CSCE 221 Cover Page
Homework Assignment #3
Due April 20 at 23:59 pm to eCampus

First Name Jialu Last Name Zhao UIN 426000712

User Name zhaojialu123 E-mail address zhaojialu123@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more on Aggie Honor System Office website: `http://aggiehonor.tamu.edu/`

Type of sources				
People				
Web pages (provide URL)				
Printed material				
Other Sources				

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.
On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Your Name Jialu Zhao Date 04/10/2017

Homework 3 (100 points)

due April 20 at 11:59 pm to eCampus.

Write clearly and give full explanations to solutions for all the problems. Show all steps of your work.

Reading assignment.

- Heap and Priority Queue, Chap. 8
- Graphs, Chap. 13

Problems.

1. (10 points) R-8.7 p. 361

   An airport is developing a computer simulation of air-traffic control that handles events such as landings and takeoffs. Each event has a *time-stamp* that denotes the time when the event occurs. The simulation program needs to efficiently perform the following two fundamental operations:

   - Insert an event with a given time-stamp (that is, add a future event)
   - Extract the event with smallest time-stamp (that is, determine the next event to process)

   Which data structure should be used for the above operations? Why? Provide big-oh asymptotic notation for each operation.

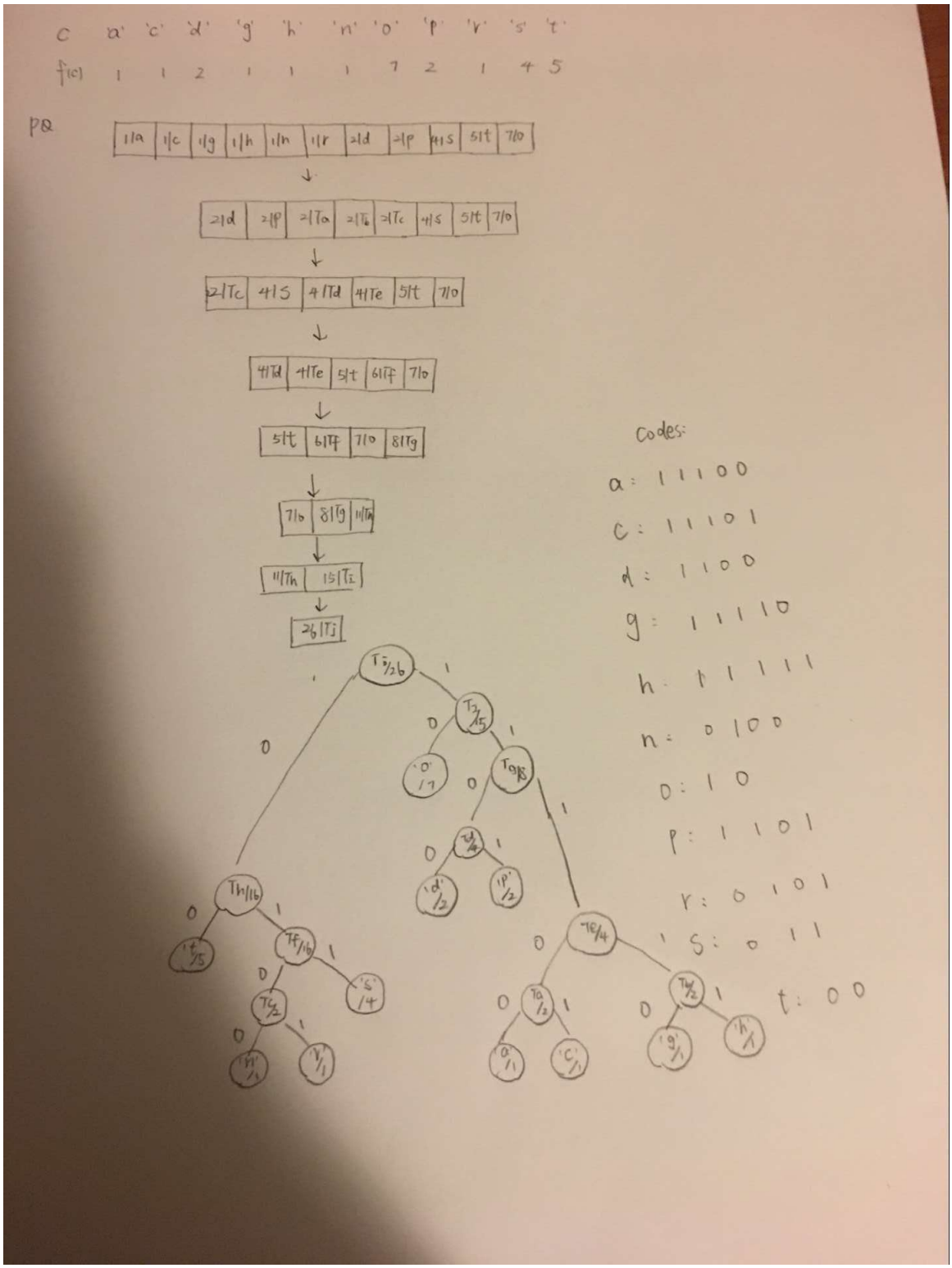
   Solution:

   If we want to have these two questions, we need to use priority queue to do this. We should use Binary Heap data structure used for above operations to implement priority queue. Because both insert function and search function will be done in O(logn) which is most efficient. The key can be time-stamp.

