Spring 1997 | **Written Assignment #7: Sample Answers** | CMSC 420 David Traum

Written Assignment #7: Sample Answers

1. The basic data structure is a graph: cities are nodes, and railroad links are edges with weights. A vertex based representation will be easiest for finding the shortest path, using a varient of the shortest elementary chain algorithm for finding the shortest path. The complication is that we want to find the nearest city to a given point, so we don't just want a list of cities (vertices) - we want an efficient structure for this kind of search. Thus, we also store the vertices in a quadtree. If we choose a point quadtree, then we can just use the nearest-neighbor algorithm written for project 1, as the answer to part (c)!

   (a)
```
#define NRDIRS 4

typedef char * nameptr;
typedef enum {NW, NE, SW, SE} quadrant;

typdef struct link_s {
 nameptr CITY1;
 nameptr CITY2;
 int DIST;
} link;

typedef struct link_list {
 link FIRST;
 struct link_list NEXT;
} ll;

typedef struct node_s {
  int XCOORD;    /* X coordinate */
  int YCOORD;    /* Y coordinate */
  nameptr NAME;  /* Name */
  struct node_s * SON[NRDIRS];  /* Four sons in quadtree */
  ll LINKS;
} node;
typedef node * nodeptr;

nodeptr quadtree; /* Declare a quadtree */
```

   (b) Justification is given above. The quadtree is better than a simple list of cities, since it allows effecient pruning of nodes in a nearest city search. Since a particular point would lie in one quadrant relative to a node,a s we are searching down from the root, it always allows pruning the oppositie quadrant (which will be further than the root), and usually one or two other quadrants. Thus we can get O(log n) search time (assuming we can prune away half the points on average). A simple list of points would require O(n) time.

(c) use nearest-neighbor algorithm from Programming assignment 1.

(d) Average case should be O(log n). Assuming we don't get many new insertions or deletions, we could sort and balance the tree to insure this.

2. (a) If we just needed to find whether an item was in the database or not, the hash table will be the most efficient. We can find the record (or its absence) in constant time. Since hash tables aren't good for finding predecessors or sucessors (except for OPLH, which is not as fast), we can store this information with each record. The downside is that when we delete or insert, we have to update the predecessors and sucessors.

   The problem with this solution is that it's not easy to find the predecessor or successor to a value that's not stored in the database, since there's no way to know where that would hash to.

   A range tree would be a good solution, since we can find the item in log(n) time, and then just follow the links for the predecessors and successors.

   (b) range trees are not so good if we have to insert and delete a lot, since we'd have to rebuld the tree. However, a skip list would give us on average log(n) time, and we could use the links at the lowest level to find successors. If we also add links going the other way, we could get predecessors, as well.

   (c) A range tree would be a good solution.

3.
```
      (4  9)
     /  |  \
(1 2) (5 7 8) (10 15)
```

4.
```
        4
      /b  \r
     2      9
   /r     /b \b
  1      7     10
       /r \r    \r
      5    8     15
```

5. The values for the hash function are:

X	1	10	2	15	16
h(x)	1	3	2	1	2
(g(x)	1	4	2	3	4

(a)

loc	chain
0	
1	$1 \rightarrow 15$
2	$2 \rightarrow 16$
3	10
4	
5	
6	

(b)

loc	item	next
0		
1	1	6
2	2	5
3	10	
4		
5		
6		

(c)

0	
1	1
2	2
3	10
4	15
5	16
6	

(d)

0	
1	1
2	2
3	10
4	15
5	
6	16

(e) h(8)=1, g(8) = 2. The insertion sequence for 8 is: 1,3,5. The table of possible values is:

1	3	5
2	0	
3		

Moving 1 down to 2 is impossible, as is moving 1 down to 3. So, depending on direction, could either move 10 to 0 and 8 to 3, or 8 to 5. Either way the added cost is 2.

