Yash Avinash Patole

Course: CS600

Homework 11

CWID: 10460520

Chapter 21

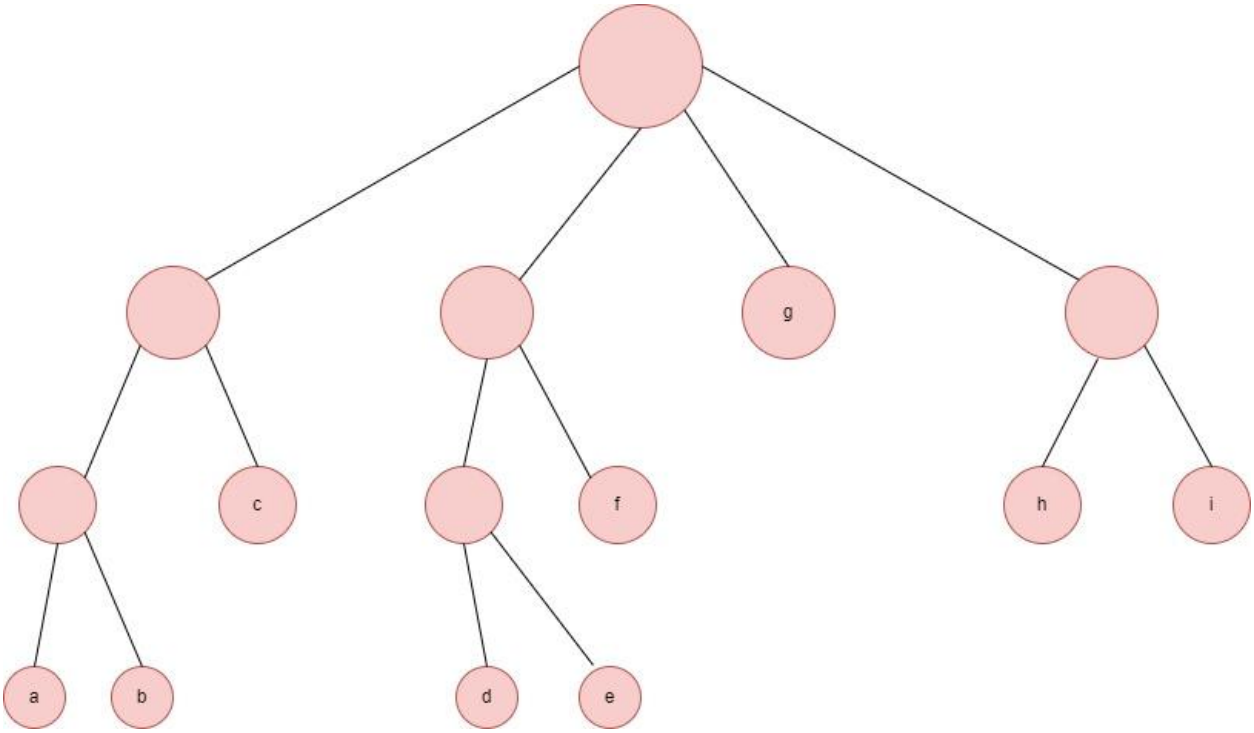
Exercise 21.5.7

Draw a quadtree for the following set of points, assuming a 16 x 16 bounding box: {(1,2), (4, 10), (14, 3), (6, 6), (3, 15), (2, 2), (3, 12), (9, 4), (12, 14)}.

Answer:

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
01		a														
02		f														
03												g			e	
04										b						
05																
06						d										
07																
08																
09				h												
10																
11																
12														i		
13																
14			c													
15																
16																

Quadtree:



Exercise 21.5.13

Answer:

```
/* Returns a hash value for the specified object. */
static int hash(Object x) {
   int h = x.hashCode();
   return h;
}

/* Returns index for hash code h. */
static int indexFor(int h, int length) {
   return h & (length-1);
}

/* Returns the value to which the supplied key is mapped in this identity hash map,
or null if no mapping for this key exists in the map.*/
public Object get(Object key) {
```

```
    int hash = hash(key);
    int i = indexFor(hash, table.length);
    Entry e = table[i];
    while (true) {
      if (e == null)
        return e;
      if (e.hash == hash && eq(k, e.key))
        return e.value;
      e = e.next;
    }
}

/**
* This hash table associates the provided value with the specified key.
* If there was already a mapping for this key in the hash table, the old value is
* replaced.
*/
public Object put(Object key, Object value) {
    int hash = hash(key);
    int i = indexFor(hash, table.length);

    for (Entry e = table[i]; e != null; e = e.next) {
      if (e.hash == hash && eq(k, e.key)) {
        Object oldValue = e.value;
        e.value = value;
        return oldValue;
      }
    }
    addEntry(hash, k, value, i);
    return null;
}

/**
* Add a new entry to the selected bucket with the specified key, value, and hash
* code. This method is responsible for resizing the table if necessary.
*/
void addEntry(int hash, Object key, Object value, int bucketIndex) {
```

```
    table[bucketIndex] = new Entry(hash, key, value, table[bucketIndex]);
    if (size++ >= threshold)
      resize(2 * table.length);
}

/**
* Removes the mapping for this key from this hash table if present.
*/
public Object remove(Object key) {
    Entry e = removeEntryForKey(key);
    return (e == null ? e : e.value);
}

/**
* Removes and returns the Hash Table entry associated with the specified key. If
* there is no mapping for this key in the Hash Table, this method returns null.
*/
Entry removeEntryForKey(Object key) {
    int hash = hash(key);
    int i = indexFor(hash, table.length);
    Entry prev = table[i];
    Entry e = prev;

    while (e != null) {
      Entry next = e.next;
      if (e.hash == hash && eq(k, e.key)) {
        size--;
        if (prev == e)
          table[i] = next;
        else
          prev.next = next;
        return e;
      }
      prev = e;
      e = next;
    }
    return e;
```

```
}

/**
* This hash table's contents are rehashes into a new array with a greater size. When
* the number of keys in this hash table reaches a certain threshold, this function is
* automatically invoked.
*/
void resize(int newCapacity) {
   Entry[] oldTable = table;
   int oldCapacity = oldTable.length;
   if (oldCapacity == MAXIMUM_CAPACITY) {
     threshold = Integer.MAX_VALUE;
     return;
   }

   Entry[] newTable = new Entry[newCapacity];
   transfer(newTable);
   table = newTable;
   threshold = (int)(newCapacity * loadFactor);
}

/**
* Transfer all entries from current table to newTable.
*/
void transfer(Entry[] newTable) {
   Entry[] src = table;
   int newCapacity = newTable.length;
   for (int j = 0; j < src.length; j++) {
     Entry e = src[j];
     if (e != null) {
       src[j] = null;
       do {
         Entry next = e.next;
         int i = indexFor(e.hash, newCapacity);
         e.next = newTable[i];
         newTable[i] = e;
         e = next;
```

```
      } while (e != null);
    }
  }
}

/* Each item contains a (key, value) pair, its hash value, and a link to the next entry
*  with the same hash value. */
class Entry {
  Object key;
  Object value;
  final int hash;
  Entry next;

  /**
  * Create new entry.
  */
  Entry(int h, Object k, Object v, Entry n) {
    value = v;
    next = n;
    key = k;
    hash = h;
  }
}
```

Exercise 21.5.27

***Answer*:**

A collection of points R 2 and a rectangular plate P with dimensions (a,/3)

Output: The bottom-right corner of P must be optimal for the objective 11 function.

Let R indicate a collection of isothetic rectangles of size (,) in which Ri is drawn.

with the i-th point in the top-left corner

Make a skeletal interval tree (T) with all members' x-coordinates in R's left and right bounds. Insert the interval corresponding to the floor's roof in T with degree zero.

In R, process the rectangles' top and bottom limits in the correct sequence. */ For each boundary, let Ii be the current border with the horizontal interval [g, r] at height h.

FIN := set of nodes from root to a node v* (fork node) such that 1 < < r;
FL set of nodes from on the path from v* to g;
FR := set of nodes from on the path from v* to r;
if I is a top boundary then
begin
if problem = min-enclosure then SET-MIN-DEGRE;
for each node V belongs to FIN do PROCESS-FIN(V);
for each node V belongs to FL do PROCESS-FL(V);
for each node V belongs FR do PROCESS-FR(v);
end;
if I is a bottom boundary then
begin
begin for each node v •.FIN do PROCESS-FIN(V); for each node v • FL do PROCESS-FL(V); for each node v • FR do PrOCESS-FR(v); if problem = max-enclosure then SET-MAX-DEGRE; end; traverse from leaf nodes a and b to the root along the search path to find the window with maximum degree and to set the max or mindegree fields and target point PROCESS-FIN(V): Begin propagating excess(v) to v's left and right children and set excess(v) to O;
Procedure PROCESS-FIN(V)begin propagate excess(v) to the left and right children of v and set excess(v) to 0; letW[x, y] be the window associated to node v with degree 5; if x < ~ < y then split Ix, y] intoIx, ~] and [~, y] with degrees 5 and 5+1 respectively; if x < r < y then split [x, y] into [x, r]and [r, y] with degrees 6+1 and 5 respectively; if [~, r] • [x, y] then split [x, y] into Ix, g], [g,r] and [r, y] with degrees 5, 5+1 and 5 respectively; end; Procedure PROCESS-FL (v)begin propagate excess(v) to the left and right children of v and set excess(v) to 0;
Let W[x, y] be the window associated to node v with degree 5; if Ix, Y] • [~,r] then begin if [g, r] corresponds to the top boundary then increase degree(v) by I; if [g, r] corresponds to the bottom boundary then decrease degree(v) by I; end; if Ix, y] overlaps [~, r] then split [Ix, y] into [Ix, ~] and [g, y] with degrees 5 and 5+1 respectively; if search path proceeds to the left of v then add (subtract) 1 to (from)

