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

First Name Rong Last Name Xu UIN 928009312

User Name abby-xu E-mail address rongx0915@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 Rong Xu Date 11/21

Homework 3 (100 points)

due November 23 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:

- Balanced Binary Search Trees
- Skip Lists
- Hash Tables
- Heap and Priority Queue
- Graphs

Problems.

1. (10 points) For the following statements about red-black trees, provide a justification for each true statement and a counterexample for each false one.

   (a) A subtree of a red-black tree is itself a red-black tree.

   i. False

   ii. counterexample: if there is a red node, then the subtree of this node is not a red-black tree, because a red-black tree must have a black root node.

   (b) The sibling of an external node is either external or red.

   i. True

   ii. justification: We could use contradiction to prove. Suppose that there is an external node has a black internal sibling and h is the depth of the external node. Then the internal sibling would be at least h+1 which violates the depth rule of red-black tree, becasue all external nodes nust have the same black height.

   (c) There is a unique 2-4 tree associated with a given red-black tree.

   i. True

   ii. justification: Set there is a red-black tree T and an unique 2-4 tree T' associated with T. Since a node in T that contains two red children is represented as a 4-node in T; a node in T contains one red child is represented as a 3-node in T'; and a node with no red children is represented as the 2-node in T', the T' is uniquely associated with a given red-black tree.

   (d) There is a unique red-black tree associated with a given 2-4 tree.

   i. False

   ii. counterexample: The 3-node in a 2-4 tree could have two different representations in the red-black tree.

2. (10 points) Modify this skip list after performing the following series of operations: `erase(38)`, `insert(48,x)`, `insert(24,y)`, `erase(42)`. Provided the recorded coin flips for `x` and `y`.

$-\infty$	—	—	—	—	—	$+\infty$
$-\infty$	—	17	—	—	—	$+\infty$
$-\infty$	—	17	—	—	42	$+\infty$
$-\infty$	—	17	—	—	42	$+\infty$
$-\infty$	12	17	—	38	42	$+\infty$
$-\infty$	12	17	20	38	42	$+\infty$

To erase 38, we will have:

$-\infty$	—	—	—	—	$+\infty$
$-\infty$	—	17	—	—	$+\infty$
$-\infty$	—	17	—	42	$+\infty$
$-\infty$	—	17	—	42	$+\infty$
$-\infty$	12	17	—	42	$+\infty$
$-\infty$	12	17	20	42	$+\infty$

To insert 48, we will have (head -> tail) x = 1 (1+1=2):

$-\infty$	—	—	—	—	—	$+\infty$
$-\infty$	—	17	—	—	—	$+\infty$
$-\infty$	—	17	—	42	—	$+\infty$
$-\infty$	—	17	—	42	—	$+\infty$
$-\infty$	12	17	—	42	48	$+\infty$
$-\infty$	12	17	20	42	48	$+\infty$

To insert 24, we will have (head -> head -> tail) y = 2 (2+1=3):

$-\infty$	—	—	—	—	—	—	$+\infty$
$-\infty$	—	17	—	—	—	—	$+\infty$
$-\infty$	—	17	—	—	42	—	$+\infty$
$-\infty$	—	17	—	24	42	—	$+\infty$
$-\infty$	12	17	—	24	42	48	$+\infty$
$-\infty$	12	17	20	24	42	48	$+\infty$

To erase 42, we will have:

$-\infty$	—	—	—	—	—	$+\infty$
$-\infty$	—	17	—	—	—	$+\infty$
$-\infty$	—	17	—	—	—	$+\infty$
$-\infty$	—	17	—	—24–	—	$+\infty$
$-\infty$	12	17	—	—24–	48	$+\infty$
$-\infty$	12	17	20	24	48	$+\infty$

3. (10 points) Draw the 17-entry hash table that results from using the has function: $h(k) = ((3k + 5) \bmod 11)$, to hash the keys: 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5, assuming collisions are handled by double hashing using the secondary hash function: $h_s(k) = (7 - (k \bmod 7))$.