6. (a)

x	y
D	E
F	D
B	A
G	I
A	G
I	F
C	C
E	B
H	H

(b)

	0-4	4-8	8-12	12-16	16-20
0-4					
5-8		D		E	
8-12			A G I		
12-16		B F		C	
16-20					H

(c) (kids shown in order: NW,NE,SW,SE)

```
            A
        /  / \  \
      /   |   |  \
    B     C   D   E
  //|\   /|\\
     FG   H  I
```

(d) Use a grid from 32 to 32 to get even divisions:

```
 1234 5 6 7 8 9 0 1 2 3 4 56|7890123456789012
1    |       |       |      |
2    |       |       |      |
3    |       |       |      |
4    |       |       |      |
 ----+-------|-------+------|
5    |   |   |       | | |  |
     |   |   |       |-+-|  |
6    |   |   |       | |E|  |
     |---+---|       |---+--|
7    |D| |   |       |   |  |
     |-+-|   |       |   |  |
8    | | |   |       |   |  |
 ------------+--------------|
9    |   |   | | |   |      |
     |   |   |-+-|   |      |
0    |   |   | |A|I  |      |
     |---+---|---+---|      |
1    |   | |G|   |   |      |
     |   |-+-|   |   |      |
2    |F  | | |   |   |      |
 ----+-------|-------+------|
3    |   |   |   | | |      |
     |   |   |   |-+-|      |
4    |   |   |   | |C|      |
     |---+---|---+---|      |
5    | |B|   |   |   |      |
     |-+-|   |   |   |      |
6    | | |   |   |   |      |H
----------------------------+----------------
7                           |
8                           |
9                           |
0                           |
...                         |
2                           |
```

(e)

```
 12345|67 890|12 345|67890
1     |      |      |
2     |      |      |
3     |      |      |
4     |      |      |
5     |      |      |
------+------|------+------
6     |      |  | E |
7    D|      |  |   |
      |      |--+---|
8     |      |  |   |
9     |      |  |   |
0     |     A|I |   |
-------------+------------
1     |  |G  |      |
2    F|  |   |      |
      |--+---|      |
3     |  |   |      |
4     |  |   | C    |
5     |B |   |      |
------+------|------+------
6     |      |      | H
7     |      |      |
8     |      |      |
9     |      |      |
0     |      |      |
```

(f)

```
                 A         (x-test)
               /   \
              B     C      (y-test)
             /     / \
            D     E   H    (x-test)
           / \   /
          F   G  I
```

(g)

```
 12345 67 890 12 345 67890
1     .      |      .
2     .      |      .
3     .      |      .
4     .      |      .
5     .      |      .
6     .      |    E .
7    D.      |      .
8     .      |      .
9     .      |      .
0     .     A|I     .
 ------------|-------------
1     |   G  |      .
2    F|      |      .
3     |      |      .
4     |      | C    .
5     |B     |      .
6     |      |      . H
7     |      |      .
8     |      |      .
9     |      |      .
0     |      |      .
```

(h)

point	x	x bin	y	y bin	k bin	k
A	10	01010	10	01010	0011001100	204
B	6	00110	15	01111	0001111101	125
C	12	01100	14	01110	0011110100	244
D	5	00101	7	00111	0000110111	55
E	14	01110	6	00110	0010111100	188
F	5	00101	12	01100	0001110010	114
G	8	01000	11	01011	0011000101	197
H	17	10001	16	10000	1100000010	770
I	11	01011	10	01010	0011001110	206

(i) Assuming bucket size of 2. Buckets are shown as columns, each row shows the hash table after that step. Overflow buckets are shown in parentheses.

0	1	2	3	4	5	6
A						
A B						
A	B					
A C	B					
A C	B					
A C	B D					
A C (E)	B		D			
A C (E)	B	F	D			
A C (E)	B G	F	D			
	B G	F	D	A C (E)		
		F H	D	A C (E)	B G	
		F H (I)	D	A C (E)	B G	

(j) Just reverse the 10 bits in k bin of part h (e.g., B is 1011111000 = 760), and do the same operations as for linear hashing.

(k) Range trees and range priority trees are too hard to draw here. For the range tree. In the outer dimension, use X values to make a range tree (binary tree with all data points in leaves, and linked together). At each node, also make a range tree using the Y values, including all the data points that are descendants of the node. If there is not an even power of two, just make some nodes to the right have only one child.

(l) for the priority range tree, make a range tree with the y values on the outside. Then, for each node which has non-leaf children, associate a priority seach tree with each of its children, containing all the leaves that are descendants of that child: for the left child make a normal priority search tree, with a heap of highest values. For the right child, make an inverse priority search tree, with heap of lowest values.