Laboratorio di Algoritmi e Strutture Dati

Laboratorio, esercizio 2

"It's the latest innovation in office safety.
When your computer crashes, an air bag is activated
so you won't bang your head in frustration."

Programma concettuale e videolezioni

	video	argomento
lista, array	13	Liste: inserimento, ricerca, cancellazione
	14	Pile e code: inserimento e cancellazione
	15	Heap, code di prioritá, e *HeapSort*
	16	Tabelle hash
	17	Strutture per insiemi disgiunti
	18	Esercizi su pile, code, heap, tabelle hash, e insiemi disgiunti
albero	19	Alberi: visite e problemi collegati
	20	Alberi binari di ricerca: inserimento, ricerca, cancellazione
	21	Alberi red-black: inserimento, ricerca
	22	Laboratorio: esercizio 2
	23	Alberi B: inserimento, ricerca
	24	Esercizi su alberi

Secondo esercizio

Nel secondo esercizio di laboratorio vogliamo confrontare l'efficienza di due strutture dati, le tabelle hash e gli alberi red-black, nel contesto delle operazioni di inserimento e di ricerca. L'obbiettivo e verificare se una delle due strutture si comporta meglio dell'altra in maniera sperimentale. Le chiavi saranno interi di dimensione inferiore alla parola di memoria.

Utiliziamo liste collegate e funzione modulo opportunamente instanziata.

Secondo esercizio: idea

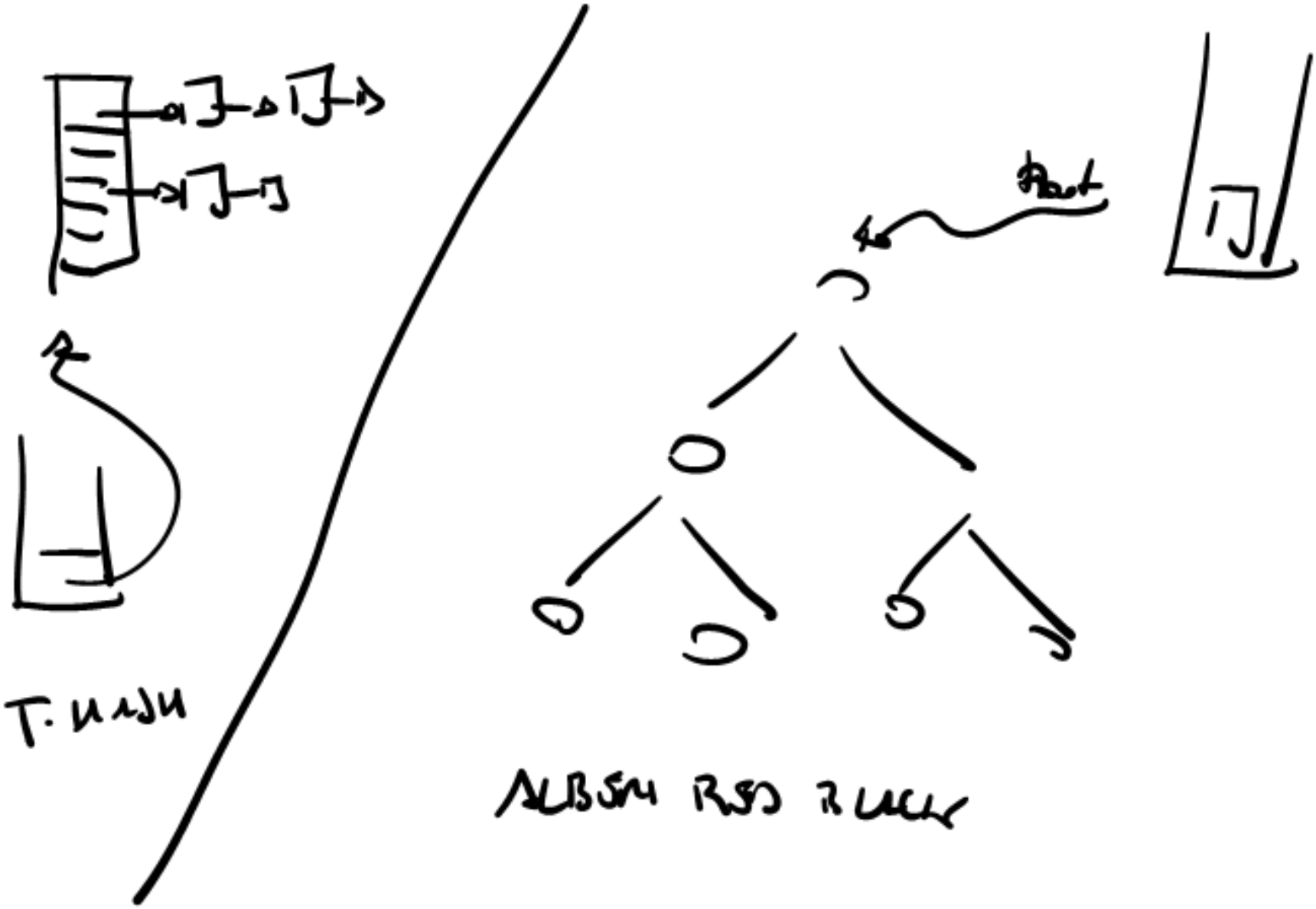
n elementi presenti, m posti nella tabella

