FUNCTIONS AND THEIR EXPLANATION

```c
struct socialNetListNode *newsocialNetListNode(int destination, int weight);
```

- This function takes destination and weight, then adds them to new node.

```c
struct graph *newGraph(int size)
```

- This function created with using struct graph and it creates new graph and returns it.

```c
struct minHeap* newMinHeap(int capacity)
```

- This function is using to create a new min heap.

```c
void addVertex(struct graph *graph, int source, int destination, int weight)
```

- This function takes values from input file and adds them into node with using newsocialNetListNode function.

```c
struct minHeapNode* newMinHeapNode(int maxSize, int destination)
```

- This function creates min heap ‘node’ according to size of vertices and their destination.

```c
void changeMinHeap(struct minHeapNode** firstHeapNode, struct minHeapNode** secondHeapNode)
```

- This function swap the position of two nodes from minHeapNode to organize min heap.

```c
void heapifyingMinHeap(struct minHeap* minHeap, int index)
```

- This function organizes the unorganized minheap with using changeMinHeap function. Also we can say that in a single word ‘heapifying’.

```c
int isInMinHeap(struct minHeap *minHeap, int destination)
```

- This function checks if given destination is in minheap or not. Then returns 1 or 0.

```c
int isEmpty(struct minHeap* minHeap)
```

- isEmpty function checks given minheap whether minheap is empty or not. If minheap is empty it return 1, else returns 0

```c
struct minHeapNode *getMin(struct minHeap* minHeap)
```

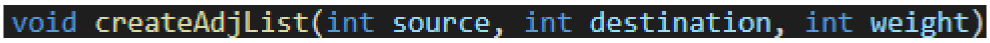
- getMin function returns the minimum node from minheap, but it checks first if minheap is empty or not.

```c
void reduce(struct minHeap* minHeap, int maxSize, int destination)
```

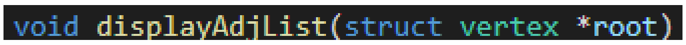
- reduce function reduces the position of node.

```c
void algorithmDijkstra(struct graph *graph, int source)
```

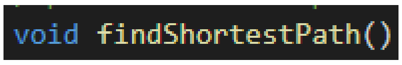
- This might be the most important function. Because it creates new minheap then it applies destination and source with their weights. It organizes the minheap and send destination, source and their weights to another function after calculation.


```
void createAdjList(int source, int destination, int weight)
```

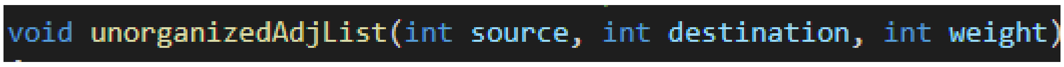
- This function takes values organized from dijsktra algorithm function. Then adds them into linked list to maket them prepared to display.


```
void displayAdjList(struct vertex *root)
```

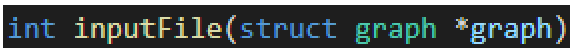
- displayAdjList function prints adjacency list like A: B,3 C,5 D,6 and so on.


```
void findShortestPath()
```

- This function takes source vertex and destination vertex as an input from user like A and B, then it prints the shortest path between those vertices.


```
void unorganizedAdjList(int source, int destination, int weight)
```

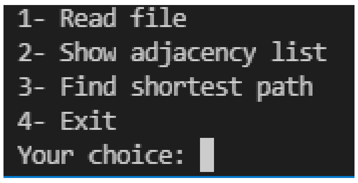
- This function takes values from input file sources with their destinations and weights to add them into nodes.


```
int inputFile(struct graph *graph)
```

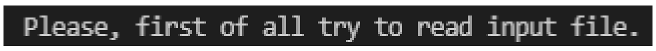
- inputFile function reads the input file, then takes values and send them to functions.

