CCP 6214 – ALGORITHM DESIGN & ANALYSIS ASSIGNMENT

Lecture Section: TC4L

Tutorial Section: TT13L

Group No.: 6

Num	Student ID	Student Name	Task descriptions	Percentage %
1	1211102398	Nicholas Tiow Kai Bo (Group Leader)	Q4: 0/1 Knapsack	25
2	1211101699	Low Kai Yan	Q3: Dijkstra's and Kruskal	25
3	1211101452	Wong Jui Hong	Q2: Heap and Selection Sort	25
4	1211102080	Wong Wei Ping	Q1: Dataset 1 and 2	25

1 Table of Contents

Table of Contents

2 Q1: Dataset 1 and 2

2.1 Dataset 1 Generation

2.1.1 Algorithm

```cpp
// Extract all the digits from the ID
vector<int> extractDigits(int id) {
    vector<int> digits;
    while (id > 0) {
        digits.push_back(id % 10);
        id /= 10;
    }
    return digits;
}
```

- **extractDigits:** Function to extract digits from an integer id.
- **vector<int> digits:** Vector to store the digits.
- **while (id > 0):** Loop to extract digits from id using modulo and division operations, pushing them into the digits vector.
- **return digits:** Return the vector containing the digits.

```cpp
// Generate all the posible combinations from the extracted digits in 2, 3, 4 digits
vector<int> generateCombinations(vector<int>& digits) {
    vector<int> combinations;
    // 2-digits combination
    for(size_t i = 0; i < digits.size(); ++i) {
        for(size_t j = 0; j < digits.size(); ++j) {
            if(i != j) {
                combinations.push_back((digits[i]*10) + digits[j]);
            }
        }
    }
    //3-digits combination
    for(size_t i = 0; i < digits.size(); ++i){
        for(size_t j = 0; j < digits.size(); ++j) {
            for(size_t k = 0; k < digits.size(); ++k) {
                if(i != j && i != k && i != k) {
                    combinations.push_back((digits[i] * 100) + (digits[j] * 10) + digits[k]);
                }
            }
        }
    }
    //4-digits combinations
    for(size_t i = 0; i < digits.size(); ++i) {
        for(size_t j = 0; j < digits.size(); ++j) {
            for(size_t k = 0; k < digits.size(); ++k) {
                for(size_t l = 0; l < digits.size(); l++) {
                    if(i != j && i != k && i != l && j != k && j != l && k != l) {
                        combinations.push_back((digits[i] * 1000) + (digits[j] * 100) + (digits[k] * 10) + digits[l]);
                    }
                }
            }
        }
    }

    return combinations;
}
```

- **generateCombinations:** Function to generate combinations of digits.
- **vector<int> combinations:** Vector to store the combinations.
- **2-digits combination:** Nested loops to create 2-digit combinations, ensuring no repeated digits.
- **3-digits combination:** Nested loops to create 3-digit combinations, ensuring no repeated digits.
- **4-digits combination:** Nested loops to create 4-digit combinations, ensuring no repeated digits.
- **return combinations:** Return the vector containing all the generated combinations.

```
57  // Get a random number combinations
58  int getRandomCombinations(const vector<int>& combinations) {
59      int randomIdx = rand() % combinations.size();
60      return combinations[randomIdx];
61  }
```

- **getRandomCombinations:** Function to get a random combination from the combinations vector.
- **rand() % combinations.size():** Generate a random index within the range of the combinations vector.
- **return combinations[randomIdx]:** Return the combination at the random index.

```
64  // Generate datasets
65  vector<vector<int>> generateDataset(vector<int>& combinations) {
66      srand(time(0));
67      vector<vector<int>> datasets;
68      vector<int> sizes = {100, 1000, 10000, 100000, 250000, 500000};
69
70      for(int size: sizes) {
71          vector<int> dataset;
72          for(size_t i = 0; i < size; ++i) {
73              dataset.push_back(getRandomCombinations(combinations));
74          }
75          datasets.push_back(dataset);
76      }
77
78      return datasets;
79  }
```

- **generateDataset:** Function to generate datasets of various sizes.
- **srand(time(0)):** Seed the random number generator with the current time to ensure different results each run.
- **vector<vector<int>> datasets:** Vector to store multiple datasets.
- **vector<int> sizes:** Sizes for different datasets.
- **for(int size: sizes):** Loop through each size.
- **for(size_t i = 0; i < size; ++i):** Generate size number of random combinations.
- **datasets.push_back(dataset):** Store each generated dataset in the datasets vector.
- **return datasets:** Return the vector containing all datasets.

```
82  void storeDatasets(const vector<vector<int>>& datasets) {
83      for(size_t i = 0; i < datasets.size(); ++i) {
84          string fileName = "./dataset1/dataset_" + to_string(i+1) + ".txt";
85          ofstream outfile(fileName);
86          for(int value: datasets[i]) {
87              outfile << value << endl;
88          }
89          outfile.close();
90          cout << "Dataset " << i + 1 << " stored in " << fileName << endl;
91      }
92  }
```

- **storeDatasets:** Function to store datasets into files.
- **for(size_t i = 0; i < datasets.size(); ++i):** Loop through each dataset.
- **string fileName:** Create a filename for each dataset file.
- **ofstream outfile(fileName):** Open the file for writing.
- **for(int value: datasets[i]):** Write each value in the dataset to the file.
- **outfile.close():** Close the file.
- **cout << "Dataset " << i + 1 << " stored in " << fileName << endl:** Print a message indicating that the dataset has been stored.

```
96  int main() {
97      int leaderID = 1211102398;
98
99      vector<int> digits = extractDigits(leaderID);
100
101     vector<int> combinations = generateCombinations(digits);
102
103     vector<vector<int>> datasets = generateDataset(combinations);
104
105     storeDatasets(datasets);
106
107     return 0;
108 }
```

- **int main():** The main function of the program.
- **int leaderID = 1211102398:** Initialize the leader ID.
- **vector<int> digits = extractDigits(leaderID):** Extract digits from the leader ID.
- **vector<int> combinations = generateCombinations(digits):** Generate combinations from the extracted digits.
- **vector<vector<int>> datasets = generateDataset(combinations):** Generate datasets from the combinations.
- **storeDatasets(datasets):** Store the generated datasets into files.
- **return 0:** Return 0 to indicate successful execution.

2.1.2 Program

The dataset 1 generation process is generated via the below program.

When the user runs the program, the program will auto generate random numbers for 6 different datasets of sizes as follows:

Set 1: 100
Set 2: 1,000
Set 3: 10,000
Set 4: 100,000
Set 5: 250,000
Set 6: 500,000

The auto generated random numbers for 6 different datasets are based on the group leader's student ID (1211102398), in which for this case the random numbers can only compose of 0, 1, 2, 3, 8, and 9. The program will then store the random numbers into different .txt files based on the size of datasets mentioned above. For example, **Dataset 1** is stored in **dataset_1.txt** under folder named **dataset1** and so on.

```
Dataset 1 stored in ./dataset1/dataset_1.txt
Dataset 2 stored in ./dataset1/dataset_2.txt
Dataset 3 stored in ./dataset1/dataset_3.txt
Dataset 4 stored in ./dataset1/dataset_4.txt
Dataset 5 stored in ./dataset1/dataset_5.txt
Dataset 6 stored in ./dataset1/dataset_6.txt
```

Example of **Dataset 1**

2281	2011	8129	220	1918	10	3012	3112	2139	(91) more…

Example of **Dataset 2**

3011	9111	3121	8101	931	1130	9131	3111	1113	(991) more…

Example of **Dataset 3**

3901	2139	9121	1022	1232	191	2318	1238	3112	(9991) more…

Example of **Dataset 4**

21	931	9212	3182	8201	311	211	921	1389	(99991) more…

Example of **Dataset 5**

381	2811	1198	8	8902	931	1301	8119	3201	(249991) more…

Example of **Dataset 6**

2011	1118	3210	9028	3201	2211	3280	1122	1181	(499991) more…

2.2 Dataset 2 Generation

2.2.1 Algorithm

```
// Function to generate star data
vector<Star> generateStars(int numberOfStars, const vector<int>& combinations) {
    srand(time(0));
    vector<Star> stars;
    vector<string> alps = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T"};
    for (int i = 0; i < numberOfStars; ++i) {
        Star star;
        star.name = "Star_" + alps[i];
        star.x = getRandomCombination(combinations);
        star.y = getRandomCombination(combinations);
        star.z = getRandomCombination(combinations);
        star.weight = 10 + getRandomCombination(combinations) % 300;
        star.profit = 10 + getRandomCombination(combinations) % 300;
        stars.push_back(star);
    }
    return stars;
}
```

- **generateStars:** Function to generate data for stars.
- **srand(time(0)):** Seed the random number generator with the current time to ensure different results each run.
- **vector<Star> stars:** Vector to store the stars.
- **vector<string> alps:** Vector containing alphabet characters for naming stars.
- **for (int i = 0; i < numberOfStars; ++i):** Loop to generate numberOfStars stars.
- **Star star:** Create a new Star object.
- **star.name:** Set the name of the star using the alphabet.
- **star.x, star.y, star.z:** Assign random coordinates to the star.
- **star.weight, star.profit:** Assign random weight and profit to the star, ensuring a minimum value of 10, result in values within the range 10 to 309.
- **stars.push_back(star):** Add the star to the stars vector.
- **return stars:** Return the vector containing all generated stars.

```cpp
// Function to generate edges ensuring each star connects to at least 3 others
vector<pair<string, string>> generateEdges(const vector<Star>& stars, int numberOfEdges) {
    vector<pair<string, string>> edges;
    vector<int> degree(stars.size(), 0);
    srand(time(0));

    // Ensure each star has at least 3 connections
    for (size_t i = 0; i < stars.size(); ++i) {
        while (degree[i] < 3) {
            int target;
            do {
                target = rand() % stars.size();
            } while (target == i || find(edges.begin(), edges.end(), make_pair(stars[i].name, stars[target].name)) != edges.end() ||
                     find(edges.begin(), edges.end(), make_pair(stars[target].name, stars[i].name)) != edges.end());

            edges.push_back(make_pair(stars[i].name, stars[target].name));
            degree[i]++;
            degree[target]++;
        }
    }

    // Generate remaining edges
    while (edges.size() < numberOfEdges) {
        int from = rand() % stars.size();
        int to;
        do {
            to = rand() % stars.size();
        } while (from == to || find(edges.begin(), edges.end(), make_pair(stars[from].name, stars[to].name)) != edges.end() ||
                 find(edges.begin(), edges.end(), make_pair(stars[to].name, stars[from].name)) != edges.end());

        edges.push_back(make_pair(stars[from].name, stars[to].name));
    }

    return edges;
}
```

- **generateEdges:** Function to generate edges between stars.
- **vector<pair<string, string>> edges:** Vector to store edges as pairs of star names.
- **vector<int> degree(stars.size(), 0)**: Vector to keep track of the degree (number of connections) of each star.
- **srand(time(0)):** Seed the random number generator.
- **Ensure each star has at least 3 connections:** Loop through each star and connect it to other stars until it has at least 3 connections.
- **while (...):** Ensure there is no duplicate connection from stars[target] to stars[i] and stars[i] to stars[target].
- **degree[i]++:** Increment the degree of the current star.
- **degree[target]++:** Increment the degree of the target star.
- **Generate remaining edges:** Continue random generating edges until the total number of edges equals numberOfEdges.

```cpp
// Function to calculate distance between sconnected stars
string calculateDistance(const Star& star1, const Star& star2) {
    double distance = sqrt(pow(star2.x - star1.x, 2) + pow(star2.y - star1.y, 2) + pow(star2.z - star1.z, 2));

    ostringstream oss;
    oss << fixed << setprecision(1) << distance;
    return oss.str();
}
```

- **calculateDistance:** Function to calculate the Euclidean distance between two stars.
- **double distance:** Calculate the distance using the Euclidean formula.
- **ostringstream oss:** Use a string stream to format the distance to one decimal place.
- **return oss.str():** Return the formatted distance as a string.

```
// Function to store stars and edges into text files
void storeData(const vector<Star>& stars, const vector<pair<string, string>>& edges) {
    ofstream starFile("./dataset2/stars.txt");
    for (const auto& star : stars) {
        starFile << star.name << " " << star.x << " " << star.y << " " << star.z << " " << star.weight << " " << star.profit << endl;
    }
    starFile.close();

    ofstream edgeFile("./dataset2/edges.txt");
    for (const auto& edge : edges) {
        int fromIndex = edge.first.back() - 'A';
        int toIndex = edge.second.back() - 'A';
        edgeFile << edge.first << " " << edge.second << " " << calculateDistance(stars[fromIndex], stars[toIndex]) << endl;
    }
    edgeFile.close();
}
```

- **storeData:** Function to store star and edge data into text files.
- **ofstream starFile("./dataset2/stars.txt"):** Open a file to write star data.
- **for (const auto& star : stars):** Loop through each star and write its data to the file.
- **starFile.close():** Close the star file.
- **ofstream edgeFile("./dataset2/edges.txt"):** Open a file to write edge data.
- **for (const auto& edge : edges):** Loop through each edge and write its data with calculated distance to the file, including the distance between stars.
- **edgeFile.close():** Close the edge file.

```
void printStars(const vector<Star>& stars) {
    // Print table header
    cout << "\n\nStars Information:" << endl;
    cout << string(70, '-') << endl;

    cout << left << setw(5) << "Idx"
         << left << setw(15) << "Name"
         << right << setw(10) << "X"
         << right << setw(10) << "Y"
         << right << setw(10) << "Z"
         << right << setw(10) << "Weight"
         << right << setw(10) << "Profit" << endl;

    cout << string(70, '-') << endl;

    // Print each star information in a formatted way
    for (size_t i = 0; i < stars.size(); ++i) {
        const auto& star = stars[i];
        cout << left << setw(5) << i
             << left << setw(15) << star.name
             << right << setw(10) << star.x
             << right << setw(10) << star.y
             << right << setw(10) << star.z
             << right << setw(10) << star.weight
             << right << setw(10) << star.profit << endl;
    }
}
```

- **printStars:** Function to print star data in a formatted way.
- **Print table header:** Print the header of the table.
- **for (size_t i = 0; i < stars.size(); ++i):** Loop through each star and print its data.

