TCP 2101
ALGORITHM DESIGN AND ANALYSIS

Trimester 2 (2020/2021)

Assignment Report

Lecturer: Ts. Dr. Ng Kok Why
Tutorial Section: TT4V

Student ID	Student Name
1171103619	Terence Tan Kah Chee
1171103355	Low Zi Jian
1181302867	Ghada Saeed

Table of Contents

Data Generation

The main.cpp program will generate information about 10 planets, after entering our 3 student ids. The output of the program consists of the number, the name, and the 3d coordinate of the planets, as shown below.

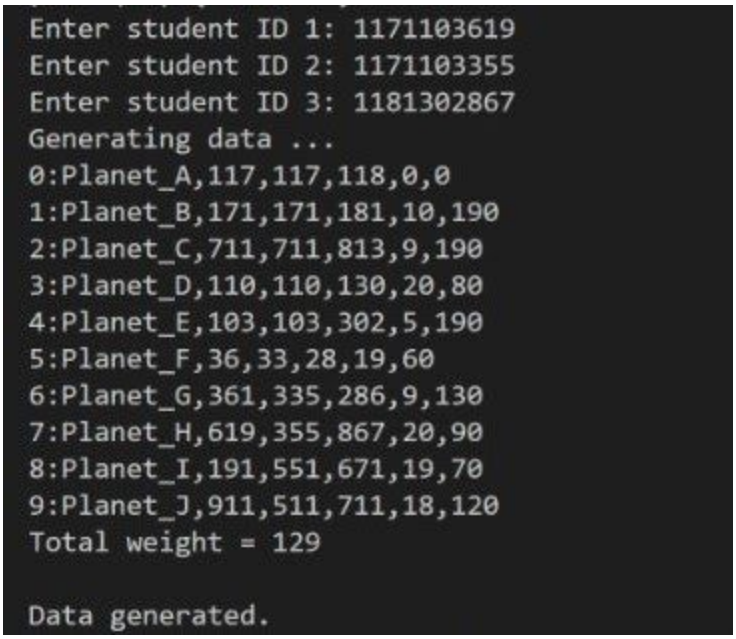


Figure 1 : Screenshot of" main.cpp" output

The main.cpp program will also generate a text file "A2planets.txt" which contains the data of the planets. This text file will be used throughout the 4 programs that will be shown in this report.

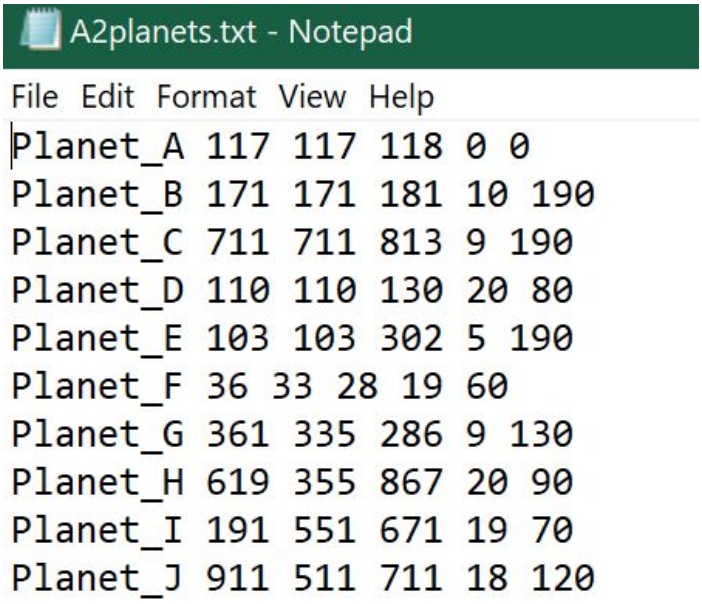


Figure 2 : Screenshot of "A2planets.txt"

The text data set consists of 10 rows, each of the rows carries information about a particular planet. Like the planet name(alphabet A-J), the planet x-coordinate,y-coordinate, and the z-coordinate respectively.

Table 1:Planets data

Planets	x	y	z	item weight	item profit
0:Planet_A	117	117	118	0	0
1:Planet_B	171	171	181	10	190
2:Planet_C	711	711	813	9	190
3:Planet_D	110	110	130	20	80
4:Planet_E	103	103	302	5	190
5:Planet_F	36	33	28	19	60

6:Planet_G	361	335	286	9	130
7:Planet_H	619	355	867	20	90
8:Planet_I	191	551	671	19	70
9:Planet_J	911	511	711	18	120
Total weight =				129	

$$distance_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$

By using the above formula we calculated the distance between the planets as shown in the following figure and table.