OUTPUTS

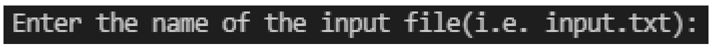
- When the program executes, this menu is shown and waits user to enter choice.


```
1- Read file
2- Show adjacency list
3- Find shortest path
4- Exit
Your choice:
```

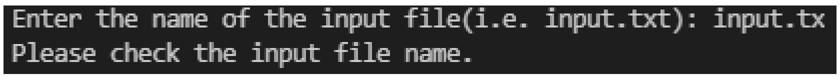
- If the user tries to choice menu 2 or 3, there will be printed error, because input file was not given.


```
Please, first of all try to read input file.
```

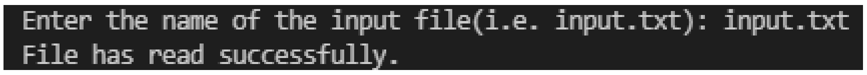
- When user select menu 1, this menu is going to be shown.


```
Enter the name of the input file(i.e. input.txt):
```

Then user should enter the input file correctly as ‘input.txt’, if the user enters wrong, this message will be printed.


```
Enter the name of the input file(i.e. input.txt): input.tx
Please check the input file name.
```

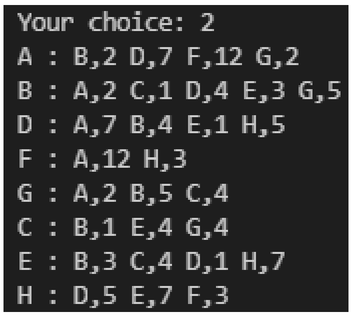
Then user should enter the input file correctly.



```
Enter the name of the input file(i.e. input.txt): input.txt
File has read successfully.
```

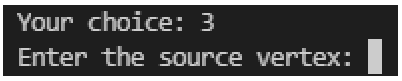
After reading the input file, user can print the adjancency list or try to find shortest path between two vertices.

- After reading if user wants to show adjancency list, user should enter 2, then this output will be on screen if everything goes correctly.



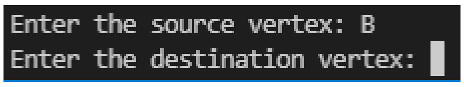
```
Your choice: 2
A : B,2 D,7 F,12 G,2
B : A,2 C,1 D,4 E,3 G,5
D : A,7 B,4 E,1 H,5
F : A,12 H,3
G : A,2 B,5 C,4
C : B,1 E,4 G,4
E : B,3 C,4 D,1 H,7
H : D,5 E,7 F,3
```

- If user wants to find shortest path between two vertices, user should enter 3.



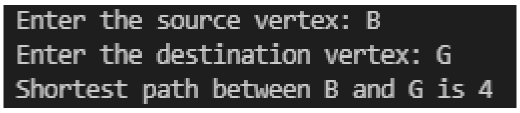
```
Your choice: 3
Enter the source vertex:
```

On this screent user should enter the source vertex. Lets try source as B,



```
Enter the source vertex: B
Enter the destination vertex:
```

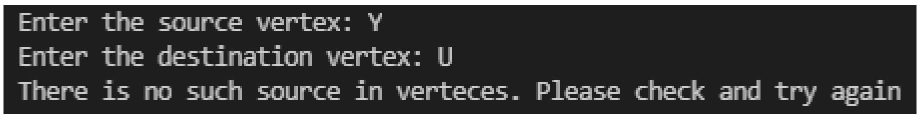
And user should enter destination vertex. We can try to enter G,



```
Enter the source vertex: B
Enter the destination vertex: G
Shortest path between B and G is 4
```

Finally, there is the output. The shortest path between B and G has written.

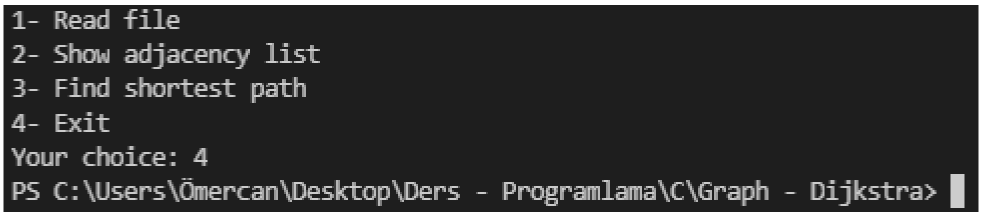
- What if user tries to enter a vertex which is not in the node list?



```
Enter the source vertex: Y
Enter the destination vertex: U
There is no such source in verteces. Please check and try again
```

And there it is. There is an error. Also if user tries to enter destination wrongly, there also be the error message.

- And the final choice.



```
1- Read file
2- Show adjacency list
3- Find shortest path
4- Exit
Your choice: 4
PS C:\Users\Ömercan\Desktop\Ders - Programlama\C\Graph - Dijkstra>
```

Up to this time, there is no incompleted part. All of the menus and functions work properly.