
Nombre:

DNI / NIE:

Programación Avanzada y Estructuras de Datos
Examen final C2 - 17 de enero de 2024

Instrucciones:

- **Antes de comenzar el examen** poned vuestros datos en el cuadernillo de preguntas y también en la hoja de respuestas.
- **No olvidéis** poner la modalidad en la hoja de respuestas.
- **Dejad** encima de la mesa vuestro DNI o vuestra TIU.
- No podéis consultar ningún material ni hablar con nadie.
- Cada pregunta solo tiene una opción correcta, marcadla como se indica en la hoja de respuestas.
- Cada respuesta incorrecta resta la mitad de una correcta. Las preguntas sin contestar ni suman ni restan puntos.
- Tenéis 45 min. para hacer este test.
- Esta sección del examen (tipo test) cuenta 4 puntos sobre la nota final del examen.
- Una vez finalizada, comenzará la sección de problemas, que cuenta 6 puntos sobre la nota final del examen.
- No es necesario obtener una nota mínima en ninguna de las dos partes (test y problemas).

Normativa: (*Reglamento para la evaluación de los aprendizajes*, 27-11-2015)

- Está prohibido acceder al aula del examen con cualquier tipo de dispositivo electrónico.
- Además de dos bolígrafos o lápices, del documento de identificación personal y del material suministrado por el profesorado, **no** se permite tener ningún objeto o documento, ni en la mesa ni en sus inmediaciones.
- Si tienes dudas acerca de un objeto o dispositivo concreto pregunta al profesor antes de que comience la prueba.
- El incumplimiento de esta normativa puede conllevar, entre otras, la expulsión del aula del examen sin posibilidad de realizar la prueba.

Modalidad 1

Preguntas:

1. En los tipos abstractos de datos:
   (a) Todas las implementaciones de la misma especificación deben emplear la misma representación (vectorial, enlazada, etc.).
   (b) Una implementación puede corresponder a múltiples especificaciones distintas.
   (c) Ninguna de las opciones anteriores es correcta.
2. El modificador `const` a la derecha de la declaración de un método en el fichero `.h`, como en `vector<int>categoriasMasFrecuentes(vector<int>&v) const;`, quiere decir:
   (a) Que el método puede ser invocado por objetos constantes. Si no pusiéramos el modifcador `const`, el método no podría ser invocado por objetos constantes.
   (b) Que los parámetros que se pasan al método no se modifican durante su ejecución.
   (c) Que los valores de retorno del método no se modifican.
3. Encontrar un arco de retroceso en el árbol extendido en profundidad de un grafo dirigido nos garantiza que:
   (a) El grafo contiene un ciclo
   (b) El grafo es conexo
   (c) Ambas opciones son correctas
4. En la operación de inserción en un árbol AVL:
   (a) Siempre se realiza al menos una rotación
   (b) Se realiza como máximo una rotación
   (c) Ninguna de las opciones anteriores es correcta
5. La complejidad en el peor caso del algoritmo de búsqueda binaria (buscar un elemento en un vector ordenado) en un TAD vector implementado mediante un array de C++ es:
   (a) $O(1)$
   (b) $O(\log n)$
   (c) $O(n)$
6. ¿Cuál de estas implementaciones ofrece menor complejidad temporal para las operaciones de inserción y búsqueda en un árbol AVL?
   (a) Implementación mediante vector
   (b) Implementación enlazada
   (c) Ambas ofrecen las mismas complejidades temporales asintóticas. La diferencia entre ambas es la complejidad espacial

7. A la hora de calcular la complejidad temporal asintótica de una función, un bucle que siempre se ejecuta 1 000 veces y no depende del parámetro de entrada de la función:
   (a) Constituye 1 paso de programa
   (b) Constituye 1 000 pasos de programa.
   (c) Ninguna de las opciones anteriores es correcta.
8. El algoritmo heapsort:
   (a) Ordena un vector aprovechádose del hecho de que el mayor elemento de un heap máximo (o menor, en caso de un heap mínimo) es su raíz
   (b) Además del vector que se desea ordenar, necesita memoria adicional para almacenar el heap auxiliar
   (c) Ambas opciones son correctas
9. Para implementar un grafo, es preferible emplear la representación mediante matriz frente a la representación mediante lista de adyacencia cuando:
   (a) El grafo es muy disperso (tiene pocos arcos)
   (b) El grafo es muy denso (tiene muchos arcos)
   (c) Ambas representaciones son equivalentes en cuanto a tiempos de ejecución y requisitos de memoria y se pueden usar indistintamente
10. Respecto a las estrategias de redispersión "hash abierto" y "hash cerrado con segunda función de hash" en la implementación del TAD conjunto mediante tabla hash:
    (a) Ambas estrategias de redispersión ofrecen la misma complejidad espacial asintótica
    (b) La estrategia "hash abierto" necesita más operaciones de *rehashing* para mantener constante la complejidad de la búsqueda
    (c) En la estrategia "hash cerrado con segunda función de hash" sólo se producen colisiones entre claves sinónimas
11. El operador `delete` en C++:
    (a) Debe usarse obligatoriamente para liberar la memoria ocupada por todas las variables empleadas en un programa.
    (b) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, el programa siempre acabará con una violación de segmento o *segmentation fault*.
    (c) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, se incrementará el uso de memoria del programa durante su ejecución.

12. ¿Cuál es la complejidad en el peor caso de añadir un elemento (operación `push`) en una pila implementada como una lista enlazada?
    (a) $O(1)$
    (b) $O(\log n)$
    (c) $O(n)$

13. Imagina que la biblioteca C++ STL no dispone del tipo `queue` y debes implementar el TAD cola mediante un objeto de tipo `vector`. ¿Cuál sería la complejidad de la operación `dequeue` (sacar de la cola) en la implementación más eficiente posible mediante un `vector`?
    (a) $\Theta(1)$
    (b) $\Theta(\log n)$
    (c) $\Theta(n)$

14. Imagina que tenemos un progama en el que vamos a emplear un vector en el que se van a realizar muchas inserciones al principio del mismo, y necesitamos que cada inserción se ejecute en tiempo constante. ¿Qué implementación es más adecuada?
    (a) Array estático: si conocemos de antemano el número máximo de elementos que deberán almacenarse, no será necesario redimensionar
    (b) Array dinámico: las redimensiones no son un problema porque su coste amortizado es $O(1)$
    (c) Lista enlazada: las inserciones al principio de la lista siempre se realizan en tiempo constante

15. ¿Cuál es la complejidad en el peor caso de eliminar un elemento (operación `erase`) en un vector implementado como un array?
    (a) $O(1)$
    (b) $O(\log n)$
    (c) $O(n)$

16. En el TAD cola (no confundir con cola de prioridad):
    (a) Sólo podemos insertar elementos en uno de los extremos
    (b) Sólo podemos insertar elementos en ambos extremos.
    (c) Podemos insertar elementos en cualquier posición, siempre que la cola se mantenga ordenada.

17. El TAD `map` en C++ está implementado como:
    (a) Un árbol binario de búsqueda con un algoritmo que permite mantenerlo equilibrado
    (b) Una tabla hash con redispersión cerrada
    (c) Una tabla hash con redispersión abierta

18. Siempre es posible reconstruir un único arbol binario si conocemos esta pareja de recorridos:
    (a) inorden y niveles
    (b) inorden y preorden
    (c) Las dos opciones anteriores son correctas
19. La complejidad espacial **asintótica** de un heap (montículo):
    (a) Es mayor (necesita significativamente más memoria) si se representa mediante un vector que si se implementa mediante punteros
    (b) Es menor (necesita significativamente menos memoria) si se representa mediante un vector que si se implementa mediante punteros
    (c) Ninguna de las opciones anteriores es correcta
20. Decimos que un algoritmo presenta caso mejor y caso peor cuando:
    (a) Se pueden obtener tiempos de ejecución distintos para entradas distintas
    (b) Se pueden obtener tiempos de ejecución distintos para entradas de la misma talla
    (c) Se pueden obtener tiempos de ejecución distintos para entradas idénticas

Respuestas para la modalidad **1**

1. En los tipos abstractos de datos:
   - (a) Todas las implementaciones de la misma especificación deben emplear la misma representación (vectorial, enlazada, etc.).
   - (b) Una implementación puede corresponder a múltiples especificaciones distintas.
   - [(c)] Ninguna de las opciones anteriores es correcta.

2. El modificador `const` a la derecha de la declaración de un método en el fichero `.h`, como en `vector<int>categoriasMasFrecuentes(vector<int>&v) const;`, quiere decir:
   - [(a)] Que el método puede ser invocado por objetos constantes. Si no pusiéramos el modifcador `const`, el método no podría ser invocado por objetos constantes.
   - (b) Que los parámetros que se pasan al método no se modifican durante su ejecución.
   - (c) Que los valores de retorno del método no se modifican.

