Algorithm Design and Analysis

Assignment 2 Report

Prepared by

John C. G. Escobia

February 10, 2021

Abstract

This report examines the experimental results for sorting edges according to distance and value with merge-sort, calculating shortest distances with Dijikstra's algorithm, producing a Minimum Spanning Tree with Kruskal's Algorithm, and calculating the maximum benefit with 0/1 Knapsack Algorithm.

Table of Contents

Overview **2**
Environment 2
Pre-Execution Notes 2
Program Execution 2
Get Data **3**
Introduction 3
Algorithm 3
Program 1 **6**
Algorithm 6
Experimental Results 9
Program 2 **11**
Algorithm 11
Experimental Results 13
Program 3 **14**
Algorithm 14
Experimental Results 16
Program 4 **17**
Algorithm 17
Experimental Results 19

Overview

Environment

Operating System: [Ubuntu 20.04]

Integrated Development Environment: [Geany] v1.36

Pre-Execution Notes

Running *main.cpp* file will take user input of three student IDs, generate 10 planet data and store the generated data in a text file — *A2planets.txt*.

Input	Output
The following three student IDs were taken from the user: ● Student ID 1: 1132701350 ● Student ID 2: 1193540154 ● Student ID 3: 1156548216	10 planets were produced with each planet having the following data: ● Planet name ● X-coordinate ● Y-coordinate ● Z-coordinate ● Weight ● Profit

Table 1: Shows the input and output data upon running main.cpp file.

Program Execution

There are four programs in total. Each program runs by executing a main file with the program name which will call other source code files as follows:

No	Program	Main File	Files Included
1	Program 1	program_1.cpp	planet.cpp, getData.cpp
2	Program 2	program_1.cpp	planet.cpp, getData.cpp
3	Program 3	program_1.cpp	planet.cpp, getData.cpp
4	Program 4	program_1.cpp	planet.cpp, getData.cpp

Table 2: Shows the main file and files included for each program.

Get Data

Introduction

As observed in Table 2, the files included for each program are the same two source code files. These source files are *planet.cpp* and *getData.cpp*.

Algorithm

planet.cpp

```
START STRUCT Connection
  ATTRIBUTE string name
  ATTRIBUTE char initial
  ATTRIBUTE bool isConnected
  ATTRIBUTE int distance
END STRUCT Connection

START CLASS Planet
  ATTRIBUTE string name
  ATTRIBUTE char initial
  ATTRIBUTE int x, y, z, weight, profit, value
  ATTRIBUTE vector< Connection > con

  CONSTRUCTOR Planet (string arr[])
    name = arr[0]
    initial = arr[0].at(7)
    x = arr[1]
    y = arr[2]
    z = arr[3]
    weight = arr[4]
    profit = arr[5]

    IF weight == 0 or profit == 0 THEN value = 0
    ELSE value = profit / weight

    CALL FUNCTION setConnections (name)
  END CONSTRUCTOR Planet

  START setDistance (string name, int distance)
    FOR elements in con as c
      IF c.name == name THEN c.distance = distance
  END setDistance

  START setConnections (string name)
    // Set the connections for planets that are directly connected
  END setConnections
END CLASS Planet
```

