Algoritmos 2024-25

Grado en Ingeniería Informática

Práctica 3

Diccionario de datos

Fecha límite de entrega: sábado, 9 de noviembre

En esta práctica se pide construir un programa de consulta de sinónimos de una palabra dada, usando los datos en el archivo `sinonimos.txt` con 19062 entradas con el objetivo de comparar diferentes implementaciones de Diccionarios de Datos.

Las búsquedas se realizarán a partir de la palabra de cuyo significado se buscan sinónimos, y usaremos las dos siguientes funciones de dispersión:

```
#include <string.h>
#define MIN(X,Y) ((X) < (Y) ? (X) : (Y))

unsigned int dispersionA(char *clave, int tamTabla) {
  int i, n = MIN(8, strlen(clave));
  unsigned int valor = clave[0];
  for (i = 1; i < n; i++)
    valor += clave[i];
  return valor % tamTabla;
}
unsigned int dispersionB(char *clave, int tamTabla) {
  int i, n = MIN(8, strlen(clave));
  unsigned int valor = clave[0];
  for (i = 1; i < n; i++)
    valor = (valor<<5) + clave[i]; /* el desplazamiento de 5 bits equivale a */
  return valor % tamTabla;         /* multipicar por 32 */
}
```

Como diccionario de datos se usarán *tablas de dispersión cerrada* con *exploración lineal*, *exploración cuadrática*, y *exploración doble* con $10007 - x$ mód 10007 como función de dispersión secundaria (siendo x el resultado de la función de dispersión principal). En todos los casos el factor de carga λ será aproximadamente $0,5$ ($N = 38197$).

Se pide:

1. Implemente los diccionarios de datos con las operaciones `inicializar`, `insertar` (que devuelve el número de colisiones que se hayan producido al poner el elemento en la tabla), `buscar` y `mostrar` (que muestra el contenido de toda la tabla).

   ```
   typedef struct entrada_ {
     int ocupada;
     char clave [LONGITUD_CLAVE];
     char sinonimos [LONGITUD_SINONIMOS];
   } entrada;
   typedef int pos;
   typedef entrada *tabla_cerrada;
   tabla_cerrada d = malloc (38197 * sizeof(entrada));
   ```

```
void inicializar_cerrada(tabla_cerrada *diccionario, int tam);

pos buscar_cerrada(char *clave, tabla_cerrada diccionario, int tam,
                   int *colisiones, unsigned int (*dispersion)(char *, int),
                   unsigned int (*resol_colisiones)(int pos_ini, int num_intento));
int insertar_cerrada(char *clave, char *sinonimos,
                     tabla_cerrada *diccionario, int tam,
                     unsigned int (*dispersion)(char *, int),
                     unsigned int (*resol_colisiones)(int pos_ini, int num_intento));
void mostrar_cerrada(tabla_cerrada diccionario, int tam);
```

Y valide que funciona correctamente con el ejemplo visto en clase de teoría usando la siguiente función de dispersión (véase la fig. 1) .

```
#include <string.h>
int ndispersion(char *clave, int tamTabla) {
  if (strcmp(clave, "ANA")==0) return 7;
  if (strcmp(clave, "JOSE")==0) return 7;
  if (strcmp(clave, "OLGA")==0) return 7;
  return 6;
}
```

2. Para cada uno de los diccionarios de datos y con cada una de las funciones de dispersión, **indique el número total de colisiones producidas al insertar *todos* los datos en `sinonimos.txt`** (véase la figura 2).

3. Calcule empíricamente la complejidad computación de la búsqueda de n elementos en cada uno de los tres diccionarios y con cada una de las dos funciones de dispersión. Consejo: no busque las claves en el orden en que se insertan, búsquelas al azar.

