Estructures de Dades
Pràctica 1.3: Llista doblement encadenada i Skip List

Josep Bello Curto

2019/2020

ÍNDEX

Contenido

1. Llista doblement encadenada

1.1. Estructures

Utilitzo una estructura Node a la que guardo l'element d'aquella posició i punters al element anterior i següent.

A l'estructura de la llista, "llista_encadenada" guardo:

elems: Nombre d'elements que conte aquesta, així el cost de longitud es constant, i puc utilitzar aquest valor per tractar els casos en que tinc 0, 1 o +1 element de forma diferent fàcilment.

pdi: Punter al punt d'interès demanat a l'enunciat de la pràctica.

ghost: A causa d'implementar la llista amb un pdi es útil l'ús d'un element fantasma per tal de poder inserir en la primera posició de la llista. El punter next apuntarà sempre a first, i el punter previous sempre a nul.

first: Punter al primer element de la llista, el punter a la seva esquerra (previous) es sempre ghost. Aquest punter es el que utilitzo per anar iterant sobre la llista.

last: Punter al últim element de la llista, el punter a la seva dreta (next) es sempre nul. Tot i que en aquest cas no es crític, ja que la inserció es fa sempre al pdi, no al final de la llista, he decidit guardar aquest punter ja que així puc moure el pdi al final amb un cost constant. A més a més l'utilitzo per saber si estic al final de la llista, això també es podria fer mirant si el punter next es nul, però així queda el codi mes comprensible.

1.2. Codis d'error

SUCCESS: Tot ha anat be.

ERROR_CREAR: Ha sorgit un error al crear la llista.

ERROR_DESTRUIR: Ha sorgit un error al destruir la llista.

LLISTA_NO_CREADA: S'està intentant accedir a un punter de llista encadenada que és nul.

LLISTA_BUIDA: La llista està buida i s'està intentant eliminar un element o destruir-la.

ELEMENT_NO_CREAT: L'element al que es vol accedir no ha estat creat.

MEMORIA_INSUFICIENT: No hi ha prou memòria per afegir un element a la llista.

OPERACIO_NO_PERMITIDA: S'està intentant fer una operació que no es pot ja que trenca la integritat de la llista. Ex: eliminar el fantasma.

1.3. Detalls d'implementació

Destruir(llista_encadenada): Aquesta funció comença al segon element, si existeix, i va fent un free de l'anterior sempre.

Inserir(llista_encadenada, elem): Insereix l'element passat per paràmetre a la dreta del pdi, es per això que tinc l'element fantasma al principi de la llista.

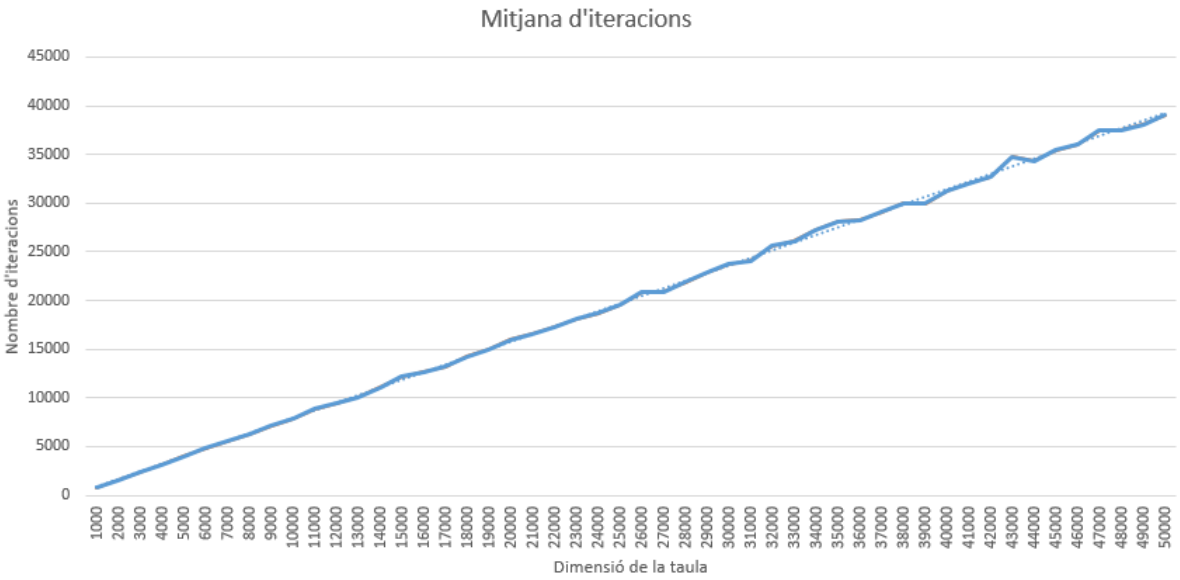
Esborrar(llista_encadenada): Esborra l'element apuntat pel pdi, però si aquest es el fantasma retorna un codi d'error (OPERACIO_NO_PERMITIDA).