getData.cpp

```
DEFINE Edge as pair<char, char>

START setEdges (vector<pair<int, Edge>> edges, char planetOne,
  char planetTwo, int distance)
  // A subroutine that sets distance as weight of edge between planetOne
  // and planetTwo and pushes them to vector edges
END setEdges

START calculateDistance (vector<Planet> vecPlanet)
  FOR elements in vecPlanet as planetOne
    FOR elements in planetOne.con as connection
      IF connection.isConnected == true THEN
        index = 0
        FOR elements in vecPlanet as planetTwo
          IF planetTwo.name == planetOne.name THEN
            int x = pow (planetOne.x-planetTwo.x, 2)
            int y = pow (planetOne.y-planetTwo.y, 2)
            int z = pow (planetOne.z-planetTwo.z, 2)
            int distance = sqrt (x + y + z)
            planetOne.setDistance (connection.name, distance)
          index++
END calculateDistance

START calculateDistance (vector<Planet> vecPlanet,
  vector<pair<int, Edge>> edges)
  // Similar to the calculateDistance function defined above. The only
  // difference is that this function calls the following function
  // inside the if statement prior to the increment of index:

  setEdges (edges, planetOne.initial, planetTwo.initial, distance)
END calculateDistance

START createPlanet (vector<Planet> vecPlanet, string planetData)
  istringstream iss (planetData)
  string planetDataWord

  string arr[6]
  int i = 0

  WHILE iss >> planetDataWord
    Arr[i] = planetDataWord
    ++i

  Planet planet (arr)

  vecPlanet.push_back (planet)
END

START getPlanetData (vector<Planet> vecPlanet)
  OPEN A2planets.txt
  Initialize ReadFile = contents of A2planets.txt
```

```
  Initialize planet = ""

  START infinite while-loop
    GETLINE from ReadFile and store to planet

    IF ReadFile reaches end of file THEN stop while-loop

    Call function createPlanet (vecPlanet, planet)

    NEXTLINE
  END while-loop

  CLOSE A2planets.txt
END getPlanetData
```

Program 1

Algorithm

program_1.cpp

```
DEFINE Edge as pair<char, char>

START merge (vector<pair<int, Edge>> A, vector<pair<int, Edge>> T,
  int p, int m, int r, string type)
  int i, j

  FOR i from m+1 to p-1
    Temp[i - 1] = A[i - 1]
  NEXT --i

  FOR j from m to r-1
    Temp[r + m - j] = A[j + 1]
  NEXT j

  FOR k from p to r
    IF type == "ascending" THEN
      IF Temp[j] < Temp[i] THEN A[k] = Temp[j--]
      ELSE A[k] = Temp[i++]
    ELSE
      IF Temp[j] > Temp[i] THEN A[k] = Temp[j--]
      ELSE A[k] - Temp[i++]
  NEXT k
END merge

START mergeSort (vector<pair<int, Edge>> A, vector<pair<int, Edge>> T,
  int p, int m, int r, string type)

  IF p < r THEN
    int m = (p + r) / 2
    mergeSort (A, Temp, p, m, type)
    mergeSort (A, Temp, m + 1, r, type)
    merge (A, Temp, p, m, r, type)
END mergeSort

START storeEdges (vector<pair<int, Edge>> edges)
  Populate vector edges with edges of directly connected planets
END storeEdges

START displayAdjacencyList (vector<Planet> vecPlanet)
  PRINT "---ADjACENCY LIST---"
  int j = 0

  FOR elements in vecPlanet as planet
    PRINT planet.initial
    FOR elements in planet.con as connection
```

```
      IF connection.name == vecPlanet[j].name and
      connection.isConnected == true THEN
        PRINT "-->" + connection.initial "|" + connection.distance
      j++
    j = 0
    PRINT two newline
END displayAdjacencyList

START displayAdjacencyMatrix (vector<Planet> vecPlanet)
  PRINT "---ADJACENCY MATRIX---" + two newline

  FOR elements in vecPlanet as planet
    PRINT planet.initial

  PRINT two new lines
  string inf = "   " + infinity symbol
  int j = 0

  FOR elements in vecPlanet as planet
    PRINT planet.initial

    FOR elements in planet.con as connection
      IF connection.name == vecPlanet[j].name and
      connection.isConnected == true
        PRINT connection.distance
      ELSE PRINT inf
      j++
    j = 0
    PRINT two newline
  PRINT three newline
END displayAdjacencyMatrix

START main ()
  Initialize vector<Planet> vecPlanet

  Call subroutine getPlanetData (vecPlanet)

  Initialize vector<pair<int, Edge>> edges

  Call subroutine storeEdges (edges)

  Call subroutine calculateDistance (vecPlanet, edges)

  Call subroutine displayAdjacencyMatrix (vecPlanet)

  Call subroutine displayAdjacencyList (vecPlanet)

  Random_shuffle (edges)

  vector<pair<int, Edge>> Temp (18)
  mergeSort (edges, Temp, 0, edges.size()-1, "ascending")

  PRINT three newline + "---PLANET DISTANCE ---" + one newline
```

```
  PRINT "Edge" + "    " + "Distance" + one newline

  FOR elements in edges as edge
    PRINT "(" + edge.second.first + ",  " + edge.second.second + ")"
    + "   " + edge.first + one newline

  vector<pair<int, Edge>> values
  FOR elements in vecPlanet as planet
    Push to vector values planet.value and Edge(planet.initial,
    planet.initial)

  vector<pair<int, Edge>> valTemp (10)

  mergeSort (values, valTemp, 0, values.size()-1, "descending")

  PRINT three newline + "---PLANET VALUES---" + one newline
  PRINT "Planet" + "   " + "Value" + one newline

  FOR elements in values as val
    PRINT " " + val.second.first + "   " + val.first + one newline

  RETURN 0
END main
```