(a)

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Elements	13	94			39	44			12	16	20	88		23	11	5	

(b) $h(12) = (3(12) + 5)\%11 = 41\%11 = 8$ => put 12 on 8th

(c) $h(44) = (3(44) + 5)\%11 = 137\%11 = 5$ => put 44 on 5th

(d) $h(13) = (3(13) + 5)\%11 = 44\%11 = 0$ => put 13 on 0th

(e) $h(88) = (3(88) + 5)\%11 = 269\%11 = 5$ => collision with 44 on 5th

i. $h(88, 0) = h(88)\%17 = 5\%17 = 5$ => collision with 44 on 5th

ii. $h(88, 1) = (h(88) + h_s(88))\%17 = (5 + (7 - 88\%7))\%17 = 8\%17 = 8$ => collision with 12 on 8th

iii. $h(88, 2) = (h(88) + 2h_s(88))\%17 = (5 + 6)\%17 = 11\%17 = 11$ => put 88 on 11th

(f) $h(23) = (3(23) + 5)\%11 = 74\%11 = 8$ => collision with 12 on 8th

i. $h(23, 0) = h(23)\%17 = 11\%17 = 11$ => collision with 88 on 11th

ii. $h(23, 1) = (h(23) + h_s(23))\%17 = (8 + (7 - 23\%7))\%17 = 13\%17 = 13$ => put 23 on 13th

(g) $h(94) = (3(94) + 5)\%11 = 287\%11 = 1$ => put 94 on 1st

(h) $h(11) = (3(11) + 5)\%11 = 38\%11 = 5$ => collision with 44 on 5th

i. $h(11, 0) = h(11)\%17 = 5\%17 = 5$ => collision with 44 on 5th

ii. $h(11, 1) = (h(11) + h_s(11))\%17 = (5 + (7 - 11\%7))\%17 = 8\%17 = 8$ => collision with 12 on 8th

iii. $h(11, 2) = (h(11) + 2h_s(11))\%17 = (5 + 6)\%17 = 11\%17 = 11$ => collision with 88 on 11th

iv. $h(11, 3) = (h(11) + 3h_s(11))\%17 = (5 + 9)\%17 = 14\%17 = 14$ => put 11 on 14th

(i) $h(39) = (3(39) + 5)\%11 = 122\%11 = 1$ => collision with 94 on 1st

i. $h(39, 0) = h(39)\%17 = 1\%17 = 1$ => collision with 94 on 1st

ii. $h(39, 1) = (h(39) + h_s(39))\%17 = (1 + (7 - 39\%7))\%17 = 4\%17 = 4$ => put 39 on 4th

(j) $h(20) = (3(20) + 5)\%11 = 65\%11 = 10$ => put 20 on 10th

(k) $h(16) = (3(16) + 5)\%11 = 53\%11 = 9$ => put 16 on 9th

(l) $h(5) = (3(5) + 5)\%11 = 20\%11 = 9$ => collision with 16 on 9th

i. $h(5, 0) = h(5)\%17 = 9\%17 = 9$ => collision with 16 on 9th

ii. $h(5, 1) = (h(5) + h_s(5))\%17 = (9 + (7 - 5\%7))\%17 = 11\%17 = 11$ => collision with 88 on 11th

iii. $h(5, 2) = (h(5) + 2h_s(5))\%17 = (9 + 4)\%17 = 13\%17 = 13$ => collision with 23 on 13th

iv. $h(5, 3) = (h(5) + 3h_s(5))\%17 = (9 + 6)\%17 = 15\%17 = 15$ => put 5 on 15th

4. (10 points) 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 a smallest time-stamp (that is, determine the next event to process)