2.2.2 Program

The dataset 2 generation process is generated via the below program.

At first, the program will prompt user to choose whether they want to enter the 3 student IDs themselves or use the default IDs.

```
Dataset2 Random Generator
--------------------------------------------------
Enter 1 to enter student IDs manually or 2 to use default IDs:
```

In this case, the user chooses 1 to enter student IDs manually.

```
Dataset2 Random Generator
--------------------------------------------------
Enter 1 to enter student IDs manually or 2 to use default IDs: 1
```

The group members' student IDs (except for the group leader's student ID) are then being inputted:

1. 1211101699
2. 1211101452
3. 1211102080

The program then will add up the 3 group members' student IDs and shows up the resulting Group ID sum.

```
Dataset2 Random Generator
--------------------------------------------------
Enter 1 to enter student IDs manually or 2 to use default IDs: 1
Enter the 1st student ID: 1211101699
Enter the 2nd student ID: 1211101452
Enter the 3rd student ID: 1211102080
--------------------------------------------------
Student IDs: 1211101699 + 1211101452 + 1211102080
Group ID sum: 3633305231
--------------------------------------------------
```

The data is then generated from the Group ID sum obtained. We have the 5 columns being the x, y, and z coordinates, weight, and profit of the corresponding stars. The values for these 5 columns are auto generated from the Group ID sum (3633305231) in which for this case the values can only compose of the integers of 0, 1, 2, 3, 5, and 6.

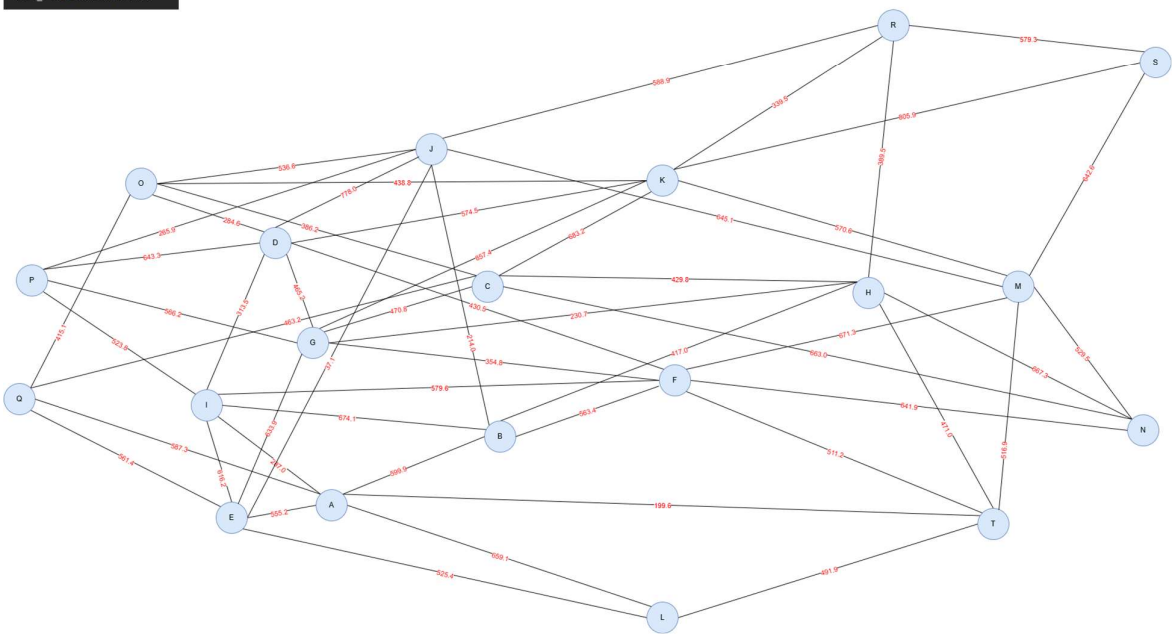
```
Stars Information:
------------------------------------------------------------------------
Idx  Name                 X         Y         Z    Weight    Profit
------------------------------------------------------------------------
0    Star_A              26        52       126        36       170
1    Star_B              12       520       501        13       160
2    Star_C             260       335       562       215        46
3    Star_D             533       253        30        11        62
4    Star_E              52       335       603        30       216
5    Star_F             313       623        36        33        23
6    Star_G             536       623       312        31        15
7    Star_H             320       605       233        15        40
8    Star_I             313        50       123        42       271
9    Star_J              15       333       605        75       261
10   Star_K              10        16        12        71       143
11   Star_L             330       621       261        15        40
12   Star_M             501        36       302       211        33
13   Star_N              62        33         6       140        15
14   Star_O             361       156       235        26       220
15   Star_P             260       230       612       135        63
16   Star_Q             315       562       162        42       172
17   Star_R             235       265        63       223       240
18   Star_S             236       561       561       163        42
19   Star_T              32       233       210       270        36
```

stars.txt

File Edit View

```
Star_A 26 52 126 36 170
Star_B 12 520 501 13 160
Star_C 260 335 562 215 46
Star_D 533 253 30 11 62
Star_E 52 335 603 30 216
Star_F 313 623 36 33 23
Star_G 536 623 312 31 15
Star_H 320 605 233 15 40
Star_I 313 50 123 42 271
Star_J 15 333 605 75 261
Star_K 10 16 12 71 143
Star_L 330 621 261 15 40
Star_M 501 36 302 211 33
Star_N 62 33 6 140 15
Star_O 361 156 235 26 220
Star_P 260 230 612 135 63
Star_Q 315 562 162 42 172
Star_R 235 265 63 223 240
Star_S 236 561 561 163 42
Star_T 32 233 210 270 36
```

In this **stars.txt** file, we have the first column as the name of the stars (20 stars in total) while the second, third and fourth columns are the stars' coordinates (x, y, z) respectively. The fifth column is the stars' weight and the sixth and last column is the stars' profit (weight and profit refer to the points to be collected from conquering the star).



Star Location Design with All Routes (2D Projection)

edges.txt

File Edit View

```
Star_A Star_Q 587.3
Star_A Star_B 599.9
Star_A Star_E 555.2
Star_B Star_J 214.0
Star_B Star_I 674.1
Star_C Star_G 470.8
Star_C Star_Q 463.2
Star_C Star_N 663.0
Star_D Star_G 465.2
Star_D Star_F 430.5
Star_D Star_I 313.5
Star_E Star_L 525.4
Star_E Star_I 616.2
Star_F Star_N 641.9
Star_F Star_T 511.2
Star_G Star_P 566.2
Star_H Star_R 389.5
Star_H Star_B 417.0
Star_H Star_T 471.0
Star_J Star_R 588.9
Star_J Star_O 536.6
Star_K Star_R 339.5
Star_K Star_O 438.8
Star_K Star_M 570.6
Star_L Star_T 491.9
Star_L Star_A 659.1
Star_M Star_N 529.5
Star_M Star_J 645.1
Star_O Star_C 386.2
Star_P Star_D 643.3
Star_P Star_I 523.8
Star_Q Star_O 415.1
Star_S Star_R 579.3
Star_S Star_M 642.6
Star_S Star_K 805.9
Star_T Star_A 199.6
Star_F Star_G 354.8
Star_F Star_I 579.6
Star_C Star_H 429.8
Star_K Star_G 857.4
Star_F Star_B 563.4
Star_K Star_D 574.5
Star_F Star_M 671.3
Star_G Star_E 633.9
Star_A Star_I 287.0
Star_G Star_H 230.7
Star_K Star_C 683.2
Star_E Star_J 37.1
Star_Q Star_E 561.4
Star_D Star_O 284.6
Star_M Star_T 516.9
Star_P Star_J 265.9
Star_J Star_D 778.0
Star_H Star_N 667.3
```

We have also created an **edges.txt** file which stores the edges (54 edges in total) from one node to another as the image, **Star Location Design with All Routes (2D Projection)** shown above. We have the first column and second column to be the stars' name that are being connected to each other. The third column is the distance between the two connected stars. The distance is calculated using the formula below:

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

For example, the first row, means that Star_A is connected Star_R with the distance between them of 192.9. Same goes to the following rows.

3 Q2: Heap and Selection Sort

Note: *All the data involved for this* ***Q2: Heap and Selection*** *are based on the* ***Dataset 1 Generation*** *(6 different datasets generated based on the group leader's student ID – 1211102398 in this case).*

3.1 Heap Sort

3.1.1 Algorithm

This algorithm leverages a priority queue implemented as a min-heap to perform heap sort. The primary purpose of this algorithm is to efficiently sort datasets by first inserting all data for 6 different datasets into a priority queue that implements the min-heap concept, which ensures that the smallest element is always at the top. After all elements are inserted, the algorithm repeatedly dequeues the smallest element, thereby creating a sorted list for each 6 different datasets. The algorithm also measures and stores the time taken for each sorting phase, highlighting its efficiency across various dataset sizes.

Filenames of datasets to be read

```
50    // Filenames of datasets to be read
51    vector<string> filenames = {
52        "./Q1_dataset1Output/dataset_1.txt", "./Q1_dataset1Output/dataset_2.txt", "./Q1_dataset1Output/dataset_3.txt",
53        "./Q1_dataset1Output/dataset_4.txt", "./Q1_dataset1Output/dataset_5.txt", "./Q1_dataset1Output/dataset_6.txt"
54    };
55
```

Purpose: Define the list of **filenames** for the datasets that will be read.
Data Structure: A vector of strings, where each string is the path to a dataset file.

Read datasets

```
56    // Read datasets
57    vector<vector<int>> datasets = readDatasets(filenames);
```

Function Call: readDatasets(filenames)
Purpose: Read the datasets from the specified files.
Return Value: A vector of **vector<int>** where each inner vector contains the data from one dataset file.

Measure time for inserting all data into the priority queue

```
59  // Measure time for inserting all data into the priority queue
60  auto start_insert = chrono::high_resolution_clock::now();
61  vector<priority_queue<int, vector<int>, greater<int>>> heaps;
62
63  for (const auto& dataset : datasets) {
64      priority_queue<int, vector<int>, greater<int>> heap;
65      for (const auto& elem : dataset) {
66          heap.push(elem);
67      }
68      heaps.push_back(move(heap));
69  }
70
71  auto end_insert = chrono::high_resolution_clock::now();
72  chrono::duration<double> duration_insert = end_insert - start_insert;
73  cout << "Time taken to insert all data into the priority queue: " << duration_insert.count() << " seconds." << endl;
```

Purpose: Measure and output the time taken to insert all data into heaps (min-heaps).
Parameters (vector<priority_queue<int, vector<int>, greater<int>>> heaps): A vector of min-heaps (priority queues) for each dataset.
Steps:

1. **Start timer (Line 60): start_insert** records the start time.
2. **Loop through each dataset (Line 63 - 69):**
   a. Create a min-heap for each dataset.
   b. Insert each element of the dataset into the min-heap using **heap.push(elem)**.
   c. Move the heap into the heaps vector.
3. **end_insert (Line 72):** Records the end time.
4. **duration_insert (Line 72):** Elapsed time.
5. **Output time (Line 73):** Print the time taken for insertion.

Measure time for dequeuing all data from the priority queue

```
76  // Measure time for dequeuing all data from the priority queue, row by row
77  vector<vector<int>> sorted_datasets;
78  vector<double> sortingTimes;
79
80  for (size_t i = 0; i < heaps.size(); ++i) {
81      auto start_dequeue = chrono::high_resolution_clock::now();
82      vector<int> sorted_dataset;
83
84      while (!heaps[i].empty()) {
85          sorted_dataset.push_back(heaps[i].top());
86          heaps[i].pop();
87      }
88
89      auto end_dequeue = chrono::high_resolution_clock::now();
90      chrono::duration<double> duration_dequeue = end_dequeue - start_dequeue;
91      cout << "Time taken to dequeue dataset " << i + 1 << " from the priority queue: " << duration_dequeue.count() << " seconds." << endl;
92
93      sortingTimes.push_back(duration_dequeue.count());
94      sorted_datasets.push_back(sorted_dataset);
95  }
```

Purpose: Measure and output the time taken to dequeue (extract) all data from the heaps (thus sorting the datasets).
Parameters:

- **vector<vector<int>> sorted_datasets:** Stores the sorted datasets.
- **vector<double> sortingTimes:** Stores the time taken to sort each dataset.

Steps:

Loop through each heap (Line 80 - 95):

1. **Start timer: start_dequeue** records the start time for dequeuing.
2. **Extract elements:** While the heap is not empty, extract the top element (smallest element due to min-heap) and add it to **sorted_dataset**.
3. **End timer: end_dequeue** records the end time for dequeuing.
4. **Calculate duration: duration_dequeue** is the elapsed time for dequeuing.
5. **Output time:** Print the time taken for dequeuing.
6. Add the sorting duration to **sortingTimes**.
7. Add the sorted dataset to **sorted_datasets**.