Experimental Results

Adjacency Matrix:

------------------------------ ADJACENCY MATRIX ------------------------------

	A	B	C	D	E	F	G	H	I	J
A	∞	∞	∞	608	∞	473	∞	259	∞	584
B	∞	∞	∞	541	773	∞	688	∞	∞	∞
C	∞	∞	∞	∞	544	625	∞	∞	590	∞
D	608	541	∞	∞	∞	∞	∞	∞	∞	268
E	∞	773	544	∞	∞	∞	818	∞	435	∞
F	473	∞	625	∞	∞	∞	∞	495	∞	∞
G	∞	688	∞	∞	818	∞	∞	∞	919	465
H	259	∞	∞	∞	∞	495	∞	∞	419	580
I	∞	∞	590	∞	435	∞	919	419	∞	∞
J	584	∞	∞	268	∞	∞	465	580	∞	∞

Adjacency List:

```
-----------  ADJACENCY LIST  ------------

A --> D|608 --> F|473 --> H|259 --> J|584

B --> D|541 --> E|773 --> G|688

C --> E|544 --> F|625 --> I|590

D --> A|608 --> B|541 --> J|268

E --> B|773 --> C|544 --> G|818 --> I|435

F --> A|473 --> C|625 --> H|495

G --> B|688 --> E|818 --> I|919 --> J|465

H --> A|259 --> F|495 --> I|419 --> J|580

I --> C|590 --> E|435 --> G|919 --> H|419

J --> A|584 --> D|268 --> G|465 --> H|580
```

Planet Distance (ascending order):

```
---PLANET DISTANCE---
 Edge         Distance
(A, H)          259
(D, J)          268
(H, I)          419
(I, E)          435
(J, G)          465
(A, F)          473
(H, F)          495
(D, B)          541
(C, E)          544
(J, H)          580
(A, J)          584
(I, C)          590
(A, D)          608
(F, C)          625
(B, G)          688
(B, E)          773
(G, E)          818
(G, I)          919
```

Planet Values (descending order):

```
---PLANET VALUES---
Planet         Value
  B             25
  F             21
  J             12
  I             11
  D             11
  E             8
  G             7
  H             5
  C             5
  A             0
```