3. Encontrar un arco de retroceso en el árbol extendido en profundidad de un grafo dirigido nos garantiza que:
   - [(a)] El grafo contiene un ciclo
   - (b) El grafo es conexo
   - (c) Ambas opciones son correctas

4. En la operación de inserción en un árbol AVL:
   - (a) Siempre se realiza al menos una rotación
   - [(b)] Se realiza como máximo una rotación
   - (c) Ninguna de las opciones anteriores es correcta

5. La complejidad en el peor caso del algoritmo de búsqueda binaria (buscar un elemento en un vector ordenado) en un TAD vector implementado mediante un array de C++ es:
   - (a) $O(1)$
   - [(b)] $O(\log n)$
   - (c) $O(n)$

6. ¿Cuál de estas implementaciones ofrece menor complejidad temporal para las operaciones de inserción y búsqueda en un árbol AVL?
   - (a) Implementación mediante vector
   - [(b)] Implementación enlazada
   - (c) Ambas ofrecen las mismas complejidades temporales asintóticas. La diferencia entre ambas es la complejidad espacial

7. A la hora de calcular la complejidad temporal asintótica de una función, un bucle que siempre se ejecuta 1 000 veces y no depende del parámetro de entrada de la función:
   - [x] (a) Constituye 1 paso de programa
   - [ ] (b) Constituye 1 000 pasos de programa.
   - [ ] (c) Ninguna de las opciones anteriores es correcta.

8. El algoritmo heapsort:
   - [x] (a) Ordena un vector aprovechádose del hecho de que el mayor elemento de un heap máximo (o menor, en caso de un heap mínimo) es su raíz
   - [ ] (b) Además del vector que se desea ordenar, necesita memoria adicional para almacenar el heap auxiliar
   - [ ] (c) Ambas opciones son correctas

9. Para implementar un grafo, es preferible emplear la representación mediante matriz frente a la representación mediante lista de adyacencia cuando:
   - [ ] (a) El grafo es muy disperso (tiene pocos arcos)
   - [x] (b) El grafo es muy denso (tiene muchos arcos)
   - [ ] (c) Ambas representaciones son equivalentes en cuanto a tiempos de ejecución y requisitos de memoria y se pueden usar indistintamente

10. Respecto a las estrategias de redispersión "hash abierto" y "hash cerrado con segunda función de hash" en la implementación del TAD conjunto mediante tabla hash:
    - [x] (a) Ambas estrategias de redispersión ofrecen la misma complejidad espacial asintótica
    - [ ] (b) La estrategia "hash abierto" necesita más operaciones de *rehashing* para mantener constante la complejidad de la búsqueda
    - [ ] (c) En la estrategia "hash cerrado con segunda función de hash" sólo se producen colisiones entre claves sinónimas

11. El operador `delete` en C++:
    - [ ] (a) Debe usarse obligatoriamente para liberar la memoria ocupada por todas las variables empleadas en un programa.
    - [ ] (b) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, el programa siempre acabará con una violación de segmento o *segmentation fault*.
    - [x] (c) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, se incrementará el uso de memoria del programa durante su ejecución.

12. ¿Cuál es la complejidad en el peor caso de añadir un elemento (operación `push`) en una pila implementada como una lista enlazada?

    [(a)] $O(1)$
    (b) $O(\log n)$
    (c) $O(n)$

13. Imagina que la biblioteca C++ STL no dispone del tipo `queue` y debes implementar el TAD cola mediante un objeto de tipo `vector`. ¿Cuál sería la complejidad de la operación `dequeue` (sacar de la cola) en la implementación más eficiente posible mediante un `vector`?

    [(a)] $\Theta(1)$
    (b) $\Theta(\log n)$
    (c) $\Theta(n)$

14. Imagina que tenemos un progama en el que vamos a emplear un vector en el que se van a realizar muchas inserciones al principio del mismo, y necesitamos que cada inserción se ejecute en tiempo constante. ¿Qué implementación es más adecuada?

    (a) Array estático: si conocemos de antemano el número máximo de elementos que deberán almacenarse, no será necesario redimensionar
    (b) Array dinámico: las redimensiones no son un problema porque su coste amortizado es $O(1)$
    [(c)] Lista enlazada: las inserciones al principio de la lista siempre se realizan en tiempo constante

15. ¿Cuál es la complejidad en el peor caso de eliminar un elemento (operación `erase`) en un vector implementado como un array?

    (a) $O(1)$
    (b) $O(\log n)$
    [(c)] $O(n)$

16. En el TAD cola (no confundir con cola de prioridad):

    [(a)] Sólo podemos insertar elementos en uno de los extremos
    (b) Sólo podemos insertar elementos en ambos extremos.
    (c) Podemos insertar elementos en cualquier posición, siempre que la cola se mantenga ordenada.

17. El TAD `map` en C++ está implementado como:

    [(a)] Un árbol binario de búsqueda con un algoritmo que permite mantenerlo equilibrado
    (b) Una tabla hash con redispersión cerrada
    (c) Una tabla hash con redispersión abierta

18. Siempre es posible reconstruir un único arbol binario si conocemos esta pareja de recorridos:

    (a) inorden y niveles

    (b) inorden y preorden

    [(c)] Las dos opciones anteriores son correctas

19. La complejidad espacial **asintótica** de un heap (montículo):

    (a) Es mayor (necesita significativamente más memoria) si se representa mediante un vector que si se implementa mediante punteros

    (b) Es menor (necesita significativamente menos memoria) si se representa mediante un vector que si se implementa mediante punteros

    [(c)] Ninguna de las opciones anteriores es correcta

20. Decimos que un algoritmo presenta caso mejor y caso peor cuando:

    (a) Se pueden obtener tiempos de ejecución distintos para entradas distintas

    [(b)] Se pueden obtener tiempos de ejecución distintos para entradas de la misma talla

    (c) Se pueden obtener tiempos de ejecución distintos para entradas idénticas

---

Nombre:

DNI / NIE:

Programación Avanzada y Estructuras de Datos
Examen final C2 - 17 de enero de 2024

Instrucciones:

- **Antes de comenzar el examen** poned vuestros datos en el cuadernillo de preguntas y también en la hoja de respuestas.
- **No olvidéis** poner la modalidad en la hoja de respuestas.
- **Dejad** encima de la mesa vuestro DNI o vuestra TIU.
- No podéis consultar ningún material ni hablar con nadie.
- Cada pregunta solo tiene una opción correcta, marcadla como se indica en la hoja de respuestas.
- Cada respuesta incorrecta resta la mitad de una correcta. Las preguntas sin contestar ni suman ni restan puntos.
- Tenéis 45 min. para hacer este test.
- Esta sección del examen (tipo test) cuenta 4 puntos sobre la nota final del examen.
- Una vez finalizada, comenzará la sección de problemas, que cuenta 6 puntos sobre la nota final del examen.
- No es necesario obtener una nota mínima en ninguna de las dos partes (test y problemas).

Normativa: (*Reglamento para la evaluación de los aprendizajes*, 27-11-2015)

- Está prohibido acceder al aula del examen con cualquier tipo de dispositivo electrónico.
- Además de dos bolígrafos o lápices, del documento de identificación personal y del material suministrado por el profesorado, **no** se permite tener ningún objeto o documento, ni en la mesa ni en sus inmediaciones.
- Si tienes dudas acerca de un objeto o dispositivo concreto pregunta al profesor antes de que comience la prueba.
- El incumplimiento de esta normativa puede conllevar, entre otras, la expulsión del aula del examen sin posibilidad de realizar la prueba.

Modalidad 2

Preguntas:

1. Imagina que tenemos un progama en el que vamos a emplear un vector en el que se van a realizar muchas inserciones al principio del mismo, y necesitamos que cada inserción se ejecute en tiempo constante. ¿Qué implementación es más adecuada?
   (a) Array estático: si conocemos de antemano el número máximo de elementos que deberán almacenarse, no será necesario redimensionar
   (b) Array dinámico: las redimensiones no son un problema porque su coste amortizado es $O(1)$
   (c) Lista enlazada: las inserciones al principio de la lista siempre se realizan en tiempo constante
2. Para implementar un grafo, es preferible emplear la representación mediante matriz frente a la representación mediante lista de adyacencia cuando:
   (a) El grafo es muy disperso (tiene pocos arcos)
   (b) El grafo es muy denso (tiene muchos arcos)
   (c) Ambas representaciones son equivalentes en cuanto a tiempos de ejecución y requisitos de memoria y se pueden usar indistintamente
3. En los tipos abstractos de datos:
   (a) Todas las implementaciones de la misma especificación deben emplear la misma representación (vectorial, enlazada, etc.).
   (b) Una implementación puede corresponder a múltiples especificaciones distintas.
   (c) Ninguna de las opciones anteriores es correcta.
4. En la operación de inserción en un árbol AVL:
   (a) Siempre se realiza al menos una rotación
   (b) Se realiza como máximo una rotación
   (c) Ninguna de las opciones anteriores es correcta
5. ¿Cuál es la complejidad en el peor caso de eliminar un elemento (operación `erase`) en un vector implementado como un array?
   (a) $O(1)$
   (b) $O(\log n)$
   (c) $O(n)$