3.1.2 Program for Algorithm Implementation

When we run the program named **Q2_heapSort.cpp**, it will start to insert all the data for 6 different datasets into their respective priority queues and displays the time taken to insert all data into the priority queues. Then, the program will start to dequeue the data for the 6 different datasets from their respective priority queues and display the time taken to dequeue the data from each priority queue. The program will finally store the 6 sorted datasets into their respective **.txt** files, e.g., sorted dataset 1 using heap sort stored in **heap_sorted_dataset_1.txt** and so on for the rest. The resulting sorting time for the 6 different datasets will be stored in **heap_sorting_times.txt**.

```
Time taken to insert all data into the priority queue: 0.0760521 seconds.
Time taken to dequeue dataset 1 from the priority queue: 2.67e-05 seconds.
Time taken to dequeue dataset 2 from the priority queue: 0.0003008 seconds.
Time taken to dequeue dataset 3 from the priority queue: 0.0038111 seconds.
Time taken to dequeue dataset 4 from the priority queue: 0.0505537 seconds.
Time taken to dequeue dataset 5 from the priority queue: 0.136013 seconds.
Time taken to dequeue dataset 6 from the priority queue: 0.283869 seconds.
Sorted dataset 1 using heap stored in ./Q2_AnswerOutput/heapSorted_dataset1/heap_sorted_dataset_1.txt
Sorted dataset 2 using heap stored in ./Q2_AnswerOutput/heapSorted_dataset1/heap_sorted_dataset_2.txt
Sorted dataset 3 using heap stored in ./Q2_AnswerOutput/heapSorted_dataset1/heap_sorted_dataset_3.txt
Sorted dataset 4 using heap stored in ./Q2_AnswerOutput/heapSorted_dataset1/heap_sorted_dataset_4.txt
Sorted dataset 5 using heap stored in ./Q2_AnswerOutput/heapSorted_dataset1/heap_sorted_dataset_5.txt
Sorted dataset 6 using heap stored in ./Q2_AnswerOutput/heapSorted_dataset1/heap_sorted_dataset_6.txt
Sorting times stored in ./Q2_AnswerOutput/heapSorted_dataset1/heap_sorting_times.txt
```

3.1.3 Experimental Results

Figure 3.1 shows the graph of timing vs dataset size for our heap sort.

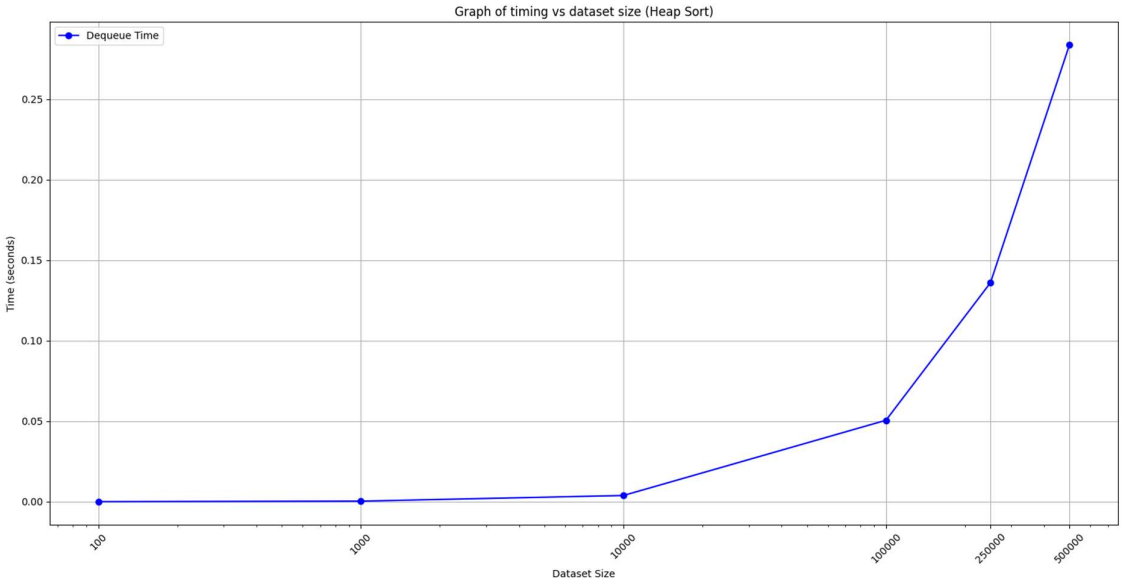


Figure 3.1 Graph of timing vs dataset size for Heap Sort

3.1.4 Discussions

3.1.4.1 Time Complexity

Insertion Time Complexity

1. **Heap insertion:** Inserting an element into a binary heap has a time complexity of **O(log n)**, where **n** is the number of elements already in the heap.
2. **Total insertion for n elements:** For inserting **n** elements, the overall time complexity is **O(n log n)**.

Dequeue Time Complexity

1. **Heap dequeue:** Removing the smallest element (the root) from a heap also has a time complexity of **O(log n)**, because it requires reheapification of the remaining elements.
2. **Total dequeue for n elements:** Similar to insertion, dequeuing **n** elements has an overall time complexity of **O(n log n)**.

Thus, the overall time complexity of this heap sort, including both insertion and dequeue operations, is **O(n log n)**.

3.1.4.2 Space Complexity

1. **In-place sorting:** The space complexity of heap sort is **O(1)** if we are using an array-based binary heap because it can be done in-place without requiring extra space for another array.
2. **Priority queue implementation:** If using an explicit priority, the space complexity is **O(n)** for storing **n** elements.

3.1.4.3 Discussion Based on Figure 3.1

Figure 3.1 illustrates the time taken to dequeue datasets from the priority queue versus the dataset size based on our heap sort algorithm. The graph shows an upward trend, which aligns with the expected **O(n log n)** time complexity for heap sort. Here are some key observations:

1. **Initial low times**
   For smaller datasets (sizes 100, 1000), the dequeue times are very low, almost negligible. This is typical since the **O(log n)** component of the complexity is very small for small **n**.

2. **Exponential growth**
   As the dataset size increases, the time taken to dequeue the elements increase exponentially. This is consistent with the logarithmic component in the time complexity, which grows slowly at first but accelerates as **n** becomes large.

3. **Large datasets**
   For very large datasets (sizes 250000, 500000), the dequeue time becomes significantly higher. This shows the impact of the **log n** factor, where even though **log n** grows slower than **n**, its effect becomes more pronounced with very large values of **n**.

3.1.4.4 Conclusion

Heap sort exhibits a time complexity of **O(n log n)**, which is efficient and suitable for a wide range of datasets. The space complexity is **O(1)** for in-place sorting using an array-based heap, or **O(n)** if using an explicit priority queue data structure. Figure 3.1 confirms the theoretical time complexity, showing a slow initial growth and a rapid increase in dequeue time as the dataset size increases, consistent with the logarithmic factor in the heap sort algorithm.

3.2 Selection Sort

3.2.1 Algorithm

This Selection Sort algorithm is a simple comparison-based sorting algorithm that works by repeatedly finding the minimum element from the unsorted portion of the array and swapping it with the first unsorted element. This process continues until the entire array is sorted. In short, this algorithm is used to sort the data for 6 different datasets. The algorithm reads datasets from text files, sorts each dataset using Selection Sort, measures and stores the time taken for each sort operation, and then saves the sorted datasets and their corresponding sorting times to their respective **.txt** files. The primary purpose of this algorithm is to demonstrate and measure the performance of Selection Sort on the 6 different datasets, providing insights into its time complexity and efficiency.

Selection sort function

```
7  // Selection Sort function
8  template<typename T>
9  void selectionSort(std::vector<T>& arr) {
10     int n = arr.size();
11     for (int i = 0; i < n - 1; ++i) {
12         int minIndex = i;
13         for (int j = i + 1; j < n; ++j) {
14             if (arr[j] < arr[minIndex])
15                 minIndex = j;
16         }
17         std::swap(arr[i], arr[minIndex]);
18     }
19 }
```

Purpose: Sort a vector using the selection sort algorithm.
Template function: The function is templated to work with any data type T.
Steps:

1. Loop through each element except the last one.
2. Find the minimum element in the unsorted part of the vector.
3. Swap the minimum element with the current element.

Sort datasets and measure time

```
70     // Vector to store sorting times for each dataset
71     std::vector<double> sortingTimes;
72
73     // Sort datasets using Selection Sort and record time for each dataset
74     for (auto& dataset : datasets) {
75         auto start = std::chrono::high_resolution_clock::now();
76         selectionSort(dataset);
77         auto end = std::chrono::high_resolution_clock::now();
78         std::chrono::duration<double> duration = end - start;
79         sortingTimes.push_back(duration.count());
80     }
```

Purpose: Sorting datasets using selection sort, measuring sorting times.
Steps:
Loop through each dataset (Line 74 - 80):

1. Start the timer before sorting.
2. Call **selectionSort** function to sort the dataset.
3. Stop the timer after sorting.
4. Calculate the duration and store it in the **sortingTimes** vector.

3.2.2 Program for Algorithm Implementation

When we run the program named **Q2_selection_sort.cpp**, the program will start to sort the data for the 6 different datasets using the selection sort algorithm as mentioned in 3.2.1 and display the time taken to sort the data for each dataset. The program will finally store the 6 sorted datasets into their respective **.txt** files, e.g., sorted dataset 1 using selection sort stored in **selection_sorted_dataset_1.txt** and so on for the rest. The resulting sorting time for the 6 different datasets will be stored in **selection_sorting_times.txt**.

```
Time taken to sort dataset 1: 2.68e-05 seconds.
Time taken to sort dataset 2: 0.0021401 seconds.
Time taken to sort dataset 3: 0.215863 seconds.
Time taken to sort dataset 4: 20.6428 seconds.
Time taken to sort dataset 5: 137.846 seconds.
Time taken to sort dataset 6: 549.83 seconds.
Sorted dataset 1 using selection stored in ./Q2_AnswerOutput/selectionSorted_dataset1/selection_sorted_dataset_1.txt
Sorted dataset 2 using selection stored in ./Q2_AnswerOutput/selectionSorted_dataset1/selection_sorted_dataset_2.txt
Sorted dataset 3 using selection stored in ./Q2_AnswerOutput/selectionSorted_dataset1/selection_sorted_dataset_3.txt
Sorted dataset 4 using selection stored in ./Q2_AnswerOutput/selectionSorted_dataset1/selection_sorted_dataset_4.txt
Sorted dataset 5 using selection stored in ./Q2_AnswerOutput/selectionSorted_dataset1/selection_sorted_dataset_5.txt
Sorted dataset 6 using selection stored in ./Q2_AnswerOutput/selectionSorted_dataset1/selection_sorted_dataset_6.txt
Sorting times stored in ./Q2_AnswerOutput/selectionSorted_dataset1/selection_sorting_times.txt
```

3.2.3 Experimental Results

Figure 3.2 shows the graph of timing vs dataset size for our selection sort.

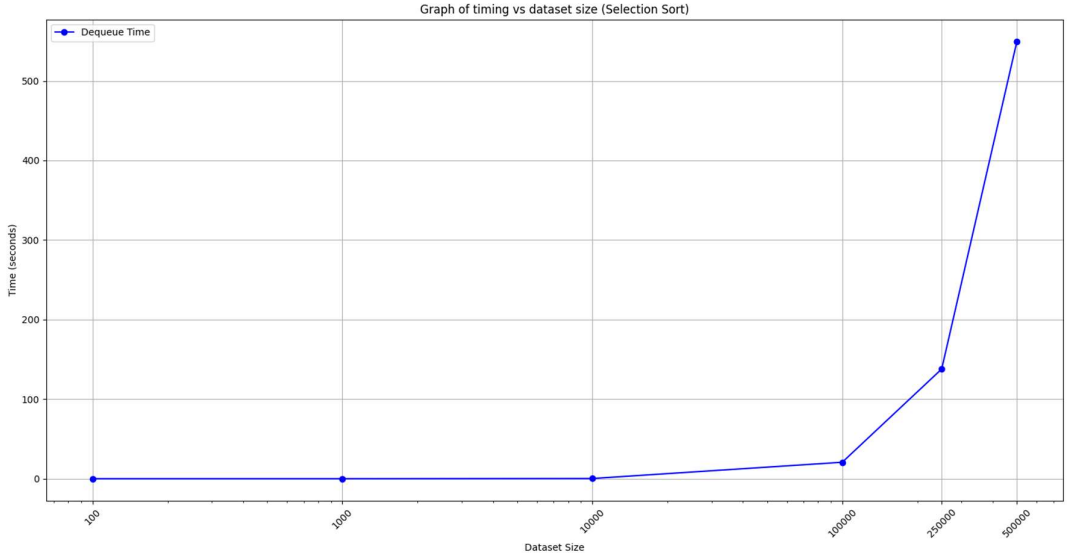


Figure 3.2 Graph of timing vs dataset size for Selection Sort

3.2.4 Discussions

3.2.4.1 Time Complexity

Selection process for each element

1. For each element in the array, the algorithm scans the remaining elements to find the minimum. This involves **n-1** comparisons for the first element, **n-2** comparisons for the second element, and so on.
2. Therefore, the total number of comparisons is **(n-1) + (n-2) + ... + 1** = $\frac{n(n-1)}{2}$, which is $O(n^2)$.

Swapping

1. Each swap operation takes a constant time, but the number of swaps is at most **n-1**.

Thus, the overall time complexity of selection sort is $O(n^2)$.

3.2.4.2 Space Complexity

1. **In-place sorting:** Selection sort is an in-place sorting algorithm, meaning it requires only a constant amount of extra space, **O(1)**, apart from the input array itself.

3.2.4.3 Discussion Based on Figure 3.2

Figure 3.2 illustrates the time taken to sort datasets using the selection sort algorithm versus the dataset size. The graph shows an upward trend, which aligns with the expected $O(n^2)$ time complexity for selection sort. Here are some key observations:

1. **Initial low times**
   For smaller datasets (sizes 100, 1000), the sorting times are very low, almost negligible. This is typical since the $O(n^2)$ complexity is more noticeable with larger values of **n**.

2. **Quadratic growth**
   As the dataset size increases, the time taken to sort the elements increases significantly. This is consistent with the quadratic nature of the time complexity. The impact of n^2 becomes more pronounced as **n** grows larger, leading to a rapid increase in sorting time.

3. **Large datasets**
   For very large datasets (sizes 250000, 500000), the sorting time becomes significantly higher. This illustrates the inefficiency of selection sort for large datasets, where the time complexity leads to impractically long sorting times.

3.2.4.4 Conclusion

Selection sort exhibits a time complexity of $\boldsymbol{O(n^2)}$ which makes it inefficient for large datasets. The space complexity is **O(1)** since it is an in-place sorting algorithm. Figure 3.2 confirms the theoretical time complexity, showing low initial sorting times for small datasets and a rapid increase in sorting time as the dataset size increases. This increase is consistent with the quadratic factor in the selection sort algorithm. Therefore, while selection sort can be useful for small datasets, it is not suitable for large datasets due to its poor time complexity.