Da un punto di vista asintotico immaginiamo una tabella hash implementata con conflitti risolti via chaining, funzione modulo, e liste singolarmente o doppiamente collegate. La teoria ci da le seguenti prestazioni, nel caso medio:

- Inserimento: $\Theta(1)$ ($\Theta(1 + \frac{n}{m})$ se non si ammettono ripetizioni);
- Ricerca: $\Theta(1 + \frac{n}{m})$.

Le stesse operazioni possono essere eseguite usando un albero red-black, che, sempre nel caso medio, si comporta cosi:

- Inserimento: $\Theta(lg(n))$;
- Ricerca: $\Theta(lg(n))$.



Secondo esercizio

Esercizio 2

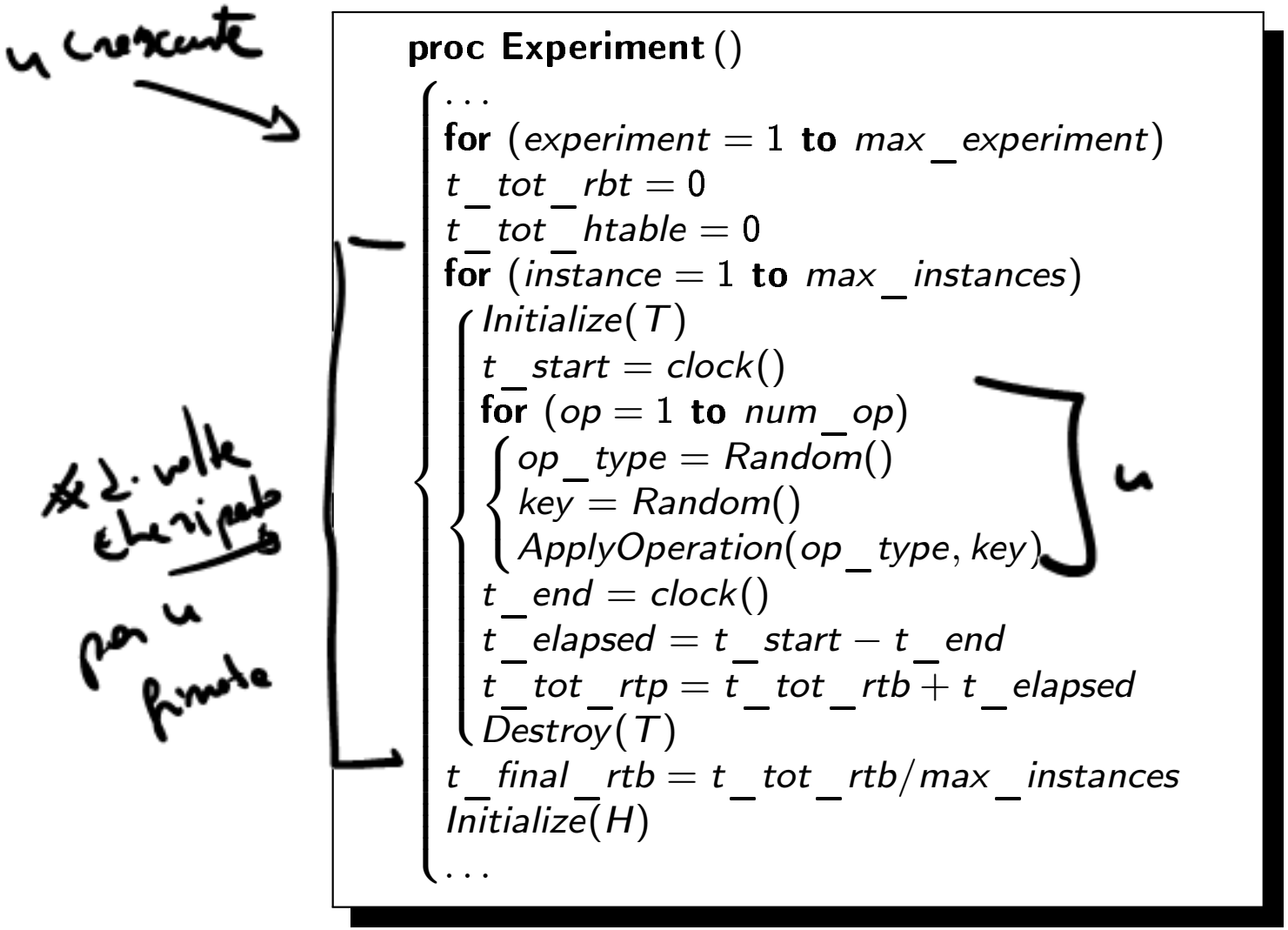
Realizzare un esperimento implementando sia una tabella hash con conflitti risolti via chaining sia un albero red-black. Misurare il tempo medio per n operazioni con n crescente, chiavi intere maggiori di zero casuali, su entrambe le strutture, e confrontarlo. Realizzare il confronto per diversi valori di m (dimensione fissata della tabella hash) e diverse proporzioni tra numero di inserimenti e numero di ricerche.