6. Decimos que un algoritmo presenta caso mejor y caso peor cuando:
   (a) Se pueden obtener tiempos de ejecución distintos para entradas distintas
   (b) Se pueden obtener tiempos de ejecución distintos para entradas de la misma talla
   (c) Se pueden obtener tiempos de ejecución distintos para entradas idénticas
7. Encontrar un arco de retroceso en el árbol extendido en profundidad de un grafo dirigido nos garantiza que:
   (a) El grafo contiene un ciclo
   (b) El grafo es conexo
   (c) Ambas opciones son correctas
8. A la hora de calcular la complejidad temporal asintótica de una función, un bucle que siempre se ejecuta 1 000 veces y no depende del parámetro de entrada de la función:
   (a) Constituye 1 paso de programa
   (b) Constituye 1 000 pasos de programa.
   (c) Ninguna de las opciones anteriores es correcta.
9. Respecto a las estrategias de redispersión "hash abierto" y "hash cerrado con segunda función de hash" en la implementación del TAD conjunto mediante tabla hash:
   (a) Ambas estrategias de redispersión ofrecen la misma complejidad espacial asintótica
   (b) La estrategia "hash abierto" necesita más operaciones de *rehashing* para mantener constante la complejidad de la búsqueda
   (c) En la estrategia "hash cerrado con segunda función de hash" sólo se producen colisiones entre claves sinónimas
10. La complejidad en el peor caso del algoritmo de búsqueda binaria (buscar un elemento en un vector ordenado) en un TAD vector implementado mediante un array de C++ es:
    (a) $O(1)$
    (b) $O(\log n)$
    (c) $O(n)$
11. El TAD `map` en C++ está implementado como:
    (a) Un árbol binario de búsqueda con un algoritmo que permite mantenerlo equilibrado
    (b) Una tabla hash con redispersión cerrada
    (c) Una tabla hash con redispersión abierta

12. El operador `delete` en C++:
    (a) Debe usarse obligatoriamente para liberar la memoria ocupada por todas las variables empleadas en un programa.
    (b) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, el programa siempre acabará con una violación de segmento o *segmentation fault*.
    (c) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, se incrementará el uso de memoria del programa durante su ejecución.
13. La complejidad espacial **asintótica** de un heap (montículo):
    (a) Es mayor (necesita significativamente más memoria) si se representa mediante un vector que si se implementa mediante punteros
    (b) Es menor (necesita significativamente menos memoria) si se representa mediante un vector que si se implementa mediante punteros
    (c) Ninguna de las opciones anteriores es correcta
14. Siempre es posible reconstruir un único arbol binario si conocemos esta pareja de recorridos:
    (a) inorden y niveles
    (b) inorden y preorden
    (c) Las dos opciones anteriores son correctas
15. En el TAD cola (no confundir con cola de prioridad):
    (a) Sólo podemos insertar elementos en uno de los extremos
    (b) Sólo podemos insertar elementos en ambos extremos.
    (c) Podemos insertar elementos en cualquier posición, siempre que la cola se mantenga ordenada.
16. Imagina que la biblioteca C++ STL no dispone del tipo `queue` y debes implementar el TAD cola mediante un objeto de tipo `vector`. ¿Cuál sería la complejidad de la operación `dequeue` (sacar de la cola) en la implementación más eficiente posible mediante un `vector`?
    (a) $\Theta(1)$
    (b) $\Theta(\log n)$
    (c) $\Theta(n)$
17. ¿Cuál es la complejidad en el peor caso de añadir un elemento (operación `push`) en una pila implementada como una lista enlazada?
    (a) $O(1)$
    (b) $O(\log n)$
    (c) $O(n)$

18. ¿Cuál de estas implementaciones ofrece menor complejidad temporal para las operaciones de inserción y búsqueda en un árbol AVL?

    (a) Implementación mediante vector

    (b) Implementación enlazada

    (c) Ambas ofrecen las mismas complejidades temporales asintóticas. La diferencia entre ambas es la complejidad espacial

19. El modificador `const` a la derecha de la declaración de un método en el fichero `.h`, como en `vector<int>categoriasMasFrecuentes(vector<int>&v) const;`, quiere decir:

    (a) Que el método puede ser invocado por objetos constantes. Si no pusiéramos el modifcador `const`, el método no podría ser invocado por objetos constantes.

    (b) Que los parámetros que se pasan al método no se modifican durante su ejecución.

    (c) Que los valores de retorno del método no se modifican.

20. El algoritmo heapsort:

    (a) Ordena un vector aprovechádose del hecho de que el mayor elemento de un heap máximo (o menor, en caso de un heap mínimo) es su raíz

    (b) Además del vector que se desea ordenar, necesita memoria adicional para almacenar el heap auxiliar

    (c) Ambas opciones son correctas

Respuestas para la modalidad **2**

1. Imagina que tenemos un progama en el que vamos a emplear un vector en el que se van a realizar muchas inserciones al principio del mismo, y necesitamos que cada inserción se ejecute en tiempo constante. ¿Qué implementación es más adecuada?

   (a) Array estático: si conocemos de antemano el número máximo de elementos que deberán almacenarse, no será necesario redimensionar

   (b) Array dinámico: las redimensiones no son un problema porque su coste amortizado es $O(1)$

   [(c)] Lista enlazada: las inserciones al principio de la lista siempre se realizan en tiempo constante

2. Para implementar un grafo, es preferible emplear la representación mediante matriz frente a la representación mediante lista de adyacencia cuando:

   (a) El grafo es muy disperso (tiene pocos arcos)

   [(b)] El grafo es muy denso (tiene muchos arcos)

   (c) Ambas representaciones son equivalentes en cuanto a tiempos de ejecución y requisitos de memoria y se pueden usar indistintamente

3. En los tipos abstractos de datos:

   (a) Todas las implementaciones de la misma especificación deben emplear la misma representación (vectorial, enlazada, etc.).

   (b) Una implementación puede corresponder a múltiples especificaciones distintas.

   [(c)] Ninguna de las opciones anteriores es correcta.

4. En la operación de inserción en un árbol AVL:

   (a) Siempre se realiza al menos una rotación

   [(b)] Se realiza como máximo una rotación

   (c) Ninguna de las opciones anteriores es correcta

5. ¿Cuál es la complejidad en el peor caso de eliminar un elemento (operación `erase`) en un vector implementado como un array?

   (a) $O(1)$

   (b) $O(\log n)$

   [(c)] $O(n)$

6. Decimos que un algoritmo presenta caso mejor y caso peor cuando:
   (a) Se pueden obtener tiempos de ejecución distintos para entradas distintas
   [(b)] Se pueden obtener tiempos de ejecución distintos para entradas de la misma talla
   (c) Se pueden obtener tiempos de ejecución distintos para entradas idénticas
7. Encontrar un arco de retroceso en el árbol extendido en profundidad de un grafo dirigido nos garantiza que:
   [(a)] El grafo contiene un ciclo
   (b) El grafo es conexo
   (c) Ambas opciones son correctas
8. A la hora de calcular la complejidad temporal asintótica de una función, un bucle que siempre se ejecuta 1 000 veces y no depende del parámetro de entrada de la función:
   [(a)] Constituye 1 paso de programa
   (b) Constituye 1 000 pasos de programa.
   (c) Ninguna de las opciones anteriores es correcta.
9. Respecto a las estrategias de redispersión "hash abierto" y "hash cerrado con segunda función de hash" en la implementación del TAD conjunto mediante tabla hash:
   [(a)] Ambas estrategias de redispersión ofrecen la misma complejidad espacial asintótica
   (b) La estrategia "hash abierto" necesita más operaciones de *rehashing* para mantener constante la complejidad de la búsqueda
   (c) En la estrategia "hash cerrado con segunda función de hash" sólo se producen colisiones entre claves sinónimas
10. La complejidad en el peor caso del algoritmo de búsqueda binaria (buscar un elemento en un vector ordenado) en un TAD vector implementado mediante un array de C++ es:
    (a) $O(1)$
    [(b)] $O(\log n)$
    (c) $O(n)$
11. El TAD `map` en C++ está implementado como:
    [(a)] Un árbol binario de búsqueda con un algoritmo que permite mantenerlo equilibrado
    (b) Una tabla hash con redispersión cerrada
    (c) Una tabla hash con redispersión abierta

12. El operador `delete` en C++:

    (a) Debe usarse obligatoriamente para liberar la memoria ocupada por todas las variables empleadas en un programa.

    (b) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, el programa siempre acabará con una violación de segmento o *segmentation fault*.

    [(c)] Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, se incrementará el uso de memoria del programa durante su ejecución.

13. La complejidad espacial **asintótica** de un heap (montículo):

    (a) Es mayor (necesita significativamente más memoria) si se representa mediante un vector que si se implementa mediante punteros

    (b) Es menor (necesita significativamente menos memoria) si se representa mediante un vector que si se implementa mediante punteros

    [(c)] Ninguna de las opciones anteriores es correcta

14. Siempre es posible reconstruir un único arbol binario si conocemos esta pareja de recorridos:

    (a) inorden y niveles

    (b) inorden y preorden

    [(c)] Las dos opciones anteriores son correctas

15. En el TAD cola (no confundir con cola de prioridad):

    [(a)] Sólo podemos insertar elementos en uno de los extremos

    (b) Sólo podemos insertar elementos en ambos extremos.

