Universidad Nacional de Cuyo
Facultad de Ingeniería
Licenciatura en Ciencias de la Computación

Trabajo Práctico Nº 4

Algoritmos y Estructuras de Datos

Hash Tables

2024

Gonzalo Padilla Lumelli

Mayo 2024

Parte 1

Ejercicio 1

Ejemplificar que pasa cuando insertamos las llaves 5, 28, 19, 15, 20, 33, 12, 17, 10 en un HashTable con la colisión resulta por el método de chaining. Permita que la tabla tenga 9 slots y la función de hash:

$$H(k) = k \mod 9 \tag{1}$$

Solución

Ejercicio 2

A partir de una definición de diccionario como la siguiente:

dictionary = Array(m,0) # una sugerencia de implementación, se puede usar una lista de python.

Crear un módulo de nombre dictionary.py que implemente las siguientes especificaciones de las operaciones elementales para el TAD diccionario.

Nota: dictionary puede ser redefinido para lidiar con las colisiones por encadenamiento.

insert(D, key, value)

Descripción: Inserta un key en una posición determinada por la función de hash (1) en el diccionario (dictionary). Resolver colisiones por encadenamiento. En caso de keys duplicados se anexan a la lista.
Entrada: el diccionario sobre el cual se quiere realizar la inserción y el valor del key a insertar.
Salida: Devuelve D.

search(D, key)

Descripción: Busca un key en el diccionario.
Entrada: El diccionario sobre el cual se quiere realizar la búsqueda (dictionary) y el valor del key a buscar.
Salida: Devuelve el value de la key. Devuelve None si el key no se encuentra.

delete(D, key)

Descripción: Elimina un key en la posición determinada por la función de hash (1) del diccionario (dictionary).
Poscondición: Se debe marcar como None el key a eliminar.
Entrada: El diccionario sobre el se quiere realizar la eliminación y el valor del key que se va a eliminar.
Salida: Devuelve D.

Solución

```
def insert(d, key, value):
    tableIndex = hash(key, len(d))
    lista = d[tableIndex]

    if not lista:
        d[tableIndex] = [(key, value)]
        return d

    for i, (k, v) in enumerate(lista):
```

```
        if k == key:
            lista[i] = (key, value)
            return d

    lista.append((key, value))
    return d
```

```
def search(d, key):
    tableIndex = hash(key, len(d))
    lista = d[tableIndex]

    if not lista:
        return None

    for i, (k, v) in enumerate(lista):
        if k == key:
            return v

    return None
```

```
def delete(d, key):
    tableIndex = hash(key, len(d))
    lista = d[tableIndex]

    if not lista:
        return d

    for i, (k, v) in enumerate(lista):
        if k == key:
            lista.pop(i)
            if len(lista) == 0:
                d[tableIndex] = None
            break

    return d
```

Parte2

Ejercicio 3

Considerar una tabla hash de tamaño $m = 1000$ y una función de hash correspondiente al método de la multiplicación donde $A = (\sqrt{5} - 1)/2)$. Calcular las ubicaciones para las claves 61, 62, 63, 64 y 65.

Solución

Ejercicio 4

Implemente un algoritmo lo más eficiente posible que devuelva True o False a la siguiente proposición: dado dos strings $s_1 \dots s_k$ y $p_1 \dots p_k$, se quiere encontrar si los caracteres de $p_1 \dots p_k$ corresponden a una permutación de $s_1 \dots s_k$. Justificar el costo en tiempo de la solución propuesta.

Solución

```python
def isPermutation(stringS, stringP):
    if not stringS or not stringP:
        return False

    # Si son de distinto largo, no es permutación
    if len(stringS) != len(stringP):
        return False

    stringS = stringS.lower()
    stringP = stringP.lower()

    # Contamos los caracteres de cada palabra
    charsCountsS = dict()
    charsCountsP = dict()
    for i in range(len(stringS)):
        if stringS[i] in charsCountsS:
            charsCountsS[stringS[i]] += 1
        else:
            charsCountsS[stringS[i]] = 1
        if stringP[i] in charsCountsP:
            charsCountsP[stringP[i]] += 1
        else:
            charsCountsP[stringP[i]] = 1

    # Verificamos que coincidan
    for c in stringS:
        if c not in charsCountsP or charsCountsP[c] != charsCountsS[c]:
            return False

    return True
```

Ejercicio 5

Implemente un algoritmo que devuelva True si la lista que recibe de entrada tiene todos sus elementos únicos, y Falso en caso contrario. Justificar el costo en tiempo de la solución propuesta.

Solución

Ejercicio 6

Los nuevos códigos postales argentinos tienen la forma *cddddccc*, donde *c* indica un carácter (A - Z) y *d* indica un dígito $0, ..., 9$. Por ejemplo, C1024CWN es el código postal que representa a la calle XXXX a la altura 1024 en la Ciudad de Mendoza. Encontrar e implementar una función de hash apropiada para los códigos postales argentinos.