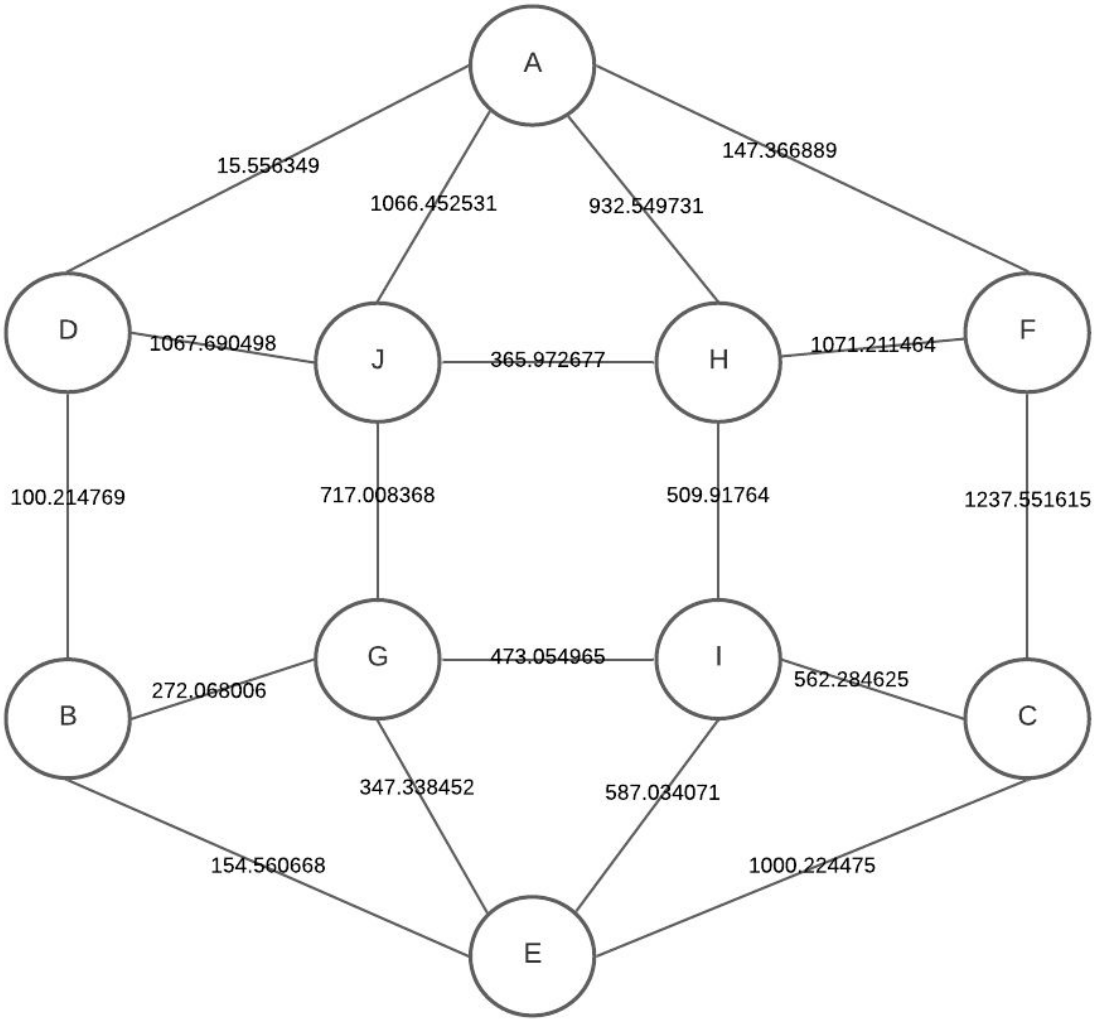


Figure 3 : Map of the planets

Table 2: Distance between planets

Planets	(X, Y, Z)	Distance
A -> D	(X1, Y1, Z1) = (117, 117, 118) (X2, Y2, Z2) = (110, 110, 130)	15.556349
A -> J	(X1, Y1, Z1) = (117, 117, 118)	1066.452531

	(X2, Y2, Z2) = (911, 511, 711)	
A -> H	(X1, Y1, Z1) = (117, 117, 118) (X2, Y2, Z2) = (619, 355, 867)	932.549731
A -> F	(X1, Y1, Z1) = (117, 117, 118) (X2, Y2, Z2) = (36, 33, 28)	147.366889
D -> J	(X1, Y1, Z1) = (110, 110, 130) (X2, Y2, Z2) = (911, 511, 711)	1067.690498
D -> B	(X1, Y1, Z1) = (110, 110, 130) (X2, Y2, Z2) = (171, 171, 181)	100.214769
J -> H	(X1, Y1, Z1) = (911, 511, 711) (X2, Y2, Z2) = (619, 355, 867)	365.972677
J -> G	(X1, Y1, Z1) = (911, 511, 711) (X2, Y2, Z2) = (361, 335, 286)	717.008368
H -> F	(X1, Y1, Z1) = (619, 355, 867) (X2, Y2, Z2) = (36, 33, 28)	1071.211464
H -> I	(X1, Y1, Z1) = (619, 355, 867) (X2, Y2, Z2) = (191, 551, 671)	509.91764
F -> C	(X1, Y1, Z1) = (36, 33, 28) (X2, Y2, Z2) = (711, 711, 813)	1237.551615
B -> G	(X1, Y1, Z1) = (171, 171, 181) (X2, Y2, Z2) = (361, 335, 286)	272.068006
G -> I	(X1, Y1, Z1) = (361, 335, 286) (X2, Y2, Z2) = (191, 551, 671)	473.054965
I -> C	(X1, Y1, Z1) = (191, 551, 671)	562.284625