En aquest cas el programa principal ens pregunta si volem fer un test unitari de cada funció o executar la prova de buscar sobre una llista de 1000 elements.

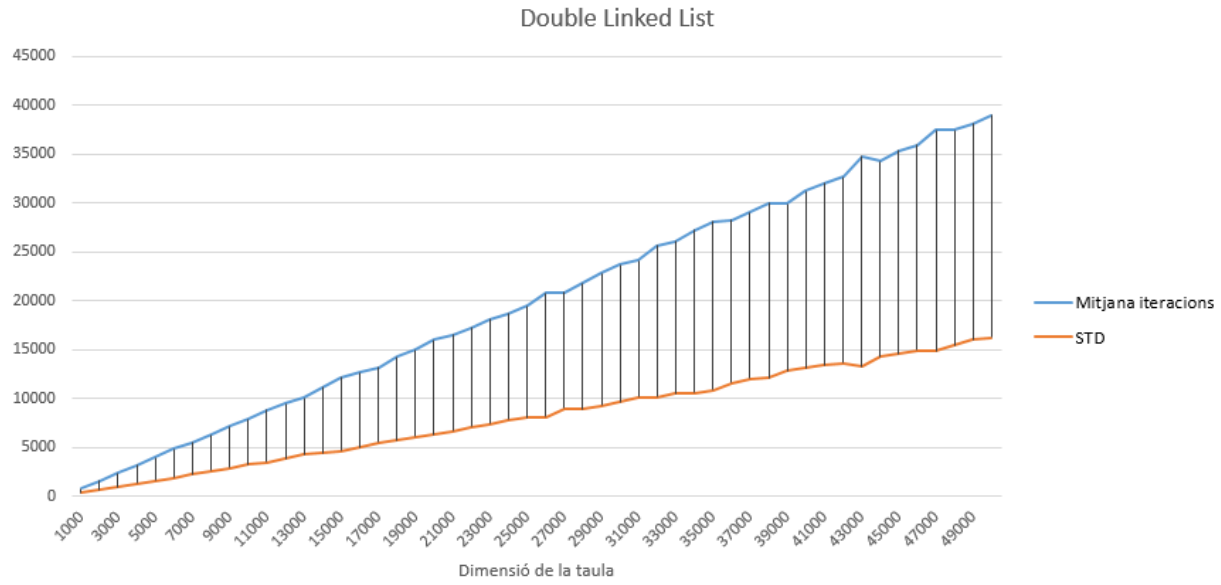
1.4. Funcions extra

Imprimir_Llista(llista_encadenada ll): Mostra per pantalla tota la informació de la llista i la llista començant des de first fins last i des de last fins first.

1.5. Gràfiques de cost i anàlisi



En aquesta gràfica observem com el cost de buscar un element a la llista es O(n). Amb línia continua hi s'observen les dades extretes del programa, i en discontinua la línia de tendència



En aquesta gràfica observem com la desviació estàndard de buscar un element es també lineal i proporcional a la mitjana d'iteracions.

