Laboratorio di Algoritmi e Strutture Dati

Secondo esercizio, prima parte: alberi binari di ricerca (punti: 0)

"Shouldn't we hold off on artificial intelligence until we figure out actual intelligence?"

Esperimenti con strutture dati

Come si effettuano esperimenti con strutture dati?

La soluzione più ovvia è quella di costruire un esperimento nel quale chiavi vengono generate casualmente in maniera sistematica e utilizzate nelle operazioni che si vogliono testare.

Misuriamo il tempo per un numero fissato di operazioni e lo salviamo; poi distruggiamo tutta la struttura e ricominciamo per un numero di operazioni più alto.

Alberi binari di ricerca: una struttura ricorsiva

Tutti i linguaggi di programmazione imperativi permettono la dichiarazione di strutture dati ricorsive.

La prima struttura ricorsiva che si vede normalmente è la lista: una lista è una chiave seguita da una lista.

Gli alberi, e in particolare gli alberi binari di ricerca, si dichiarano esattamente nello stesso modo.

```
struct tree_node{
    int key
    *struct tree_node parent
    *struct tree_node left_child
    *struct tree_node right_child
}
```

```
struct tree{
    int cardinality
    *struct tree_node root
}
```

Alberi binari di ricerca: gestione dell'esperimento singolo

Procediamo dunque come nel caso del nostro primo esperimento, con *InsertionSort*.

Al livello più basso, il singolo passo consiste nel generare un numero fissato di chiavi casuali, inserirle nella struttura dati e scegliere un'operazione tra ricerca e cancellazione, ed eseguirla.

In questo modo, però, rischiamo di non avere un buona misura, perchè se si hanno tante cancellazioni quanti inserimenti, la struttura potrebbe essere sempre vuota, nascondendo l'efficienza di una struttura migliore rispetto ad una peggiore.

Per risovere questo problema, facciamo un'esperimento semi-casuale.

Alberi binari di ricerca: gestione dell'esperimento

```
proc SingleExperiment (max_keys, max_search, max_delete, max_instances)
{ t_tot = 0
  for (instance = 1 to max_instances)
  { Initialize(T)
    t_start = clock()
    for (key = 1 to max_keys)
    { BSTTreeInsert(T, Random())
    for (search = 1 to max_search)
    { BSTTreeSearch(T, Random())
    for (delete = 1 to max_delete)
    { BSTTreeDelete(T, Random())
    t_end = clock()
    t_elapsed = t_end − t_start
    t_tot = t_tot + t_elapsed
    Empty(T)
  return t_tot/max_keys
```

Alberi binari di ricerca: distruzione della struttura

Alla fine di ogni esperimento singolo, possiamo operare una distruzione precisa della struttura in maniera ricorsiva, per assicurare una buona gestione della memoria. Non calcoliamo il tempo di questa operazione.

Quindi, possiamo ripetere lo stesso esperimento un certo numero di volte per assicurare che non ci siano **outliers** statistici, ed infine organizzare l'esperimento completo con numero di chiavi crescente.

Alberi binari di ricerca: esperimento

```
proc Experiment (min_keys, max_keys)
{ step = 10
  max_instances = 5
  percentage_search = 60
  for (keys = min_keys to max_keys step step)
  { srand(SEED)
    max_search = keys * percentage_search/100
    max_delete = keys - max_search
    time = SingleExperiment(keys, max_search, max_delete, max_instances)
    print(time)
    SEED = SEED + 1
  }
}
```

Alberi binari di ricerca: esperimento

Che tipo di funzione antagonista si può usare in un esercizio come questo?

Sappiamo che la **visita in ordine** di un albero binario di ricerca deve restituire un array ordinato; se visitamo l'albero e non troviamo un insieme ordinato, questo certamente è indice di un errore strutturale che deve essere corretto. Questa potrebbe essere una buona funzione antagonista.

Quindi vogliamo realizzare un esperimento che consiste nel misurare il tempo necessario ad effettuare un certo numero, crescente, di operazioni standard di inserimento e cancellazione in un albero binario di ricerca inizialmente vuoto.

Il risultato richesto prevede una rappresentazione grafica delle curve di tempo e una dimostrazione sperimentale di correttezza attraverso test randomizzati e funzioni antagoniste.

Conclusione

Sperimentare di prima mano con le strutture dati ci dà l'occasione di capire davvero come queste sono fatte. Questa è condizione necessaria per un uso corretto delle strutture già implementate in librerie comuni.