2. (10 points) R-12.14 p. 588

   Draw the frequency array. Use the minimum priority queue based on sorted array to build the Huffman tree for the string below. What is the code for each character and the compression ratio for this algorithm?

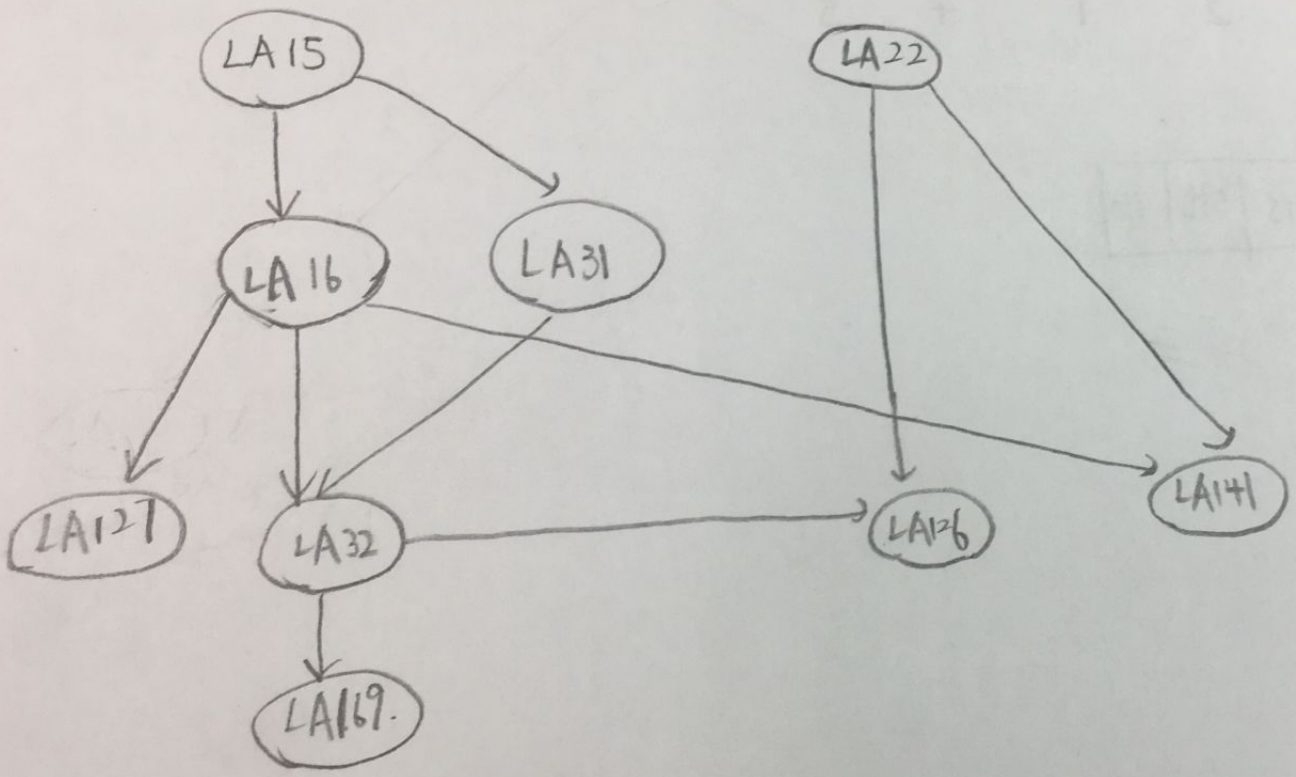
   ‘‘`dogs do not spot hot pots or cats`’’.