Program 2

Algorithm

program_2.cpp

```
TEMPLATE <typename T>

START getIndex (T object, string name)
  auto itCon = find_if(object.begin(), object.end(), [name](auto member)
               {return member.name == name;})
  auto index = std::distance(object.begin(), itCon)
  RETURN index
END getIndex

START dijkstra (vector<Planet> vecPlanet)
  FOR elements in vecPlanet[0].con as connections
    IF connections.initial != 'A' THEN
      connections.isConnected = false
    ELSE
      connections.isConnected = true

    int counter = 0

    WHILE --counter > 0
      vector<pair<int, string>> distances

      FOR elements in vecPlanet[0].con as connections
        IF connections.isConnected == false THEN
          make_pair of (connections.distance, connections.name) and
          push_back to vector distances

      sort vector distances in ascending order according to first
      element

      string nearestPlanetName = distances[0].second
      int nearestPlanetDist = distances[0].first

      int indexCon = getIndex (vecPlanet[0].con, nearestPlanetName)

      vecPlanet[0].con[indexCon].isConnected = true

      int indexPlanet = getIndex (vecPlanet, nearestPlanetName)

      Planet relaxPivot = vecPlanet[indexPlanet]

      FOR elements in relaxPivot.con as connections
        IF connections.isConnected THEN
          FOR elements in vecPlanet[0].con as con
            IF con.name == connections.name && !con.isConnected THEN
              IF con.distance > connections.distance+nearestPlanetDist
```

```
                THEN con.distance = connections.distance +
                nearestPlanetDist
       Clear contents of vector distances
END dijkstra

START main ()
  Initialize vector<Planet> vecPlanet

  Call subroutine getPlanetData (vecPlanet)

  Call subroutine calculateDistance (vecPlanet)

  Call subroutine dijkstra (vecPlanet)

  PRINT "Shortest distance from Planet_A:" + two newline

  FOR i = 1 until vecPlanet[0].con.size()-1
    PRINT vecPlanet[0].con[i].name + " " + vecPlanet[0].con[i].distance
  NEXT i

  RETURN 0
END main
```

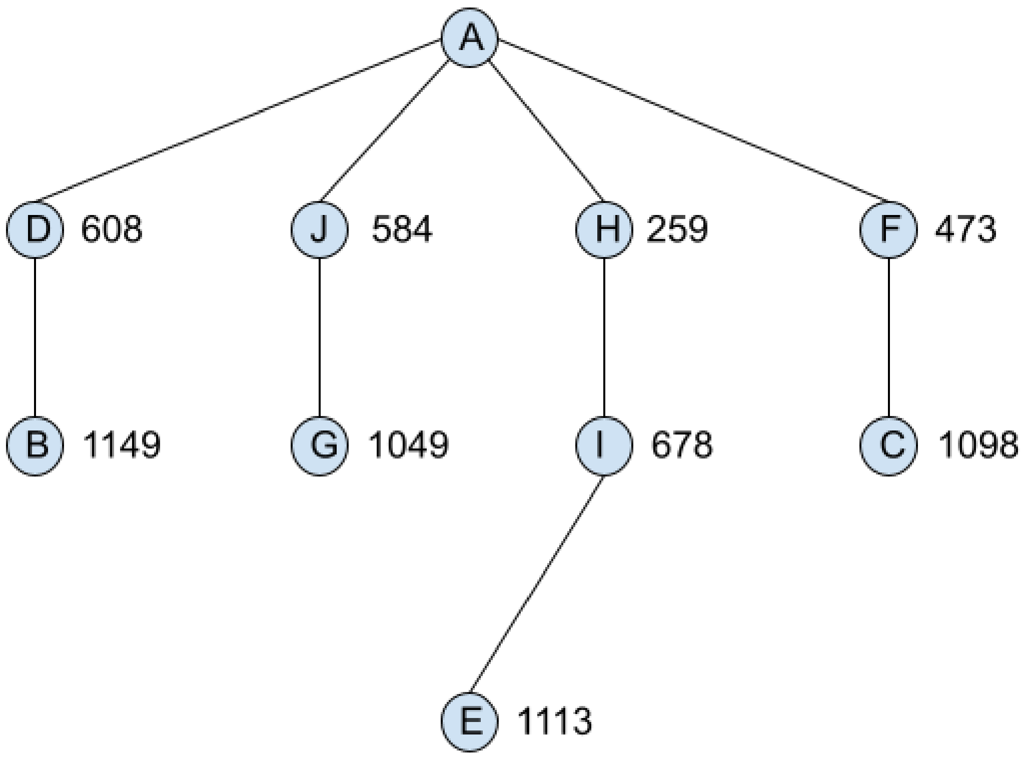
Experimental Results

Shortest Distance from Planet_A:

```
Shortest distance from Planet_A:

Planet_B 1149
Planet_C 1098
Planet_D 608
Planet_E 1113
Planet_F 473
Planet_G 1049
Planet_H 259
Planet_I 678
Planet_J 584
```