Il risultato richesto prevede:

- Una rappresentazione grafica delle curve di tempo;
- Una dimostrazione di correttezza delle implementazioni attraverso funzioni antagoniste e confronto;
- L'implementazione delle strutture dati in maniera esplicita e corretta.

Secondo esercizio

Una bozza dello pseudo-codice della soluzione:



```
proc Experiment ()
  ...
  for (experiment = 1 to max_experiment)
  t_tot_rbt = 0
  t_tot_htable = 0
  for (instance = 1 to max_instances)
    Initialize(T)
    t_start = clock()
    for (op = 1 to num_op)
      op_type = Random()
      key = Random()
      ApplyOperation(op_type, key)
    t_end = clock()
    t_elapsed = t_start − t_end
    t_tot_rtp = t_tot_rtb + t_elapsed
    Destroy(T)
  t_final_rtb = t_tot_rtb/max_instances
  Initialize(H)
  ...
```

Secondo esercizio: possibili risultati

Alcuni possibili risultati. Andamento con $m = 3$, 40 % inserimenti, 60 % ricerche, duplicati ammessi.

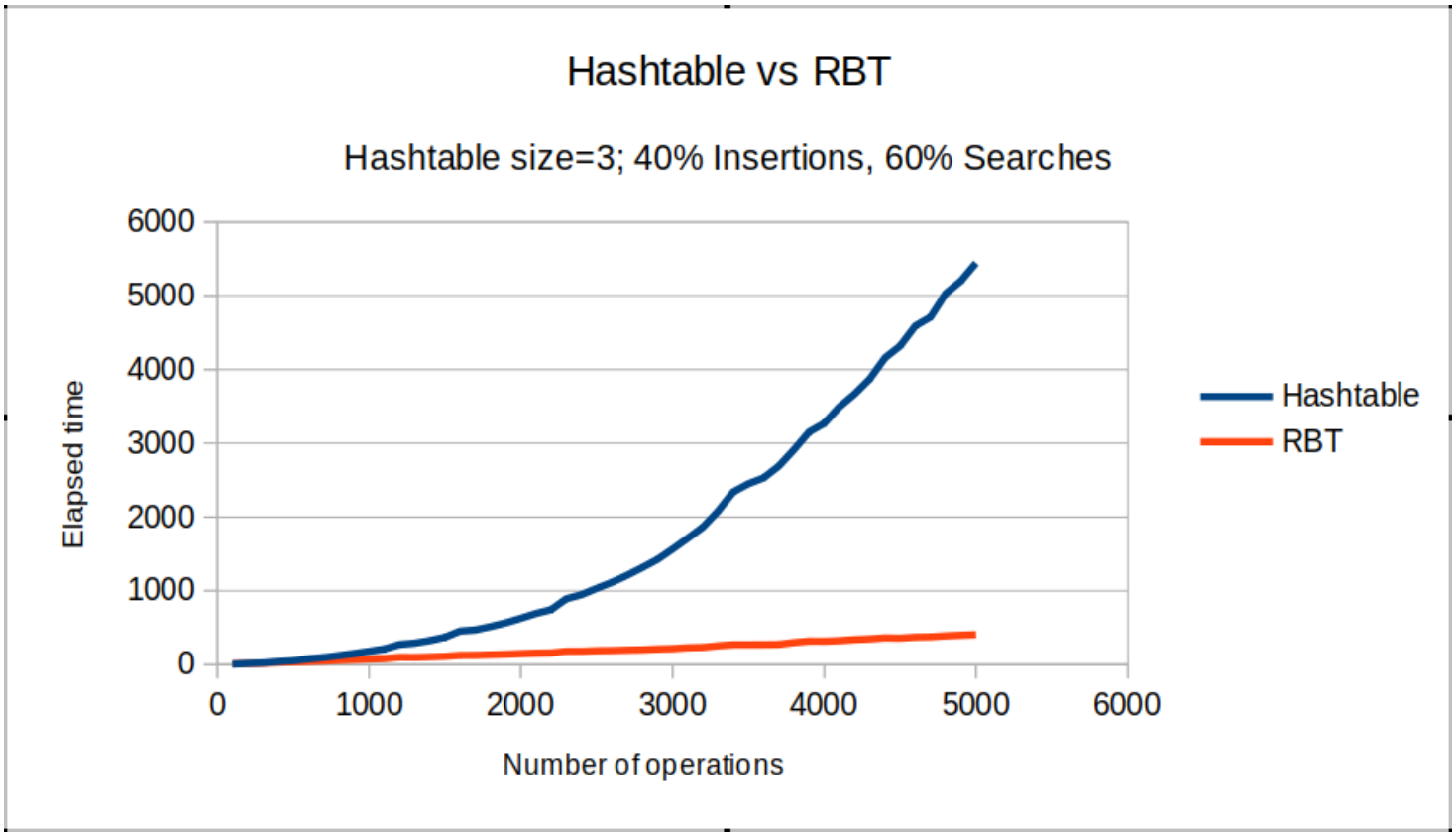


Figura: Andamento globale per il secondo problema.

Secondo esercizio: possibili risultati

Alcuni possibili risultati. Andamento con $m = 3$, 70 % inserimenti, 30 % ricerche, duplicati ammessi.

