Diccionarios.

ED Grupo A. Profesor: Isabel Pita.
Tipo B

Nombre del alumno:

1. Si la ocupación de una tabla hash sobrepasa su factor de carga debemos redimensionarla. Explica como se redimensiona una tabla hash cerrada e indica el coste de hacerlo. Justifica tu respuesta.

   Respuesta:

   Para redimensionar una tabla hash abierta, se reserva memoria para un vector de tamaño el primer primo mayor que el doble del tamaño de la tabla existente. A continuación se recorre la tabla original, calculando el nuevo hash de cada clave y situando los elementos en su nueva posición en la nueva tabla. Si existe colisión al añadir un elemento a su nueva posición se aplicará el método de resolución de colisiones empleado en la tabla hash. Se puede evitar la copia de los elementos utilizando la semántica de movimiento de C++.

   El coste de redimensionar la tabla es $\mathcal{O}(max(k, N))$, siendo k el coste de reservar la memoria de la nueva tabla y N el número de elementos de la tabla original. El coste depende de dos cantidades, por una parte se debe reservar la memoria del nuevo vector, lo que depende de la cantidad de memoria que se reserva. Por otra parte se deben desplazar a su nueva posición todas las claves de la tabla original, por lo que depende del número de elementos de la tabla. Si no se copian los valores el coste de desplazar cada elemento es constante.

   Como el número de elementos de la nueva tabla es mayor que el número de elementos de la tabla original, el máximo será el número de elementos de la nueva tabla.

2. Indica las ventajas e inconvenientes de utilizar un diccionario ordenado frente a uno no ordenado.

   Respuesta:

   Un diccionario ordenado nos permite recorrer las claves guardadas en orden, mientras que al recorrer las claves de un diccionario no ordenado no podemos suponer que se realiza en ningún orden.

   Las operaciones de añadir un elemento, buscar un elemento y borrar un elemento sobre un diccionario ordenado tienen coste logarítmico en el número de elementos del diccionario, mientras que sobre un diccionario no ordenado tienen coste constante.

   La operación de crear un diccionario ordenado tiene coste constante, mientras que esta operación tiene coste lineal en el tamaño de la tabla si se utiliza un diccionario no ordenado.

   Los diccionarios ordenados aprovechan mejor el espacio, ya que toda la memoria reservada está ocupada, mientras que los diccionarios no ordenados no ocupan toda la memoria reservada, sino solo hasta alcanzar el factor de carga definido.

   Los diccionarios no ordenados se redimensionan cuando se alcanza el factor de carga, mientras que los diccionarios ordenados no se redimensionan.

3. Escribe dos formas diferentes de modificar el valor asociado a una clave en un diccionario y explica sus diferencias. Indica si existe alguna diferencia en las instrucciones cuando se añaden los elementos en un diccionario ordenado respecto a cuando se añaden en uno no ordenado. Explica el coste de cada una de ellas sobre los dos tipos de diccionarios.

   Respuesta:

   Las instrucciones son las mismas si se está utilizando un diccionario ordenado (`map`) o uno sin orden (`unordered_map`).

   a)
   ```
   tabla.at(clave) = valor;
   ```

Si se utiliza un `map`, el coste del método `at`, es logarítmico respecto al número de elementos de la tabla, ya que estos diccionarios están implementados sobre un árbol binario de búsqueda equilibrado.

Si se utiliza un `unordered_map` el coste del método `at` es constante ya que estos diccionarios están implementados sobre una tabla hash, que suponemos bien diseñada.

b) `tabla[clave] = valor;`

Si se utiliza un `map`, el coste del método `[]`, es logarítmico respecto al número de elementos de la tabla, ya que estos diccionarios están implementados sobre un árbol binario de búsqueda equilibrado.

Si se utiliza un `unordered_map` el coste del método `[]` es constante ya que estos diccionarios están implementados sobre una tabla hash, que suponemos bien diseñada.

Diferencias entre las dos formas. Si la calve no existe en el diccionario, el método `at` lanza una excepción, mientras que el operador `[]` la añade.

Otra forma de modificar un valor es obtener un iterador con el método `find` y si existe la clave modificarla con el iterador.

```
auto it = tabla.find(clave);
if (it != tabla.end()) it->second = valor;
```

Si se utiliza un `map`, el coste del método `find`, es logarítmico respecto al número de elementos de la tabla, ya que estos diccionarios están implementados sobre un árbol binario de búsqueda equilibrado.

Si se utiliza un `unordered_map` el coste del método `find` es constante ya que estos diccionarios están implementados sobre una tabla hash, que suponemos bien diseñada.

El coste de modificar el valor utilizando el iterador es constante.

En este caso si el valor no existe no se realiza ninguna acción.

4. Indica cuál es el coste de crear un diccionario ordenado y el coste de crear un diccionario no ordenado. Justifica tu respuesta.