4 Q3: Dijkstra's and Kruskal

Note: *All the data involved for this* ***Q3: Dijkstra's and Kruskal*** *are based on the* ***Dataset 2 Generation*** *(generated based on the sum of others group members' student ID – 3633305231 in this case).*

4.1 Shortest Path (Dijkstra's Algorithm)

4.1.1 Algorithm

Dijkstra's algorithm is a well-known graph traversal algorithm used to find the shortest paths from a single source node to all other nodes in a weighted graph with non-negative edge weights. The purpose of this algorithm is to determine the minimum distance from a given starting point (**Star_A** in this case) to all other stars in the graph, effectively mapping out the most efficient routes. The algorithm utilizes a priority queue to iteratively explore the nearest unvisited star, updating the shortest known distances to each neighboring star until all stars have been processed. By maintaining and updating a set of tentative distances and predecessors for each star, Dijkstra's algorithm not only computes the shortest path lengths but also reconstructs the actual paths taken.

Dijkstra's algorithm function

```
63  // Dijkstra's algorithm to find the shortest paths from a source star
64  pair<unordered_map<string, double>, unordered_map<string, string>> dijkstra(const string& source, const vector<Star>& stars, const vector<Edge>& edges) {
65      unordered_map<string, vector<pair<string, double>>> graph;
66      // Build the graph
67      for (const auto& edge : edges) {
68          graph[edge.from].emplace_back(edge.to, edge.distance);
69          graph[edge.to].emplace_back(edge.from, edge.distance);
70      }
71      // Priority queue to select the next node with the shortest distance
72      priority_queue<pair<double, string>, vector<pair<double, string>>, greater<>> pq;
73      unordered_map<string, double> distances;
74      unordered_map<string, string> previous;
75      for (const auto& star : stars) {
76          distances[star.name] = numeric_limits<double>::infinity();
77          previous[star.name] = ""; // Initialize with empty string
78      }
79      if (distances.find(source) == distances.end()) {
80          cerr << "Source node " << source << " not found in the stars data." << endl;
81          return {distances, previous};
82      }
83      distances[source] = 0.0;
84      pq.emplace(0.0, source);
85      while (!pq.empty()) {
86          double currentDistance = pq.top().first;
87          string currentNode = pq.top().second;
88          pq.pop();
89          if (currentDistance > distances[currentNode]) continue;
90          for (const auto& neighbor : graph[currentNode]) {
91              double distance = currentDistance + neighbor.second;
92              if (distance < distances[neighbor.first]) {
93                  distances[neighbor.first] = distance;
94                  previous[neighbor.first] = currentNode; // Track the previous node
95                  pq.emplace(distance, neighbor.first);
96              }
97          }
98      }
99      return {distances, previous};
100 }
```

Purpose: This function implements Dijkstra's algorithm to find the shortest paths from a source star (**Star_A** in this case) to all other stars in the graph.

Parameters:

- **const string& source:** The name of the source star from which to calculate shortest paths.
- **const vector<Star>& stars:** A vector of Star objects, representing all stars in the graph.
- **const vector<Edge>& edges:** A vector of Edge objects, representing all edges (connections) between stars in the graph.

Return:

A pair consisting of:

- **unordered_map<string, double>:** Maps each star to its shortest distance from the source star.
- **unordered_map<string, string>:** Maps each star to the previous star on the shortest path from the source star.

Steps:

1. **Graph construction (Line 67 - 70):** The function starts by constructing an adjacency list representation of the graph using an unordered map. Each star name maps to a vector of pairs, where each pair contains a neighboring star's name and the distance to that neighbor.
2. **Initialization (Line 72 - 84):**
   - Initialize a priority queue (pq) for selecting the next node with the shortest distance.
   - Initialize an unordered map (distances) to store the shortest distances from the source to each star, initially set to infinity.
   - Initialize an unordered map (previous) to store the previous star on the shortest path, initially empty.
   - Set the distance from the source star to itself as 0 and push the source star into the priority queue.
3. **Main Loop (Line 85 - 98):**
   While the priority queue is not empty:
   - Extract the star with the current shortest distance.
   - For each neighbor of this star, calculate the distance through this star.
   - If this new distance is shorter, update the shortest distance and record the current star as the previous star for this neighbor.
   - Push the neighbor with the updated distance into the priority queue.
4. **Return:** The function returns the distances and previous maps.

Reconstruct path function

```
102  // Function to reconstruct the path from source to each node
103  vector<string> reconstructPath(const string& target, const unordered_map<string, string>& previous) {
104      vector<string> path;
105      for (string at = target; !at.empty(); at = previous.at(at)) {
106          path.push_back(at);
107      }
108      reverse(path.begin(), path.end());
109      return path;
110  }
```

Purpose: This function reconstructs the shortest path from the source star to a target star using the previous map generated by Dijkstra's algorithm.

Parameters:

- **const string& target:** The name of the target star to which the path is to be reconstructed.
- **const unordered_map<string, string>& previous:** A map where each star maps to the previous star on the shortest path from the source.

Return:

- **vector<string>:** A vector of star names representing the path from the source to the target star.

Steps:

1. **Initialize path vector (Line 104):** Initialize an empty vector path.
2. **Backtrack from target to source (Line 105 - 107):**
   - Starting from the target star, use the previous map to backtrack to the source star.
   - For each star, add it to the path vector.
3. **Reverse path (Line 108):** Since the path is constructed from the target to the source, reverse the path vector to get the correct order from source to target.
4. **Return (Line 109):** The function returns the path vector.

Store shortest paths function

```
122 // Function to store the shortest paths in a text file
123 void storeShortestPaths(const string& source, const unordered_map<string, double>& distances, const unordered_map<string, string>& previous) {
124     ofstream file("./Q3_AnswerOutput/dijkstra_shortestPaths.txt");
125     file << fixed << setprecision(2); // Set precision to 2 decimal places
126     file << "Shortest distances from " << source << ":\n";
127 
128     // Sort distances by star name
129     vector<pair<string, double>> sortedDistances(distances.begin(), distances.end());
130     sort(sortedDistances.begin(), sortedDistances.end());
131 
132     for (const auto& [star, distance] : sortedDistances) {
133         if (distance == numeric_limits<double>::infinity()) {
134             file << "To " << star << ": inf units\n";
135         } else {
136             file << "To " << star << ": " << distance << " units";
137             vector<string> path = reconstructPath(star, previous);
138             if (!path.empty()) {
139                 file << " (path: ";
140                 for (size_t i = 0; i < path.size() - 1; ++i) {
141                     file << path[i] << " -> ";
142                 }
143                 file << path.back() << ")";
144             }
145             file << "\n";
146         }
147     }
148     file.close();
149 }
```

Purpose: This function writes the shortest distances and paths from the source star to all other stars to a text file.

Parameters:

- **const string& source:** The name of the source star.
- **const unordered_map<string, double>& distances:** A map where each star maps to its shortest distance from the source.
- **const unordered_map<string, string>& previous:** A map where each star maps to the previous star on the shortest path from the source.

Steps:

1. **Open output file (Line 124):** Open an output file stream to write the results to a file.
2. **Set output formatting (Line 125):** Set the output stream to fixed-point notation and a precision of 2 decimal places.
3. **Write header (Line 126):** Write a header line indicating the source star.
4. **Sort distances (Line 130):** Convert the distances map to a vector of pairs and sort it by star names to ensure the output is ordered.
5. **Write distances and paths (Line 132 - 147):**
   For each star in the sorted distances:
   - If the distance is infinity, write **inf units**.
   - Otherwise, write the distance in units.
   - Reconstruct the path from the source to this star using **reconstructPath**.

- Write the reconstructed path in the format **source -> ... -> target**.

6. **Close file (Line 148):** Close the output file stream.

4.1.2 Program for Algorithm Implementation

When we run the program named **Q3_shortestPath.cpp**, it starts by reading the stars and edges data from **stars.txt** and **edges.txt** respectively, which contain the coordinates (x, y, z) and attributes (weight and profit) of the stars and the distances between them, respectively. It constructs a graph representation and then uses Dijkstra's algorithm to compute the shortest distances from the source star, **Star_A**, in this case to all other stars. The algorithm maintains a priority queue to explore the shortest unvisited nodes and updates the distances and paths accordingly. After computing the shortest paths, the program outputs the distances and paths to the terminal and saves them to a text file, **dijkstra_shortestPaths.txt**, with a formatted output showing the distances and the sequence of stars in each path, as shown in the figure below.

```
Shortest distances from Star_A:
To Star_A: 0 units (path: Star_A)
To Star_B: 599.9 units (path: Star_A -> Star_B)
To Star_C: 1050.5 units (path: Star_A -> Star_Q -> Star_C)
To Star_D: 600.5 units (path: Star_A -> Star_I -> Star_D)
To Star_E: 555.2 units (path: Star_A -> Star_E)
To Star_F: 710.8 units (path: Star_A -> Star_T -> Star_F)
To Star_G: 901.3 units (path: Star_A -> Star_T -> Star_H -> Star_G)
To Star_H: 670.6 units (path: Star_A -> Star_T -> Star_H)
To Star_I: 287 units (path: Star_A -> Star_I)
To Star_J: 592.3 units (path: Star_A -> Star_E -> Star_J)
To Star_K: 1175 units (path: Star_A -> Star_I -> Star_D -> Star_K)
To Star_L: 659.1 units (path: Star_A -> Star_L)
To Star_M: 716.5 units (path: Star_A -> Star_T -> Star_M)
To Star_N: 1246 units (path: Star_A -> Star_T -> Star_M -> Star_N)
To Star_O: 885.1 units (path: Star_A -> Star_I -> Star_D -> Star_O)
To Star_P: 810.8 units (path: Star_A -> Star_I -> Star_P)
To Star_Q: 587.3 units (path: Star_A -> Star_Q)
To Star_R: 1060.1 units (path: Star_A -> Star_T -> Star_H -> Star_R)
To Star_S: 1359.1 units (path: Star_A -> Star_T -> Star_M -> Star_S)
To Star_T: 199.6 units (path: Star_A -> Star_T)
Shortest paths have been stored in './Q3_AnswerOutput/dijkstra_shortestPaths.txt'.
```

4.1.3 Experimental Results

Figure 4.1 shows the resulting graph that represents the shortest paths from **Star_A** to other stars using Dijkstra's algorithm.

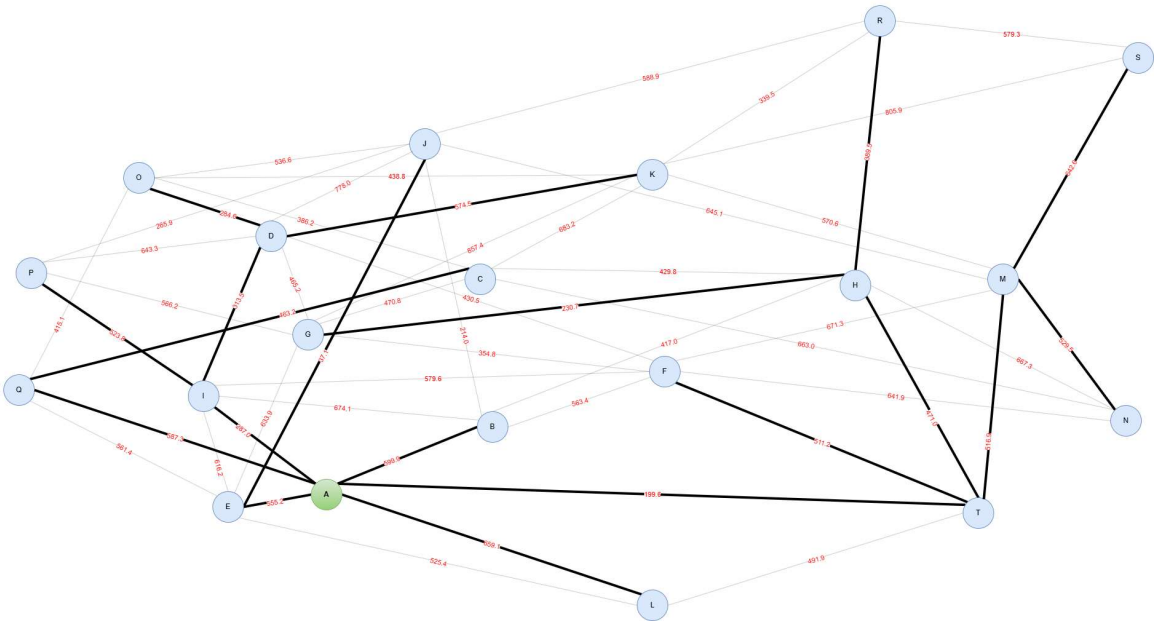


Figure 4.1 Graph Representing the Shortest Paths

4.1.4 Discussions

Notation: n *is the number of stars/vertices and* ***m*** *is the number of edges.*

4.1.4.1 Time Complexity

The time complexity of Dijkstra's algorithm depends on the data structures used to implement the priority queue and adjacency list. Here, a priority queue and adjacency list are used.

1. **Initialization**
   Initializing the distances and previous node maps takes **O(n)** time.

2. **Building the graph**
   Creating the adjacency list from the list of edges takes **O(m)** time.

3. **Priority queue operations**
   Each vertex is pushed and popped from the priority queue at most once, leading to **O(log n)** time for each operation. Therefore, for **n** vertices, this results in **O(n log n)** time.

4. **Relaxation of edges**
   For each edge, the algorithm performs a relaxation step which involves updating the priority queue. In the worst case, every edge causes an update, leading to **O(m log n)** time.

By combining these steps, the overall time complexity of Dijkstra's algorithm is **O((n + m) log n)**.

4.1.4.2 Space Complexity

The space complexity of the algorithm is determined by the storage requirements for the graph, distances, previous node information, and the priority queue.

