Part I.

Estructuras de datos

Última modificación: March 28, 2016 16:38.

1. Primer parcial

Pregunta 1: Operaciones y su complejidad. Level M

Considere cada una de las siguientes operaciones y especifique su complejidad. Su respuesta debe estar claramente justificada. Sin embargo, no se requiere la implementación de la operación.

1. Determinar cuántos elementos duplicados existen en una lista doblemente enlazada.
2. Insertar un elemento en una lista ordenada. Es decir, usted puede asumir que antes de producirse la inserción, los elementos dentro de la lista se encuentran en orden. Después de la inserción, la lista debe estar ordenada.
3. Insertar un elemento en un vector ordenado. Asuma que antes de la inserción los elementos del vector están en orden. Después de la inserción el vector debe quedar ordenado.

Solución, pg. 8

Pregunta 2: Conceptos de estructuras de datos y sus operaciones. Level M

Responda *falso* o *verdadero* a las siguientes preguntas justificando de forma clara y concisa su respuesta.

1. Insertar un elemento al inicio de una secuencia almacenada en una lista es más fácil (computacionalmente) que si se encuentra almacenada en un vector.
2. Una secuencia va a contener entre 1000 y 1200 elementos. El uso de una lista simplemente enlazada para representar esta secuencia utiliza más memoria que un vector con capacidad inicial 150 y política de crecimiento de 3.

Solución, pg. 8

Pregunta 3: Máximo elemento en una pila. Level M

Escriba la operación `max` de la clase `Stack` que debe retornar el máximo elemento insertado hasta el momento en la pila.

```
T& max() { /* Su código aquí */ }
```

Considere el siguiente fragmento como *un posible* uso de la operación:

```
Stack<int> s;
for (int i = 0; i < 6; i++) { s.push(rand()%10); }
s.max() = 15;
```

En el siguiente ejemplo usted puede apreciar el efecto de la operación. La pila de la izquierda es como queda después de ser llenada. La de la derecha es el resultado final.

1
5
9
0
1

1
5
15
0
1

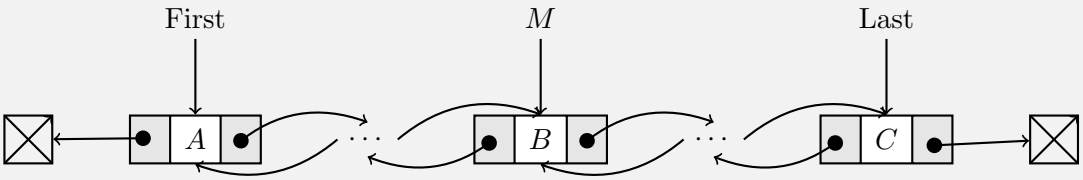
Recuerde que una pila puede ser representada de diferentes formas (por ejemplo, utilizando listas, colas, ...). Usted debe especificar claramente que representación utiliza antes de implementar la operación. Es necesario especificar la complejidad de su implementación.

Solución, pg. 9

Pregunta 4: Operaciones internas en listas. Level MH

La siguiente es una representación de una lista doblemente enlazada. Cada nodo de la lista tiene un dato, un apuntador al nodo anterior y un apuntador al nodo siguiente. En el diagrama $\cdots$ representa la existencia de un número arbitrario de nodos adicionales. M representa un nodo que está dentro de la lista pero que no es

ni el primero ni el último. ☒ debe ser interpretado como `nullptr`.



Teniendo en cuenta la descripción anterior escriba las líneas de código necesarias para realizar las siguientes operaciones. Puede suponer que los apuntadores ya se encuentran posicionados. Para ninguna de las operaciones usted deberá intercambiar los datos de los nodos. Recuerde especificar la complejidad temporal de cada respuesta.

1. Eliminar todos los nodos a excepción de los que contienen los datos A y C.
2. Eliminar k nodos después del nodo apuntado por M. Si después de M existen menos de k elementos entonces el nodo apuntado por M se convertirá en el último de la lista.

Pregunta 5: Mezcla de listas. Level **MH**

Como parte de la implementación de un algoritmo de ordenamiento se requiere implementar una operación que mezcle dos listas ordenadas. Por ejemplo, Las listas L y M representan las secuencias $\langle 1, 3, 5, 7, 9\rangle$ y $\langle 2, 4, 6, 8, 10\rangle$. El resultado de `L.sortedMerge(M)` será la lista que representa la secuencia $\langle 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\rangle$. Escriba una implementación eficiente de la operación `sortedMerge` y proponga una cota para su complejidad.