Solución

Ejercicio 7

Implemente un algoritmo para realizar la compresión básica de cadenas utilizando el recuento de caracteres repetidos. Por ejemplo, la cadena 'aabcccccaaa' se convertiría en 'a2blc5a3'. Si la cadena "comprimida" no se vuelve más pequeña que la cadena original, su método debería devolver la cadena original. Puedes asumir que la cadena sólo tiene letras mayúsculas y minúsculas (a - z, A - Z). Justificar el costo en tiempo de la solución propuesta.

Solución

Ejercicio 8

Se requiere encontrar la primera ocurrencia de un string $p_1 \dots p_k$ en uno más largo $a_1 \dots a_L$. Implementar esta estrategia de la forma más eficiente posible con un costo computacional menor a $O(K * L)$ (solución por fuerza bruta). Justificar el coste en tiempo de la solución propuesta.

Solución

Ejercicio 9

Considerar los conjuntos de enteros $S = s_1, \dots, s_n$ y $T = t_1, \dots, t_m$. Implemente un algoritmo que utilice una tabla de hash para determinar si $S \subseteq T$ (S subconjunto de T). ¿Cuál es la complejidad temporal del caso promedio del algoritmo propuesto?

Solución

Parte3

Ejercicio 10

Considerar la inserción de las siguientes llaves: 10; 22; 31; 4; 15; 28; 17; 88; 59 en una tabla hash de longitud $m = 11$ utilizando direccionamiento abierto con una función de hash $h'(k) = k$. Mostrar el resultado de insertar estas llaves utilizando:

1. Linear probing.
2. Quadratic probing con $c_1 = 1$ y $c_2 = 3$.
3. Double hashing con $h_1(k) = k$ y $h_2(k) = 1 + (k \mod (m - 1))$.

Solución

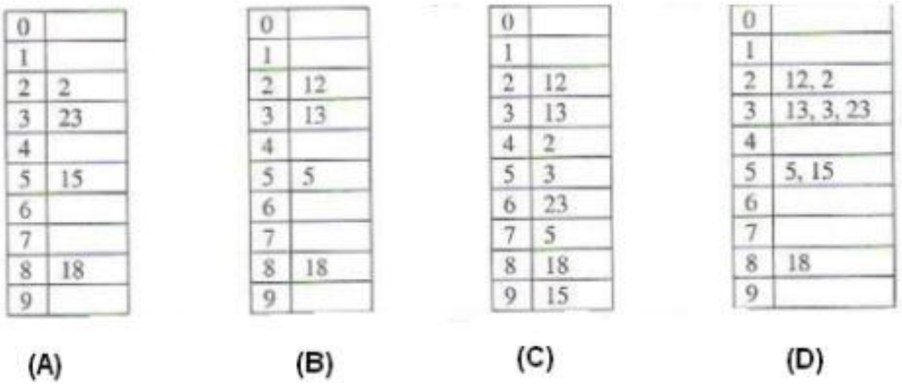
Ejercicio 11 (opcional)

Implementar las operaciones de insert() y delete() dentro de una tabla hash vinculando todos los nodos libres en una lista. Se asume que un slot de la tabla puede almacenar un indicador (flag), un valor, junto a una o dos referencias (punteros). Todas las operaciones de diccionario y manejo de la lista enlazada deben ejecutarse en O(1). La lista debe estar doblemente enlazada o con una simplemente enlazada alcanza?

Solución

Ejercicio 12

Las llaves 12, 18, 13, 2, 3, 23, 5 y 15 se insertan en una tabla hash inicialmente vacía de longitud 10 utilizando direccionamiento abierto con función hash $h(k) = k \mod 10$ y exploración lineal (linear probing). ¿Cuál es la tabla hash resultante? Justifique.



0	
1	
2	2
3	23
4	
5	15
6	
7	
8	18
9	

(A)

0	
1	
2	12
3	13
4	
5	5
6	
7	
8	18
9	

(B)

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

(C)

0	
1	
2	12, 2
3	13, 3, 23
4	
5	5, 15
6	
7	
8	18
9	

(D)

Solución

Ejercicio 13

Una tabla hash de longitud 10 utiliza direccionamiento abierto con función hash $h(k) = k \mod 10$, y exploración lineal (linear probing). Después de insertar 6 valores en una tabla hash vacía, la tabla es como se muestra a continuación.

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

¿Cuál de las siguientes opciones da un posible orden en el que las llaves podrían haber sido insertadas en la tabla? Justifique

(A) 46, 42, 34, 52, 23, 33

(B) 34, 42, 23, 52, 33, 46

(C) 46, 34, 42, 23, 52, 33

(D) 42, 46, 33, 23, 34, 52

Solución