1. **Graph representation**
   The adjacency list requires **O(n + m)** space because each vertex and edge is stored once.

2. **Distances and previous node maps**
   The distances and previous node maps each require **O(n)** space to store the shortest distance and path information for each vertex.

3. **Priority queue**
   The priority queue holds up to n vertices, leading to **O(n)** space.

By combining these, the overall space complexity of Dijkstra's algorithm is **O(n + m)**.

4.1.4.3 Conclusion

In conclusion, the implementation of Dijkstra's algorithm using a priority queue and adjacency list efficiently finds the shortest paths from a source vertex/star (**Star_A** for this case) to all other vertices in a graph with non-negative edge weights. The time complexity of the algorithm, **O((n + m) log n)**, reflects its dependency on both the number of vertices **n** and the number of edges **m**. This complexity is achieved through efficient priority queue operations and edge relaxation steps. The space complexity, **O(n + m)**, accounts for the storage of the graph representation, distance mappings, previous node information, and the priority queue itself. These characteristics make Dijkstra's algorithm well-suited for practical applications in networks and routing problems where an efficient and reliable shortest path solution is required.

4.2 Minimum Spanning Tree (Kruskal's Algorithm)

4.2.1 Algorithm

Kruskal's algorithm is a popular algorithm used to find the Minimum Spanning Tree (MST) of a connected, undirected graph. The purpose of this algorithm is to construct a subset of the edges that connects all vertices/stars in the graph without any cycles and with the minimum possible total edge weight. The algorithm works by first sorting all the edges in the graph by their weight, then repeatedly adding the smallest edge to the MST as long as it does not form a cycle, using a Union-Find data structure to efficiently manage and check for cycles. This process continues until the MST contains exactly **n-1** edges, where **n** is the number of vertices/stars. In the provided code, Kruskal's algorithm is applied to a set of stars and edges (distances between stars), constructing the MST that connects all the stars with the minimum total distance, ensuring efficient interstellar connections.

Union-Find data structure

```cpp
// Union-Find structure to support Kruskal's algorithm
class UnionFind {
    private:
        vector<int> parent;
        vector<int> rank;

    public:
        UnionFind(int n) : parent(n), rank(n, 0) {
            for (int i = 0; i < n; ++i)
                parent[i] = i;
        }

        int find(int u) {
            if (parent[u] != u)
                parent[u] = find(parent[u]);
            return parent[u];
        }

        bool unite(int u, int v) {
            int rootU = find(u);
            int rootV = find(v);
            if (rootU != rootV) {
                if (rank[rootU] > rank[rootV]) {
                    parent[rootV] = rootU;
                } else if (rank[rootU] < rank[rootV]) {
                    parent[rootU] = rootV;
                } else {
                    parent[rootV] = rootU;
                    ++rank[rootU];
                }
                return true;
            }
            return false;
        }
};
```

Purpose: Union-Find (also known as Disjoint Set Union or DSU) is a data structure that keeps track of a partition of a set into disjoint (non-overlapping) subsets. It supports efficient union and find operations.

Components:

1. **Parent Array**
    - **vector<int> parent:** This array holds the parent or representative of each element. Initially, each element is its own parent.

2. **Rank Array**
   - **vector<int> rank:** This array holds the rank (or depth) of the trees for union by rank optimization.

Operations:

1. **Find (Line 38 - 42)**
   - Finds the representative (root) of the set containing element **u**.
   - Uses path compression to flatten the structure, making future operations faster.
2. **Union (Line 44 - 59)**
   - Merges the sets containing elements **u** and **v**.
   - Uses union by rank to attach the smaller tree under the root of the larger tree, keeping the tree shallow.

Kruskal's algorithm to find the minimum spanning tree (MST)

```
95  // Kruskal's algorithm to find the MST
96  vector<Edge> kruskalMST(const vector<Edge> &edges, int numberOfNodes) {
97      vector<Edge> mst;
98      UnionFind uf(numberOfNodes);
99      auto sortedEdges = edges;
100
101     sort(sortedEdges.begin(), sortedEdges.end(), [](const Edge &a, const Edge &b) {
102         return a.weight < b.weight;
103     });
104
105     for (const auto &edge : sortedEdges) {
106         int u = edge.from.back() - 'A';
107         int v = edge.to.back() - 'A';
108         if (uf.unite(u, v)) {
109             mst.push_back(edge);
110             if (mst.size() == numberOfNodes - 1)
111                 break;
112         }
113     }
114     return mst;
115 }
```

Purpose: Kruskal's algorithm is used to find the Minimum Spanning Tree (MST) of a graph. An MST is a subset of edges that connects all vertices together, without any cycles and with the minimum possible total edge weight.

Steps:

1. **Sort edges (Line 101- 103):** Sort all the edges in the graph in non-decreasing order of their weight (distance in this case).
2. **Initialize Union-Find (Line 98):** Initialize the Union-Find data structure to keep track of which vertices are in which components (connected subgraphs).
3. **Iterate through sorted edges (Line 105 - 113):**
   For each edge in the sorted list:
   - Check if adding this edge to the MST will form a cycle.
   - If it does not form a cycle, add it to the MST.
   - Use Union-Find to check for cycles and unite components.
4. **Terminate (Line 110):** The algorithm stops when we have **numberOfNodes - 1** edges in the MST.

4.2.2 Program for Algorithm Implementation

When we run the program named **Q3_minSpanningTree.cpp**, it starts by reading the stars and edges data from **stars.txt** and **edges.txt** respectively, which contain the coordinates (x, y, z) and attributes (weight and profit) of the stars and the distances between them, respectively. The program then uses a Union-Find data structure to efficiently manage and unite disjoint sets during the minimum spanning tree (MST) construction process. By sorting the edges by distance and adding the shortest edges to the MST while avoiding cycles, this Kruskal's algorithm ensures the resulting MST has the minimum total edge weight. The edges in the MST are then printed and saved to a text file named **kruskal_mstEdges.txt**, providing a list of edges that connect all stars with the least total distance, as shown in the figure below.

```
Edges in the Minimum Spanning Tree:
Star_E - Star_J : 37.1
Star_T - Star_A : 199.6
Star_B - Star_J : 214
Star_G - Star_H : 230.7
Star_P - Star_J : 265.9
Star_D - Star_O : 284.6
Star_A - Star_I : 287
Star_D - Star_I : 313.5
Star_K - Star_R : 339.5
Star_F - Star_G : 354.8
Star_O - Star_C : 386.2
Star_H - Star_R : 389.5
Star_Q - Star_O : 415.1
Star_H - Star_B : 417
Star_C - Star_H : 429.8
Star_L - Star_T : 491.9
Star_M - Star_T : 516.9
Star_M - Star_N : 529.5
Star_S - Star_R : 579.3

Minimum Spanning Tree stored in './Q3_AnswerOutput/kruskal_mstEdges.txt'
```

4.2.3 Experimental Results

Figure 4.2 shows the resulting graph that represents the minimum spanning tree (MST) using Kruskal's algorithm.

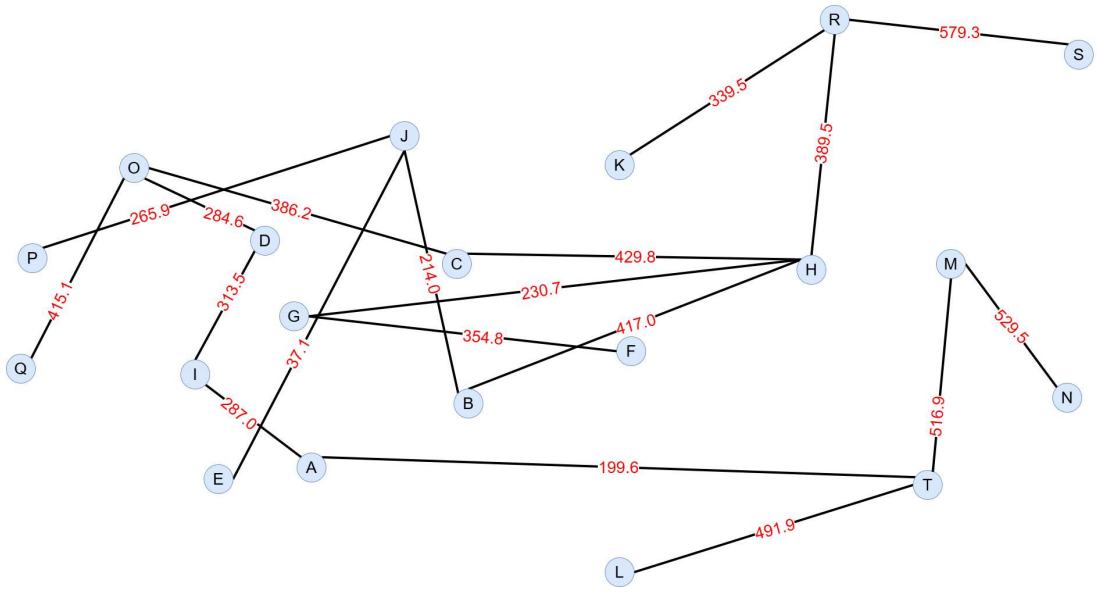


Figure 4.2 Graph Representing the Minimum Spanning Tree (MST)

4.2.4 Discussion

Notation: ***n*** *is the number of stars/vertices and* ***m*** *is the number of edges.*

4.2.4.1 Time complexity

The time complexity of Kruskal's algorithm is primarily influenced by two main operations: sorting the edges and performing union-find operations. Below is the breakdown of the key components:

1. **Sorting the edges:** The edges are sorted based on their weights. Given that there are m edges, the time complexity for sorting is **O(m log m)**.

2. **Union-Find operations:**
   - **Find:** Determines the root of the set containing a particular element, thus it takes **O(1)** time.
   - **Union:** Merges two sets by moving the elements of the smaller set to the sequence of the larger set and update their references. This takes **O(log n)** time.
   - Whenever an element is processed, it goes into a set of size at least double, hence each element is processed at most **O(log n)** times.

By combining these, the overall time complexity of Kruskal's algorithm is **O(m log m)**.

4.2.4.2 Space complexity

The space complexity of Kruskal's algorithm involves:

1. **Storing the stars and edges:** This requires **O(n)** space to store the vertices/stars and **O(m)** space to store the edges respectively.
2. **Storing the Union-Find data structures:** This requires **O(n)** space for the parent and rank arrays.
3. **Storing the resulting MST:** This requires **O(n)** space since the MST will have **n - 1** edges.

By combining these, the overall space complexity of Kruskal's algorithm is **O(n + m)**.

4.2.4.3 Conclusion

Kruskal's algorithm is efficient for finding the Minimum Spanning Tree (MST) in a graph, especially when the graph is sparse, i.e., the number of edges **m** is much larger than the number of vertices **n**. The algorithm's time complexity, dominated by the sorting of edges and performing union-find operations, is **O(m log m)**, which is very efficient for large graphs. The space complexity is **O(n + m)**, ensuring that it uses memory proportional to the size of the input graph. Overall, Kruskal's algorithm is a robust and efficient method for MST problems, leveraging the union-find data structure to ensure near-constant time complexity for its key operations.

5 Q4: 0/1 Knapsack

Note: *All the data involved for this* ***Q4: 0/1 Knapsack*** *are based on the* ***Dataset 2 Generation*** *(generated based on the sum of others group members' student ID – 3633305231 in this case).*

5.1 Algorithm

The provided algorithm implements the 0/1 knapsack problem using dynamic programming. The 0/1 knapsack problem is a classic optimization problem where the objective is to maximize the total profit of items placed in a knapsack without exceeding its maximum weight capacity (800 kg in this case). Each item can either be included (1) or excluded (0) from the knapsack, hence the name 0/1 knapsack. In this implementation, the algorithm reads data about various stars from **stars.txt**, each with a weight and a profit. It constructs a dynamic programming matrix (DP Matrix) to store the maximum profit achievable for each subproblem defined by the first **i** stars and weight limit **w**. The DP Matrix is built iteratively, considering whether to include each star based on its weight and profit. The purpose of this algorithm is to determine the optimal subset of stars to maximize the profit while keeping the total weight within a specified limit which is 800 kg in this case.

Initialize Variables and DP Matrix

```
68  void knapsack(const vector<Star>& stars, int maxWeight, const string& outputFileName) {
69      int n = stars.size();
70      vector<vector<int>> dp(n + 1, vector<int>(maxWeight + 1, 0));
```

Purpose: Initialize the number of stars **n** and a 2D vector **dp** to store the maximum profit for each subproblem. The dimensions of **dp** are **(n+1) x (maxWeight+1)**.

Build the DP Matrix

```
72      // Build the dp matrix
73      for (int i = 1; i <= n; ++i) {
74          for (int w = 1; w <= maxWeight; ++w) {
75              if (stars[i - 1].weight <= w) {
76                  dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - stars[i - 1].weight] + stars[i - 1].profit);
77              } else {
78                  dp[i][w] = dp[i - 1][w];
79              }
80          }
81      }
```

Purpose: Populate the DP matrix to solve subproblems iteratively.
Steps:

- Loop through each star (**i** from 1 to **n**).
- Loop through each possible weight (**w** from 1 to **maxWeight** = 800).
- If the current star can fit in the knapsack (**stars[i-1].weight <= w**), update the DP value as the maximum of not including the star or including the star.
- Otherwise, carry forward the value from the previous row (**dp[i-1][w]**).

Output the DP Matrix in Segments

```
85      // Output the dp matrix in segments
86      int segmentSize = 20;
87      for (int startCol = 0; startCol <= maxWeight; startCol += segmentSize) {
88          int endCol = min(startCol + segmentSize - 1, maxWeight);
89
90          outputFile << string(70, '-') << endl;
91          outputFile << "Segment [" << startCol << "- " << endCol << "]:" << endl;
92          outputFile << string(70, '-') << endl;
93
94          cout << string(70, '-') << endl;
95          cout << "Segment [" << startCol << "- " << endCol << "]:" << endl;
96          cout << string(70, '-') << endl;
97
98          printSegment(outputFile, dp, startCol, endCol, n);
99
100         outputFile << endl;
101         cout << endl;
102     }
```

Purpose: Output the DP matrix in segments to make it more readable.