the maxdegree and excess field of its right child depending on whether [g, r] is a top (bottom) boundary; end; Procedure PROCESS-FR(v) begin Similar to processing nodes in FL; end; Procedure SET-MIN-DEGREE begin /* let current-minimum contains the minimum degree up to the current instant of time. (Pz, py) is a point having minimum degree */ for all nodes on the search path (in FIN, FL and FR) check the mindegree fields; if (mindegree < min) then begin obtain the window [a, b] having minimum degree using target pointer; A Unified Algorithm 61 if (b < c~ or h < fl) /* check boundary conditions*/ then exit else rain := mindegree; (Px,Py) := (b, h); end; end; Procedure SET-MAX-DEGREE begin the procedure is identical to SET-MIN-DEGREE;
before setting (Px,Py), the boundary conditions need not be checked; end;

Chapter 22

Exercise 22.6.7

Give a pseudocode description of the plane-sweep algorithm for finding a closest pair of points among a set of n points in the plane.

Answer:

Pseudo-code:
1. First we can know that both x axis and y axis and draw the lines.
2. Next, arrange the list of x-axis points from left to right.
3. Next, look for the nearest point on the x axis.
4. Locate the nearest pair in the x plane.
5. Next, sort the list of y axis points from right to left.
6. Next, look for the nearest point on the y axis.
7. Find the closest pair in the y plane.
8. And compare the closest pair of x plane and y plane.
9. If X plane elements is closer than y plane then display the result of x coordinate elements.
10. If Y plane elements is closer than x plane then display the result of y coordinate elements.
11. Display the closest pair of the plane.

Exercise 22.6.16

Let C be a collection of n horizontal and vertical line segments. Describe an O(n log n)-time algorithm for determining whether the segments in C form a simple polygon.

Answer:

Each vertex can be in no more than two vertex-edge visible pairings, for a total of no more than 2n vertices. Divide P into trapezoids and triangles by drawing a horizontal line segment through the interior of P between each visible pair (of either type). Each edge-edge visible pair corresponds to the bottom boundary of exactly one such trapezoid or triangle, the top boundary of which is either one- or two-line segments corresponding to vertex-edge visible pairs (in the case of a trapezoid) or a vertex that is not in any visible pairs (in the case of a triangle).

A vertex can give rise to no more than two top boundary segments of a trapezoid or one top boundary of a triangle. As a result, there are only 2n trapezoids and triangles whose bottom bounds match to edge-edge visible pairings, and hence only 2n such pairs. The second pillar of our solution is the close relationship between visibility computation and the Jordan sorting problem. The Jordan sorting issue for a basic polygon P and a horizontal line L is to sort the intersection points of OP and L by x-coordinate, given just a list of the intersections in the order in which they occur clockwise around t P as input. (The list of C's vertices is not part of the input.) have described a linear-time Jordan sorting algorithm that works for any simple curve, open or closed. This technique, which we term "the Jordan sorting algorithm," requires that OP truly cross L wherever it touches it, but it is easily adjusted to accommodate tangent points if each intersection point is designated in the input as crossing or tangent. Computing visible pairings is at least as difficult as Jordan sorting.

Algorithm:

Step 1: Given C, select a vertex v of C with no maximum or lowest y-coordinate. If no such v exists, the computation is terminated because there are no observable pairings to compute. Otherwise, consider L to be the horizontal line passing through v.

Step 2: Determine the crossing points of 0P and L along the boundary of P in the order they occur.

Step 3: Jordan sorts the intersection points and reports the visible pairs that correspond to consecutive intersection locations along L.

Step 4: Slice P along L, dividing P into a collection of sub polygons.

Step 5: Apply the algorithm recursively to each sub polygon produced in Step 4.

While the running time of this algorithm is O (n logn)-time.

Exercise 22.6.30

Answer:

SVM finds a dividing line or hyperplane between two classes of data. Because this is a classification problem, SVM will provide the algorithm for assessing whether there is a line, l, that divides the red and blue points in S. It is an algorithm that takes data as input and generates a line that separates the classes, which in this case are the red and blue labels that determine whether a person tests positive or negative for a certain ailment. The best separator is a line or hyperplane that maximizes the distance to the nearest point, often known as a 'margin.'

Let us take an example of covid 19:

S = {Cough, cold, fever, sneezing, headache}

data_dict = {'cough':1,'cold':0,'fever':1,'sneezing':0,'headache':0}

Cough, cold, fever, sneezing, headache are the attributes of the disease or features

used by an algorithm

here the labels 0 is for a person who has covid19

and label 1 is for a person who is free from covid 19

Now, splitting the dataset into training and testing datasets.

```
from sklearn.model_selection import train_test_split
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.3,random_state=109
```

```
from sklearn import svm
```

```
#Create a svm Classifier
clf = svm.SVC(kernel='linear')

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Hence, the running time of this algorithm is O(nfeatures*nsamples^3)