Graph of Shortest Paths from Planet_A:



Program 3

Algorithm

program_3.cpp

```
DEFINE Edge pair<char, char>

START find (int x)
  IF fathers[x] == x THEN RETURN x
  RETURN find (fathers[x])
END find

START unite (int x, int y)
  int fx = find (x)
  int fy = find (y)
  fathers[fx] = fy
END unite

START kruskal (vector<Planet> vecPlanet)
  vector<pair<int, Edge>> edges

  make_pair of (distance, Edge(Planet 1 initial, Planet 2 initial)) and
  push_back to vector edges

  Sort vector edges in ascending order according to first element

  FOR i from 0 to 99
    father[i] = i
  NEXT i

  int mst_weight = 0, mst_edges = 0, mst_ni = 0
  int totalVertices = 10

  PRINT "KRUSKAL'S ALGORITHM" + two newline "Edge" + "      " + "Weight"
  + two newline

  WHILE mst_edges < totalVertices-1
    char a = edges[mst_ni].second.first
    char b = edges[mst_ni].second.second
    int w = edges[mst_ni].first

    IF find(a) != find(b) THEN
      unite (a, b)
      mst_weight += w

      PRINT "(" + a + ", " + b + ")" + "   " + w + newline
      increment{mst_edges}

  PRINT newline + "Minimum Cost Spanning Tree: " + mst_weight + newline
END kruskal
```

```
START main ()
   Initialize vector<Planet> vecPlanet

   Call subroutine getPlanetData (vecPlanet)

   Call subroutine calculateDistance (vecPlanet)

   Call subroutine kruskal (vecPlanet)

   RETURN 0
END main
```

Experimental Results

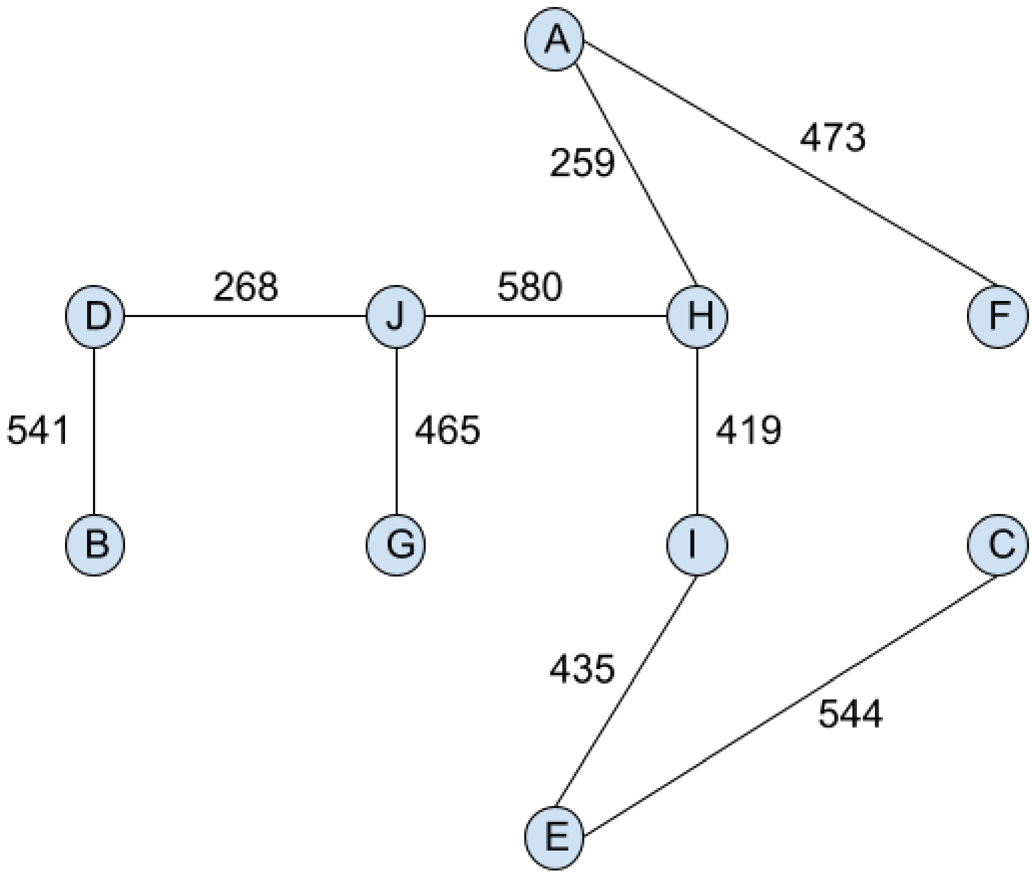
Kruskal's Algorithm:

```
KRUSKAL'S ALGORITHM

 Edge        Weight
(A, H)        259
(D, J)        268
(H, I)        419
(I, E)        435
(J, G)        465
(A, F)        473
(D, B)        541
(C, E)        544
(J, H)        580

Minimum Cost Spanning Tree: 3984
```