compression ratio: (5*4+6*4+4*3+2*12)/26*8=38.46%

3. (10 points) R-13.5, p. 654

R 13.5

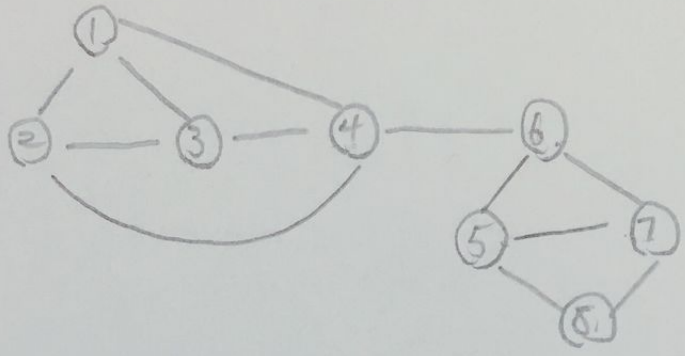


Sequence: LA15, LA22, LA16, LA31, LA127, LA32, LA126, LA141, LA16

4.(10 points) R-13.7, p. 655

R-13.7

a)



b) DFS:

1-2-3-4-6-5-8-7

c) BFS:

1-2-3-4-6-5-7-8

5.(10 points) R-13.8, p. 655

Would you use the adjacency list structure or the adjacency matrix structure in each of the following cases? Justify your choice.

a. The graph has 10,000 vertices and 20,000 edges, and it is important to use as little space as possible.

solution:

(1) Adjacency List needs: 10000(vertices) + 20000(edges)= 30000 (entries)

(2) Adjacency Matrix needs: 10000(vertices)*10000(vertices)=100000000(entries)

So we choose to use adjacency list structure.

b. The graph has 10,000 vertices and 20,000,000 edges, and it is important to use as little space as possible.

(1) Adjacency List needs: 10000(vertices) + 20000000(edges)= 20010000 (entries)

(2) Adjacency Matrix needs: 10000(vertices)*10000(vertices)=100000000(entries)

So we choose to use adjacency list structure.

c. You need to answer the query isAdjacentTo as fast as possible, no matter how much space you use.

Solution:

If we use adjacency matrix, we can answer the query isAdjacentTo in constant time, we can see directly from the matrix. But if we use adjacency list, we need to traverse the whole list to answer this query which is not fast. So we choose to use adjacency matrix structure.

6.(10 points) R-13.16, p. 656

```
Algorithm:
Dijkstra(G,w,s):
for(each vertex){
d[v]=∞;
p[v]=NULL;
}
d[s]=0;
built a tree T which root is s // built a tree
initialize PQ to have all v
while(!PQ.isEmpty()){
u=min(PQ)
if(u!=v)
add edge from u to v in the tree T // add edges
for(each vertex adj to u){
if(d[v]>d[u]+w(u,v)){
d[v]=d[u]+w(u,v);
p[v]=u;
}
}
}
```

7.(10 points) R-13.17, p. 656
Using Kruskal's algorithm:

2 - 13.17

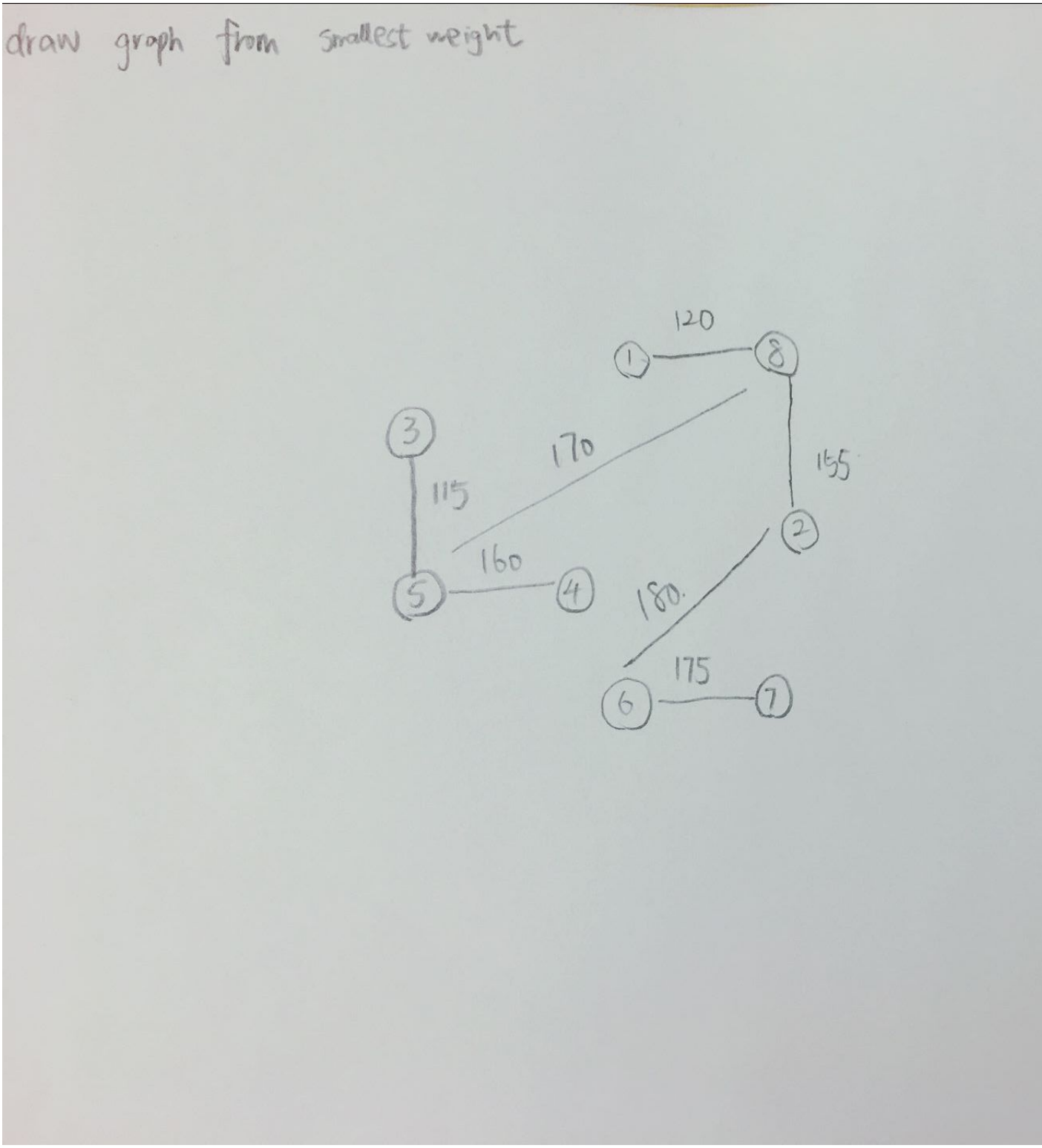
Edge List

IVI	Nodes	weight
1	1.2	240 ✓
2	1.3	210 ✓
3	1.4	340 ✓
4	1.5	280 ✓
5	1.6	200 ✓
6	1.7	345 ✓
7	1.8	120 ✓
8	2.3	265 ✓
9	2.4	175 ✓
10	2.5	215 ✓
11	2.6	180 ✓
12	2.7	185 ✓
13	2.8	155 ✓
14	3.4	260 ✓
15	3.5	115 ✓
16	3.6	350 ✓
17	3.7	435 ✓
18	3.8	195 ✓
19	4.5	160 ✓
20	4.6	330 ✓
21	4.7	295 ✓
22	4.8	230 ✓
23	5.6	360 ✓
24	5.7	400 ✓
25	5.8	170 ✓
26	6.7	175
27	6.8	205 ✓

Sort them by weight

Edge List

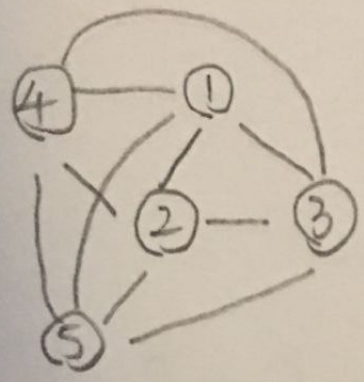
IVI	Nodes	weight
1	3.5	115
2	1.8	120
3	2.8	155
4	4.5	160
5	5.8	170
6	6.7	175
7	2.4	175
8	2.6	180
9	2.7	185
10	3.8	195
11	1.6	200
12	6.8	205
13	1.3	210
14	2.5	215
15	4.8	230
16	1.2	240
17	3.4	260
18	2.3	265
19	1.5	280
20	4.7	295
21	7.8	305
22	4.8	330
23	1.4	340
24	1.7	345
25	3.6	350
26	5.6	360
27	5.7	400
28	3.7	435



8.(10 points) R-13.31, p. 657

Solution: A depth-first search tree of a complete graph look like a simple graph, and a simple graph is a graph with no self-loops and mutliple edges,for example:

R 13.31



DFS

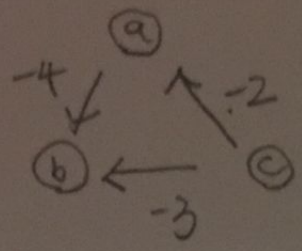
① → ② → ③ → ④ → ⑤

9.(10 points) C-13.10, p. 658

If we want to get O(m+n) algorithm, we need to avoid backtracking which means we need to visit more edges when when get some nodes. So it looks like DFS, we can touch every edges when we get a node which cost constant number of times before we go back to this node. And then we can visit next node and their edges until there is no node we can visit. In the end, we got the time complexity is O(m+n).

10.(10 points) C-13.15, p. 659

C-13.15



find the shortest path from c → b using D

iter	P[v]	d[a]	d[b]	d[c]
0	–	∞	∞	0
1	c	-2(c)	-3(c)	+
2	b	–	-3(c)	–

we get -3 which correct answer i