    (c) Podemos insertar elementos en cualquier posición, siempre que la cola se mantenga ordenada.

16. Imagina que la biblioteca C++ STL no dispone del tipo `queue` y debes implementar el TAD cola mediante un objeto de tipo `vector`. ¿Cuál sería la complejidad de la operación `dequeue` (sacar de la cola) en la implementación más eficiente posible mediante un `vector`?

    [(a)] $\Theta(1)$

    (b) $\Theta(\log n)$

    (c) $\Theta(n)$

17. ¿Cuál es la complejidad en el peor caso de añadir un elemento (operación `push`) en una pila implementada como una lista enlazada?

    [(a)] $O(1)$

    (b) $O(\log n)$

    (c) $O(n)$

18. ¿Cuál de estas implementaciones ofrece menor complejidad temporal para las operaciones de inserción y búsqueda en un árbol AVL?

    (a) Implementación mediante vector

    [(b)] Implementación enlazada

    (c) Ambas ofrecen las mismas complejidades temporales asintóticas. La diferencia entre ambas es la complejidad espacial

19. El modificador `const` a la derecha de la declaración de un método en el fichero `.h`, como en `vector<int>categoriasMasFrecuentes(vector<int>&v) const;`, quiere decir:

    [(a)] Que el método puede ser invocado por objetos constantes. Si no pusiéramos el modifcador `const`, el método no podría ser invocado por objetos constantes.

    (b) Que los parámetros que se pasan al método no se modifican durante su ejecución.

    (c) Que los valores de retorno del método no se modifican.

20. El algoritmo heapsort:

    [(a)] Ordena un vector aprovechádose del hecho de que el mayor elemento de un heap máximo (o menor, en caso de un heap mínimo) es su raíz

    (b) Además del vector que se desea ordenar, necesita memoria adicional para almacenar el heap auxiliar

    (c) Ambas opciones son correctas

Nombre:

DNI / NIE:

Programación Avanzada y Estructuras de Datos
Examen final C2 - 17 de enero de 2024

Instrucciones:

- **Antes de comenzar el examen** poned vuestros datos en el cuadernillo de preguntas y también en la hoja de respuestas.
- **No olvidéis** poner la modalidad en la hoja de respuestas.
- **Dejad** encima de la mesa vuestro DNI o vuestra TIU.
- No podéis consultar ningún material ni hablar con nadie.
- Cada pregunta solo tiene una opción correcta, marcadla como se indica en la hoja de respuestas.
- Cada respuesta incorrecta resta la mitad de una correcta. Las preguntas sin contestar ni suman ni restan puntos.
- Tenéis 45 min. para hacer este test.
- Esta sección del examen (tipo test) cuenta 4 puntos sobre la nota final del examen.
- Una vez finalizada, comenzará la sección de problemas, que cuenta 6 puntos sobre la nota final del examen.
- No es necesario obtener una nota mínima en ninguna de las dos partes (test y problemas).

Normativa: (*Reglamento para la evaluación de los aprendizajes*, 27-11-2015)

- Está prohibido acceder al aula del examen con cualquier tipo de dispositivo electrónico.
- Además de dos bolígrafos o lápices, del documento de identificación personal y del material suministrado por el profesorado, **no** se permite tener ningún objeto o documento, ni en la mesa ni en sus inmediaciones.
- Si tienes dudas acerca de un objeto o dispositivo concreto pregunta al profesor antes de que comience la prueba.
- El incumplimiento de esta normativa puede conllevar, entre otras, la expulsión del aula del examen sin posibilidad de realizar la prueba.

Modalidad 3

Preguntas:

1. ¿Cuál de estas implementaciones ofrece menor complejidad temporal para las operaciones de inserción y búsqueda en un árbol AVL?
   (a) Implementación mediante vector
   (b) Implementación enlazada
   (c) Ambas ofrecen las mismas complejidades temporales asintóticas. La diferencia entre ambas es la complejidad espacial
2. Siempre es posible reconstruir un único arbol binario si conocemos esta pareja de recorridos:
   (a) inorden y niveles
   (b) inorden y preorden
   (c) Las dos opciones anteriores son correctas
3. El TAD `map` en C++ está implementado como:
   (a) Un árbol binario de búsqueda con un algoritmo que permite mantenerlo equilibrado
   (b) Una tabla hash con redispersión cerrada
   (c) Una tabla hash con redispersión abierta
4. La complejidad en el peor caso del algoritmo de búsqueda binaria (buscar un elemento en un vector ordenado) en un TAD vector implementado mediante un array de C++ es:
   (a) $O(1)$
   (b) $O(\log n)$
   (c) $O(n)$
5. Imagina que tenemos un progama en el que vamos a emplear un vector en el que se van a realizar muchas inserciones al principio del mismo, y necesitamos que cada inserción se ejecute en tiempo constante. ¿Qué implementación es más adecuada?
   (a) Array estático: si conocemos de antemano el número máximo de elementos que deberán almacenarse, no será necesario redimensionar
   (b) Array dinámico: las redimensiones no son un problema porque su coste amortizado es $O(1)$
   (c) Lista enlazada: las inserciones al principio de la lista siempre se realizan en tiempo constante
6. Decimos que un algoritmo presenta caso mejor y caso peor cuando:
   (a) Se pueden obtener tiempos de ejecución distintos para entradas distintas
   (b) Se pueden obtener tiempos de ejecución distintos para entradas de la misma talla
   (c) Se pueden obtener tiempos de ejecución distintos para entradas idénticas

7. ¿Cuál es la complejidad en el peor caso de eliminar un elemento (operación `erase`) en un vector implementado como un array?
   (a) $O(1)$
   (b) $O(\log n)$
   (c) $O(n)$
8. Para implementar un grafo, es preferible emplear la representación mediante matriz frente a la representación mediante lista de adyacencia cuando:
   (a) El grafo es muy disperso (tiene pocos arcos)
   (b) El grafo es muy denso (tiene muchos arcos)
   (c) Ambas representaciones son equivalentes en cuanto a tiempos de ejecución y requisitos de memoria y se pueden usar indistintamente
9. En la operación de inserción en un árbol AVL:
   (a) Siempre se realiza al menos una rotación
   (b) Se realiza como máximo una rotación
   (c) Ninguna de las opciones anteriores es correcta
10. La complejidad espacial **asintótica** de un heap (montículo):
    (a) Es mayor (necesita significativamente más memoria) si se representa mediante un vector que si se implementa mediante punteros
    (b) Es menor (necesita significativamente menos memoria) si se representa mediante un vector que si se implementa mediante punteros
    (c) Ninguna de las opciones anteriores es correcta
11. ¿Cuál es la complejidad en el peor caso de añadir un elemento (operación `push`) en una pila implementada como una lista enlazada?
    (a) $O(1)$
    (b) $O(\log n)$
    (c) $O(n)$
12. El operador `delete` en C++:
    (a) Debe usarse obligatoriamente para liberar la memoria ocupada por todas las variables empleadas en un programa.
    (b) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, el programa siempre acabará con una violación de segmento o *segmentation fault*.
    (c) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, se incrementará el uso de memoria del programa durante su ejecución.

13. En los tipos abstractos de datos:
    (a) Todas las implementaciones de la misma especificación deben emplear la misma representación (vectorial, enlazada, etc.).
    (b) Una implementación puede corresponder a múltiples especificaciones distintas.
    (c) Ninguna de las opciones anteriores es correcta.

14. Respecto a las estrategias de redispersión "hash abierto" y "hash cerrado con segunda función de hash" en la implementación del TAD conjunto mediante tabla hash:
    (a) Ambas estrategias de redispersión ofrecen la misma complejidad espacial asintótica
    (b) La estrategia "hash abierto" necesita más operaciones de *rehashing* para mantener constante la complejidad de la búsqueda
    (c) En la estrategia "hash cerrado con segunda función de hash" sólo se producen colisiones entre claves sinónimas

15. A la hora de calcular la complejidad temporal asintótica de una función, un bucle que siempre se ejecuta 1 000 veces y no depende del parámetro de entrada de la función:
    (a) Constituye 1 paso de programa
    (b) Constituye 1 000 pasos de programa.
    (c) Ninguna de las opciones anteriores es correcta.

16. El modificador `const` a la derecha de la declaración de un método en el fichero `.h`, como en `vector<int>categoriasMasFrecuentes(vector<int>&v) const;`, quiere decir:
    (a) Que el método puede ser invocado por objetos constantes. Si no pusiéramos el modifcador `const`, el método no podría ser invocado por objetos constantes.
    (b) Que los parámetros que se pasan al método no se modifican durante su ejecución.
    (c) Que los valores de retorno del método no se modifican.