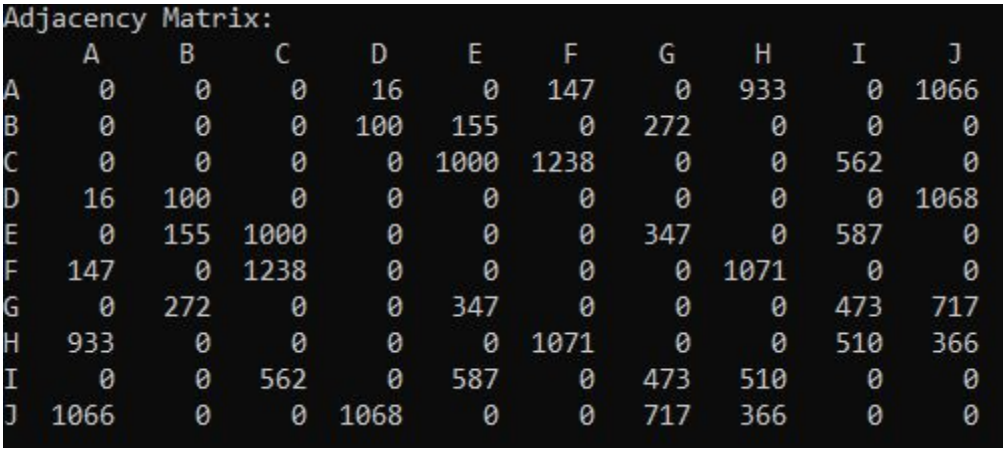
	(X2, Y2, Z2) = (711, 711, 813)	
E -> B	(X1, Y1, Z1) = (103, 103, 302) (X2, Y2, Z2) = (171, 171, 181)	154.560668
E -> G	(X1, Y1, Z1) = (103, 103, 302) (X2, Y2, Z2) = (361, 335, 286)	347.338452
E -> I	(X1, Y1, Z1) = (103, 103, 302) (X2, Y2, Z2) = (191, 551, 671)	587.034071
E -> C	(X1, Y1, Z1) = (103, 103, 302) (X2, Y2, Z2) = (711, 711, 813)	1000.224475

Program 1 – Display and Sort(adjacency matrix/list)

First, all edges of the map of planets are calculated using the formula :

$$distance_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$

Then, the value of each edge is inserted into a matrix of size 10 x 10 which is a 2d array. For example, planet A has an edge to planet D and the distance is 16, hence the position AD and DA in the matrix is filled with 16. All unreachable edges are filled with 0. The matrix is printed out using a nested for loop to travel through all positions in the matrix. Below is the output of the program.



Adjacency Matrix:

	A	B	C	D	E	F	G	H	I	J
A	0	0	0	16	0	147	0	933	0	1066
B	0	0	0	100	155	0	272	0	0	0
C	0	0	0	0	1000	1238	0	0	562	0
D	16	100	0	0	0	0	0	0	0	1068
E	0	155	1000	0	0	0	347	0	587	0
F	147	0	1238	0	0	0	0	1071	0	0
G	0	272	0	0	347	0	0	0	473	717
H	933	0	0	0	0	1071	0	0	510	366
I	0	0	562	0	587	0	473	510	0	0
J	1066	0	0	1068	0	0	717	366	0	0

Figure 4 : Screenshot of adjacency matrix

Adjacency list was created using an array of vectors. Each array represents a planet and each vector stores the adjacent planets. A nested for loop is used to travel through the matrix in order to determine which planets are connected. The second for loop counter is initialized to the first for loop counter, so that only the upper diagonal of the matrix is being visited, hence duplication of edges are eliminated. The first for loop indicates the starting planet while the second for loop indicates the destination planet, if the value in the matrix is not 0, the destination planet name will be pushed back into the vector of the starting planet. After that, the array of vectors will be printed out using a nested for loop. Below is the output of the adjacency list.

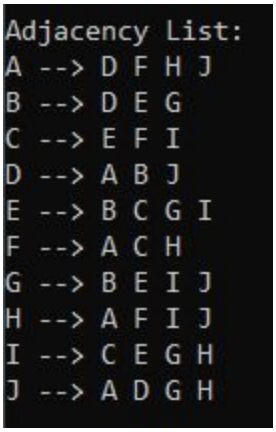


Figure 5 :Screenshot of adjacency list.

Merge sort is a divide and conquer algorithm, it divides the original array into two sub arrays and recursively calls itself to further separate the sub arrays until the array cannot be further divided into half. Then, sub arrays are merged together from bottom to top in ascending order or descending order to form a sorted array. First, the program calls the merge sort function and puts the array to be sorted,the first index as well as the last index of the array as the parameter. Then, inside of the merge sort function, middle index is calculated by using the formula:

$$m = l + (r - l) / 2$$

where l is the first index and r is the last index of the array. The function then recursively calls itself twice where the in the first time, the parameter last index changed to m. As for the second time, the first index changed to m+1. Hence, the function separated the array into two smaller sub arrays. After that, the sub arrays are merged together using a merge function. The process of separating the array into sub arrays repeated until there is only one element in the sub array. Figure below shows the list of edges in ascending order of distance sorted using merge sort.

