1. (20) (i) Please explain why the algorithm that uses Stack to compute the spans of stock prices is better than the algorithm that does not use stack. (15)

(ii) Discuss the difference between Vectors and Lists. (5)

1. By analyzing the following Java code,

while( !D.isEmpty() &&Q[ i ].getPrice() >= (( Quote )D.top()).getPrice() )

D.pop();

if( D.isEmpty() )

prevHigh = -1;

else

prevHigh = (( Quote )D.top()).getDay();

Q[ i ].setSpan( i - prevHigh );

D.push( Q[ i ]);

We can see that even though there is a while loop nested within a for loop, there are only n elements can be pushed into the stack in total, so the D.pop() statement can be executed n times at most. So the time complexity for this method is O(n). However, by the method without Stack O(n2) time is required. So, the method with stack is much faster.

1. Comparing Lists and vectors, the following difference can be observed:

* Lists use position to access data while Vectors use rank to access data. Therefore, by vectors the data entries can be randomly visited. But by lists the data nodes can be accessed only from the first to the last.
* By vectors, we may have some empty entries (i.e., space occupied but not used.) By lists each node is with a data. Thus, no space is wasted. In addition, the size of a list can be dynamically increased and shrunk.

2. (50) Assume that in an array A[10], we have stored ten strings: Baltimore, Rome, Seattle, Toronto, Berlin, London, Athen, Bejing, Vancouver, Winnipeg,

(i) write a method to store them in a singly linked list. (10)

(ii) write a method to store them in a doubly linked list. (10)

(iii) write a method to remove the last node from the singly linked list. (15)

(iv) write a method which is able to remove any indicated node from the doubly linked list. (15)

(i)

public class StoreSinglyLink {

static Node head = null;

static Node tail = null;

public static void main(String[] args)

{

String[] cityName = {"Baltimore", "Rome", "Seattle", "Toronto", "Berlin", "London", "Athen", "Bejing", "Vancouver", "Winnipeg"};

StoreList(cityName);

Node temp = head.getNext();

System.out.println("The Elements in the Singly Linked Lists are: ");

while(temp != tail) { System.out.print(temp.getElement()+" "); temp = temp.getNext(); }

}

public static void StoreList(Object[] ele)

{

head = tail;

for (int i = 0; i < ele.length; i++)

{

Node x = new Node();

x.setElement(ele[i]);

if (i == 0 ) {x.setNext(null); tail = x;}

else x.setNext(head);

head = x;

}

}

}

(ii)

String[] city = {"Baltimore", "Rome", "Seattle", "Toronto", "Berlin", "London", "Athen", "Bejing", "Vancouver", "Winnipeg"};

public void NodeList generateList(Object[] ele)

{

NodeList x = new NodeList();

// Store the cities in a doublely linked list

for (int i = 0; i < ele.length; i++)

{

x.insertFirst(ele[i]);

}

Return x;

}

(iiii)

public void removeLast(SinglyList list)

{

Node temp = list.header (); DNode tempPrev = temp;

if(list.isEmpty()) System.out.println(“The list is Empty”);

else {

while(temp.getNext() != list.trailer)

// Stop when next Node of temp equal trailer

{ tempPrev = temp; // use tempPrev to keep the previous Node of last Node

temp = tempPrev.getNext();}

tempPrev.setNext(trailer);

//link tempprev next Node to trailer

}

}

(iiii)

public static void removeSelect(NodeList list,Object ele)

{

DNode temp = list.header;

if(list.isEmpty()) System.out.println("The list is Empty");

else

{

while(temp != list.trailer)

{

if(temp.element().equals(ele)) list.remove(temp);

temp = temp.getNext();

}

}

}

3.(30) A breadth-first search traverses a tree as shown in Fig. 1. Please write an algorithm (not a Java program) to search a tree in the breadth-first manner by using the queue data structure to control the process.

enqueue(root);

while (the queue is not empty) do

{ x := dequeue; print(x);

let x1, x2, …, xk be the children of x;

for (i = k to 1) do {enqueue(xi);}

}