PES UNIVERSITY

Department of Computer Science and Engineering

UE19CS251: Design and Analysis of Algorithms (4-0-0-4-4)

Unit 4: Space and Time Tradeoffs

Question and Answers

1. Write an algorithm for Comparison counting.

```
ALGORITHM ComparisonCountingSort(A[0..n − 1])
    //Sorts an array by comparison counting
    //Input: An array A[0..n − 1] of orderable elements
    //Output: Array S[0..n − 1] of A's elements sorted in
    nondecreasing order
    for i ←0 to n − 1 do Count[i]←0
    for i ←0 to n − 2 do
        for j ←i + 1 to n − 1 do
            if A[i]<A[j ]
                Count[j ]←Count[j ]+ 1
            else Count[i]←Count[i]+ 1
    for i ←0 to n − 1 do S[Count[i]]←A[i]
    return S
```

2. Show how comparison counting method sorts the list: 55,2,19,11,35,22,1,24

Array[0..7]		55	2	19	11	35	22	1	24
Initial	Count[]	0	0	0	0	0	0	0	0

After Pass i=0	Count[]	7	0	0	0	0	0	0	0
After Pass i=1	Count[]		1	1	1	1	1	0	1
After Pass i=2	Count[]			3	1	2	2	0	2
After Pass i=3	Count[]				2	3	3	0	3
After Pass i=4	Count[]					6	3	0	3
After Pass i=5	Count[]						4	0	4
After Pass i=6	Count[]							0	5
Array S[0…7]		1	2	11	19	22	24	35	55

3. Discuss the time complexity of Comparison counting sort and Distribution Counting sorting methods.

The time complexity **of Comparison counting sort** is quadratic because the algorithm considers all the different pairs of an *n*-element array, whereas the time complexity **of *Distribution Counting Sort*** *method is linear.*

4. Apply Horspool's algorithm to search for the pattern BAOBAB in the text BESS NEW ABOUT BAOBABS. Also find the total number of comparisons made.

Shift Table:

Character(c)	A	B	C	D	.....	O	......	Z	_
Shift(c)	1	2	6	6	6	3	6	6	6

B	E	S	S		K	N	E	W		A	B	O	U	T		B	A	O	B	A	B	S
B	A	O	B	A	B																	
						B	A	O	B	A	B											
										B	A	O	B	A	B							
																B	A	O	B	A	B	

Number of Comparisons : 11

5. Summarize the Boyer-Moore Algorithm.

The Boyer-Moore algorithm

Step 1: For a given pattern and the alphabet used in both the pattern and the
text, construct the bad-symbol shift table.

Step 2: Using the pattern, construct the good-suffix shift table.

Step 3: Align the pattern against the beginning of the text.

Step 4: Repeat the following step until either a matching substring is found or the pattern reaches beyond the last character of the text. Starting with the last character in the pattern, compare the corresponding characters in the pattern and the text until either all m character pairs are matched(then stop) or a mismatching pair is encountered after $k \geq 0$ character pairs are matched successfully. In the latter case, retrieve the entry $t_1(c)$ from the c's column of the bad-symbol table where c is the text's mismatched character. If $k > 0$, also retrieve the corresponding d_2 entry from the good-suffix table. Shift the pattern to the right by the number of positions computed by the formula

$$d = d_1 \text{ if } k = 0, \text{ or } \max\{d_1, d_2\} \text{ if } k > 0,$$

$$\text{where } d_1 = \max\{t_1(c) - k, 1\}$$

Shifting by the maximum of the two available shifts when ***k* > 0** is quite logical.
The two shifts are based on the observations—the first one about a text's mismatched character, and the second one about a matched group of the pattern's rightmost characters—that imply that shifting by less than d_1 and d_2 characters, respectively, cannot lead to aligning the pattern with a matching substring in the text. Since we are interested in shifting the pattern as far as possible without missing a possible matching substring, we take the maximum of these two numbers.

6. Define minimum cost spanning tree

A spanning tree of an undirected connected graph is its connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph. If such a graph has weights assigned to its edges, a minimum spanning tree is its spanning tree of the smallest weight, where the weight of a tree is defined as the sum of the weights on all its edges.

7. Define the following functions:

a) makeset(x),

b) find(x),

c) union(x,y).

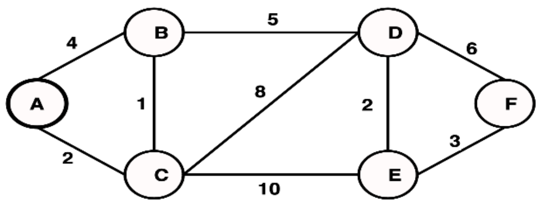
Solution:

makeset(x) creates a one-element set {x}. It is assumed that this operation can be applied to each of the elements of set S only once.

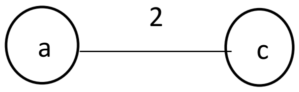
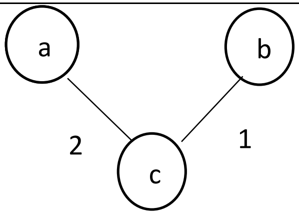
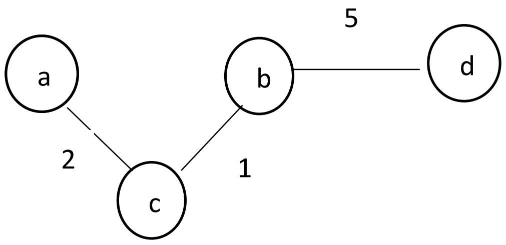
find(x) returns a subset containing x.

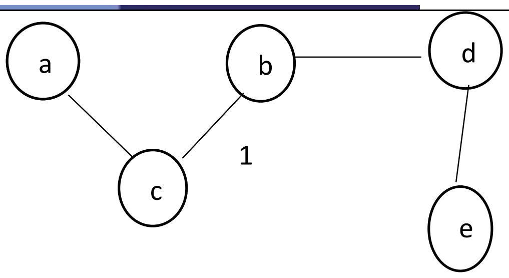
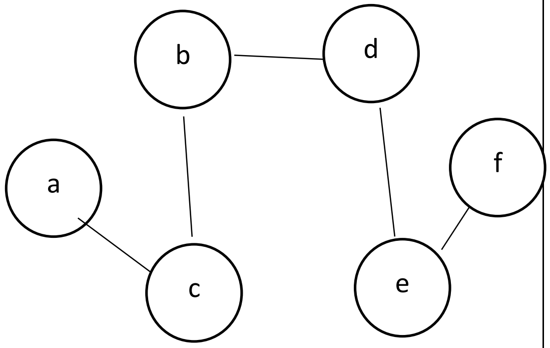
union(x, y) constructs the union of the disjoint subsets S_x and S_y containing x and y, respectively, and adds it to the collection to replace S_x and S_y, which are deleted from it.

8. Apply Dijkstra's algorithm to find single source shortest paths for the following graph taking vertex 'A' as source.



Solution:

Tree Vertices	Remaining Vertices	Illustration
a(-, 0)	b(a,4), c(a,2), d(-,∞), e(-,∞), f(-,∞)	
c(a,2)	b(c, 2+1), d(c,2+8), e(c, 2+10), f(-,∞)	
b(c,3)	d(b, 3+5), e(c,12), f(-,∞)	

d(b,8)	e(d, 8+2), f(d,8+6)	
e(d,10)	f(e, 10+3)	
f(e,13)		

Shortest Path:

a-c-b :3

a-c : 2

a-c-b-d : 8

a-c-b-d-e : 10

a-c-b-d-e-f :13

9.