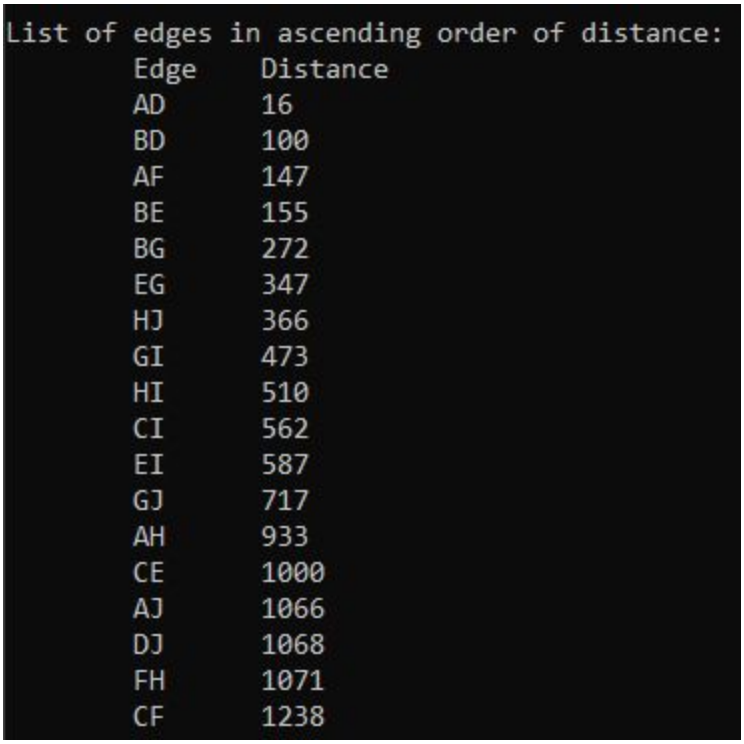


Figure 6 : Screenshot of list edges in ascending order of distance

In order to sort in descending order, the merge function needs to be modified. During the process of merging the sub arrays, a larger value will be first inserted into the array instead of a smaller value. This results in the array to be sorted in descending order. Figure below shows the result of the list of planets in descending order of value sorted using merge sort.

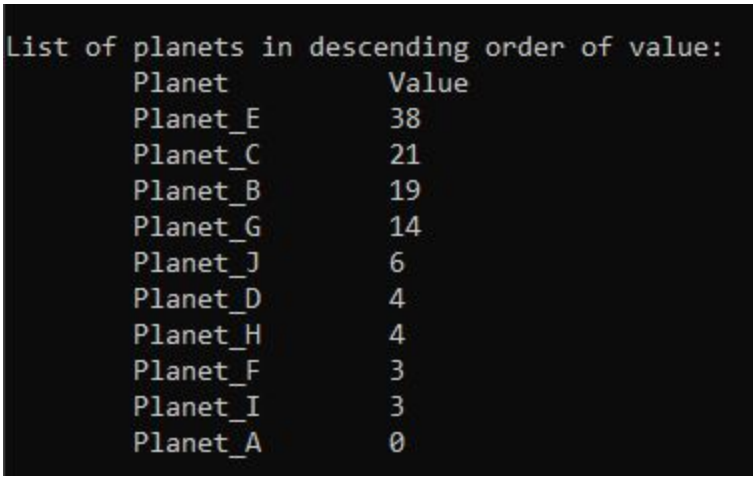


Figure 7 : Screenshot of list of planets in descending order of value

Program 2 – Shortest Paths(Dijkstra's Algorithm)

Algorithm

1) Create an array of booleans named "sptSet" to keep track of the planets included in the shortest path tree and initialize it to be empty.

2) The distance for all planets to other planets is calculated using formula where x_j, y_j *and* z_j are the coordinates of a planet and x_i, y_i *and* z_i are the coordinates of another planet that forms a link with a planet:

$$distance_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$

Then, initialize all distance values as INFINITE and sptSet values as false. Assign distance value to the source planet as 0 so that it is picked first.

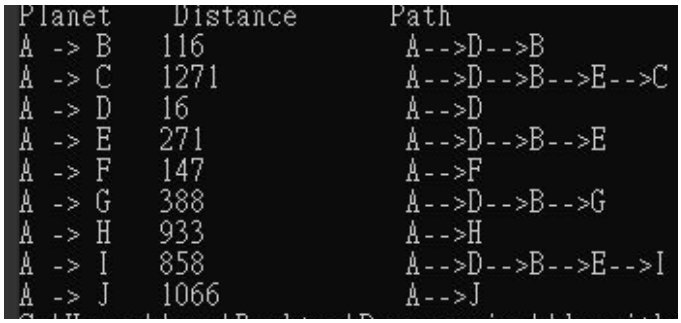
3) While sptSet doesn't include all planets

….a) Pick a planet, u which is not there in sptSet and has a minimum distance value.

….b) Include planet u to sptSet since u is picked.

….c) Update the distance value of all adjacent planets of u by iterating through all adjacent planets. For every adjacent planet v, if the sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v using sum of distance value of u (from source) and weight of edge u-v

Program Output



Planet	Distance	Path
A -> B	116	A-->D-->B
A -> C	1271	A-->D-->B-->E-->C
A -> D	16	A-->D
A -> E	271	A-->D-->B-->E
A -> F	147	A-->F
A -> G	388	A-->D-->B-->G
A -> H	933	A-->H
A -> I	858	A-->D-->B-->E-->I
A -> J	1066	A-->J

Figure 8 : Dijikstra.cpp output

As the planet A is the source planet, the shortest distance is 0 and there is no shortest path since planet A is the source planet itself and the destination is also itself.

From the source planet, A to planet B. The shortest distance is 116 and the shortest path from planet A to planet B is from A to D to B.

From the source planet, A to planet C. The shortest distance is 1271 and the path from planet A to planet C is from A to D to B to E to C.

From the source planet, A to planet D. The shortest distance is 16 and the path from planet A to planet D is from A to D.

From the source planet, A to planet E. The shortest distance is 271 and the path from planet A to planet E is from A to D to B to E.

From the source planet, A to planet F. The shortest distance is 147 and the path from planet A to planet F is A to F.

From the source planet, A to planet G. The shortest distance is 388 and the path from planet A to planet G is from A to D to B to G.

From the source planet, A to planet H. The shortest distance is 933 and the path from planet A to planet H is from A to H.

From the source planet, A to planet I. The shortest distance is 858 and the path from planet A to planet I is from A to D to B to E to I.

From the source planet, A to planet J. The shortest distance is 1066 and the path from planet A to planet J is from A to J.

