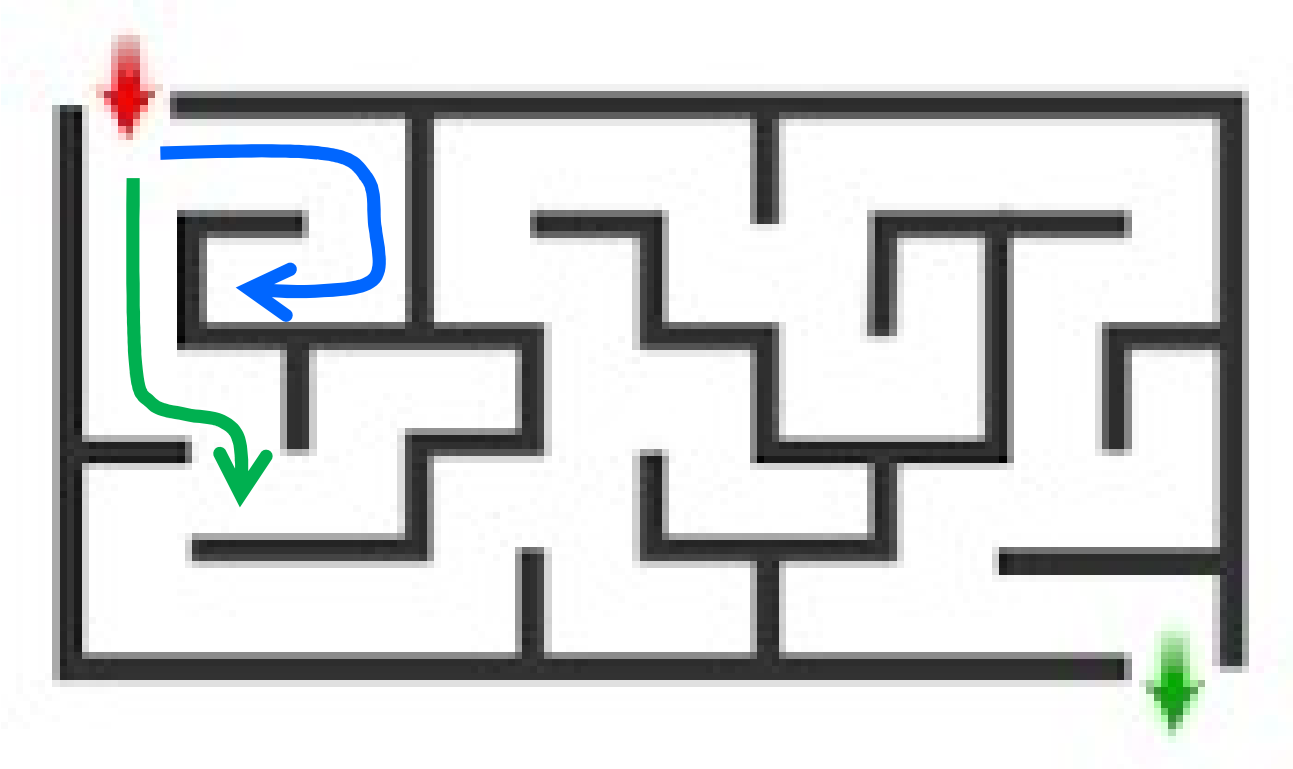
Esercizio 3: trovare la strada in un labirinto



- Un programma tradizionale quando trova un bivio deve:
  - Esplorare il primo ramo
  - Se non ha trovato l'uscita, tornare indietro ed esplorare il ramo alternativo: serve un programma ricorsivo con backtrack.

Esercizio 3: trovare la strada in un labirinto

- Un programma multithread quando trova un bivio può proseguire da una parte e generare un thread che esplori la strada alternativa



- NB: quando un thread arriva in un posto già visitato da un altro thread può terminare.