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

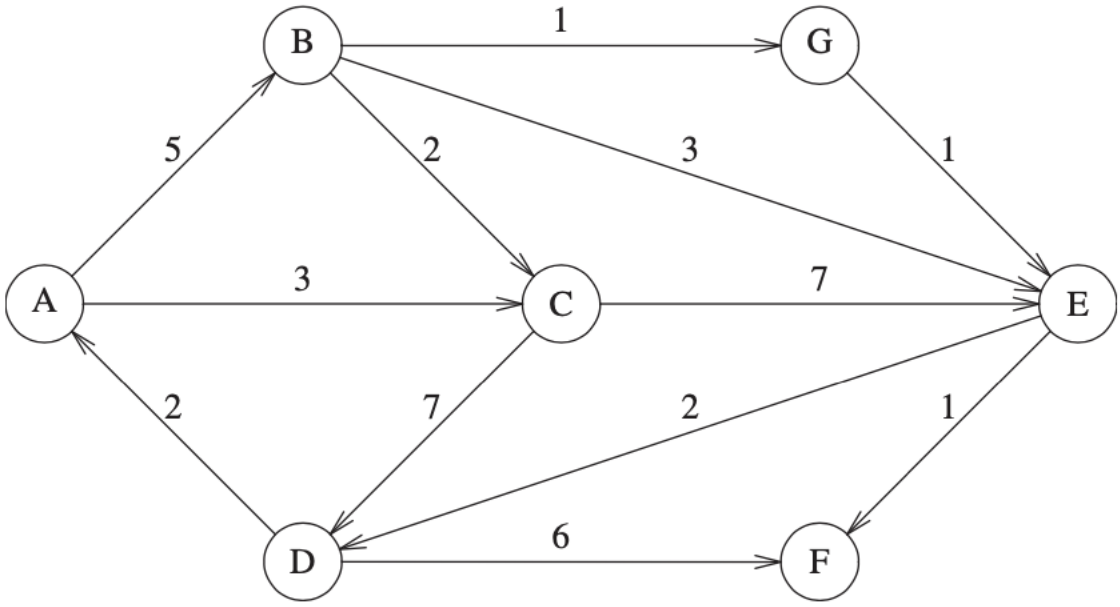
   (a) I think in this case, we can use the binary heap data structure to implement.

   (b) Becasuse we can use the time-stamp of each flight as the key to store each event into the priority queue. Besides, using the binary heap will have more efficient time complexity.

   (c) The big-O of the insert operation is $O(logn)$

   (d) The big-O of extracting the event with a smallest time-stamp is as same as min() function in binary heap, which is $O(1)$

5. (15 points) Find the shortest path from D to all other vertices for the graph below.



(a)

(b) Code for find the shortest path with the Dijkstra's algorithm(using the vector data structure)

```
#include <iostream>
#include <vector>
using namespace std;
const int INF = 1000000000;
void Dijkstra(int n, int s, vector<vector<int>> G, vector<bool>& vis, vector<int>& d, vector<int>& pre) {
        fill(d.begin(), d.end(), INF);
        for (int i = 0; i < n; ++i)
                pre[i] = i;
        d[s] = 0;
        for (int i = 0; i < n; ++i) {
                int u = -1;
                int MIN = INF;
                for (int j = 0; j < n; ++j) {
                        if (vis[j] == false && d[j] < MIN) {
                                u = j;
                                MIN = d[j];
                        }
                }
                if (u == -1)
                        return;
                vis[u] = true;
                for (int v = 0; v < n; ++v) {
                        if (vis[v] == false && d[u] + G[u][v] < d[v]) {
                                d[v] = d[u] + G[u][v];
                                pre[v] = u;
                        }
                }
        }
}

int main() {
        int n = 7;
        vector<vector<int>> G = {
                {0,5,3,INF,INF,INF,INF},
                {INF,0,2,INF,3,INF,1},
                {INF,INF,0,7,7,INF,INF},
                {2,INF,INF,0,INF,6,INF},
                {INF,INF,INF,2,0,1,INF,},
                {INF,INF,INF,INF,INF,0,INF},
                {INF,INF,INF,INF,1,INF,0}};
        vector<bool> vis(n);
        vector<int> d(n);
        vector<int> pre(n);
        Dijkstra(n,3,G,vis,d,pre);
        for (auto x : d)
```

```
                cout << x << " ";
        cout << endl;
        return 0;
}
```

(c)

```
~ » /Users/abby/Desktop/2020_fall/CSCE221_511/hw3/Q5_Dijkstra
2 7 5 0 9 6 8

[Process completed]_
```

(d) Code by using priority queue

```
void Graph::shortestPath(int src) {
        priority_queue< iPair, vector <iPair> , greater<iPair> > pq;
        vector<int> dist(V, INF);
        pq.push(make_pair(0, src));
        dist[src] = 0;
        while (!pq.empty()) {
                int u = pq.top().second;
                pq.pop();
                list< pair<int, int> >::iterator i;
                for (i = adj[u].begin(); i != adj[u].end(); ++i) {
                        int v = (*i).first;
                        int weight = (*i).second;
                        if (dist[v] > dist[u] + weight) {
                                dist[v] = dist[u] + weight;
                                pq.push(make_pair(dist[v], v));
                        }
                }
        }
        printf("Vertex Distance from Source\n");
        for (int i = 0; i < V; ++i)
                printf("%d \t\t %d\n", i, dist[i]);
}
```