Steps:

- Define the segment size.
- Loop through the DP matrix in segments of **segmentSize**.
- For each segment, print a header and call **printSegment** to print the segment to the file and console.

Find the Stars to Visit

```
104     // Find the stars to visit
105     int w = maxWeight;
106     vector<Star> resultStars;
107     for (int i = n; i > 0 && w > 0; --i) {
108         if (dp[i][w] != dp[i - 1][w]) {
109             resultStars.push_back(stars[i - 1]);
110             w -= stars[i - 1].weight;
111         }
112     }
```

Purpose: Determine which stars to visit to maximize profit without exceeding the weight limit, 800 kg.

Steps:

- Backtrack from **dp[n][maxWeight]** to find which stars are included in the optimal solution.
- If **dp[i][w]** is different from **dp[i-1][w]**, it means star **i-1** is included.
- Add the star to **resultStars** and reduce the remaining weight **w** accordingly.

Output the result and calculate total profit and weight

```
114     // Output the result & Calculate the total profit and weight
115     int totalProfit = 0;
116     int totalWeight = 0;
117
118     outputFile << "Stars to visit:" << endl;
119     for (const Star& star : resultStars) {
120         totalProfit += star.profit;
121         totalWeight += star.weight;
122         outputFile << star.name << " - Weight: " << star.weight << "kg, Profit: " << star.profit << endl;
123         cout << star.name << " - Weight: " << star.weight << "kg, Profit: " << star.profit << endl;
124     }
125
126     outputFile << "-------------------------------" << endl;
127     outputFile << "Total Weight: " << totalWeight << "kg" << endl;
128     outputFile << "Total Profit: " << totalProfit << endl;
129
130     cout << "-------------------------------" << endl;
131     cout << "Total Weight: " << totalWeight << "kg" << endl;
132     cout << "Total Profit: " << totalProfit << endl;
133
134     outputFile.close();
```

Purpose: Output the selected stars, their weights, and profits, and calculate the total profit and weight.

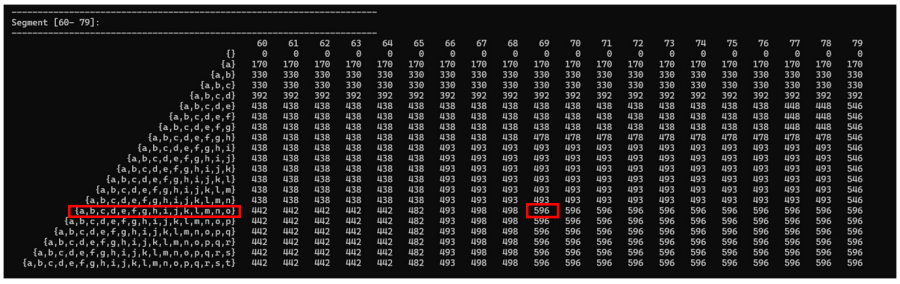
Steps:

- Initialize **totalProfit** and **totalWeight**.
- Loop through **resultStars**, adding each star's profit and weight to the totals and writing the details to the output file and console.
- Print the total weight and profit to the output file and console.
- Close the output file.

5.2 Program for Algorithm Implementation

When we run the program named **Q4_dynamicProgramming.cpp**, the program will go through the below important steps of this algorithm:

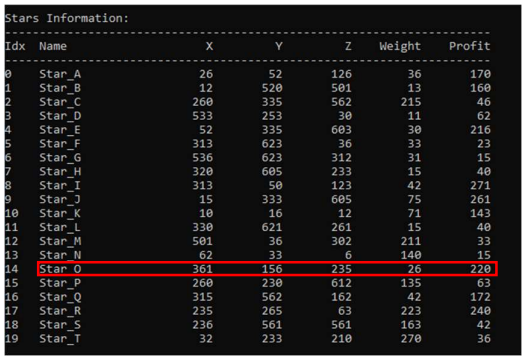
1. **Dynamic Programming Matrix Calculation (Example)**
   For this example, let's see how the maximum profit (**596**) at the capacity of **69 kg** is achieved by including the stars from **Star_A** to **Star_O**.



Segment [60- 79]:

	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c,d}	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392
{a,b,c,d,e}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	448	448	546
{a,b,c,d,e,f}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	448	448	546
{a,b,c,d,e,f,g}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	448	448	546
{a,b,c,d,e,f,g,h}	438	438	438	438	438	438	438	438	438	478	478	478	478	478	478	478	478	478	478	546
{a,b,c,d,e,f,g,h,i}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	493	546
{a,b,c,d,e,f,g,h,i,j}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	493	546
{a,b,c,d,e,f,g,h,i,j,k}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	493	546
{a,b,c,d,e,f,g,h,i,j,k,l}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	493	546
{a,b,c,d,e,f,g,h,i,j,k,l,m}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	493	546
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	493	546
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	442	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	442	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	442	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	442	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596	596
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596	596

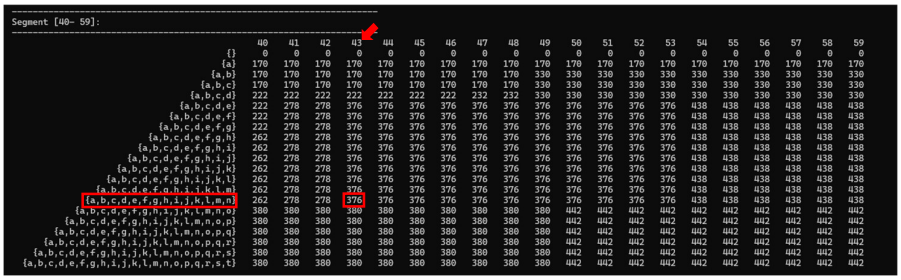
We first deduct the **69 kg** with **Star_O**'s weight which is **26 kg**, this results in **43 kg** (69 kg – 26 kg = 43 kg).



Stars Information:

Idx	Name	X	Y	Z	Weight	Profit
0	Star_A	26	52	126	36	170
1	Star_B	12	520	501	13	160
2	Star_C	260	335	562	215	46
3	Star_D	533	253	30	11	62
4	Star_E	52	335	603	30	216
5	Star_F	313	623	36	33	23
6	Star_G	536	623	312	31	15
7	Star_H	320	605	233	15	40
8	Star_I	313	50	123	42	271
9	Star_J	15	333	605	75	261
10	Star_K	10	16	12	71	143
11	Star_L	330	621	261	15	40
12	Star_M	501	36	302	211	33
13	Star_N	62	33	6	140	15
14	Star_O	361	156	235	26	220
15	Star_P	260	230	612	135	63
16	Star_Q	315	562	162	42	172
17	Star_R	235	265	63	223	240
18	Star_S	236	561	561	163	42
19	Star_T	32	233	210	270	36

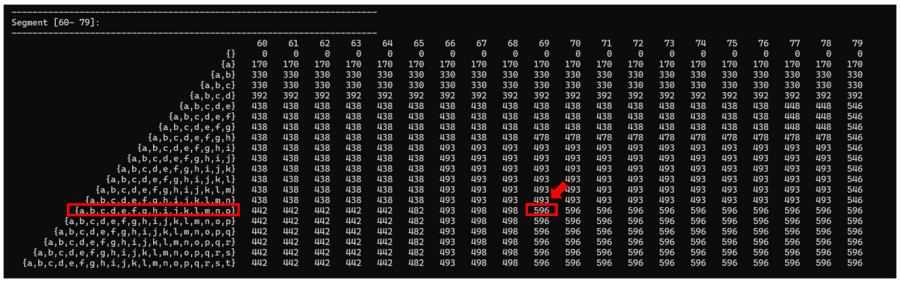
Then, we refer back to the weight of **43 kg** of the previous row at **Star_N**. At this point, we can see the profit is **376**.



Segment [40- 59]:

	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	170	170	170	170	170	170	170	170	170	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	170	170	170	170	170	170	170	170	170	330	330	330	330	330	330	330	330	330	330	330
{a,b,c,d}	222	222	222	222	222	222	232	232	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c,d,e}	222	278	278	376	376	376	376	376	376	376	376	376	376	376	438	438	438	438	438	438
{a,b,c,d,e,f}	222	278	278	376	376	376	376	376	376	376	376	376	376	376	438	438	438	438	438	438
{a,b,c,d,e,f,g}	222	278	278	376	376	376	376	376	376	376	376	376	376	376	438	438	438	438	438	438
{a,b,c,d,e,f,g,h}	262	278	278	376	376	376	376	376	376	376	376	376	376	376	438	438	438	438	438	438
{a,b,c,d,e,f,g,h,i}	262	278	278	376	376	376	376	376	376	376	376	376	376	376	438	438	438	438	438	438
{a,b,c,d,e,f,g,h,i,j}	262	278	278	376	376	376	376	376	376	376	376	376	376	376	438	438	438	438	438	438
{a,b,c,d,e,f,g,h,i,j,k}	262	278	278	376	376	376	376	376	376	376	376	376	376	376	438	438	438	438	438	438
{a,b,c,d,e,f,g,h,i,j,k,l}	262	278	278	376	376	376	376	376	376	376	376	376	376	376	438	438	438	438	438	438
{a,b,c,d,e,f,g,h,i,j,k,l,m}	262	278	278	376	376	376	376	376	376	376	376	376	376	376	438	438	438	438	438	438
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	262	278	278	376	376	376	376	376	376	376	376	376	376	376	438	438	438	438	438	438
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	380	380	380	380	380	380	380	380	380	380	442	442	442	442	442	442	442	442	442	442
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	380	380	380	380	380	380	380	380	380	380	442	442	442	442	442	442	442	442	442	442
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	380	380	380	380	380	380	380	380	380	380	442	442	442	442	442	442	442	442	442	442
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	380	380	380	380	380	380	380	380	380	380	442	442	442	442	442	442	442	442	442	442
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	380	380	380	380	380	380	380	380	380	380	442	442	442	442	442	442	442	442	442	442
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	380	380	380	380	380	380	380	380	380	380	442	442	442	442	442	442	442	442	442	442

Therefore, we add this profit, **376**, to the profit of current star, **Star_O** which is **220**, this resulting profit of **596** (376 + 220 = 596). Since the resulting **596** is bigger than **493**, we choose the maximum profit which is **596** and put this profit in for **69 kg** at **Star_O**.

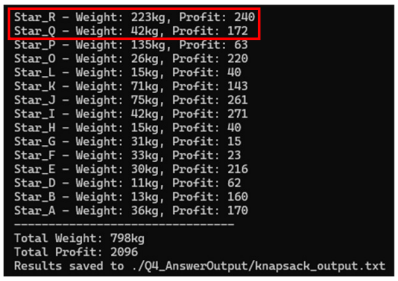


Segment [60– 79]:

	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c,d}	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392
{a,b,c,d,e}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	448	448	546	
{a,b,c,d,e,f}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	448	448	546	
{a,b,c,d,e,f,g}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	448	448	546	
{a,b,c,d,e,f,g,h}	438	438	438	438	438	438	438	438	438	478	478	478	478	478	478	478	478	478	546	
{a,b,c,d,e,f,g,h,i}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	546	
{a,b,c,d,e,f,g,h,i,j}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	546	
{a,b,c,d,e,f,g,h,i,j,k}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	546	
{a,b,c,d,e,f,g,h,i,j,k,l}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	546	
{a,b,c,d,e,f,g,h,i,j,k,l,m}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	546	
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	438	438	438	438	438	438	493	493	493	493	493	493	493	493	493	493	493	493	546	
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596	
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596	
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596	
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596	
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596	
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	442	442	442	442	442	482	493	498	498	596	596	596	596	596	596	596	596	596	596	

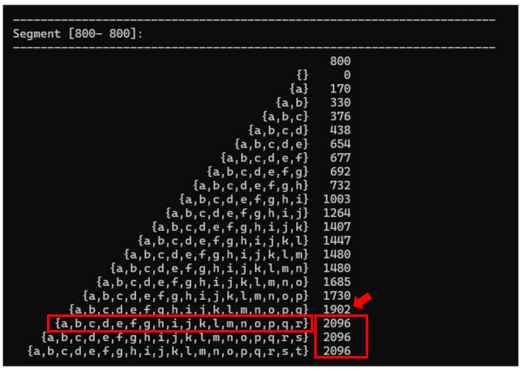
The same concept and idea for all the rest of DP Matrix calculation.

2. **Picking Stars to Visit (Example)**
   For this example, let's see why **Star_R** and **Star_Q** are included in the list for a maximum capacity of 800kg.



```
Star_R - Weight: 223kg, Profit: 240
Star_Q - Weight: 42kg, Profit: 172
Star_P - Weight: 135kg, Profit: 63
Star_O - Weight: 26kg, Profit: 220
Star_L - Weight: 15kg, Profit: 40
Star_K - Weight: 71kg, Profit: 143
Star_J - Weight: 75kg, Profit: 261
Star_I - Weight: 42kg, Profit: 271
Star_H - Weight: 15kg, Profit: 40
Star_G - Weight: 31kg, Profit: 15
Star_F - Weight: 33kg, Profit: 23
Star_E - Weight: 30kg, Profit: 216
Star_D - Weight: 11kg, Profit: 62
Star_B - Weight: 13kg, Profit: 160
Star_A - Weight: 36kg, Profit: 170
----------------------------------
Total Weight: 798kg
Total Profit: 2096
Results saved to ./Q4_AnswerOutput/knapsack_output.txt
```

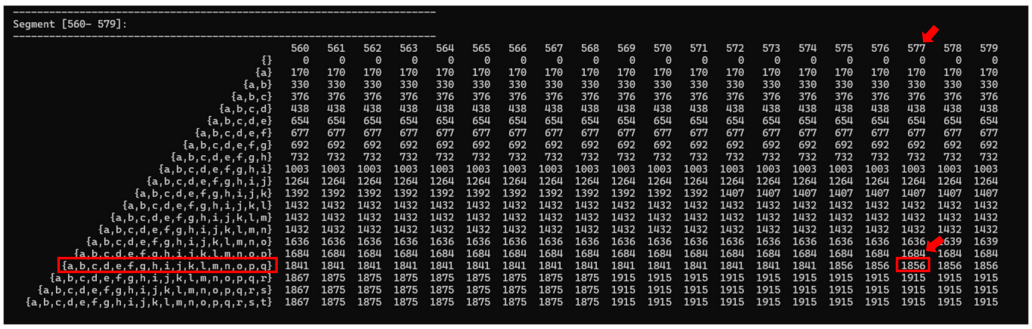
The program first starts from the most bottom-right cell of this DP matrix (**2096**) and traces back to identify the stars that contribute to the maximum profit. When examining the current cell value of **2096** in **Star_T**, it notices that this value is the same as the value in the cell directly above (**Star_S**). Therefore, **Star_T** is not selected. The same logic applies to **Star_S**. However, when the program compares the value of **2096** in **Star_R** with the value of **1902** in the cell directly above (**Star_Q**), it finds a difference. This indicates that **Star_R** is part of the optimal solution, and thus, **Star_R** is selected for the visit.