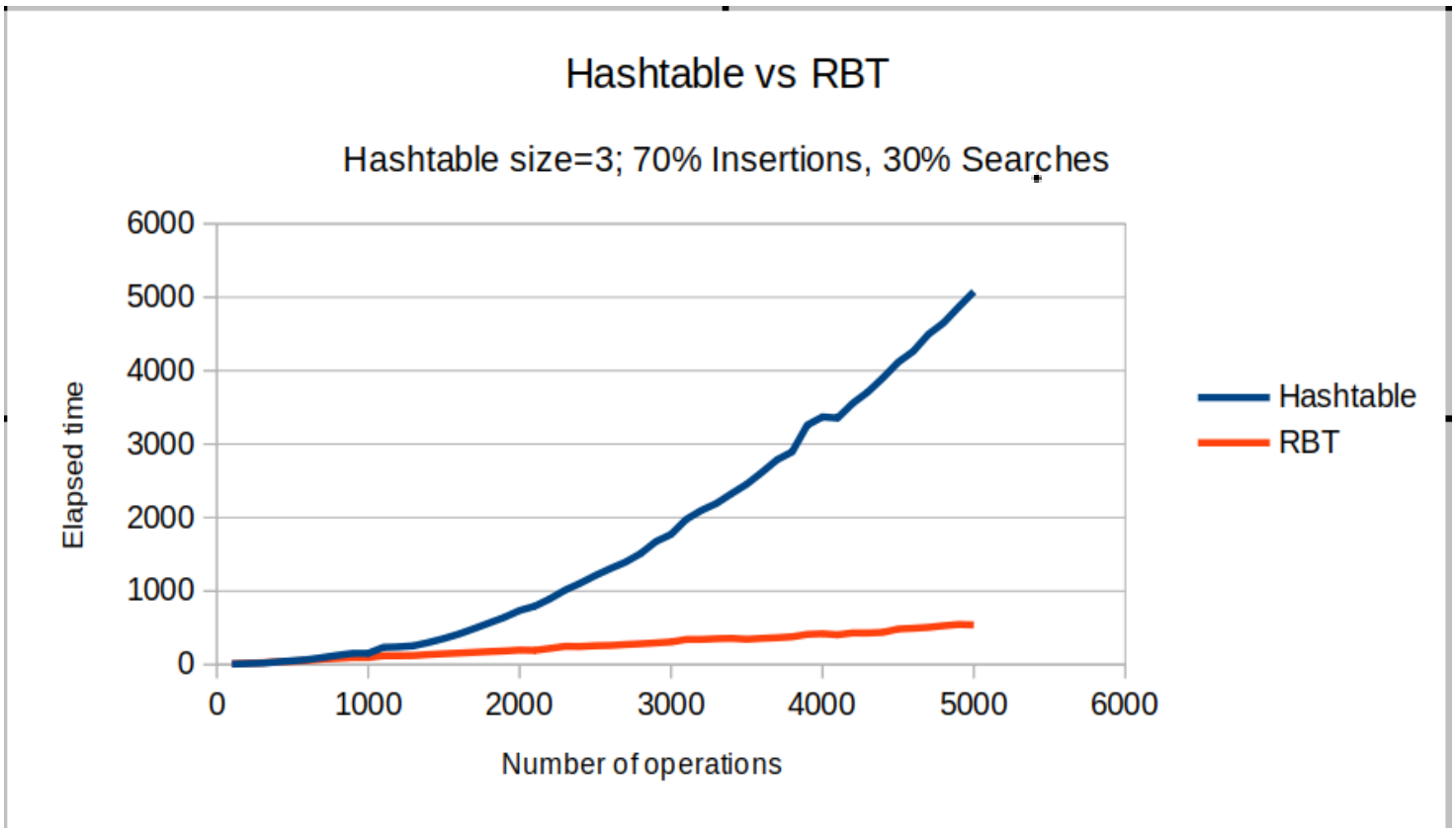


Figura: Andamento globale per il secondo problema.

Secondo esercizio: possibili risultati

Alcuni possibili risultati. Andamento con $m = 13$, 40 % inserimenti, 60 % ricerche, duplicati ammessi.