2. SkipList

2.1. Estructures

Utilitzo una estructura Node a la que guardo l'element d'aquella posició i punters al element anterior, següent, de dalt i baix.

A l'estructura "skip_list" guardo:

elemNumber: Nombre d'elements de la llista.

height: Nombre de capes de la llista.

topLeft: Punter a l'element de dalt a l'esquerra.

2.2. Codis d'error

SUCCESS: Tot ha anat be.

ERROR_CREAR: Ha sorgit un error al crear la llista.

ERROR_DESTRUIR: Ha sorgit un error al destruir la llista.

LLISTA_NO_CREADA: S'està intentant accedir a un punter de llista encadenada que és nul.

LLISTA_BUIDA: La llista està buida i s'està intentant eliminar un element o destruir-la.

ELEMENT_NO_CREAT: L'element al que es vol accedir no ha estat creat.

MEMORIA_INSUFICIENT: No hi ha prou memòria per afegir un element a la llista.

OPERACIO_NO_PERMITIDA: L'operació que l'ha retornat s'ha fet sobre un element protegit.

2.3. Detalls d'implementació

Inserir(skip_list, elem): Insereix l'element passat per paràmetre al seu lloc de la llista. He decidit utilitzar el random de C inicialitzant-lo amb time(NULL) per decidir quantes capes pujar l'element, crido a la funció rand() i faig una and amb 1. He detectat que amb aquesta forma queda una llista molt més ben distribuïda que amb el random que ens doneu. Si l'element es un 0 es retorna un error ja que es l'element utilitzat com a sentinella inicial.

insertPos(skip_list, elem): Aquesta funció es la base de moltes altres, retorna l'element que va a l'esquerra del nou node i esta a l'última capa

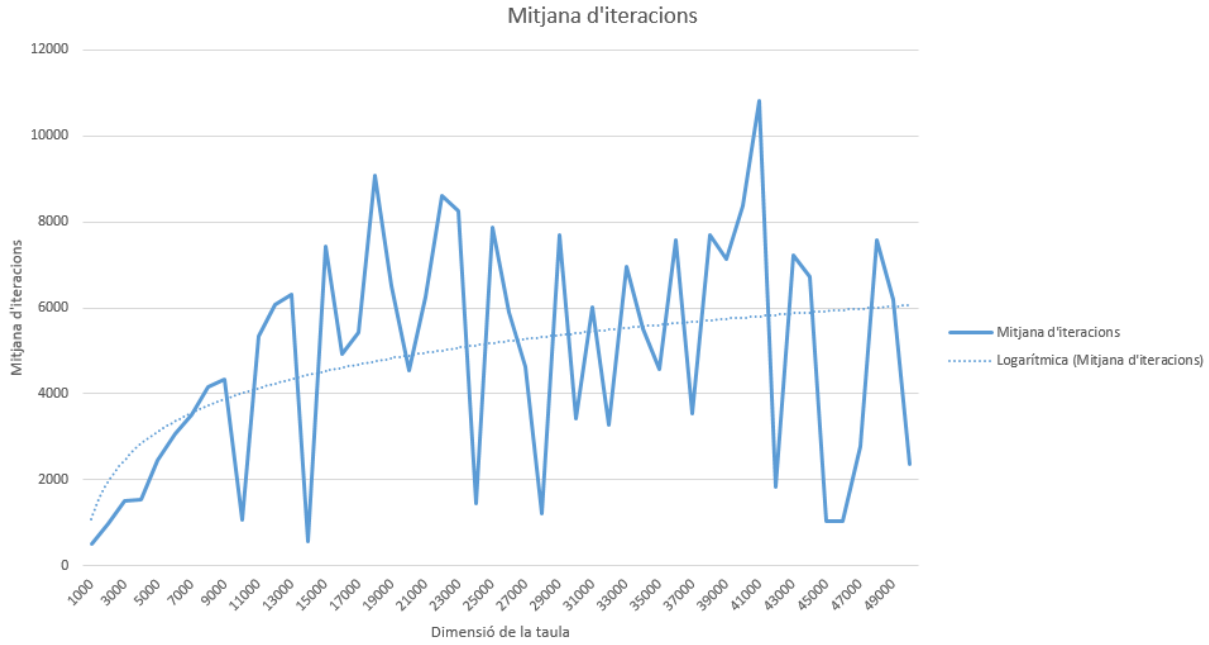
Destacar també l'ús de funcions com newLayer, deleteLayer, deleteLayers, deleteCol, etc que tenen com a finalitat única fer el codi molt mes llegible i reutilitzable.

