ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA
Universitat Rovira i Virgili

Estructures de Dades 2022

Pràctica 1: Part 2. *Taula de hash i anàlisi de costos: 6pts*

En aquesta segona part de la pràctica implementarem una estructura de tipus taula de hash, i compararem el cost de les operacions d'aquesta estructura versus les operacions de la llista enllaçada.

L'estructura haurà de respectar la següent interfície:

```
void Crear();
```
- Constructor per inicialitzar la taula.

```
void Inserir(K key, T data);
```
- Funció per tal d'inserir un element a la taula de Hash.
- Si l'element ja existia, actualitza el seu valor
- L'operació llença una excepció en cas que no es pugui inserir

```
T Obtenir(K key);
```
- Funció que retorna l'element que té la clau K.
- L'operació llença una excepció en cas que no es pugui obtenir

```
int Buscar(K key);
```
- Funció que comprova si un element està a la taula.
- La funció retorna el cost de l'operació. Nombre d'elements que s'hagin accedit per tal de comprovar si l'element existeix o no.
- L'operació llença una excepció en cas que l'element no s'hagi trobat. La mateixa excepció contindrà informació del nombre d'elements que s'han accedit per comprovar si l'element buscat existeix o no.

```
int Mida();
```
- Retorna el nombre d'elements que conté la taula en aquest moment.

```
void Esborrar(K key);
```
- Funció per tal d'esborrar un element de la taula.
- L'operació llença una excepció en cas que l'element no s'hagi trobat.

```
Llista<T> ObbtenirValors();
```
- `Retorna una llista amb tots els valors de la taula`

```
Llista<K> ObtenirClaus();
```
- `Retorna una llista amb tots les claus de la taula`

```
Float ObtenirFactorDeCàrrega();
```
- `Retorna el factor de càrrega actual`

Implementació de la taula de hash encadenada indirecta (3pts):

Es demana implementar taula de hash encadenada indirecta amb la vostra propi TAD per a cada node associat a una posició. Tingueu en compte que poden haver-hi col·lisions i per tant, haureu de tenir un "Següent Col·lisió".

Per altra banda, aprofitarem l'estructura llista que hem implementat en la primera part de la pràtica per a les llistes de retorn de mètodes com ObtenirValors o ObtenirClaus.

Haureu de decidir quina és la vostra funció de hash, l'única condició és que ha de complir amb les propietats que hem vist a classe.

La taula ha de respectar l'especificació de taula donada. Aquesta taula tindrà dos components genèrics que s'inicialitzaran quan es creï una instància de la taula: el tipus de la clau i el tipus del valor guardat.

La implementació del TAD ha de realitzar-se combinant memòria dinàmica per a les llistes de col·lisions i amb memòria estàtica per a la de posicions i punters.

Redimensionat de la taula (1pt) [OPCIONAL]

El fet d'utilitzar memòria estàtica pot suposar un problema quan treballem amb moltes més dades que posicions. Es proposa fer una redimensió de la taula en funció del factor de càrrega, és a dir, quan el factor de càrrega superi un límit establert, s'haurà de moure tot a una taula més gran.

Validació:

Per tal de posar a prova la vostra implementació, el joc de proves s'haurà de realitzar amb el `TAD: Ciutadà`. El mateix de la primera part de la pràctica. En aquesta segona part, considerarem que el DNI del ciutadà és la clau de la taula de hash.

En el mètode Main s'haurà de veure clarament tot el vostre joc de proves i quina és la intenció de cada prova a més d'afegir els resultats a l'informe.

Anàlisis del cost computacional de l'operació buscar (2 punts)

Un cop tingueu la vostra llista i taula de hash funcionant i perfectament testejades amb el TAD ciutadà, es demana que realitzeu un anàlisis comparatiu de l'operació Buscar utilitzant les dues estructures. Com a resultat de l'exercici, per a cada una de les dues implementacions, es demana fer una gràfica on es mostri el cost mitjà de buscar un element de la llista en funció de la mida de la llista.

Per fer aquesta gràfica, es crearan llistes d'enters, és a dir, per a la comparativa, en lloc d'utilitzar el TAD Ciutadà, farem servir `Integers`. Farem llistes de mides entre 1.000 i 50.000 elements, en passos de 1.000 elements. És a dir, llistes de mides: $N \in \{1.000, 2.000, \ldots, 49.000, 50.000\}$. Cada una de les llistes contindrà *N* elements que seran enters generats de manera aleatòria entre 1 i *N*/2. Per exemple, si la llista té longitud 6.000, els números aleatoris s'han de generar en el rang entre 1 i 3.000. Una vegada creada la llista, es realitzaran *N* cerques d'enters també

aleatoris a cada una de les llistes creades. Per a cada cerca es guardarà el nombre d'elements que hem hagut de consultar (valor que retorna l'operació Buscar o l'excepció corresponent).

Una vegada realitzat cada experiment, guardarem els resultats en un arxiu CSV de tres columnes, indicant: mida de la llista, el cost mitjà que ha retornat la consulta i la seva desviació estàndard; aquests estadístics es calcularan considerant els costos de les N cerques fetes (de cada mida de llista). Aquest arxiu és el que utilitzareu per a fer la gràfica, utilitzant el programa que vulgueu (gnuplot, grace, matplotlib, ggplot, full de càlcul, etc.).

Hauríeu d'obtenir una diferència considerable que reafirmi la relació de costos vista a classe i s'espera de vosaltres que expliqueu amb les vostres paraules els resultats obtinguts.

Lliurament

- Les pràctiques són individuals.
- Es fa el lliurament d'un únic arxiu `PR1-NOM_COGNOMS.zip` que conté:
  - Els arxius de codi font.
  - Els arxius amb els resultats del càlcul del cost mitjà de les cerques
  - Informe (en pdf) incloent explicacions dels aspectes més rellevants de les vostres implementacions, jocs de proves realitzats... al final de tot de l'informe també haureu d'incloure tot el vostre codi enganxat.

Puntuació

- Llista doblement encadenada (4 punts)
- Taula de hash (3 punts)
- Redimensió taula de hash (1 punt) (Opcional)
- Anàlisis del cost computacional (2 punts)