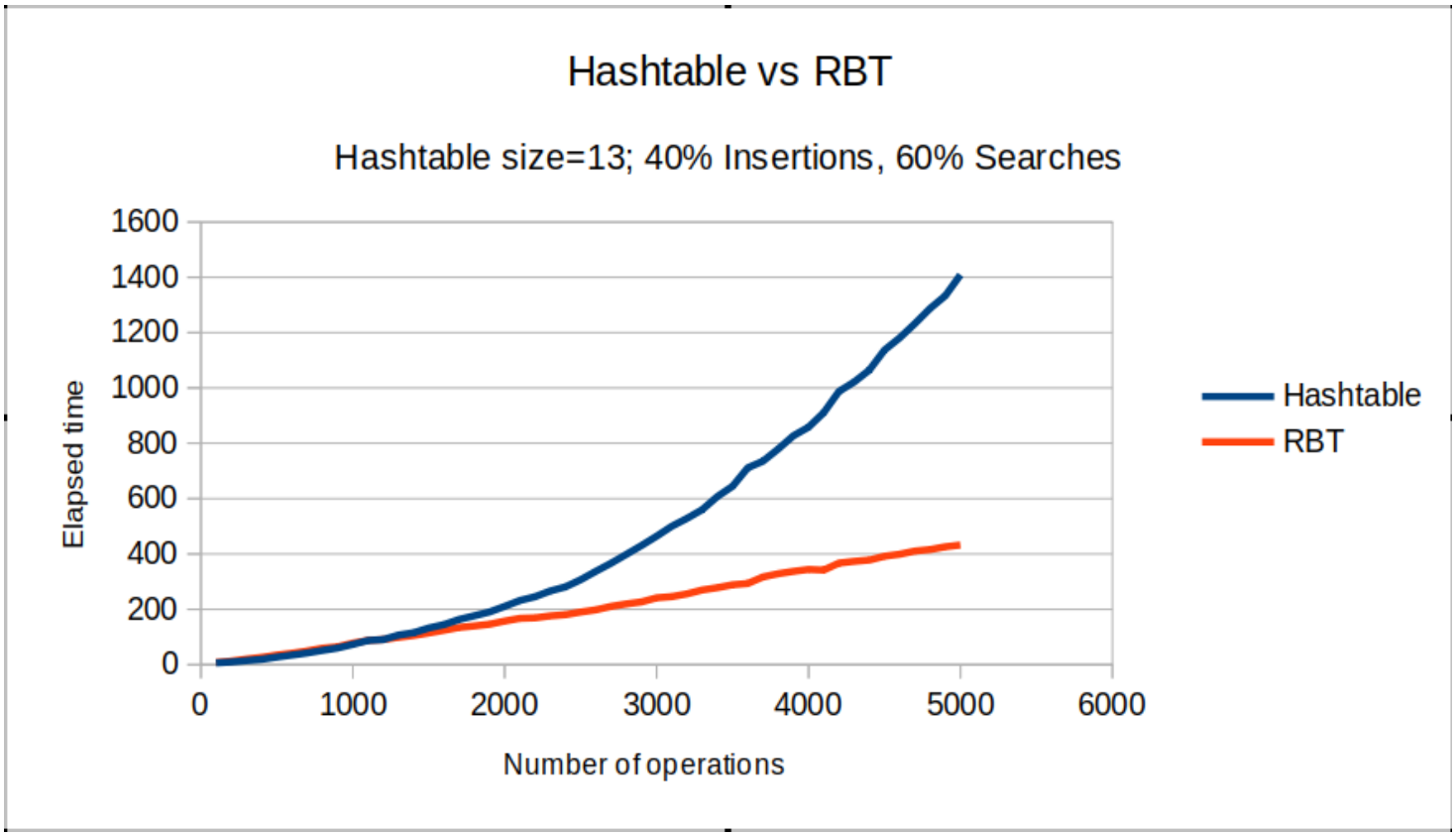


Figura: Andamento globale per il secondo problema.

Secondo esercizio: possibili risultati

Alcuni possibili risultati. Andamento con $m = 59$, 40 % inserimenti, 60 % ricerche, duplicati ammessi.