Graph of Minimum Spanning Tree:



Program 4

Algorithm

program_4.cpp

```
START knapsack (vector<Planet> planet)
  int knapsackTable[11][81]
  int num1, num2

  FOR i = 0 until 10
    FOR j = 0 until 80
      knapsackTable[i][j] = 0
    NEXT j
  NEXT i

  FOR i = 1 until 10
    FOR j = 0 until 80
      IF planet[i-1].weight <= 1 THEN
        num1 = knapsackTable[i - 1][j]
        num2 = knapsackTable[i - 1][j - planet[i - 1].weight] +
        planet[i - 1].profit
        knapsackTable[i][j] = max (num1, num2)
      ELSE
        knapsackTable[i][j] = knapsackTable[i - 1][j]
    NEXT j
  NEXT i

  int verticalKnapsackTable[81][11]

  FOR i = 0 until 10
    FOR j = 0 until 80
      verticalKnapsackTable[j][i] = knapsackTable[i][j]
    NEXT j
  NEXT i

  FOR i = 0 until 10
    PRINT i
  NEXT i

  PRINT three newline

  FOR i = 0 until 80
    PRINT i
    FOR j = 0 until 10
      PRINT verticalKnapsackTable[i][j]
    NEXT j
    PRINT newline
  NEXT i
END knapsack
```

```
START main ()
   Initialize vector<Planet> vecPlanet

   Call subroutine getPlanetData (vecPlanet)

   Call subroutine calculateDistance (vecPlanet)

   Call subroutine knapsack (vecPlanet)

   RETURN 0
END main
```

Experimental Results

Vertical 0/1 Knapsack Table (Top):