Pregunta 6: Tiempo de ejecución. Level **E**

Considere las siguientes complejidades de un conjunto de programas: $O\left(n^2\right)$, $O\left(2^n\right)$, $O\left(1\right)$, $O\left(n \cdot \log_2 n\right)$, $O\left(\log_2 n\right)$ y $O\left(n!\right)$. Ordene de mayor a menor las velocidades de los programas ante el mismo valor de n.

Solución, pg. 9

Pregunta 7: Operaciones y su complejidad. Level E

Proponga una operación de cualquiera de las estructuras de datos o programas vistos hasta el momento para cada una de las siguientes complejidades.

a) $O(n)$: Search()

b) $O(n^2)$:

c) $O(n \cdot \log_2 n)$:

d) $O(\log_2 n)$:

e) $O(2^n)$:

Solución, pg. 10

Pregunta 8: Pertenencia en listas. Level E

Escriba una implementación de la operación `member` para la clase `List` desarrollada en clase. La operación verifica si un dato dado se encuentra en la lista. Especifique su complejidad.

Solución, pg. 10

Pregunta 9: Conceptos de estructuras de datos y sus operaciones. Level M

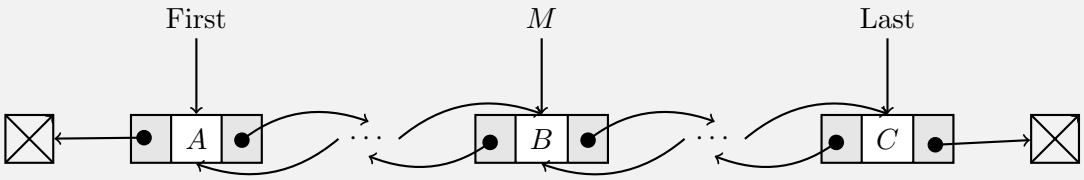
Responda *falso* o *verdadero* a las siguientes preguntas justificando clara y de manera concisa su respuesta.

1. Insertar un elemento al inicio de una secuencia almacenada en una lista es más difícil (computacionalmente) que si se encuentra almacenada en un vector. F
   La complejidad de insertar nuevo elemento en una lista es 0(1), en vector es 0(n) dado que hay que correr elementos a derecha.
2. Saber si un elemento se encuentra dentro de una secuencia ordenada representada por una lista doblemente enlazada tiene una complejidad $O(\log_2 n)$ si se usa el algoritmo de búsqueda binaria. V. Busqueda binaria reduce la complejidad por cada una de los nodos que se van visitando.
3. Una secuencia va a contener entre 1000 y 1200 elementos. El uso de una lista simplemente enlazada para representar esta secuencia utiliza menos memoria que un vector con capacidad inicial 150 elementos y política de crecimiento de 1.5. Depende del tipo de dato T. Si es de tipo Int las listas usan un mayor espacion en memoria teniendo en cuenta que el modelo que representa el desgaste es:

   1200M(int)+1200M(p)+2M(p)+M(int), contra la ecuacion de vector que es :
   1350M(int)+ 2M(int)

Pregunta 10: Operaciones internas en listas. Level MH

La siguiente es una representación de una lista doblemente enlazada. Cada nodo de la lista tiene un dato, un apuntador al nodo anterior y un apuntador al nodo siguiente. En el diagrama "$\cdots$" representa la existencia de un número arbitrario de nodos adicionales. M representa un nodo que está dentro de la lista pero que no es ni el primero ni el último. ☒ debe ser interpretado como `nullptr`.



Teniendo en cuenta la descripción anterior escriba las líneas de código necesarias para realizar las siguientes operaciones. Puede suponer que los apuntadores ya se encuentran posicionados. Para ninguna de las operaciones usted deberá intercambiar los datos de los nodos. Recuerde especificar la complejidad temporal de cada respuesta.

1. Insertar un nodo con dato P inmediatamente después de B. Realizada.
2. Eliminar el nodo que contiene a C. Realizada.
3. Intercambiar el nodo con dato M por el nodo con C. Recuerde que no puede intercambiar datos, debe intercambiar los nodos. No funciona.

Part II.

Solutions

Última modificación: March 28, 2016 16:38.