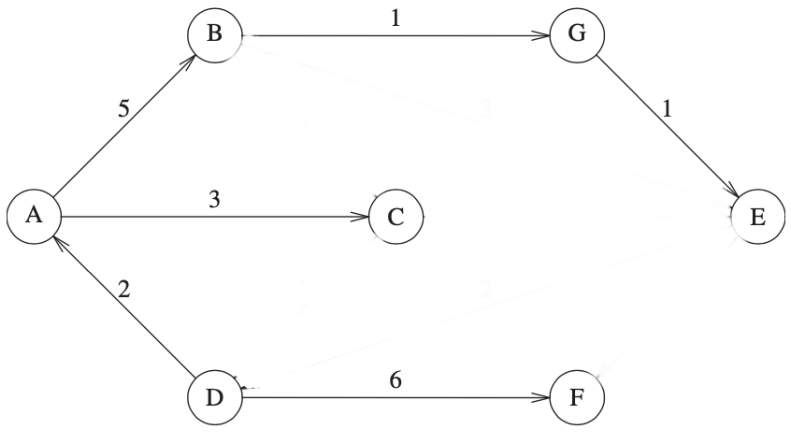
(e) Illustrate the minimum priority queue at each iteration Dijkstra's algorithm.

(f)

iteration	P[v]	d[A]	d[B]	d[C]	d[D]	d[E]	d[F]	d[G]
0	—	$+\infty$	$+\infty$	$+\infty$	0	$+\infty$	$+\infty$	$+\infty$
1	D	2[D]	$+\infty$	$+\infty$	—	$+\infty$	$+\infty$	$+\infty$
2	A	—	7[B]	5[A]	—	$+\infty$	—	$+\infty$
3	C	—	—	—	—	12[C]	—	$+\infty$
4	F	—	—	—	—	—	—	$+\infty$
5	B	—	—	—	—	10[B]	—	8[B]
6	G	—	—	—	—	9[G]	—	—
7	E	—	—	—	—	—	—	—

(g) Draw the Shortest Path Tree.

i.



(h) What is the running time of the Dijkstra's algorithm under the assumption that the graph is implemented based on an adjacency list and the minimum priority queue is implemented based on a binary heap?

i. Adding all n vertex of the graph to a binary min-heap takes $O(|n|)$ time.

ii. If we using the hash map to organize vertex, with constant time of the put operation, then removing or adding a vertex to binary min-heap will be $O(log|n|)$ time. But this only happens when the vertex at most once for each of its neighbirs. Thus, this will happen at most m times, so the total of all priority value updates takes $O(|m|log|n|)$.

iii. Since we use the adjacency matrix of edge weights, then we can access edge weights in constant time. Calculating a vertex's updated priority value a constant time operation. So all updated calculations take a total of $O(|m|)$ time.

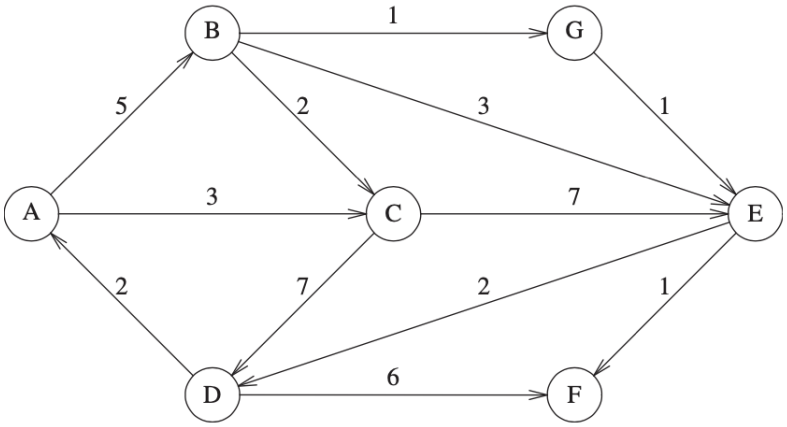
iv. Since each vertex is removed from the fringe exactly once, and never readded to the fringe. Dijkstra's algorithm only removes from the priority queue $|m|$ times, and each removal takes $O(log|n|)$ time for a total of $O(|n|log|n|)$ time for all vertex removals.

v. Checking whether the priority queue is empty is a constant time operation and happens $O(|n|)$ times. Thus, all is_empty() will take a total of $O(|n|)$ time.

vi. Iterating through a vertex;s neighbors can be done in time proportional to that vertex's degree with an adjacency list. Thus, iterating over all vertex's neighbors will take $O(|m|)$ times.

vii. By adding the operations stated above, we have the running time for Dijkstra's algorithm using a binary min-heap as a priority queue is $O((|m| + |n|)log|n|)$.