   ¿Se obtiene $O(n)$ en todos los casos? ¿Por qué?

```
***Dispersion cerrada cuadratica con dispersion B
Insertando 19062 elementos... Numero total de colisiones: 13366
Buscando n elementos...
           n           t(n)     t(n)/n^0.8         t(n)/n  t(n)/n*log(n)
(*)      125          9.705       0.203924       0.077640       0.016080
(*)      250         19.259       0.232424       0.077036       0.013952
(*)      500         38.378       0.266015       0.076756       0.012351
(*)     1000         76.756       0.305571       0.076756       0.011112
(*)     2000        153.777       0.351615       0.076888       0.010116
(*)     4000        306.615       0.402666       0.076654       0.009242
        8000        657.000       0.495557       0.082125       0.009138
       16000       1239.000       0.536754       0.077438       0.007999
```

4. Compare entre sí los resultados de los diccionarios de datos, y de las funciones de dispersión en cuanto a número de colisiones y tiempos de búsqueda.

5. Entregue los ficheros con el código C y el fichero `.txt` con el informe por medio de la tarea *Entrega Práctica 3* en la página de Algoritmos en el campus virtual. Se recuerda que el límite para completar la tarea es el sábado 9 de noviembre a las 23:59, y una vez subidos los archivos no se podrán cambiar. Todos los compañeros que forman un equipo tienen que entregar el trabajo.

```
***TABLA CERRADA LINEAL
{
 0- (IVAN )
 1-
 2-
 3-
 4-
 5-
 6- (LUIS )
 7- (ANA )
 8- (JOSE )
 9- (OLGA )
10- (ROSA )
}
Numero total de colisiones al insertar los elementos: 12

Al buscar: ANA, encuentro: ANA, colisiones: 0
Al buscar: LUIS, encuentro: LUIS, colisiones: 0
Al buscar: JOSE, encuentro: JOSE, colisiones: 1
Al buscar: OLGA, encuentro: OLGA, colisiones: 2
Al buscar: ROSA, encuentro: ROSA, colisiones: 4
Al buscar: IVAN, encuentro: IVAN, colisiones: 5
No encuentro: CARLOS, colisiones: 6


***TABLA CERRADA CUADRATICA
{0- (OLGA )
 1-
 2-
 3-
 4- (IVAN )
 5-
 6- (LUIS )
 7- (ANA )
 8- (JOSE )
 9-
10- (ROSA )}
Numero total de colisiones al insertar los elementos: 8

Al buscar: ANA, encuentro: ANA, colisiones: 0
Al buscar: LUIS, encuentro: LUIS, colisiones: 0
Al buscar: JOSE, encuentro: JOSE, colisiones: 1
Al buscar: OLGA, encuentro: OLGA, colisiones: 2
Al buscar: ROSA, encuentro: ROSA, colisiones: 2
Al buscar: IVAN, encuentro: IVAN, colisiones: 3
No encuentro: CARLOS, colisiones: 5
```

Figura 1: Test con parte de los ejemplos vistos en la clase de teoría sobre tablas de dispersión

```
#include <stdio.h>
#include <stdlib.h>
#define MIN(X,Y) ((X) < (Y) ? (X) : (Y))
#define LONGITUD_CLAVE 30
#define LONGITUD_SINONIMOS 300

typedef struct {
  char clave [LONGITUD_CLAVE];
  char sinonimos [LONGITUD_SINONIMOS];
} item;

int leer_sinonimos(item datos[]) {
  char c;
  int i, j;
  FILE *archivo;
  if ((archivo = fopen("sinonimos.txt", "r")) == NULL) {
    printf("Error al abrir 'sinonimos.txt'\n");
    return(EXIT_FAILURE);
  }
  for (i = 0; fscanf(archivo, "%s", datos[i].clave) != EOF; i++) {
    if ((c = fgetc(archivo)) != '\t') {
      printf("Error al leer el tabulador\n");
      return(EXIT_FAILURE);
    }
    for (j = 0; (c = fgetc(archivo)) != '\n'; j++) {
      if (j < LONGITUD_SINONIMOS - 1)
        datos[i].sinonimos[j] = c;
    }
    datos[i].sinonimos[MIN(j, LONGITUD_SINONIMOS -1)] = '\0';
  }
  if (fclose(archivo) != 0) {
    printf("Error al cerrar el fichero\n");
    return(EXIT_FAILURE);
  }
  return(i);
}
```

Figura 2: Lectura del archivo 'sinonimos.txt'