17. Encontrar un arco de retroceso en el árbol extendido en profundidad de un grafo dirigido nos garantiza que:
    (a) El grafo contiene un ciclo
    (b) El grafo es conexo
    (c) Ambas opciones son correctas

18. El algoritmo heapsort:
    (a) Ordena un vector aprovechádose del hecho de que el mayor elemento de un heap máximo (o menor, en caso de un heap mínimo) es su raíz
    (b) Además del vector que se desea ordenar, necesita memoria adicional para almacenar el heap auxiliar
    (c) Ambas opciones son correctas
19. En el TAD cola (no confundir con cola de prioridad):
    (a) Sólo podemos insertar elementos en uno de los extremos
    (b) Sólo podemos insertar elementos en ambos extremos.
    (c) Podemos insertar elementos en cualquier posición, siempre que la cola se mantenga ordenada.
20. Imagina que la biblioteca C++ STL no dispone del tipo `queue` y debes implementar el TAD cola mediante un objeto de tipo `vector`. ¿Cuál sería la complejidad de la operación `dequeue` (sacar de la cola) en la implementación más eficiente posible mediante un `vector`?
    (a) $\Theta(1)$
    (b) $\Theta(\log n)$
    (c) $\Theta(n)$

Respuestas para la modalidad **3**

1. ¿Cuál de estas implementaciones ofrece menor complejidad temporal para las operaciones de inserción y búsqueda en un árbol AVL?

   (a) Implementación mediante vector

   [(b)] Implementación enlazada

   (c) Ambas ofrecen las mismas complejidades temporales asintóticas. La diferencia entre ambas es la complejidad espacial

2. Siempre es posible reconstruir un único arbol binario si conocemos esta pareja de recorridos:

   (a) inorden y niveles

   (b) inorden y preorden

   [(c)] Las dos opciones anteriores son correctas

3. El TAD `map` en C++ está implementado como:

   [(a)] Un árbol binario de búsqueda con un algoritmo que permite mantenerlo equilibrado

   (b) Una tabla hash con redispersión cerrada

   (c) Una tabla hash con redispersión abierta

4. La complejidad en el peor caso del algoritmo de búsqueda binaria (buscar un elemento en un vector ordenado) en un TAD vector implementado mediante un array de C++ es:

   (a) $O(1)$

   [(b)] $O(\log n)$

   (c) $O(n)$

5. Imagina que tenemos un progama en el que vamos a emplear un vector en el que se van a realizar muchas inserciones al principio del mismo, y necesitamos que cada inserción se ejecute en tiempo constante. ¿Qué implementación es más adecuada?

   (a) Array estático: si conocemos de antemano el número máximo de elementos que deberán almacenarse, no será necesario redimensionar

   (b) Array dinámico: las redimensiones no son un problema porque su coste amortizado es $O(1)$

   [(c)] Lista enlazada: las inserciones al principio de la lista siempre se realizan en tiempo constante

6. Decimos que un algoritmo presenta caso mejor y caso peor cuando:

   (a) Se pueden obtener tiempos de ejecución distintos para entradas distintas

   [(b)] Se pueden obtener tiempos de ejecución distintos para entradas de la misma talla

   (c) Se pueden obtener tiempos de ejecución distintos para entradas idénticas

7. ¿Cuál es la complejidad en el peor caso de eliminar un elemento (operación `erase`) en un vector implementado como un array?

   (a) $O(1)$

   (b) $O(\log n)$

   [(c)] $O(n)$

8. Para implementar un grafo, es preferible emplear la representación mediante matriz frente a la representación mediante lista de adyacencia cuando:

   (a) El grafo es muy disperso (tiene pocos arcos)

   [(b)] El grafo es muy denso (tiene muchos arcos)

   (c) Ambas representaciones son equivalentes en cuanto a tiempos de ejecución y requisitos de memoria y se pueden usar indistintamente

9. En la operación de inserción en un árbol AVL:

   (a) Siempre se realiza al menos una rotación

   [(b)] Se realiza como máximo una rotación

   (c) Ninguna de las opciones anteriores es correcta

10. La complejidad espacial **asintótica** de un heap (montículo):

    (a) Es mayor (necesita significativamente más memoria) si se representa mediante un vector que si se implementa mediante punteros

    (b) Es menor (necesita significativamente menos memoria) si se representa mediante un vector que si se implementa mediante punteros

    [(c)] Ninguna de las opciones anteriores es correcta

11. ¿Cuál es la complejidad en el peor caso de añadir un elemento (operación `push`) en una pila implementada como una lista enlazada?

    [(a)] $O(1)$

    (b) $O(\log n)$

    (c) $O(n)$

12. El operador `delete` en C++:

    (a) Debe usarse obligatoriamente para liberar la memoria ocupada por todas las variables empleadas en un programa.

    (b) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, el programa siempre acabará con una violación de segmento o *segmentation fault*.

    [(c)] Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, se incrementará el uso de memoria del programa durante su ejecución.

13. En los tipos abstractos de datos:
    (a) Todas las implementaciones de la misma especificación deben emplear la misma representación (vectorial, enlazada, etc.).
    (b) Una implementación puede corresponder a múltiples especificaciones distintas.
    [(c)] Ninguna de las opciones anteriores es correcta.

14. Respecto a las estrategias de redispersión "hash abierto" y "hash cerrado con segunda función de hash" en la implementación del TAD conjunto mediante tabla hash:
    [(a)] Ambas estrategias de redispersión ofrecen la misma complejidad espacial asintótica
    (b) La estrategia "hash abierto" necesita más operaciones de *rehashing* para mantener constante la complejidad de la búsqueda
    (c) En la estrategia "hash cerrado con segunda función de hash" sólo se producen colisiones entre claves sinónimas

15. A la hora de calcular la complejidad temporal asintótica de una función, un bucle que siempre se ejecuta 1 000 veces y no depende del parámetro de entrada de la función:
    [(a)] Constituye 1 paso de programa
    (b) Constituye 1 000 pasos de programa.
    (c) Ninguna de las opciones anteriores es correcta.

16. El modificador `const` a la derecha de la declaración de un método en el fichero `.h`, como en `vector<int>categoriasMasFrecuentes(vector<int>&v) const;`, quiere decir:
    [(a)] Que el método puede ser invocado por objetos constantes. Si no pusiéramos el modifcador `const`, el método no podría ser invocado por objetos constantes.
    (b) Que los parámetros que se pasan al método no se modifican durante su ejecución.
    (c) Que los valores de retorno del método no se modifican.

17. Encontrar un arco de retroceso en el árbol extendido en profundidad de un grafo dirigido nos garantiza que:
    [(a)] El grafo contiene un ciclo
    (b) El grafo es conexo
    (c) Ambas opciones son correctas

18. El algoritmo heapsort:

    [(a)] Ordena un vector aprovechádose del hecho de que el mayor elemento de un heap máximo (o menor, en caso de un heap mínimo) es su raíz

    (b) Además del vector que se desea ordenar, necesita memoria adicional para almacenar el heap auxiliar

    (c) Ambas opciones son correctas

19. En el TAD cola (no confundir con cola de prioridad):

    [(a)] Sólo podemos insertar elementos en uno de los extremos

    (b) Sólo podemos insertar elementos en ambos extremos.

    (c) Podemos insertar elementos en cualquier posición, siempre que la cola se mantenga ordenada.

20. Imagina que la biblioteca C++ STL no dispone del tipo `queue` y debes implementar el TAD cola mediante un objeto de tipo `vector`. ¿Cuál sería la complejidad de la operación `dequeue` (sacar de la cola) en la implementación más eficiente posible mediante un `vector`?

    [(a)] $\Theta(1)$

    (b) $\Theta(\log n)$

    (c) $\Theta(n)$

Nombre:

DNI / NIE:

Programación Avanzada y Estructuras de Datos
Examen final C2 - 17 de enero de 2024

Instrucciones:

- **Antes de comenzar el examen** poned vuestros datos en el cuadernillo de preguntas y también en la hoja de respuestas.
- **No olvidéis** poner la modalidad en la hoja de respuestas.
- **Dejad** encima de la mesa vuestro DNI o vuestra TIU.
- No podéis consultar ningún material ni hablar con nadie.
- Cada pregunta solo tiene una opción correcta, marcadla como se indica en la hoja de respuestas.
- Cada respuesta incorrecta resta la mitad de una correcta. Las preguntas sin contestar ni suman ni restan puntos.
- Tenéis 45 min. para hacer este test.
- Esta sección del examen (tipo test) cuenta 4 puntos sobre la nota final del examen.
- Una vez finalizada, comenzará la sección de problemas, que cuenta 6 puntos sobre la nota final del examen.
- No es necesario obtener una nota mínima en ninguna de las dos partes (test y problemas).

Normativa: (*Reglamento para la evaluación de los aprendizajes*, 27-11-2015)

- Está prohibido acceder al aula del examen con cualquier tipo de dispositivo electrónico.
- Además de dos bolígrafos o lápices, del documento de identificación personal y del material suministrado por el profesorado, **no** se permite tener ningún objeto o documento, ni en la mesa ni en sus inmediaciones.
- Si tienes dudas acerca de un objeto o dispositivo concreto pregunta al profesor antes de que comience la prueba.
- El incumplimiento de esta normativa puede conllevar, entre otras, la expulsión del aula del examen sin posibilidad de realizar la prueba.