6. (15 points) Find the shortest unweighted path from D to all other vertices for the graph below. You can measure the distance from D by number of edges.



(a)

(b) Which graph algorithm can solve the problem?

i. I used the BFS to solve this question

ii. From the BFS we could have:

D -> A -> F
A -> B -> C
B -> C -> G -> E

iii. C -> E -> D
G -> E
E -> F -> D
F -> null

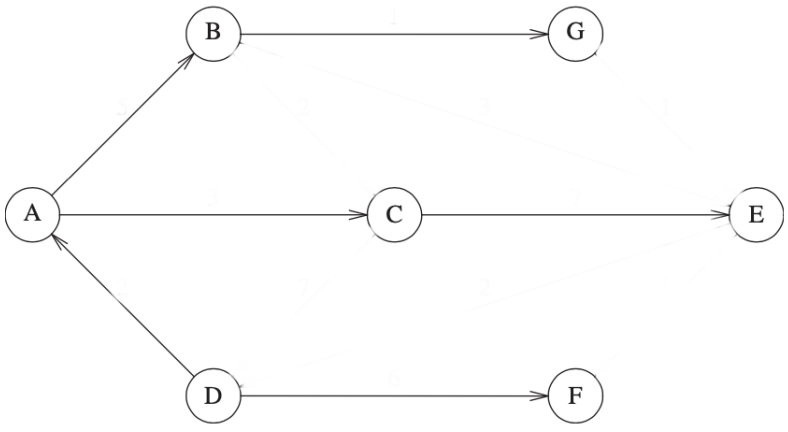
iv. The BFS queue will be:

D
A F

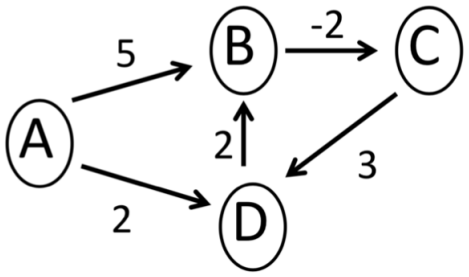
v. F B C
B C
C G E

(c) Draw the Shortest Path Tree.

i.



7. (10 points) Apply the Dijkstra's algorithm to find the shortest path from the vertex A to all the vertices in the graph below. Does the algorithm return a correct output? Justify your answer using the Dijkstra's Theorem.



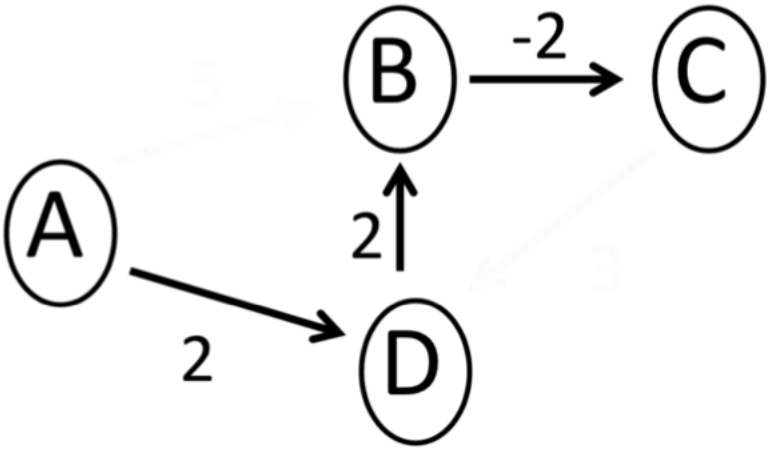
In this case, the algorithm return a correct output, by the result showing below. Basically, Dijkstra's algorithm may fial on certain graphs with negative edge weights, but it's also possible to be successful. For the Dijkstra's algorithm, having a negative cycle will make a problem for finding the shortest path of the graph. However, in this case, since the negative edge is B -> C, which would only influent the cycle B -> C -> D, so it won't influent the shortest path from A to other vertex. On the other hand, since Dijkstra's algorithm is a greedy algorithm, it will only find the shortest path instead of going over all adges so that in this case it will only go over the negative edge once to find the shortest path from A to C.