Segment [800– 800]:

	800
{}	0
{a}	170
{a,b}	330
{a,b,c}	376
{a,b,c,d}	438
{a,b,c,d,e}	654
{a,b,c,d,e,f}	677
{a,b,c,d,e,f,g}	692
{a,b,c,d,e,f,g,h}	732
{a,b,c,d,e,f,g,h,i}	1003
{a,b,c,d,e,f,g,h,i,j}	1264
{a,b,c,d,e,f,g,h,i,j,k}	1407
{a,b,c,d,e,f,g,h,i,j,k,l}	1447
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1480
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1480
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1685
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1730
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1902
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	2096
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	2096
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	2096

The weight limit (**800 kg**) is then reduced by the weight of the visited star (**Star_R**) which is **223 kg**, resulting in **577 kg** (800 kg – 223 kg = 577 kg). Now, the program moves on to **Star_Q** and compares the value (**1856**) of **Star_Q** with the value (**1684**) cell directly above (**Star_P**), the values are not the same, thus the program will pick the current star (**Star_Q**) to visit.



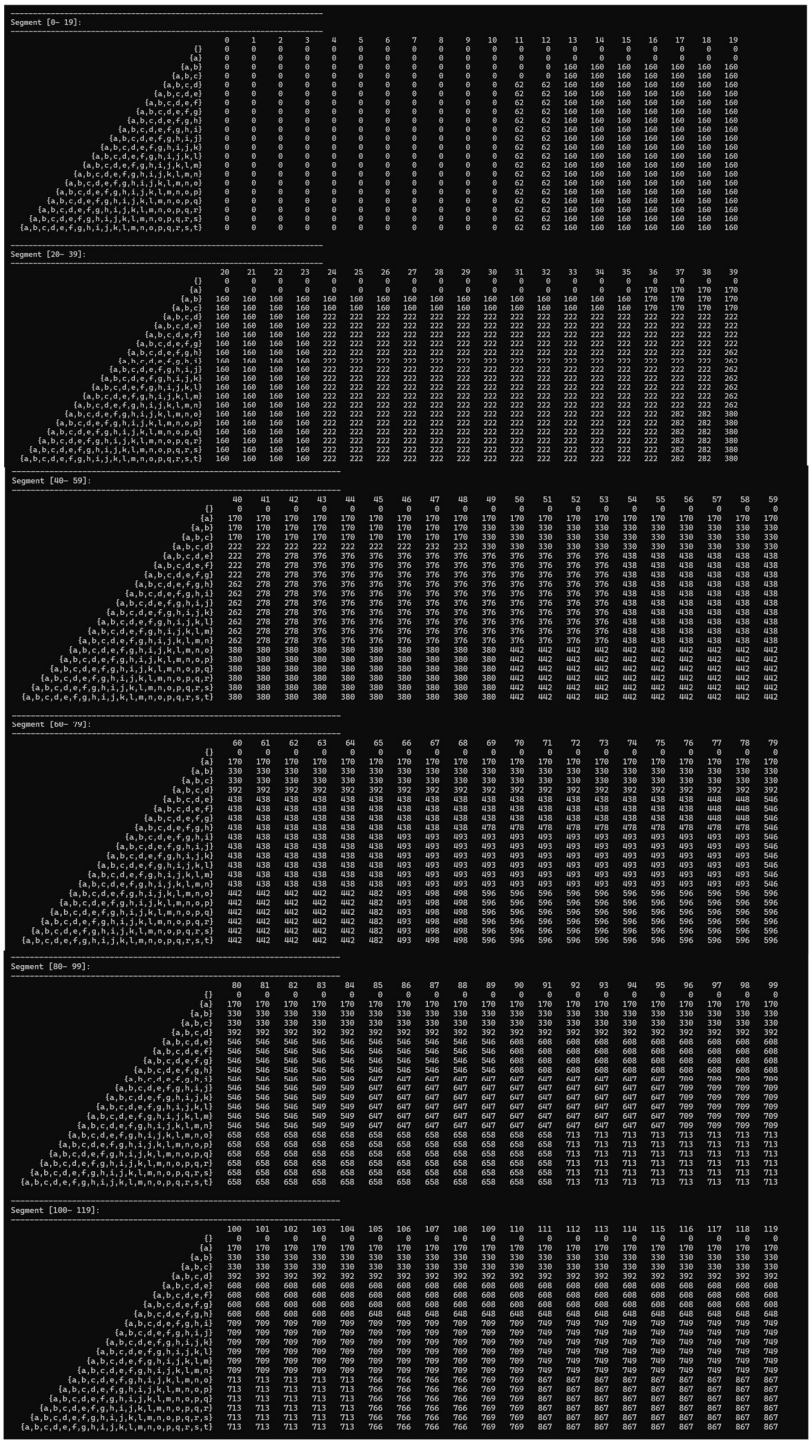
Segment [560– 579]:

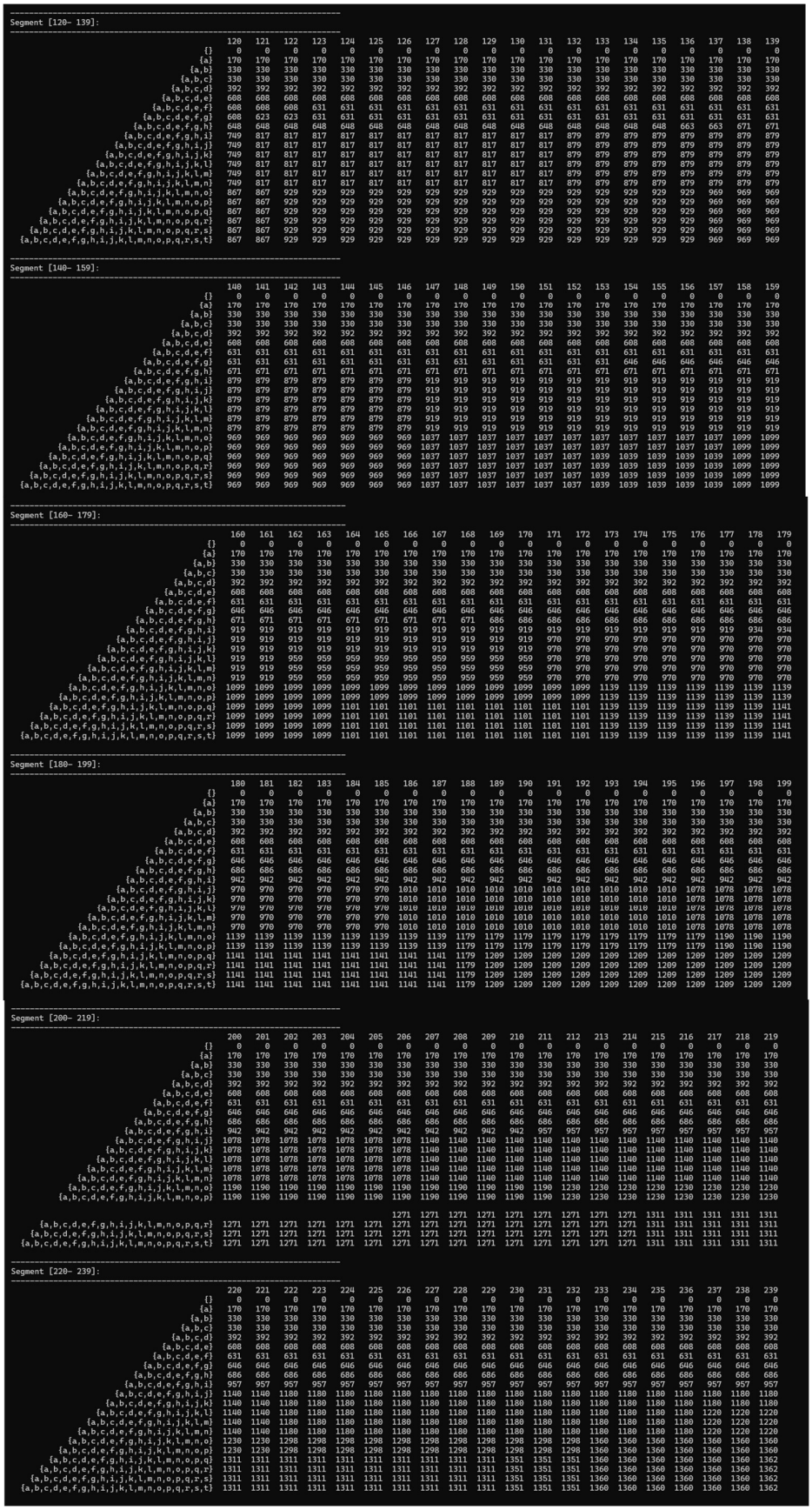
	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692
{a,b,c,d,e,f,g,h}	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732
{a,b,c,d,e,f,g,h,i}	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003
{a,b,c,d,e,f,g,h,i,j}	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264
{a,b,c,d,e,f,g,h,i,j,k}	1392	1392	1392	1392	1392	1392	1392	1392	1392	1392	1392	1392	1407	1407	1407	1407	1407	1407	1407	1407
{a,b,c,d,e,f,g,h,i,j,k,l}	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432	1432
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1636	1636	1636	1636	1636	1636	1636	1636	1636	1636	1636	1636	1636	1636	1636	1636	1636	1636	1639	1639
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1684	1684	1684	1684	1684	1684	1684	1684	1684	1684	1684	1684	1684	1684	1684	1684	1684	1684	1684	1684
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1841	1841	1841	1841	1841	1841	1841	1841	1841	1841	1841	1841	1841	1841	1841	1856	1856	1856	1856	1856
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1867	1875	1875	1875	1875	1875	1875	1875	1875	1915	1915	1915	1915	1915	1915	1915	1915	1915	1915	1915
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1867	1875	1875	1875	1875	1875	1875	1875	1875	1915	1915	1915	1915	1915	1915	1915	1915	1915	1915	1915
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1867	1875	1875	1875	1875	1875	1875	1875	1875	1915	1915	1915	1915	1915	1915	1915	1915	1915	1915	1915

Then, the weight limit (**577 kg**) is then reduced by the weight of the visited star (**Star_Q**), and the above process continues until all stars are considered or the weight limit is exhausted.