Pregunta 1 pg. 2:

1. $O\left(n^2\right)$, donde n es el tamaño de la lista. Por cada elemento de la lista, la lista completa debe ser revisada para saber si este está repetido.

2. $O\left(n\right)$, donde n es el tamaño de la lista. Primero se busca la posición que debe tener el elemento a ser insertado. Para esto es necesario iterar hasta encontrar el primer elemento mayor guardando un puntero a su anterior. Luego la inserción del nuevo elemento tiene lugar. En el peor caso, el elemento a insertar debe le corresponde estar después del actual último y entonces es necesario recorrer toda la lista.

3. $O\left(n\right)$ siendo n el tamaño del vector. Primero se busca la posición del primer elemento mayor al que va a ser insertado. Este elemento y todos los que le siguen son movidos una posición a la derecha. Por último se realiza la inserción.

Pregunta 2 pg. 2:

1. Es verdad y se puede observar comparando la complejidad de la operación `push_front` en ambas estructuras de datos. Para listas esta complejidad es $O\left(1\right)$ mientras que para vectores es $O\left(n\right)$ siendo n el tamaño del vector.

2. La incertidumbre en el número exacto de elementos hace imposible utilizar el constructor de la clase `vector` que podría reservar memoria para todos los elementos y tener una complejidad siempre constante en `push_back`.

   a) Inserta los primeros 150 elementos, se hace resize y se cuenta con una capacidad de $150 \times 3 = 450$ de los cuales 300 están libres.

   b) Inserta 300 elementos más y se hace resize. A este punto se cuenta con $450 \times 3 = 1350$ de los cuales 950 están libres.

   c) Inserta 750 elementos más y con esto completaría 1200 que es el máximo de elementos previstos por insertar. No es necesario ralizar otro resize puesto que la capacidad está en 1350. Después de terminadas las inserciones el vector queda con 150 elementos desperdiciados.

   La ecuación que describe el uso de memoria del vector M_v es:

$$M_v = 1350M(T) + 2M(int)$$

   donde $M(T)$ representa la memoria utilizada por un dato del vector.

El uso de memoria M_l de una lista simple para almacenar los elementos es:

$$M_l = 1200M(T) + 1200M(P) + 2M(P) + M(int)$$

El vectorutiliza $150M(T) + M(int)$ más memoria. Sin embargo la lista usa punteros en cada elemento y en total $1202M(P)$. $M(P)$ tiene 8 bytes en una arquitectura reciente. Esto quiere decir que la lista usa 9616 bytes en punteros. Entonces, a menos que $M(T) > 64$ bytes, el vector utilizará más memoria que la lista ($9616/150 \approx 64$).

Pregunta 3 pg. 3:

- Si la pila está representada utilizando `vector<T>`. Complejidad $O(n)$.

```
T& max() {
  assert(!empty());
  T m = storage.front();
  int mp = 0;
  for (int i = 0; i < size(); i++) {
    if (m > storage[i]) {
      m = storage[i];
      mp = i;
    }
  }
  return storage[mp];
}
```

- Si la pila está representada utilizando `list<T>`.

Pregunta 6 pg. 4:

$$O(n!), O(2^n), O(n^2), O(n \cdot \log_2 n), O(\log_2 n), O(1) \tag{1.2}$$

Pregunta 7 pg. 5:

a) $O(n)$: `push_front`, `pop_front` en la clase `vector`. `pop_back` en la clase `list` (single linked list).

b) $O(n^2)$: `insertionSort`, `selectionSort`, `duplicates` (encuentra cuántos elementos se repiten en un vector).

c) $O(n \cdot \log_2 n)$: `mergeSort`, `quickSort` (caso promedio).

d) $O(\log_2 n)$: `binarySearch`, `power` (eleva una número a un exponente haciendo uso de: $x^n = (x \cdot x)^{\frac{n}{2}}$).

e) $O(2^n)$: `hanoi`, `fibonacci` (implementación recursiva).

Pregunta 8 pg. 5:

```
bool member(T elem) const {
  if (empty()) return false;
  Node* x = first;
  while(x != nullptr) {
    if(x->getData() == elem) return true;
    x = x->getNext();
  }
  return false;
}
```

En el peor de los casos tiene que recorrer todos los elementos de la lista. Este caso se da cuando `elem` no se encuentra o cuando es el último elemento. La complejidad es $O(n)$ siendo n el número de elementos en la lsita.