   Respuesta:

   El coste de crear un diccionario ordenado es $\mathcal{O}(1)$, esto es debe a que un diccionario ordenado está implementado con un árbol binario de búsqueda y crear un árbol binario de búsqueda vacío tiene coste constante.

   El coste de crear un diccionario sin orden de tamaño `TAM` es lineal en el tamaño `TAM`. Esto se debe a que un diccionario sin orden está implementado por medio de una tabla hash y estas a su vez se implementan por medio de un vector. Reservar memoria para los `TAM` elementos de un vector tiene coste lineal en `TAM`.

5. Indica si los siguientes pares `<clave, valor>` se pueden guardar en una tabla hash cerrada de tamaño 11 con factor de carga 0,7, ajustado al valor menor (truncamiento). En caso de que haya que dimensionar la tabla indica con qué valor se realizaría el redimensionado, suponiendo que los valores se van a añadir en el orden dado.

   `<4,"xx">, <6,"yy">, <4, "zz">, <5, "bb">, <5, "cc">, <7, "vv">, <8, "jj">, <2,"ww">, <6, "ss">, <1, "kk">, <2, "ff">, <3, "ff">, <9, "ff">.`

   Respuesta:

   En una tabla hash con tamaño 11 y factor de carga 0.7 se pueden guardar 7 elementos.

$$0{,}7 = \frac{NumElem}{11} \Rightarrow NumElem = 11 * 0{,}7 = 7{,}7.$$

En la entrada los valores repetidos solo se guardan una vez, por lo tanto se van guardando: <4,"xx">, <6,"yy"> <5, "bb"> <7, "vv"> <8, "jj">, <2,"ww">, <1, "kk">

El primer valor con el que se debe redimensionar la tabla es <3, "ff">.

6. Indica como se distribuyen los siguientes pares <clave, valor> sobre una tabla hash abierta de tamaño 7, si la función hash considerada asocia a cada carácter de la cadena su posición en el abecedario y su factor de carga es 0.8. Los datos se añaden en el orden dado.

   <"Dani", 3>, <"Viki", 5>, <"Job", 1>, <"Ben", 5>, <"Cas", 5>

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

   Respuesta:

   h("Dani") = 4 + 1+ 14 + 9 = 28 % 7 = 0.
   h("Viki") = 22 + 9 + 9 + 11 = 51 % 7 = 2.
   h("Job") = 10 + 15 + 2 = 227% 7 = 6.
   h("Ben") = 2 + 5 + 14 = 21 % 7 = 0.
   h("Cas") = 3 + 1 + 19 = 23 % 7 = 2.

   En la lista de la posición 0 se sitúan <"Ben",5> y \verb¡"Dani",3¿+ en este orden, ya que en la implementación vista en clase los elementos se añaden por la parte de delante de las listas.

   En la lista de la posición 2 se sitúan <"Cas",5> y \verb¡"Viki",5¿+.

   En la lista de la posición 6 se sitúa <"Job", 1>.

7. Explica si la siguiente afirmación es cierta o falsa. *En una tabla hash cerrada, el número de elementos que se pueden almacenar depende del método de resolución de colisiones empleado. En general se pueden almacenar más valores si se utilizan varias funciones hash para resolver las colisiones que si se utiliza una única función hash con búsqueda lineal de la siguiente posición libre.*

   Respuesta:

   La afirmación es falsa. El número de elementos que se almacenan en una tabla hash depende del tamaño de la tabla y del factor de carga admitida. El método de resolución de colisiones afecta a la forma en que se guardan los elementos en la tabla, y por lo tanto a las colisiones que se originarán. Cuando se produce una colisiónel nuevo elemento se guarda en una posición que depende del método de resolución de colisiones empleado, pero al final ocupa una posición, tanto si se ha producido una colisión como si no.

8. (1 punto) Dado el siguiente fragmento de código indica qué se muestra por pantalla al ejecutarlo. Indica el coste de cada una de las instrucciones del código.

```
1  std::unordered_map<std::string, int> tabla;
2  tabla.insert({"verde", 4});
3  tabla.insert({"rojo", 6});
4  tabla.insert({"amarillo", 2});
5  std::pair<std::unordered_map<std::string, int>::iterator,bool> par1;
6  par1 = tabla.insert({"amarillo", 6});
7  if (par1.second) std::cout << "primero\n";
8  else std::cout << "segundo\n";
9  std::pair<std::unordered_map<std::string, int>::iterator,bool> par2;
10 par2 = tabla.insert({"naranja", 1});
11 if (par2.second) std::cout << "primero\n";
12 else std::cout << "segundo\n";
13 std::pair<std::unordered_map<std::string, int>::iterator,bool> par3;
14 par3 = tabla.insert({"naranja", 8});
15 std::cout << par3.first->first << ' ' << par3.first->second << '\n';
```

Respuesta:

segundo
primero
naranja 1

Explicación:

La línea 6 añade la clave `amarillo` al diccionario. Cómo la clave ya se encuentra en el diccionario la operación devuelve un iterador a la clave existente y false. No se modifica el dicconario.