En aquest cas el programa principal ,al igual que en la llista doblement enllaçada, ens pregunta si volem fer un test unitari de cada funció o executar la prova de buscar sobre una llista de 1000 elements.

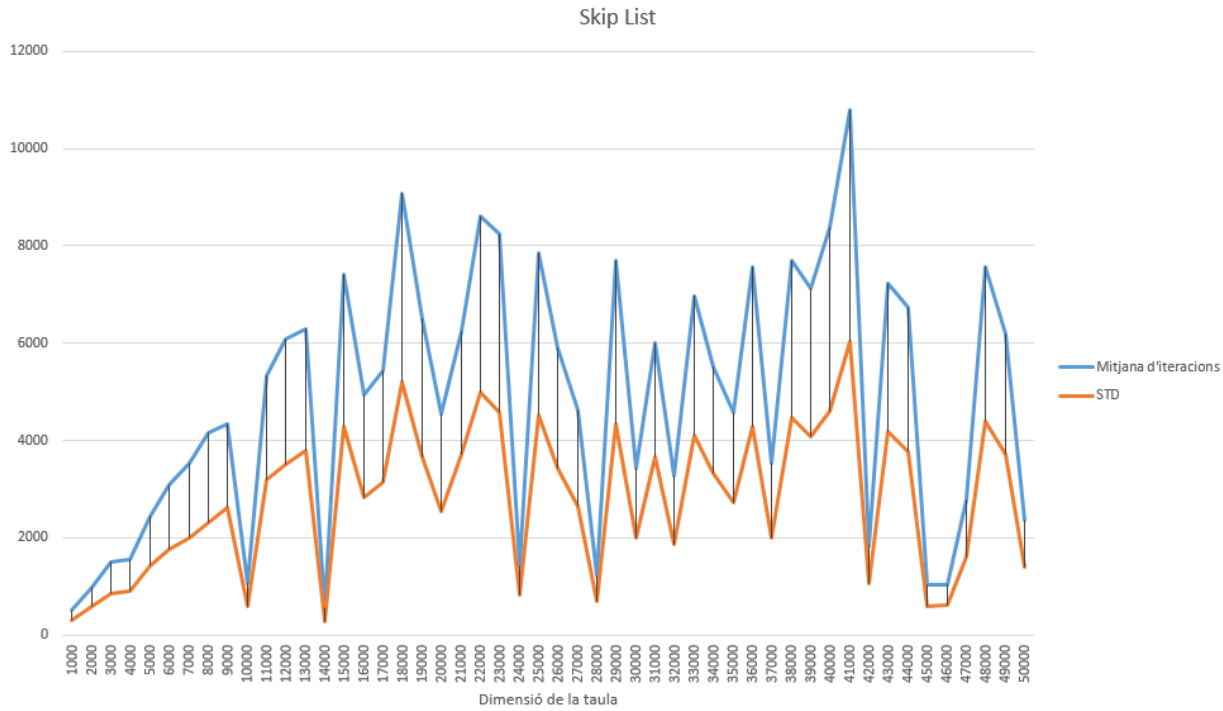
2.4. Funcions extra

Imprimir_Llista(skip_list): Imprimeix tots els elements de l'estructura "skip_list"