Modalidad 4

Preguntas:

1. El algoritmo heapsort:
   (a) Ordena un vector aprovechádose del hecho de que el mayor elemento de un heap máximo (o menor, en caso de un heap mínimo) es su raíz
   (b) Además del vector que se desea ordenar, necesita memoria adicional para almacenar el heap auxiliar
   (c) Ambas opciones son correctas
2. ¿Cuál es la complejidad en el peor caso de eliminar un elemento (operación `erase`) en un vector implementado como un array?
   (a) $O(1)$
   (b) $O(\log n)$
   (c) $O(n)$
3. Decimos que un algoritmo presenta caso mejor y caso peor cuando:
   (a) Se pueden obtener tiempos de ejecución distintos para entradas distintas
   (b) Se pueden obtener tiempos de ejecución distintos para entradas de la misma talla
   (c) Se pueden obtener tiempos de ejecución distintos para entradas idénticas
4. La complejidad en el peor caso del algoritmo de búsqueda binaria (buscar un elemento en un vector ordenado) en un TAD vector implementado mediante un array de C++ es:
   (a) $O(1)$
   (b) $O(\log n)$
   (c) $O(n)$
5. La complejidad espacial **asintótica** de un heap (montículo):
   (a) Es mayor (necesita significativamente más memoria) si se representa mediante un vector que si se implementa mediante punteros
   (b) Es menor (necesita significativamente menos memoria) si se representa mediante un vector que si se implementa mediante punteros
   (c) Ninguna de las opciones anteriores es correcta

6. El modificador `const` a la derecha de la declaración de un método en el fichero `.h`, como en `vector<int>categoriasMasFrecuentes(vector<int>&v) const;`, quiere decir:
   (a) Que el método puede ser invocado por objetos constantes. Si no pusiéramos el modifcador `const`, el método no podría ser invocado por objetos constantes.
   (b) Que los parámetros que se pasan al método no se modifican durante su ejecución.
   (c) Que los valores de retorno del método no se modifican.
7. El TAD `map` en C++ está implementado como:
   (a) Un árbol binario de búsqueda con un algoritmo que permite mantenerlo equilibrado
   (b) Una tabla hash con redispersión cerrada
   (c) Una tabla hash con redispersión abierta
8. ¿Cuál de estas implementaciones ofrece menor complejidad temporal para las operaciones de inserción y búsqueda en un árbol AVL?
   (a) Implementación mediante vector
   (b) Implementación enlazada
   (c) Ambas ofrecen las mismas complejidades temporales asintóticas. La diferencia entre ambas es la complejidad espacial
9. Imagina que la biblioteca C++ STL no dispone del tipo `queue` y debes implementar el TAD cola mediante un objeto de tipo `vector`. ¿Cuál sería la complejidad de la operación `dequeue` (sacar de la cola) en la implementación más eficiente posible mediante un `vector`?
   (a) $\Theta(1)$
   (b) $\Theta(\log n)$
   (c) $\Theta(n)$
10. En la operación de inserción en un árbol AVL:
    (a) Siempre se realiza al menos una rotación
    (b) Se realiza como máximo una rotación
    (c) Ninguna de las opciones anteriores es correcta
11. En los tipos abstractos de datos:
    (a) Todas las implementaciones de la misma especificación deben emplear la misma representación (vectorial, enlazada, etc.).
    (b) Una implementación puede corresponder a múltiples especificaciones distintas.
    (c) Ninguna de las opciones anteriores es correcta.

12. A la hora de calcular la complejidad temporal asintótica de una función, un bucle que siempre se ejecuta 1 000 veces y no depende del parámetro de entrada de la función:
    (a) Constituye 1 paso de programa
    (b) Constituye 1 000 pasos de programa.
    (c) Ninguna de las opciones anteriores es correcta.
13. En el TAD cola (no confundir con cola de prioridad):
    (a) Sólo podemos insertar elementos en uno de los extremos
    (b) Sólo podemos insertar elementos en ambos extremos.
    (c) Podemos insertar elementos en cualquier posición, siempre que la cola se mantenga ordenada.
14. Encontrar un arco de retroceso en el árbol extendido en profundidad de un grafo dirigido nos garantiza que:
    (a) El grafo contiene un ciclo
    (b) El grafo es conexo
    (c) Ambas opciones son correctas
15. ¿Cuál es la complejidad en el peor caso de añadir un elemento (operación `push`) en una pila implementada como una lista enlazada?
    (a) $O(1)$
    (b) $O(\log n)$
    (c) $O(n)$
16. Respecto a las estrategias de redispersión "hash abierto" y "hash cerrado con segunda función de hash" en la implementación del TAD conjunto mediante tabla hash:
    (a) Ambas estrategias de redispersión ofrecen la misma complejidad espacial asintótica
    (b) La estrategia "hash abierto" necesita más operaciones de *rehashing* para mantener constante la complejidad de la búsqueda
    (c) En la estrategia "hash cerrado con segunda función de hash" sólo se producen colisiones entre claves sinónimas
17. Siempre es posible reconstruir un único arbol binario si conocemos esta pareja de recorridos:
    (a) inorden y niveles
    (b) inorden y preorden
    (c) Las dos opciones anteriores son correctas

18. El operador `delete` en C++:

    (a) Debe usarse obligatoriamente para liberar la memoria ocupada por todas las variables empleadas en un programa.

    (b) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, el programa siempre acabará con una violación de segmento o *segmentation fault*.

    (c) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, se incrementará el uso de memoria del programa durante su ejecución.

19. Imagina que tenemos un progama en el que vamos a emplear un vector en el que se van a realizar muchas inserciones al principio del mismo, y necesitamos que cada inserción se ejecute en tiempo constante. ¿Qué implementación es más adecuada?

    (a) Array estático: si conocemos de antemano el número máximo de elementos que deberán almacenarse, no será necesario redimensionar

    (b) Array dinámico: las redimensiones no son un problema porque su coste amortizado es $O(1)$

    (c) Lista enlazada: las inserciones al principio de la lista siempre se realizan en tiempo constante

20. Para implementar un grafo, es preferible emplear la representación mediante matriz frente a la representación mediante lista de adyacencia cuando:

    (a) El grafo es muy disperso (tiene pocos arcos)

    (b) El grafo es muy denso (tiene muchos arcos)

    (c) Ambas representaciones son equivalentes en cuanto a tiempos de ejecución y requisitos de memoria y se pueden usar indistintamente

Respuestas para la modalidad **4**

1. El algoritmo heapsort:

   [(a)] Ordena un vector aprovechádose del hecho de que el mayor elemento de un heap máximo (o menor, en caso de un heap mínimo) es su raíz

   (b) Además del vector que se desea ordenar, necesita memoria adicional para almacenar el heap auxiliar

   (c) Ambas opciones son correctas

2. ¿Cuál es la complejidad en el peor caso de eliminar un elemento (operación `erase`) en un vector implementado como un array?

   (a) $O(1)$

   (b) $O(\log n)$

   [(c)] $O(n)$

3. Decimos que un algoritmo presenta caso mejor y caso peor cuando:

   (a) Se pueden obtener tiempos de ejecución distintos para entradas distintas

   [(b)] Se pueden obtener tiempos de ejecución distintos para entradas de la misma talla

   (c) Se pueden obtener tiempos de ejecución distintos para entradas idénticas

4. La complejidad en el peor caso del algoritmo de búsqueda binaria (buscar un elemento en un vector ordenado) en un TAD vector implementado mediante un array de C++ es:

   (a) $O(1)$

   [(b)] $O(\log n)$

   (c) $O(n)$

5. La complejidad espacial **asintótica** de un heap (montículo):

   (a) Es mayor (necesita significativamente más memoria) si se representa mediante un vector que si se implementa mediante punteros

   (b) Es menor (necesita significativamente menos memoria) si se representa mediante un vector que si se implementa mediante punteros

   [(c)] Ninguna de las opciones anteriores es correcta

6. El modificador `const` a la derecha de la declaración de un método en el fichero `.h`, como en `vector<int>categoriasMasFrecuentes(vector<int>&v) const;`, quiere decir:

   [(a)] Que el método puede ser invocado por objetos constantes. Si no pusiéramos el modifcador `const`, el método no podría ser invocado por objetos constantes.

   (b) Que los parámetros que se pasan al método no se modifican durante su ejecución.

   (c) Que los valores de retorno del método no se modifican.

7. El TAD `map` en C++ está implementado como:

   [(a)] Un árbol binario de búsqueda con un algoritmo que permite mantenerlo equilibrado

   (b) Una tabla hash con redispersión cerrada

   (c) Una tabla hash con redispersión abierta

8. ¿Cuál de estas implementaciones ofrece menor complejidad temporal para las operaciones de inserción y búsqueda en un árbol AVL?

   (a) Implementación mediante vector

   [(b)] Implementación enlazada

   (c) Ambas ofrecen las mismas complejidades temporales asintóticas. La diferencia entre ambas es la complejidad espacial

9. Imagina que la biblioteca C++ STL no dispone del tipo `queue` y debes implementar el TAD cola mediante un objeto de tipo `vector`. ¿Cuál sería la complejidad de la operación `dequeue` (sacar de la cola) en la implementación más eficiente posible mediante un `vector`?