Program 3 – Minimum Spanning Tree(Kruskal's Algorithm)

Algorithm

1. Pick the edge with the smallest weight. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed then include this edge else remove it.
2. Repeat step number 2 until there are (V-1) edges in the spanning tree.

Program Output

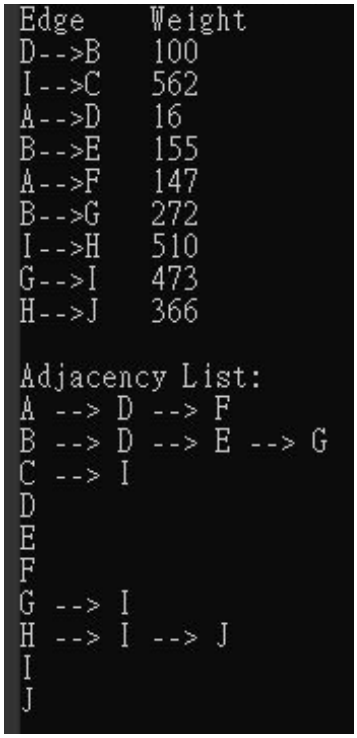


Figure 9 : Kruskal.cpp output

The distance between the 2 planets is known as weight in the program. Thus, we pick the edges from the smallest weight 1st in ascending order. In the program output, the edges of planets and its weight are listed which do not form any cycle. The following adjacency list of the Minimum Spanning Tree resulted from the Kruskal Algorithm is outputted .

Program 4 – Dynamic Programming(0/1 Knapsack Algorithm)

0/1 Knapsack Problem is a variant of Knapsack Problem that does not allow to fill the knapsack with fractional items. 0/1 Knapsack Problem solved using Dynamic Programming.

1.Create array B[][] with (n+1) number of rows and (M+1) number of columns.

2.Then will initialize the values of the array B[][] to 0.

2.The array B[][] is in a bottom-up manner.

3.Then calculate the table of options with the retrieval formula.

4.Calculate B[i][j]. If not select item planet i.

5.Then will decide if the selected planet item i, will be more beneficial than the rest of B[i][j] or not.

6.Once selected planet n, add weight M - W[n - 1].

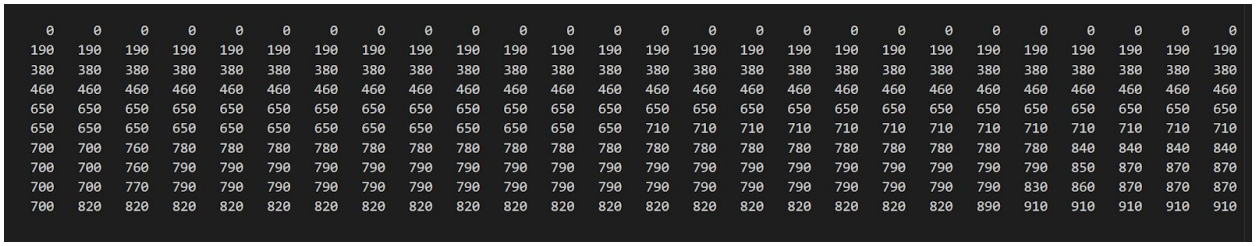
7.Then calculate until required capacity.

```
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190
   0    0    0    0    0    0    0    0    0  190  190  190  190  190  190  190  190  190  190  380  380  380  380  380  380  380
   0    0    0    0    0    0    0    0    0  190  190  190  190  190  190  190  190  190  190  380  380  380  380  380  380  380
   0    0    0    0    0  190  190  190  190  190  190  190  190  190  380  380  380  380  380  380  380  380  380  380  570  570
   0    0    0    0    0  190  190  190  190  190  190  190  190  190  380  380  380  380  380  380  380  380  380  380  570  570
   0    0    0    0    0  190  190  190  190  190  190  190  190  190  380  380  380  380  380  380  380  380  380  510  570  570
   0    0    0    0    0  190  190  190  190  190  190  190  190  190  380  380  380  380  380  380  380  380  380  510  570  570
   0    0    0    0    0  190  190  190  190  190  190  190  190  190  380  380  380  380  380  380  380  380  380  510  570  570
   0    0    0    0    0  190  190  190  190  190  190  190  190  190  380  380  380  380  380  380  380  380  380  510  570  570
Max Value:  910
Selected Planets:
    Planet J with Weight = 18 and Profit = 120
    Planet H with Weight = 20 and Profit = 90
    Planet G with Weight = 9 and Profit = 130
    Planet E with Weight = 5 and Profit = 190
    Planet C with Weight = 9 and Profit = 190
    Planet B with Weight = 10 and Profit = 190
```

Figure 10 : 0/1 Knapsack program output 1/4

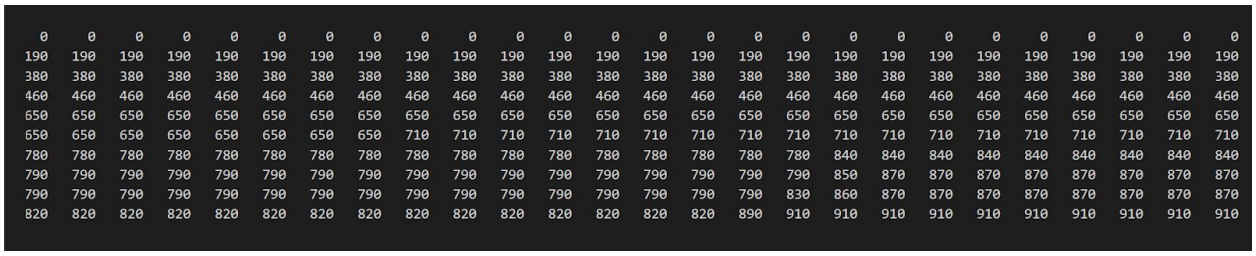
```
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190  190
 380  380  380  380  380  380  380  380  380  380  380  380  380  380  380  380  380  380  380  380  380  380  380  380  380  380
 380  380  380  380  380  380  380  380  380  380  380  380  380  460  460  460  460  460  460  460  460  460  460  460  460  460
 570  570  570  570  570  570  570  570  570  570  570  570  570  570  570  570  570  570  650  650  650  650  650  650  650  650
 570  570  570  570  570  570  570  570  570  570  570  570  570  570  570  570  570  630  650  650  650  650  650  650  650  650
 570  570  570  570  570  570  570  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700
 570  570  570  570  570  570  570  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700
 570  570  570  570  570  570  570  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700
 570  570  570  570  570  570  570  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  700  820
```