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	130	130	130	130	130
7	0	0	180	180	180	180	180	180	180	180	180
8	0	0	180	180	180	180	180	180	180	180	180
9	0	0	180	180	180	180	180	180	180	180	180
10	0	0	180	180	180	180	180	180	180	180	180
11	0	0	180	180	180	180	180	180	180	180	180
12	0	0	180	180	180	180	180	180	180	180	180
13	0	0	180	180	180	180	310	310	310	310	310
14	0	0	180	180	180	240	310	310	310	310	310
15	0	0	180	180	180	240	310	310	310	310	310
16	0	0	180	180	180	240	310	310	310	310	310
17	0	0	180	180	190	240	310	310	310	310	310
18	0	0	180	180	190	240	310	310	310	310	310
19	0	0	180	180	190	240	310	310	310	310	310
20	0	0	180	180	190	240	370	370	370	370	370
21	0	0	180	250	250	250	370	370	370	370	410
22	0	0	180	250	250	250	370	370	370	370	410
23	0	0	180	250	250	250	370	370	370	370	410
24	0	0	180	250	370	370	370	390	390	390	410
25	0	0	180	250	370	370	370	390	390	390	410
26	0	0	180	250	370	370	370	390	390	390	410
27	0	0	180	250	370	370	380	390	390	470	470
28	0	0	180	250	370	370	380	390	390	470	470
29	0	0	180	250	370	370	380	390	390	470	470
30	0	0	180	250	370	370	500	500	500	500	500
31	0	0	180	250	370	430	500	500	500	500	500
32	0	0	180	250	370	430	500	500	500	500	500
33	0	0	180	250	370	430	500	500	500	500	500
34	0	0	180	250	370	430	500	500	500	530	530
35	0	0	180	250	370	430	500	500	500	530	570
36	0	0	180	250	370	430	500	500	500	530	570
37	0	0	180	250	370	430	560	560	560	560	570
38	0	0	180	250	440	440	560	560	560	560	600
39	0	0	180	250	440	440	560	560	560	560	600
40	0	0	180	250	440	440	560	560	560	560	600
41	0	0	180	250	440	440	560	580	580	580	600
42	0	0	180	250	440	440	560	580	580	580	630
43	0	0	180	250	440	440	560	580	580	580	630
44	0	0	180	250	440	440	570	580	580	660	660
45	0	0	180	250	440	500	570	580	580	660	660
46	0	0	180	250	440	500	570	580	580	660	660
47	0	0	180	250	440	500	570	580	610	660	660
48	0	0	180	250	440	500	570	640	640	660	660
49	0	0	180	250	440	500	570	640	640	660	680
50	0	0	180	250	440	500	570	640	640	660	680
51	0	0	180	250	440	500	630	640	640	720	720
52	0	0	180	250	440	500	630	640	640	720	760
53	0	0	180	250	440	500	630	640	640	720	760
54	0	0	180	250	440	500	630	640	640	720	760
55	0	0	180	250	440	500	630	650	650	740	760
56	0	0	180	250	440	500	630	650	650	740	760
57	0	0	180	250	440	500	630	650	650	740	760
58	0	0	180	250	440	500	630	650	690	740	760
59	0	0	180	250	440	500	630	650	690	740	820
60	0	0	180	250	440	500	630	650	690	740	820

Vertical 0/1 Knapsack Table (Bottom):

61	0	0	180	250	440	500	630	650	690	770	820
62	0	0	180	250	440	500	630	710	710	800	820
63	0	0	180	250	440	500	630	710	710	800	840
64	0	0	180	250	440	500	630	710	710	800	840
65	0	0	180	250	440	500	630	710	710	800	840
66	0	0	180	250	440	500	630	710	710	800	840
67	0	0	180	250	440	500	630	710	710	800	840
68	0	0	180	250	440	500	630	710	710	800	840
69	0	0	180	250	440	500	630	710	710	810	870
70	0	0	180	250	440	500	630	710	710	810	900
71	0	0	180	250	440	500	630	710	710	810	900
72	0	0	180	250	440	500	630	710	760	850	900
73	0	0	180	250	440	500	630	710	760	850	900
74	0	0	180	250	440	500	630	710	760	850	900
75	0	0	180	250	440	500	630	710	760	850	900
76	0	0	180	250	440	500	630	710	760	870	900
77	0	0	180	250	440	500	630	710	760	870	910
78	0	0	180	250	440	500	630	710	760	870	910
79	0	0	180	250	440	500	630	710	760	870	910
80	0	0	180	250	440	500	630	710	760	870	950

Planets to Visit with Weights, Profits and Benefits:

Planet	Weight	Profit	Benefit
A	0	0	0
F	6	130	130
H	10	50	180
J	8	100	280
D	17	190	470
B	7	180	650
G	11	80	730
I	14	160	890
E	7	60	950

Table 3: Shows the weight and profit for each planet and the benefits as the spaceship travels from Planet A to Plant E.

Conclusion:

The maximum benefit the spaceship could take is 950.