1. identificación del problema

El problema consiste en el manejo de datos de los jugadores de Bsketball de la FIBA, se requiere crear un programa en el cual se permita acceder a la informeción de los jugadores a traves de diferentes criterios de busqueda, ademas se busca poder hacer que esta información sea persistente, el programa debe manejar con facilidad una gran cantidad de información, en total una cantidad superior a 200.000 datos

2. Recopilación de la información

Para la recopilación de la información se han planteado las siguientes preguntas de interes:

a. ¿Qué es FIBA?

b. ¿Cuales son las caracteristicas más relevantes del los jugadores de Basketball?

c. ¿Existen otros programas, plataformas, paginas, etc , que manejen las etadisticas de los ugadores de este deporte ?

d. ¿Cómo organiza FIBA a sus jugadores (¿qué categorias exiten dentro de este juego?)?

e.

Resultados de la Busqueda:

a. ¿Qué es FIBA?

Federación Internacional de Baloncesto(FIBA), es el organismo inetrnacional que se dedica a la vijlancia y regulación de las normas de baloncesto mundial (wikipedia, 2018).

b. ¿Cuales son las caracteristicas más relevantes del los jugadores de Basketball?

el basketball es un deporte de equipo en el cual hay dos bandos cada uno con cinco jugado res, los cuales juegan en periodos llamdos cuartos de diez (es decir 40 minutos de juego divi didos en 4 partes, sin contar los 6 minutos que se dan entre cada periono), el odjetivo del jue go es ganar anotando la mayor cantidad de canastas posibles (esto consite en hacer pasar la pelota por un aro que se encuantra a 3,05 m sobre el nivel del suelo)

3. Busqueda de Ideas Creativas

4. Diseños Preliminares

para el orden de los jugadores se han fijado cinco atributos los cuales son:

a. Nombre : el nombre del Jugador

b. Altura : la altura del jugador

c. Número de canastas Hechas

d. Número de pases hechos

e. edad : edad del jugador

6. especificaciones

TAD :

|  |
| --- |
| TAD NodeABB |
| NodeAbb <next, key, father, left, right> |
| Invariante  -     ∀ |
| Operaciones Primitivas:   * search (String key): --> NodeABB * getMax(): --> NodeABB * getSucessor(): --> NodeABB * compareTo(NodeABB): --> NodeABB |

|  |
| --- |
| Search (String key) |
| “busca un objeto en elarbol”  {Pre: key != null & key != ’ ’}  {post: el odjeto u objetos con las keys iguales sera retornado (en caso de ser mas de uno se retornara la lista )} |

|  |
| --- |
| getMax() |
| “retorna el maximo valor”  {Pre: root != null (el arbol ha sido inicializado)}  {post: el mayor de todos los nodos ha sido retornado} |

|  |
| --- |
| getSucessor() |
| “retorna el menor de los hijos mayores ”  {Pre: root != null (el arbol ha sido inicializado)}  {post: el sucesor del nodo de refencia es devuelto} |

|  |
| --- |
| comparateTo(NodeABB o) |
| “compara dos nodos del arbol deacuerdo a su llave”  {Pre: root != null (el arbol ha sido inicializado) o.key != null y o.key != ’ ’ }  {post: determina cual de los dos nodos comparados es mayor o menor } |

|  |
| --- |
| TAD NodeAVL |
| NodeAVL <next, key, father, left, right, balanceFactor> |
| Invariante  -     ∀ |
| Operaciones Primitivas:   * calculateBalanceFactor(): --> NodeAVL |

|  |
| --- |
| calculateBalancefactor () |
| “calcula el factor de balanseo del nodo segun la altura de sus sudarboles”  {Pre: no aplica}  {post: el factor de balance ha sido calculado} |

|  |
| --- |
| TAD NodeBR |
| NodeBR <next, key, father, left, right, color, uncle, grandFather> |
| Invariante  -     ∀ |
| Operaciones Primitivas:   * repaint |

|  |
| --- |
| repaint() |
| “repinta o cambia el color de los nodos por el color apuesto ”  {Pre: no aplica}  {post: los nodos repiendados ivierten su color } |

|  |
| --- |
| TAD ABBTree |
| ABBTree <root, size> |
| Invariante  -     ∀ |
| Operaciones Primitivas:   * addNode(NodeABB n): --> ABBTree() * leftRotate(NodeABB actual): --> ABBTree() * rightRotate(NodeABB actual): --> ABBTree() * search(NodeABB actual): --> ABBTree() |

|  |
| --- |
| addNode() |
| “añade un nodo nuevo al arbol”  {Pre: no aplica}  {post: el tamaño del arbol aumenta en una unidad } |

|  |
| --- |
| leftRotate(NodeABB actual) |
| “rota hacia la izquierda el nodo ’actual’ ”  {Pre: no aplica}  {post: el arbol ha sido rotado } |

|  |
| --- |
| rightRotate(NodeABB actual) |
| “rota hacia la derecha el nodo ’actual’ ”  {Pre: no aplica}  {post: el arbol ha sido rotado } |

|  |
| --- |
| search(NodeABB actual) |
| “busca al nodo ’actual’ dentro del arbol ”  {Pre: no aplica}  {post: el arbol ha sido rotado } |

|  |
| --- |
| TAD AVLTree |
| ABBTree <root, size> |
| Invariante  -     ∀ |
| Operaciones Primitivas:   * balancedTree(NodeAVL newAVL, NodeAVL father): --> AVLTree |

|  |
| --- |
| balancedTree(NodeAVL newAVL, NodeAVL father) |
| “ balancea el arbol segun el factor de carga de cada nodoAVL”  {Pre: no aplica}  {post: el arbol ha sido balanceado } |