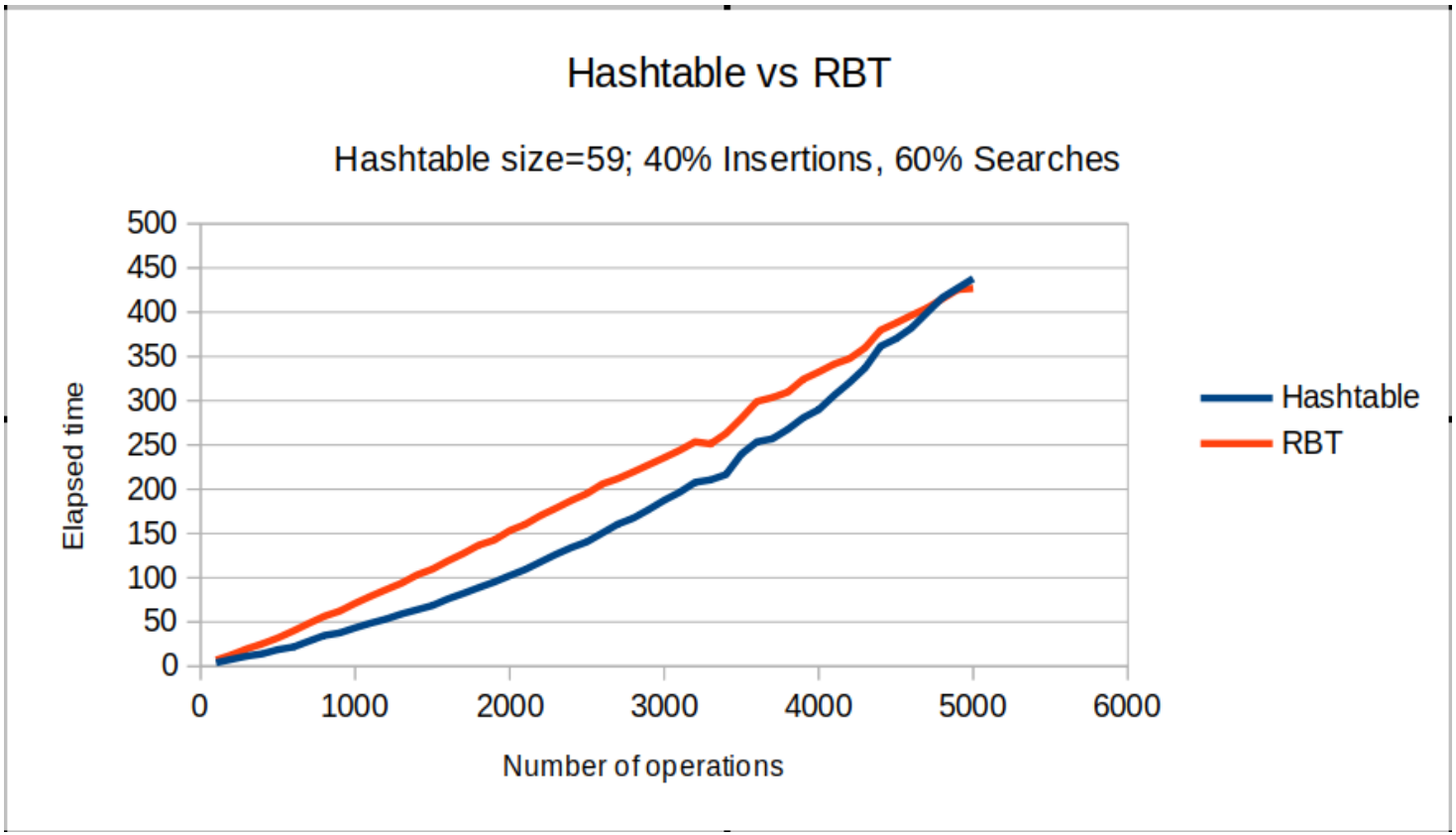


Figura: Andamento globale per il secondo problema.