2.5. Gràfiques de cost i anàlisi

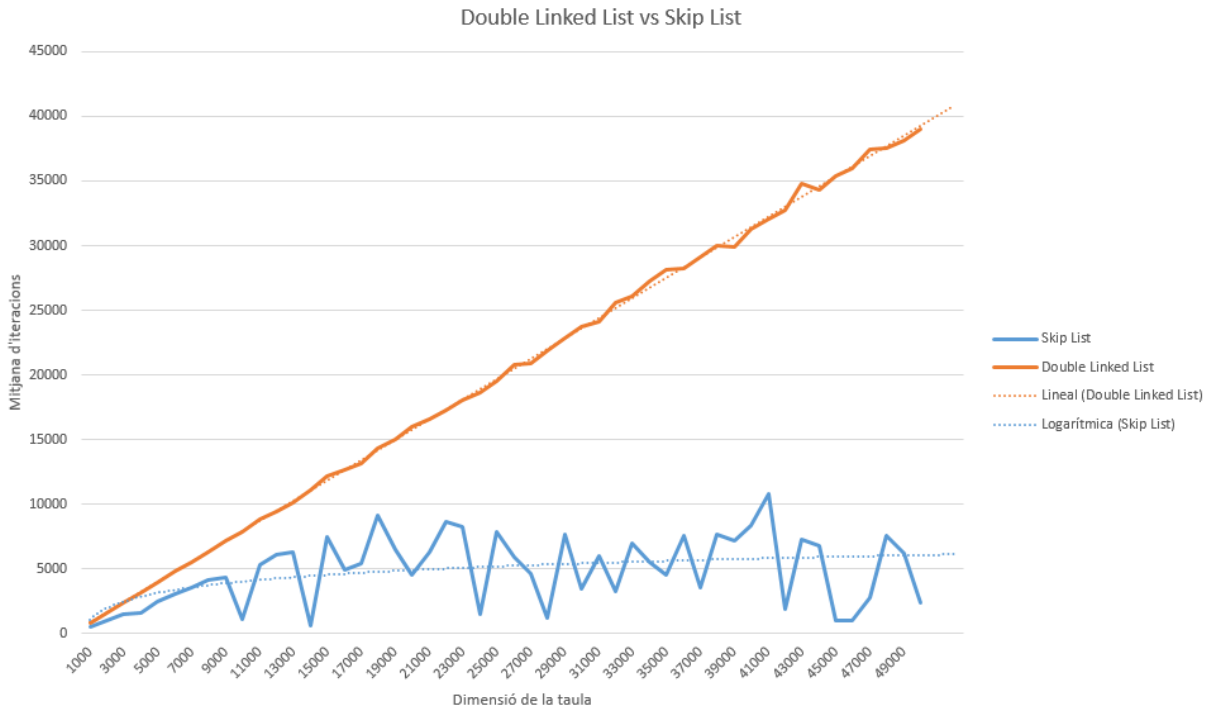


A la gràfica superior veiem el nombre d'iteracions realitzades per buscar un element, podem observar com son bastant irregulars, ja que depenem totalment del random, però que tendeixen a una línia logarítmica, com s'observa a la línia discontinua.



En aquesta gràfica observem com la desviació estàndard de buscar un element es també logarítmica i proporcional a la mitjana d'iteracions.

3. Comparació cost llista doblement encadenada amb skip list



En aquesta gràfica podem observar la gran diferència entre buscar un element a una skip list i a una linked list, tot i utilitzar un random no molt estable, com s'observa pels pics bruscs de la gràfica de la skip list, al utilitzar diferents nivells es molt mes ràpid arribar al element desitjat.

Destacar també que encara que no s'observa en aquesta gràfica, el cost d'inserció però, es molt superior a la skip list ja que es realitza una cerca del seu lloc O(log n) en primer lloc, mentre que a la linked list el cost d'inserció es constant ja que només hem de moure 4 punters.