The output:

```
~ » /Users/abby/Desktop/2020_fall/CSCE221_511/hw3/Q7 ;
0 4 2 2

[Process completed]_
```

The shortest path showed with graph:



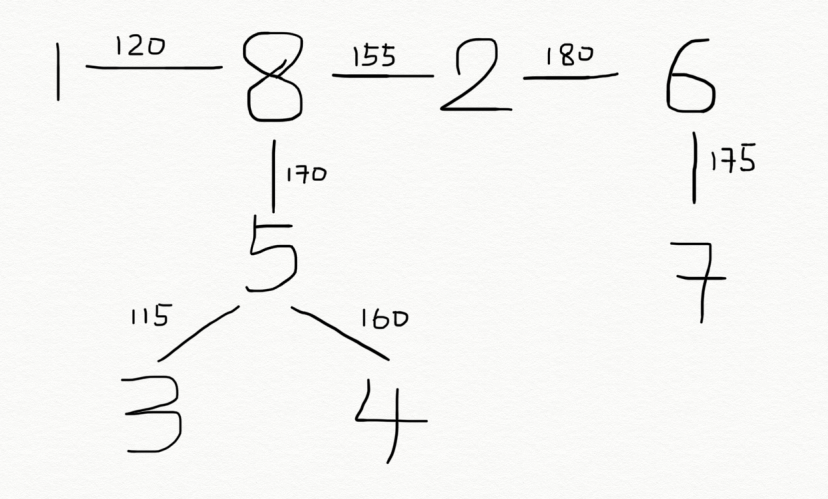
8. (20 points) There are eight small island in a lake, and the state wants to build seven bridges to connect them so that each island can be reached from any other one via one or more bridges. The cost of bridge construction is proportional to its length. The distance between pairs of islands are given in the following table.

   (a) Illustrate the Prim's algorithm using the graph below. Draw the Minimum Spanning Tree. What is the length of the bridges?

	1	2	3	4	5	6	7	8
1	-	240	210	340	280	200	345	120
2	-	-	265	175	215	180	185	155
3	-	-	-	260	115	350	435	195
4	-	-	-	-	160	330	295	230
5	-	-	-	-	-	360	400	170
6	-	-	-	-	-	-	175	205
7	-	-	-	-	-	-	-	305
8	-	-	-	-	-	-	-	-

   i. we choose the shortest edge of vertex 1, which is **1-8 : 120**
   ii. we choose the shortest edge of vertex 8, which is **8-2 : 155**
   iii. we choose the shortest edge of vertex 2, which is 2-4 : 175. However, since **8-5 : 170** is shorter than every edges of 2, we go over 5
   iv. we choose the shortest edge of vertex 5, whcih si **5-3 : 115**
   v. we choose the shortest edge of vertex 3, which is 3-8 : 195. However, since **5-4 : 160** is shorter than every edges of 3, we go over 4
   vi. we choose the shortest edge of vertex 4, which is 4-8 : 230. However, since **2-6 : 180** is shorter than every edges of 4, we go over 6
   vii. we choose the shortest edge of vertex 6, which is **6-7 : 175**

   Thus, we will use the edges mentioned above. The graph will be



(b) Illustrate the Kruskal's algorithm using the graph below. Draw the Minimum Spanning Tree. What is the length of the bridges?

	1	2	3	4	5	6	7	8
1	-	240	210	340	280	200	345	120
2	-	-	265	175	215	180	185	155
3	-	-	-	260	115	350	435	195
4	-	-	-	-	160	330	295	230
5	-	-	-	-	-	360	400	170
6	-	-	-	-	-	-	175	205
7	-	-	-	-	-	-	-	305
8	-	-	-	-	-	-	-	-

First, we can put all edges in order.

3-5	1-8	2-8	4-5	5-8	2-4	6-7	2-6	2-7	3-8	1-6	6-8	1-3	2-5
115	120	155	160	170	175	175	180	185	195	200	205	210	215
4-8	1-2	3-4	2-3	1-5	4-7	7-8	4-6	1-4	1-7	3-6	5-6	5-7	3-7
230	240	260	265	280	295	305	330	340	345	350	360	400	435

The edges we choose to make the MST are

3-5	115
1-8	120
2-8	155
4-5	160
5-8	170
6-7	175
2-6	180

Then our graph should be as same as the graph built by Prim's algorithm, which is