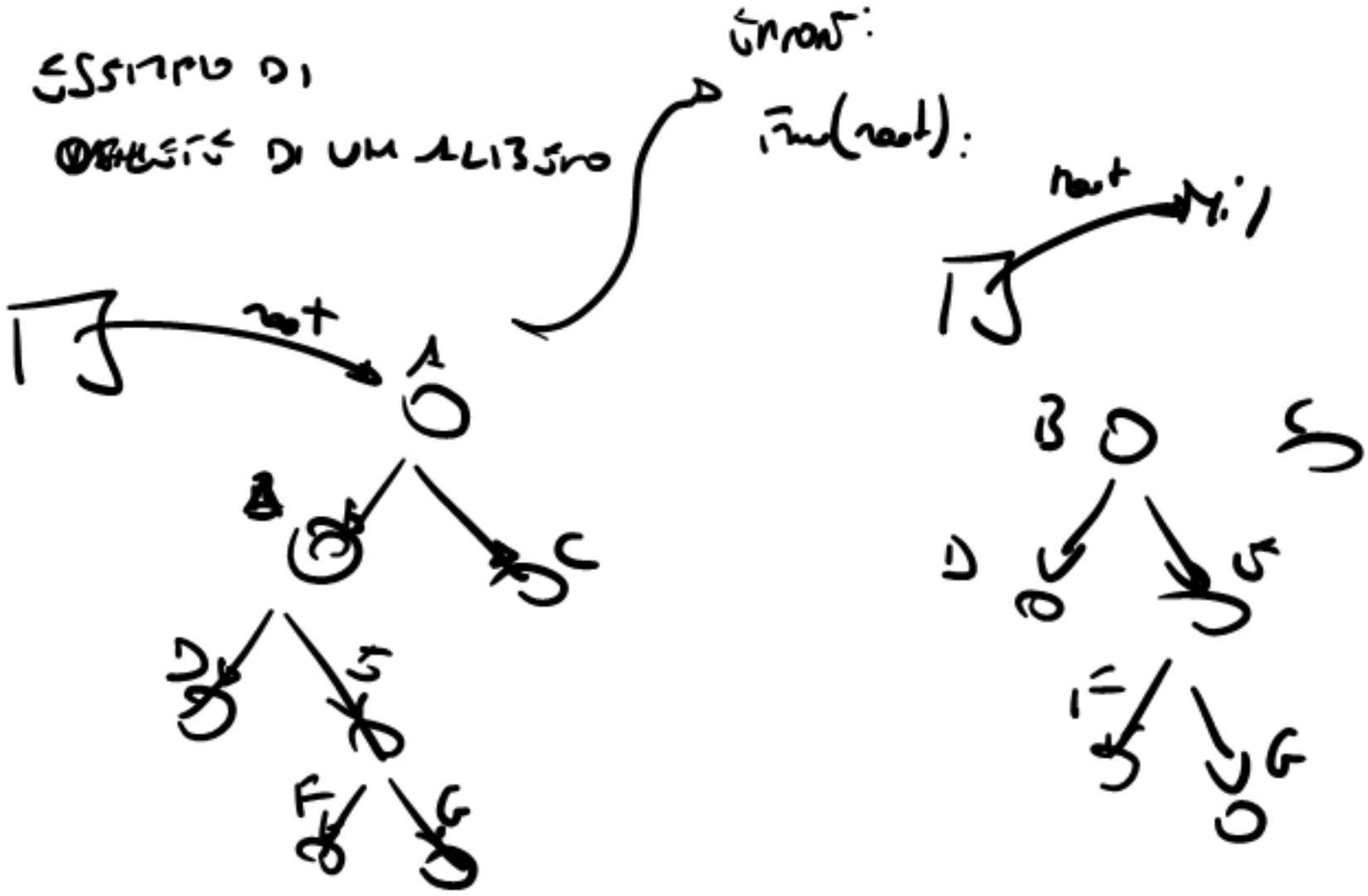
Secondo esercizio: correttezza

Come possiamo dimostrare la correttezza della nostra implementazione? Nel caso delle tabelle hash con conflitti risolti via chaining, una buona funzione antagonista è semplicemente l'inserimento e la ricerca di chiavi note; per esempio, scegliamo 10 chiavi, le insieriamo tutte, e poi le cerchiamo. Nel caso degli alberi, invece, possiamo usare una visita in order con controllo di ordinamento. Inoltre, poichè le due strutture devono fare le stesse operazioni, il risultato può essere comparato per ulteriore conferma della correttezza.

Secondo esercizio: correttezza

Cosa significa che le strutture dati devono essere esplicite? Significa semplicemente che devono essere implementate con l'uso delle *struct*, le loro caratteristiche incapsulate, e le loro operazioni tipizzate. Per esempio, per le liste concatenate avremo:

- Il tipo lista;
- Il tipo nodo;
- L'operazione di inserimento (lista, nodo) (oppure (lista, chiave));
- L'operazione di ricerca (lista, chiave) ;
- L'operazione (antagonista) di visita (lista).


root
root
root