Figure 11 : 0/1 Knapsack program output 2/4



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190
380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380
460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460
650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650
650	650	650	650	650	650	650	650	650	650	650	650	650	710	710	710	710	710	710	710	710	710	710	710	710	710
700	700	760	780	780	780	780	780	780	780	780	780	780	780	780	780	780	780	780	780	780	780	840	840	840	840
700	700	760	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	850	870	870	870
700	700	770	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	830	860	870	870	870
700	820	820	820	820	820	820	820	820	820	820	820	820	820	820	820	820	820	820	820	820	890	910	910	910	910	910

Figure 12 : 0/1 Knapsack program output 3/4



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190
380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380
460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460
650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650
650	650	650	650	650	650	650	650	710	710	710	710	710	710	710	710	710	710	710	710	710	710	710	710	710	710
780	780	780	780	780	780	780	780	780	780	780	780	780	780	780	780	780	840	840	840	840	840	840	840	840	840
790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	850	870	870	870	870	870	870	870	870
790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	790	830	860	870	870	870	870	870	870	870
820	820	820	820	820	820	820	820	820	820	820	820	820	820	820	820	890	910	910	910	910	910	910	910	910	910

Figure 13 : 0/1 Knapsack program output 4/4

In order to gain the best benefit that fit the required capacity of 80tons ,the spaceship can follow this path according to the map ,starting from planet J which will add 18 tons and 120 profit ,then will go to planet H and add 20tons with 90 profit, then will go to planet G adding 9tons with a profit of 130 ,after that will go it is next destination which is planet B adding 10tons of weight and 190 profit, then will go to planet E and add 5tons and 190 profit, lastly it will head to it is last destination planet C adding 9tons with 190 profit. By using the 0/1 Knapsack Algorithm the spaceship will gain a profit of 910 with 71tons of weight.

Conclusion

Merge Sort:

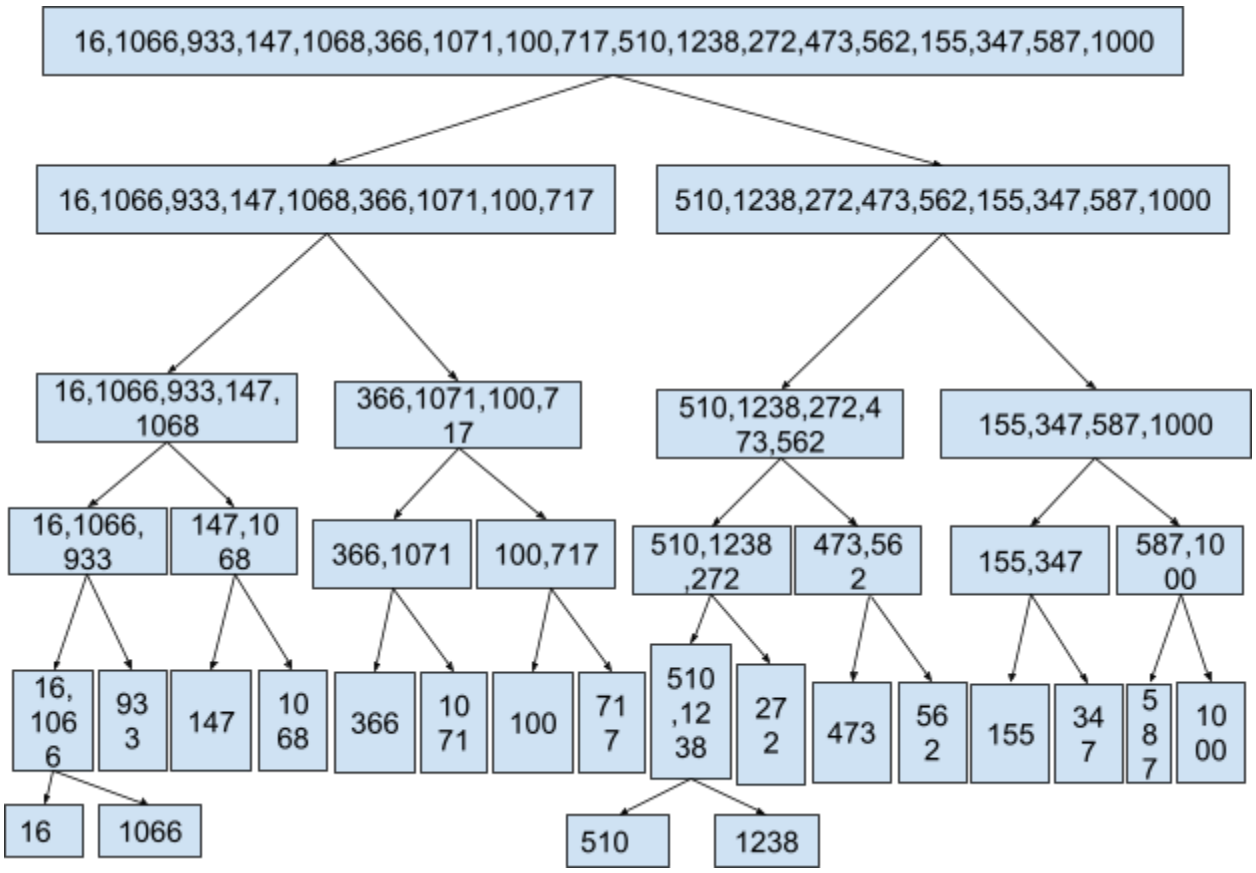


Figure 14 : Division of array to smaller size sub arrays of distance between planets

For ascending order sorting, merging starts from the bottom to the top. The program compares the elements in the sub arrays one by one and inserts the smaller value into the parent array first. For example, as we can see value 16 is the smallest element and it is inserted as the first element into the parent array followed by 1066. Then, The sub array of two elements is used to compare with the other sub array which contains only one element, 933. 16 is the smallest element, followed by 933 and 1066. The process repeated until reached the top where all elements in the array will be sorted in ascending order.

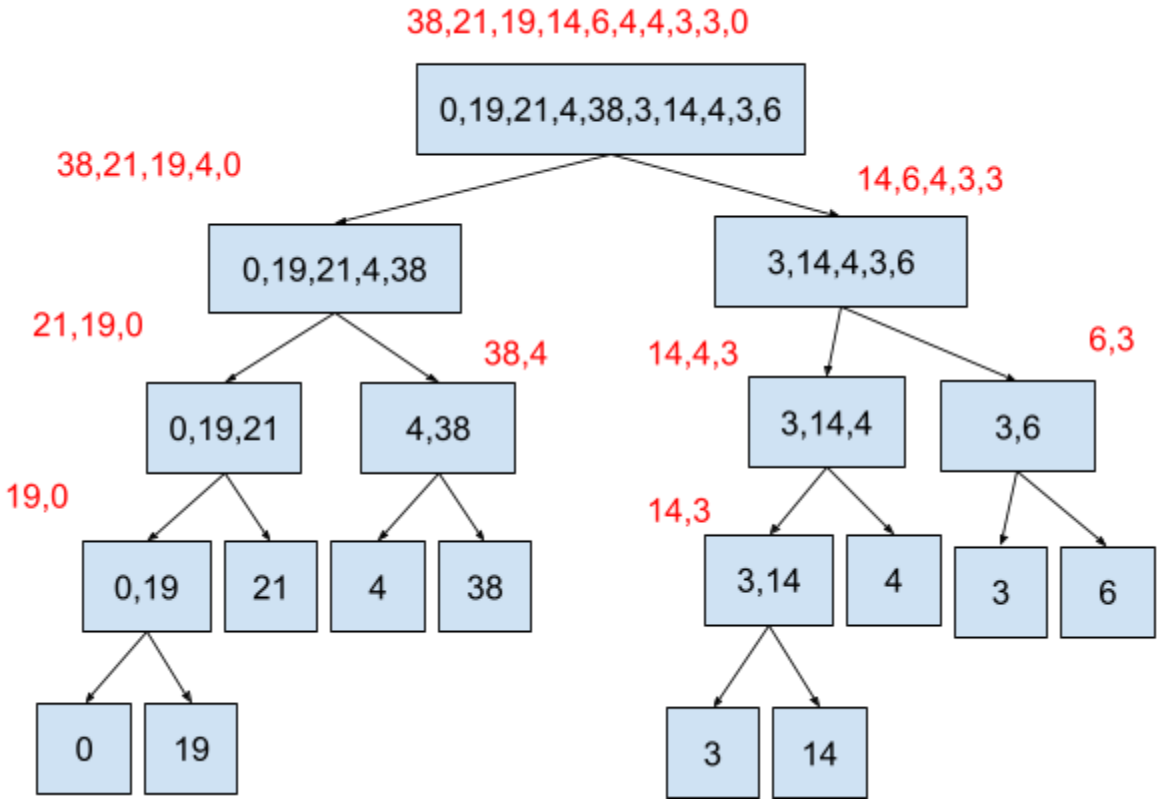


Figure 15 : Merge sort for planet value in descending order

As for the descending order, the sorting processes are mostly similar. The difference is when comparing the elements, the larger value element will be inserted into the parent array first instead of the smaller element. The red color indicates the sorted element of the arrays.