   [(a)] $\Theta(1)$

   (b) $\Theta(\log n)$

   (c) $\Theta(n)$

10. En la operación de inserción en un árbol AVL:

    (a) Siempre se realiza al menos una rotación

    [(b)] Se realiza como máximo una rotación

    (c) Ninguna de las opciones anteriores es correcta

11. En los tipos abstractos de datos:

    (a) Todas las implementaciones de la misma especificación deben emplear la misma representación (vectorial, enlazada, etc.).

    (b) Una implementación puede corresponder a múltiples especificaciones distintas.

    [(c)] Ninguna de las opciones anteriores es correcta.

12. A la hora de calcular la complejidad temporal asintótica de una función, un bucle que siempre se ejecuta 1 000 veces y no depende del parámetro de entrada de la función:
    - [x] (a) Constituye 1 paso de programa
    - [ ] (b) Constituye 1 000 pasos de programa.
    - [ ] (c) Ninguna de las opciones anteriores es correcta.

13. En el TAD cola (no confundir con cola de prioridad):
    - [x] (a) Sólo podemos insertar elementos en uno de los extremos
    - [ ] (b) Sólo podemos insertar elementos en ambos extremos.
    - [ ] (c) Podemos insertar elementos en cualquier posición, siempre que la cola se mantenga ordenada.

14. Encontrar un arco de retroceso en el árbol extendido en profundidad de un grafo dirigido nos garantiza que:
    - [x] (a) El grafo contiene un ciclo
    - [ ] (b) El grafo es conexo
    - [ ] (c) Ambas opciones son correctas

15. ¿Cuál es la complejidad en el peor caso de añadir un elemento (operación `push`) en una pila implementada como una lista enlazada?
    - [x] (a) $O(1)$
    - [ ] (b) $O(\log n)$
    - [ ] (c) $O(n)$

16. Respecto a las estrategias de redispersión "hash abierto" y "hash cerrado con segunda función de hash" en la implementación del TAD conjunto mediante tabla hash:
    - [x] (a) Ambas estrategias de redispersión ofrecen la misma complejidad espacial asintótica
    - [ ] (b) La estrategia "hash abierto" necesita más operaciones de *rehashing* para mantener constante la complejidad de la búsqueda
    - [ ] (c) En la estrategia "hash cerrado con segunda función de hash" sólo se producen colisiones entre claves sinónimas

17. Siempre es posible reconstruir un único arbol binario si conocemos esta pareja de recorridos:
    - [ ] (a) inorden y niveles
    - [ ] (b) inorden y preorden
    - [x] (c) Las dos opciones anteriores son correctas

18. El operador `delete` en C++:

    (a) Debe usarse obligatoriamente para liberar la memoria ocupada por todas las variables empleadas en un programa.

    (b) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, el programa siempre acabará con una violación de segmento o *segmentation fault*.

    [(c)] Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, se incrementará el uso de memoria del programa durante su ejecución.

19. Imagina que tenemos un progama en el que vamos a emplear un vector en el que se van a realizar muchas inserciones al principio del mismo, y necesitamos que cada inserción se ejecute en tiempo constante. ¿Qué implementación es más adecuada?

    (a) Array estático: si conocemos de antemano el número máximo de elementos que deberán almacenarse, no será necesario redimensionar

    (b) Array dinámico: las redimensiones no son un problema porque su coste amortizado es $O(1)$

    [(c)] Lista enlazada: las inserciones al principio de la lista siempre se realizan en tiempo constante

20. Para implementar un grafo, es preferible emplear la representación mediante matriz frente a la representación mediante lista de adyacencia cuando:

    (a) El grafo es muy disperso (tiene pocos arcos)

    [(b)] El grafo es muy denso (tiene muchos arcos)

    (c) Ambas representaciones son equivalentes en cuanto a tiempos de ejecución y requisitos de memoria y se pueden usar indistintamente

---

Nombre:

DNI / NIE:

Programación Avanzada y Estructuras de Datos
Examen final C2 - 17 de enero de 2024

Instrucciones:

- **Antes de comenzar el examen** poned vuestros datos en el cuadernillo de preguntas y también en la hoja de respuestas.
- **No olvidéis** poner la modalidad en la hoja de respuestas.
- **Dejad** encima de la mesa vuestro DNI o vuestra TIU.
- No podéis consultar ningún material ni hablar con nadie.
- Cada pregunta solo tiene una opción correcta, marcadla como se indica en la hoja de respuestas.
- Cada respuesta incorrecta resta la mitad de una correcta. Las preguntas sin contestar ni suman ni restan puntos.
- Tenéis 45 min. para hacer este test.
- Esta sección del examen (tipo test) cuenta 4 puntos sobre la nota final del examen.
- Una vez finalizada, comenzará la sección de problemas, que cuenta 6 puntos sobre la nota final del examen.
- No es necesario obtener una nota mínima en ninguna de las dos partes (test y problemas).

Normativa: (*Reglamento para la evaluación de los aprendizajes*, 27-11-2015)

- Está prohibido acceder al aula del examen con cualquier tipo de dispositivo electrónico.
- Además de dos bolígrafos o lápices, del documento de identificación personal y del material suministrado por el profesorado, **no** se permite tener ningún objeto o documento, ni en la mesa ni en sus inmediaciones.
- Si tienes dudas acerca de un objeto o dispositivo concreto pregunta al profesor antes de que comience la prueba.
- El incumplimiento de esta normativa puede conllevar, entre otras, la expulsión del aula del examen sin posibilidad de realizar la prueba.

Modalidad 5

Preguntas:

1. La complejidad espacial **asintótica** de un heap (montículo):
   (a) Es mayor (necesita significativamente más memoria) si se representa mediante un vector que si se implementa mediante punteros
   (b) Es menor (necesita significativamente menos memoria) si se representa mediante un vector que si se implementa mediante punteros
   (c) Ninguna de las opciones anteriores es correcta
2. ¿Cuál de estas implementaciones ofrece menor complejidad temporal para las operaciones de inserción y búsqueda en un árbol AVL?
   (a) Implementación mediante vector
   (b) Implementación enlazada
   (c) Ambas ofrecen las mismas complejidades temporales asintóticas. La diferencia entre ambas es la complejidad espacial
3. En la operación de inserción en un árbol AVL:
   (a) Siempre se realiza al menos una rotación
   (b) Se realiza como máximo una rotación
   (c) Ninguna de las opciones anteriores es correcta
4. Decimos que un algoritmo presenta caso mejor y caso peor cuando:
   (a) Se pueden obtener tiempos de ejecución distintos para entradas distintas
   (b) Se pueden obtener tiempos de ejecución distintos para entradas de la misma talla
   (c) Se pueden obtener tiempos de ejecución distintos para entradas idénticas
5. El algoritmo heapsort:
   (a) Ordena un vector aprovechádose del hecho de que el mayor elemento de un heap máximo (o menor, en caso de un heap mínimo) es su raíz
   (b) Además del vector que se desea ordenar, necesita memoria adicional para almacenar el heap auxiliar
   (c) Ambas opciones son correctas
6. Siempre es posible reconstruir un único arbol binario si conocemos esta pareja de recorridos:
   (a) inorden y niveles
   (b) inorden y preorden
   (c) Las dos opciones anteriores son correctas

7. En los tipos abstractos de datos:
   (a) Todas las implementaciones de la misma especificación deben emplear la misma representación (vectorial, enlazada, etc.).
   (b) Una implementación puede corresponder a múltiples especificaciones distintas.
   (c) Ninguna de las opciones anteriores es correcta.
8. En el TAD cola (no confundir con cola de prioridad):
   (a) Sólo podemos insertar elementos en uno de los extremos
   (b) Sólo podemos insertar elementos en ambos extremos.
   (c) Podemos insertar elementos en cualquier posición, siempre que la cola se mantenga ordenada.
9. Imagina que tenemos un progama en el que vamos a emplear un vector en el que se van a realizar muchas inserciones al principio del mismo, y necesitamos que cada inserción se ejecute en tiempo constante. ¿Qué implementación es más adecuada?
   (a) Array estático: si conocemos de antemano el número máximo de elementos que deberán almacenarse, no será necesario redimensionar
   (b) Array dinámico: las redimensiones no son un problema porque su coste amortizado es $O(1)$
   (c) Lista enlazada: las inserciones al principio de la lista siempre se realizan en tiempo constante
10. Imagina que la biblioteca C++ STL no dispone del tipo `queue` y debes implementar el TAD cola mediante un objeto de tipo `vector`. ¿Cuál sería la complejidad de la operación `dequeue` (sacar de la cola) en la implementación más eficiente posible mediante un `vector`?
    (a) $\Theta(1)$
    (b) $\Theta(\log n)$
    (c) $\Theta(n)$
11. ¿Cuál es la complejidad en el peor caso de añadir un elemento (operación `push`) en una pila implementada como una lista enlazada?
    (a) $O(1)$
    (b) $O(\log n)$
    (c) $O(n)$

12. El operador `delete` en C++:
    (a) Debe usarse obligatoriamente para liberar la memoria ocupada por todas las variables empleadas en un programa.
    (b) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, el programa siempre acabará con una violación de segmento o *segmentation fault*.
    (c) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, se incrementará el uso de memoria del programa durante su ejecución.
13. Para implementar un grafo, es preferible emplear la representación mediante matriz frente a la representación mediante lista de adyacencia cuando:
    (a) El grafo es muy disperso (tiene pocos arcos)
    (b) El grafo es muy denso (tiene muchos arcos)
    (c) Ambas representaciones son equivalentes en cuanto a tiempos de ejecución y requisitos de memoria y se pueden usar indistintamente
14. El TAD `map` en C++ está implementado como:
    (a) Un árbol binario de búsqueda con un algoritmo que permite mantenerlo equilibrado
    (b) Una tabla hash con redispersión cerrada
    (c) Una tabla hash con redispersión abierta
15. Encontrar un arco de retroceso en el árbol extendido en profundidad de un grafo dirigido nos garantiza que:
    (a) El grafo contiene un ciclo
    (b) El grafo es conexo
    (c) Ambas opciones son correctas
16. ¿Cuál es la complejidad en el peor caso de eliminar un elemento (operación `erase`) en un vector implementado como un array?
    (a) $O(1)$
    (b) $O(\log n)$
    (c) $O(n)$
17. A la hora de calcular la complejidad temporal asintótica de una función, un bucle que siempre se ejecuta 1 000 veces y no depende del parámetro de entrada de la función:
    (a) Constituye 1 paso de programa
    (b) Constituye 1 000 pasos de programa.
    (c) Ninguna de las opciones anteriores es correcta.