En la instrucción condicional de las líneas 7-8 como la condición es falsa se escribe `segundo`.

En las líneas 9-10 se declara un par y se le da el valor resultante de añadir a la tabla la clave `naranja`. Cómo la clave no se encuentra en el diccionario se añade y se devuelve un iterador a la clave que se ha añadido y el valor cierto.

La instrucción condicional de las líneas 11-12 escribe `primero` ya que la condición es cierta.

Las instrucciones de las líneas 14-15 declaran un par y le dan el valor resultante de añadir a la tabla el par `<naranja, 8>. Cómo la clave \verb`naranja+ ya se encuentra en el diccionario se devuelve un iterador a la clave ya existente en el diccionario y el valor falso. El valor de la tabla no se modifica.

En la línea 15 se escribe la clave `naranja` y el valor que tiene en el diccionario, es decir 1.

La primera instrucción en que se declara la tabla tiene coste lineal en el tamaño inicial de la tabla, ya que se crea una tabla hash. El resto de instrucciones tienen coste constante ya que el método `insert` sobre el tipo `unordered_map` tiene coste constante.

9. (1 punto) Dado el siguiente fragmento de código indica qué se muestra por pantalla al ejecutarlo. Indica el coste de cada una de las instrucciones del código.

```
1     std::map<std::string, int> tabla;
2     tabla["amarillo"] = 1;
3     tabla["verde"] = 4;
4     tabla["rojo"] = 2;
5     tabla["verde"] = 7;
6     std::cout << tabla["verde"] << '\n';
7     std::map<std::string, int>::iterator it = tabla.find("verde");
8     if (it != tabla.end()) {
9         std::cout << it->first << ' ' << it->second << '\n';
10        it->second = 5;
11        std::cout << it->first << ' ' << it->second << '\n';
12    }
13    std::cout << tabla["verde"] << '\n';
```

Respuesta:

7
verde 7
verde 5
5

Explicación:

Las instrucciones de las líneas 2-4 añaden las claves `amarillo`, `verde` y `rojo` al diccionario.

La instrucción de la línea 5 modifica el valor de la clave `verde`.

En la linea 6 se escribe el valor de la clave `verde` que se acaba de modificar.

En la línea 7 se obtiene un iterador al par con clave `verde` mediante la operación `find`.

En el condicional de la línea 8 la condición es cierta, ya que la clave `verde` se encuentra en la tabla.

La instrucción de la línea 9 escribe la clave `verde` y su valor a través del iterador anterior.

La instrucción de la línea 10 cambia el valor de la clave `verde` a través del iterador.

La instrucción de la línea 11 escribe la clave `verde` y su nuevo valor.

Por último, en la línea 13 se escribe el valor asociado a la clave `verde` que se modificó.

El coste de los métodos `[]` y `find` sobre un diccionario ordenado (`map`) es lógarítmico en el número de elementos del diccionario. El resto de operaciones e instrucciones son de coste constante.

10. (1 punto) Dado el siguiente fragmento de código indica qué se muestra por pantalla al ejecutarlo. Indica el coste de cada una de las instrucciones del código.

```
1     std::map<std::string, int> tabla;
2     tabla["rojo"] = 6;
3     tabla["verde"] = 3;
4     tabla["azul"] = 8;
5     std::map<std::string, int>::iterator it = tabla.find("azul");
6     std::cout << it->first << ' ' << it->second << '\n';
7     tabla.erase(it);
8     for (auto p : tabla)
9         std::cout << p.first << ' ' << p.second << '\n';
```

Respuesta:

azul 8
rojo 6
verde 3

Explicación:

Las instrucciones de las líneas 2-4 añaden las claves `rojo`, `verde` y `azul` al diccionario.

En la línea 5 se obtiene un iterador a la clave `azul`. y en la línea 6 se escriben los valores de la clave `azul` y su valor.

En la línea 7 se elimina del diccionario el valor asociado al iterador, es decir, la clave `azul`. Hay que tener en cuenta que al eliminar con un iterador el coste de la operación es constante.

Las líneas 8-9 recorren el diccionario mostrando los valores. Como el diccionario es ordenado las claves se recorren en orden alfabético.

El coste de los métodos `[]`y `find` sobre un diccionario ordenado (`map`) es lógarítmico en el número de elementos del diccionario. El resto de operaciones e instrucciones son de coste constante. Observad que el método `erase` tiene como argumento un iterador, por ello su coste es constante.