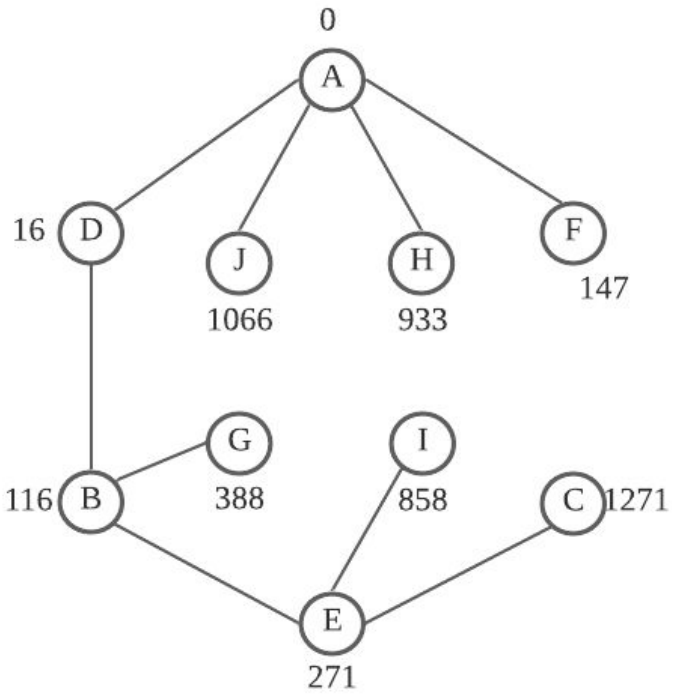


Figure 16 : Shortest Path Tree from Dijikstra.cpp

Based on the shortest path output from the source planet to every other planet, we drew a Shortest Path Tree (SPT). We list the distance travelled at each planet from planet source A. We noticed that all planets have been travelled and the path it travels forms a tree as well as the distance it travels to every planet is the shortest. Thus, it is a Shortest Path Tree (SPT).

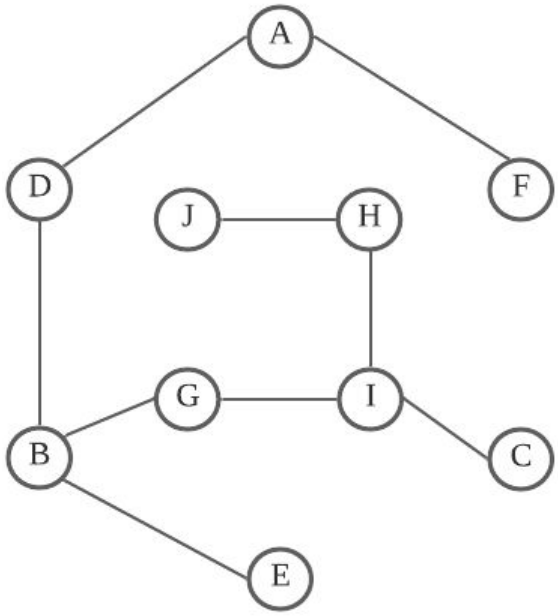


Figure 17 : Minimum Spanning Tree from Kruskal.cpp

Based on the adjacency list of Kruskal's Algorithm output, we draw the following Minimum Spanning Tree. We noticed from the number of edges in the MST is (V-1), 10 - 1 = 9 edges. Thus this is the correct Minimum Spanning Tree.

	Pl ,Iw ,Ip	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	A,0,0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	B,10,190	0	0	0	0	0	0	0	0	0	0	190	190	190	190	190	190	190	190	190	190	190
4	C,9,190	0	0	0	0	0	0	0	0	0	190	190	190	190	190	190	190	190	190	190	380	380
5	D,20,80	0	0	0	0	0	0	0	0	0	190	190	190	190	190	190	190	190	190	190	380	380
6	E,5,190	0	0	0	0	0	190	190	190	190	190	190	190	190	190	380	380	380	380	380	380	380
7	F,19,60	0	0	0	0	0	190	190	190	190	190	190	190	190	190	380	380	380	380	380	380	380
8	G,9,130	0	0	0	0	0	190	190	190	190	190	190	190	190	190	380	380	380	380	380	380	380
9	H,20,90	0	0	0	0	0	190	190	190	190	190	190	190	190	190	380	380	380	380	380	380	380
10	I,19,70	0	0	0	0	0	190	190	190	190	190	190	190	190	190	380	380	380	380	380	380	380
11	J,18,120	0	0	0	0	0	190	190	190	190	190	190	190	190	190	380	380	380	380	380	380	380

	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190
4	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380
5	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	460	460
6	380	380	380	570	570	570	570	570	570	570	570	570	570	570	570	570	570	570	570	570
7	380	380	380	570	570	570	570	570	570	570	570	570	570	570	570	570	570	570	570	570
8	380	380	510	570	570	570	570	570	570	570	570	570	700	700	700	700	700	700	700	700
9	380	380	510	570	570	570	570	570	570	570	570	570	700	700	700	700	700	700	700	700
10	380	380	510	570	570	570	570	570	570	570	570	570	700	700	700	700	700	700	700	700
11	380	380	510	570	570	570	570	570	570	570	570	570	700	700	700	700	700	700	700	700

1	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190
4	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380
5	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460
6	570	570	570	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650
7	570	570	630	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650
8	700	700	700	700	700	700	700	700	700	700	700	760	780	780	780	780	780	780	780	780
9	700	700	700	700	700	700	700	700	700	700	700	760	790	790	790	790	790	790	790	790
10	700	700	700	700	700	700	700	700	700	700	700	770	790	790	790	790	790	790	790	790
11	700	700	700	700	700	700	700	700	700	700	820	820	820	820	820	820	820	820	820	820

1	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190
4	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380	380
5	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460	460
6	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650	650
7	650	650	710	710	710	710	710	710	710	710	710	710	710	710	710	710	710	710	710	710
8	780	780	780	780	780	780	780	780	780	780	780	840	840	840	840	840	840	840	840	840
9	790	790	790	790	790	790	790	790	790	790	790	850	870	870	870	870	870	870	870	870
10	790	790	790	790	790	790	790	790	790	790	830	860	870	870	870	870	870	870	870	870
11	820	820	820	820	820	820	820	820	820	890	910	910	910	910	910	910	910	910	910	910

Figure 18 : 0/1 Knapsack program table

Considering the 0/1 Knapsack Algorithm we have tested the benefit gained after trying the possible paths ,and we found that J H G B E C is the correct path for this case.