18. El modificador `const` a la derecha de la declaración de un método en el fichero `.h`, como en `vector<int>categoriasMasFrecuentes(vector<int>&v) const;`, quiere decir:
    (a) Que el método puede ser invocado por objetos constantes. Si no pusiéramos el modifcador `const`, el método no podría ser invocado por objetos constantes.
    (b) Que los parámetros que se pasan al método no se modifican durante su ejecución.
    (c) Que los valores de retorno del método no se modifican.
19. Respecto a las estrategias de redispersión "hash abierto" y "hash cerrado con segunda función de hash" en la implementación del TAD conjunto mediante tabla hash:
    (a) Ambas estrategias de redispersión ofrecen la misma complejidad espacial asintótica
    (b) La estrategia "hash abierto" necesita más operaciones de *rehashing* para mantener constante la complejidad de la búsqueda
    (c) En la estrategia "hash cerrado con segunda función de hash" sólo se producen colisiones entre claves sinónimas
20. La complejidad en el peor caso del algoritmo de búsqueda binaria (buscar un elemento en un vector ordenado) en un TAD vector implementado mediante un array de C++ es:
    (a) $O(1)$
    (b) $O(\log n)$
    (c) $O(n)$

Respuestas para la modalidad **5**

1. La complejidad espacial **asintótica** de un heap (montículo):
   - (a) Es mayor (necesita significativamente más memoria) si se representa mediante un vector que si se implementa mediante punteros
   - (b) Es menor (necesita significativamente menos memoria) si se representa mediante un vector que si se implementa mediante punteros
   - [(c)] Ninguna de las opciones anteriores es correcta
2. ¿Cuál de estas implementaciones ofrece menor complejidad temporal para las operaciones de inserción y búsqueda en un árbol AVL?
   - (a) Implementación mediante vector
   - [(b)] Implementación enlazada
   - (c) Ambas ofrecen las mismas complejidades temporales asintóticas. La diferencia entre ambas es la complejidad espacial
3. En la operación de inserción en un árbol AVL:
   - (a) Siempre se realiza al menos una rotación
   - [(b)] Se realiza como máximo una rotación
   - (c) Ninguna de las opciones anteriores es correcta
4. Decimos que un algoritmo presenta caso mejor y caso peor cuando:
   - (a) Se pueden obtener tiempos de ejecución distintos para entradas distintas
   - [(b)] Se pueden obtener tiempos de ejecución distintos para entradas de la misma talla
   - (c) Se pueden obtener tiempos de ejecución distintos para entradas idénticas
5. El algoritmo heapsort:
   - [(a)] Ordena un vector aprovechádose del hecho de que el mayor elemento de un heap máximo (o menor, en caso de un heap mínimo) es su raíz
   - (b) Además del vector que se desea ordenar, necesita memoria adicional para almacenar el heap auxiliar
   - (c) Ambas opciones son correctas
6. Siempre es posible reconstruir un único arbol binario si conocemos esta pareja de recorridos:
   - (a) inorden y niveles
   - (b) inorden y preorden
   - [(c)] Las dos opciones anteriores son correctas

7. En los tipos abstractos de datos:

   (a) Todas las implementaciones de la misma especificación deben emplear la misma representación (vectorial, enlazada, etc.).

   (b) Una implementación puede corresponder a múltiples especificaciones distintas.

   [(c)] Ninguna de las opciones anteriores es correcta.

8. En el TAD cola (no confundir con cola de prioridad):

   [(a)] Sólo podemos insertar elementos en uno de los extremos

   (b) Sólo podemos insertar elementos en ambos extremos.

   (c) Podemos insertar elementos en cualquier posición, siempre que la cola se mantenga ordenada.

9. Imagina que tenemos un progama en el que vamos a emplear un vector en el que se van a realizar muchas inserciones al principio del mismo, y necesitamos que cada inserción se ejecute en tiempo constante. ¿Qué implementación es más adecuada?

   (a) Array estático: si conocemos de antemano el número máximo de elementos que deberán almacenarse, no será necesario redimensionar

   (b) Array dinámico: las redimensiones no son un problema porque su coste amortizado es $O(1)$

   [(c)] Lista enlazada: las inserciones al principio de la lista siempre se realizan en tiempo constante

10. Imagina que la biblioteca C++ STL no dispone del tipo `queue` y debes implementar el TAD cola mediante un objeto de tipo `vector`. ¿Cuál sería la complejidad de la operación `dequeue` (sacar de la cola) en la implementación más eficiente posible mediante un `vector`?

    [(a)] $\Theta(1)$

    (b) $\Theta(\log n)$

    (c) $\Theta(n)$

11. ¿Cuál es la complejidad en el peor caso de añadir un elemento (operación `push`) en una pila implementada como una lista enlazada?

    [(a)] $O(1)$

    (b) $O(\log n)$

    (c) $O(n)$

12. El operador `delete` en C++:

    (a) Debe usarse obligatoriamente para liberar la memoria ocupada por todas las variables empleadas en un programa.

    (b) Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, el programa siempre acabará con una violación de segmento o *segmentation fault*.

    [(c)] Debe usarse para liberar la memoria ocupada por los bloques que se han reservado con `new`. Si no se usa, se incrementará el uso de memoria del programa durante su ejecución.

13. Para implementar un grafo, es preferible emplear la representación mediante matriz frente a la representación mediante lista de adyacencia cuando:

    (a) El grafo es muy disperso (tiene pocos arcos)

    [(b)] El grafo es muy denso (tiene muchos arcos)

    (c) Ambas representaciones son equivalentes en cuanto a tiempos de ejecución y requisitos de memoria y se pueden usar indistintamente

14. El TAD `map` en C++ está implementado como:

    [(a)] Un árbol binario de búsqueda con un algoritmo que permite mantenerlo equilibrado

    (b) Una tabla hash con redispersión cerrada

    (c) Una tabla hash con redispersión abierta

15. Encontrar un arco de retroceso en el árbol extendido en profundidad de un grafo dirigido nos garantiza que:

    [(a)] El grafo contiene un ciclo

    (b) El grafo es conexo

    (c) Ambas opciones son correctas

16. ¿Cuál es la complejidad en el peor caso de eliminar un elemento (operación `erase`) en un vector implementado como un array?

    (a) $O(1)$

    (b) $O(\log n)$

    [(c)] $O(n)$

17. A la hora de calcular la complejidad temporal asintótica de una función, un bucle que siempre se ejecuta 1 000 veces y no depende del parámetro de entrada de la función:

    [(a)] Constituye 1 paso de programa

    (b) Constituye 1 000 pasos de programa.

    (c) Ninguna de las opciones anteriores es correcta.

18. El modificador `const` a la derecha de la declaración de un método en el fichero `.h`, como en `vector<int>categoriasMasFrecuentes(vector<int>&v) const;`, quiere decir:

    [(a)] Que el método puede ser invocado por objetos constantes. Si no pusiéramos el modifcador `const`, el método no podría ser invocado por objetos constantes.

    (b) Que los parámetros que se pasan al método no se modifican durante su ejecución.

    (c) Que los valores de retorno del método no se modifican.

19. Respecto a las estrategias de redispersión "hash abierto" y "hash cerrado con segunda función de hash" en la implementación del TAD conjunto mediante tabla hash:

    [(a)] Ambas estrategias de redispersión ofrecen la misma complejidad espacial asintótica

    (b) La estrategia "hash abierto" necesita más operaciones de *rehashing* para mantener constante la complejidad de la búsqueda

    (c) En la estrategia "hash cerrado con segunda función de hash" sólo se producen colisiones entre claves sinónimas

20. La complejidad en el peor caso del algoritmo de búsqueda binaria (buscar un elemento en un vector ordenado) en un TAD vector implementado mediante un array de C++ es:

    (a) $O(1)$

    [(b)] $O(\log n)$

    (c) $O(n)$