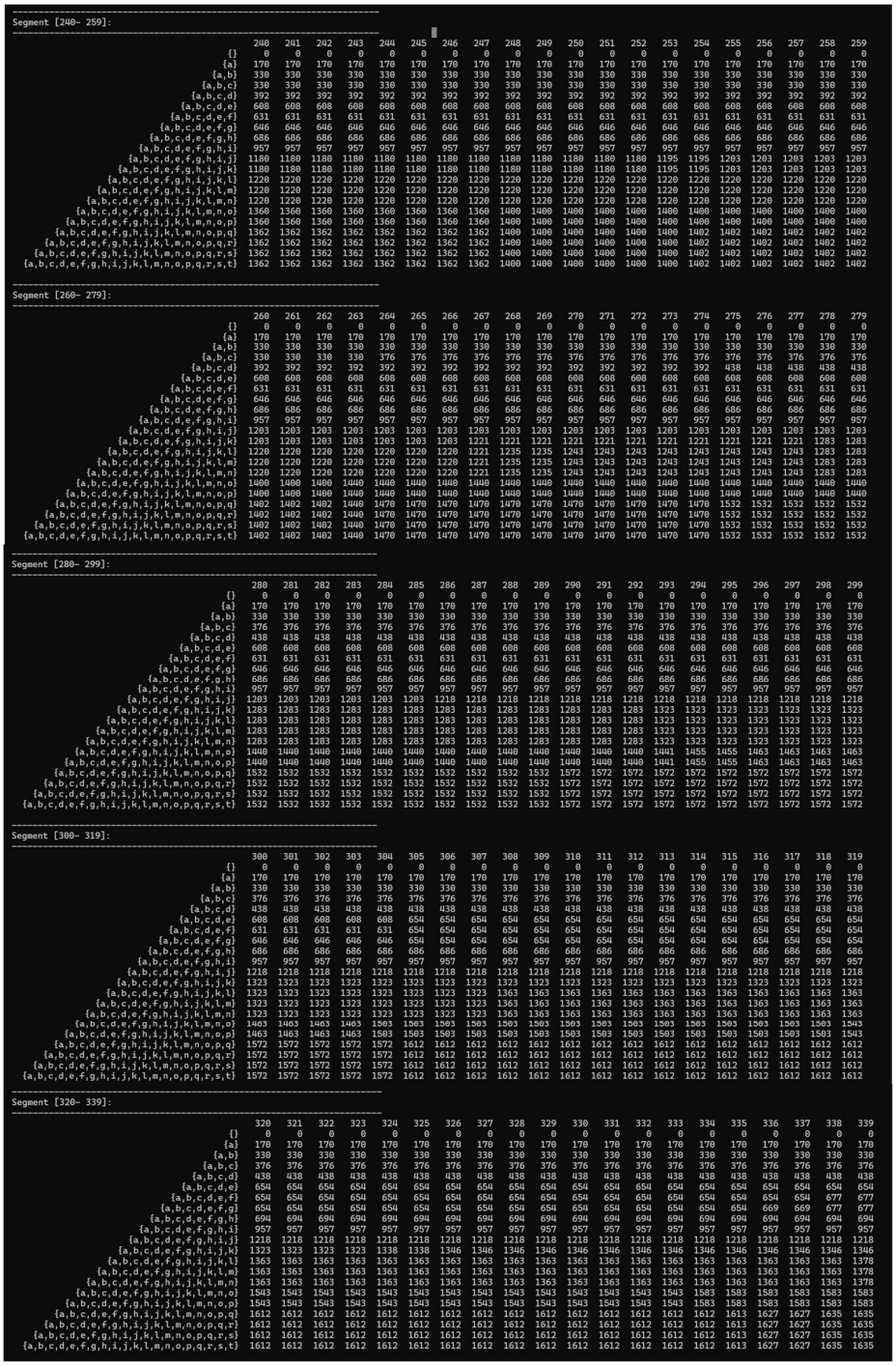
5.3 Experimental Result

Display of segmented views for the DP matrix





```
------------------------------------------------------------
Segment [120- 139]:
------------------------------------------------------------
                                  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139
                                {}  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                            {a,b,c,d}  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392
                           {a,b,c,d,e}  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608
                          {a,b,c,d,e,f}  608  608  608  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631
                         {a,b,c,d,e,f,g}  608  623  623  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631
                        {a,b,c,d,e,f,g,h}  648  648  648  648  648  648  648  648  648  648  648  648  648  648  648  648  663  663  671  671
                       {a,b,c,d,e,f,g,h,i}  749  817  817  817  817  817  817  817  817  817  817  817  879  879  879  879  879  879  879  879
                      {a,b,c,d,e,f,g,h,i,j}  749  817  817  817  817  817  817  817  817  817  817  817  879  879  879  879  879  879  879  879
                     {a,b,c,d,e,f,g,h,i,j,k}  749  817  817  817  817  817  817  817  817  817  817  817  879  879  879  879  879  879  879  879
                    {a,b,c,d,e,f,g,h,i,j,k,l}  749  817  817  817  817  817  817  817  817  817  817  817  879  879  879  879  879  879  879  879
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  749  817  817  817  817  817  817  817  817  817  817  817  879  879  879  879  879  879  879  879
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  749  817  817  817  817  817  817  817  817  817  817  817  879  879  879  879  879  879  879  879
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  867  867  929  929  929  929  929  929  929  929  929  929  929  929  929  929  929  969  969  969
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  867  867  929  929  929  929  929  929  929  929  929  929  929  929  929  929  929  969  969  969
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}  867  867  929  929  929  929  929  929  929  929  929  929  929  929  929  929  929  969  969  969
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  867  867  929  929  929  929  929  929  929  929  929  929  929  929  929  929  929  969  969  969
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  867  867  929  929  929  929  929  929  929  929  929  929  929  929  929  929  929  969  969  969
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  867  867  929  929  929  929  929  929  929  929  929  929  929  929  929  929  929  969  969  969

------------------------------------------------------------
Segment [140- 159]:
------------------------------------------------------------
                                  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159
                                {}  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                            {a,b,c,d}  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392
                           {a,b,c,d,e}  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608
                          {a,b,c,d,e,f}  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631
                         {a,b,c,d,e,f,g}  631  631  631  631  631  631  631  631  631  631  631  631  631  631  646  646  646  646  646  646
                        {a,b,c,d,e,f,g,h}  671  671  671  671  671  671  671  671  671  671  671  671  671  671  671  671  671  671  671  671
                       {a,b,c,d,e,f,g,h,i}  879  879  879  879  879  879  879  879  919  919  919  919  919  919  919  919  919  919  919  919
                      {a,b,c,d,e,f,g,h,i,j}  879  879  879  879  879  879  879  919  919  919  919  919  919  919  919  919  919  919  919  919
                     {a,b,c,d,e,f,g,h,i,j,k}  879  879  879  879  879  879  879  919  919  919  919  919  919  919  919  919  919  919  919  919
                    {a,b,c,d,e,f,g,h,i,j,k,l}  879  879  879  879  879  879  879  919  919  919  919  919  919  919  919  919  919  919  919  919
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  879  879  879  879  879  879  879  919  919  919  919  919  919  919  919  919  919  919  919  919
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  879  879  879  879  879  879  879  919  919  919  919  919  919  919  919  919  919  919  919  919
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  969  969  969  969  969  969  969 1037 1037 1037 1037 1037 1037 1037 1037 1037 1037 1037 1099 1099
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  969  969  969  969  969  969  969 1037 1037 1037 1037 1037 1037 1037 1037 1037 1037 1037 1099 1099
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}  969  969  969  969  969  969  969 1037 1037 1037 1037 1037 1037 1039 1039 1039 1039 1039 1099 1099
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  969  969  969  969  969  969  969 1037 1037 1037 1037 1037 1037 1039 1039 1039 1039 1039 1099 1099
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  969  969  969  969  969  969  969 1037 1037 1037 1037 1037 1037 1039 1039 1039 1039 1039 1099 1099
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  969  969  969  969  969  969  969 1037 1037 1037 1037 1037 1037 1039 1039 1039 1039 1039 1099 1099

------------------------------------------------------------
Segment [160- 179]:
------------------------------------------------------------
                                  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179
                                {}  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                            {a,b,c,d}  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392
                           {a,b,c,d,e}  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608
                          {a,b,c,d,e,f}  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631
                         {a,b,c,d,e,f,g}  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646
                        {a,b,c,d,e,f,g,h}  671  671  671  671  671  671  671  671  671  686  686  686  686  686  686  686  686  686  686  686
                       {a,b,c,d,e,f,g,h,i}  919  919  919  919  919  919  919  919  919  919  919  919  919  919  919  919  919  919  934  934
                      {a,b,c,d,e,f,g,h,i,j}  919  919  919  919  919  919  919  919  919  919  919  919  970  970  970  970  970  970  970  970
                     {a,b,c,d,e,f,g,h,i,j,k}  919  919  919  919  919  919  919  919  919  919  919  970  970  970  970  970  970  970  970  970
                    {a,b,c,d,e,f,g,h,i,j,k,l}  919  919  959  959  959  959  959  959  959  959  959  970  970  970  970  970  970  970  970  970
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  919  919  959  959  959  959  959  959  959  959  959  970  970  970  970  970  970  970  970  970
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  919  919  959  959  959  959  959  959  959  959  959  970  970  970  970  970  970  970  970  970
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  1099 1099 1099 1099 1099 1099 1099 1099 1099 1099 1099 1099 1099 1139 1139 1139 1139 1139 1139 1139
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  1099 1099 1099 1099 1099 1099 1099 1099 1099 1099 1099 1099 1099 1139 1139 1139 1139 1139 1139 1139
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}  1099 1099 1099 1099 1101 1101 1101 1101 1101 1101 1101 1101 1101 1139 1139 1139 1139 1139 1139 1141
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  1099 1099 1099 1099 1101 1101 1101 1101 1101 1101 1101 1101 1101 1139 1139 1139 1139 1139 1139 1141
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  1099 1099 1099 1099 1101 1101 1101 1101 1101 1101 1101 1101 1101 1139 1139 1139 1139 1139 1139 1141
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  1099 1099 1099 1099 1101 1101 1101 1101 1101 1101 1101 1101 1101 1139 1139 1139 1139 1139 1139 1141

------------------------------------------------------------
Segment [180- 199]:
------------------------------------------------------------
                                  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199
                                {}  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                            {a,b,c,d}  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392
                           {a,b,c,d,e}  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608
                          {a,b,c,d,e,f}  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631
                         {a,b,c,d,e,f,g}  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646
                        {a,b,c,d,e,f,g,h}  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686
                       {a,b,c,d,e,f,g,h,i}  942  942  942  942  942  942  942  942  942  942  942  942  942  942  942  942  942  942  942  942
                      {a,b,c,d,e,f,g,h,i,j}  970  970  970  970  970  970  970 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1078 1078 1078
                     {a,b,c,d,e,f,g,h,i,j,k}  970  970  970  970  970  970  970 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1078 1078 1078
                    {a,b,c,d,e,f,g,h,i,j,k,l}  970  970  970  970  970  970  970 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1078 1078 1078
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  970  970  970  970  970  970  970 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1078 1078 1078
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  970  970  970  970  970  970  970 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1078 1078 1078
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  1139 1139 1139 1139 1139 1139 1139 1139 1179 1179 1179 1179 1179 1179 1179 1179 1179 1179 1190 1190
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  1139 1139 1139 1139 1139 1139 1139 1139 1179 1179 1179 1179 1179 1179 1179 1179 1179 1190 1190 1190
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}  1141 1141 1141 1141 1141 1141 1141 1141 1141 1141 1179 1209 1209 1209 1209 1209 1209 1209 1209 1209
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  1141 1141 1141 1141 1141 1141 1141 1141 1141 1141 1179 1209 1209 1209 1209 1209 1209 1209 1209 1209
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  1141 1141 1141 1141 1141 1141 1141 1141 1141 1141 1179 1209 1209 1209 1209 1209 1209 1209 1209 1209
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  1141 1141 1141 1141 1141 1141 1141 1141 1141 1141 1179 1209 1209 1209 1209 1209 1209 1209 1209 1209

------------------------------------------------------------
Segment [200- 219]:
------------------------------------------------------------
                                  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219
                                {}  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                            {a,b,c,d}  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392
                           {a,b,c,d,e}  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608
                          {a,b,c,d,e,f}  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631
                         {a,b,c,d,e,f,g}  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646
                        {a,b,c,d,e,f,g,h}  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686
                       {a,b,c,d,e,f,g,h,i}  942  942  942  942  942  942  942  942  942  942  942  957  957  957  957  957  957  957  957  957
                      {a,b,c,d,e,f,g,h,i,j}  1078 1078 1078 1078 1078 1078 1078 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140
                     {a,b,c,d,e,f,g,h,i,j,k}  1078 1078 1078 1078 1078 1078 1078 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140
                    {a,b,c,d,e,f,g,h,i,j,k,l}  1078 1078 1078 1078 1078 1078 1078 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  1078 1078 1078 1078 1078 1078 1078 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  1078 1078 1078 1078 1078 1078 1078 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140 1140
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  1190 1190 1190 1190 1190 1190 1190 1190 1190 1190 1190 1190 1190 1230 1230 1230 1230 1230 1230 1230
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  1190 1190 1190 1190 1190 1190 1190 1190 1190 1190 1190 1190 1190 1230 1230 1230 1230 1230 1230 1230
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}                                    1271 1271 1271 1271 1271 1271 1271 1271 1311 1311 1311 1311 1311
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1311 1311 1311 1311 1311
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1311 1311 1311 1311 1311
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1271 1311 1311 1311 1311 1311

------------------------------------------------------------
Segment [220- 239]:
------------------------------------------------------------
                                  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239
                                {}  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                            {a,b,c,d}  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392  392
                           {a,b,c,d,e}  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608  608
                          {a,b,c,d,e,f}  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631  631
                         {a,b,c,d,e,f,g}  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646  646
                        {a,b,c,d,e,f,g,h}  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686  686
                       {a,b,c,d,e,f,g,h,i}  957  957  957  957  957  957  957  957  957  957  957  957  957  957  957  957  957  957  957  957
                      {a,b,c,d,e,f,g,h,i,j}  1140 1140 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180
                     {a,b,c,d,e,f,g,h,i,j,k}  1140 1140 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180
                    {a,b,c,d,e,f,g,h,i,j,k,l}  1140 1140 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1220 1220 1220
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  1140 1140 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1220 1220 1220
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  1140 1140 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1180 1220 1220 1220
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  1230 1230 1298 1298 1298 1298 1298 1298 1298 1298 1298 1298 1298 1360 1360 1360 1360 1360 1360 1360
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  1230 1230 1298 1298 1298 1298 1298 1298 1298 1298 1298 1298 1298 1360 1360 1360 1360 1360 1360 1360
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}  1311 1311 1311 1311 1311 1311 1311 1311 1311 1311 1311 1351 1351 1351 1360 1360 1360 1360 1360 1362
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  1311 1311 1311 1311 1311 1311 1311 1311 1311 1311 1311 1351 1351 1351 1360 1360 1360 1360 1360 1362
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  1311 1311 1311 1311 1311 1311 1311 1311 1311 1311 1311 1351 1351 1351 1360 1360 1360 1360 1360 1362
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  1311 1311 1311 1311 1311 1311 1311 1311 1311 1311 1311 1351 1351 1351 1360 1360 1360 1360 1360 1362
```



Segment [240– 259):

	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c,d}	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392
{a,b,c,d,e}	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608
{a,b,c,d,e,f}	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631
{a,b,c,d,e,f,g}	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646
{a,b,c,d,e,f,g,h}	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686
{a,b,c,d,e,f,g,h,i}	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957
{a,b,c,d,e,f,g,h,i,j}	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1195	1195	1203	1203	1203	1203	1203
{a,b,c,d,e,f,g,h,i,j,k}	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1195	1195	1203	1203	1203	1203	1203
{a,b,c,d,e,f,g,h,i,j,k,l}	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220	1220
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1360	1360	1360	1360	1360	1360	1360	1360	1400	1400	1400	1400	1400	1400	1400	1400	1400	1400	1400	1400
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1360	1360	1360	1360	1360	1360	1360	1360	1400	1400	1400	1400	1400	1400	1400	1400	1400	1400	1400	1400
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1362	1362	1362	1362	1362	1362	1362	1362	1400	1400	1400	1400	1400	1400	1402	1402	1402	1402	1402	1402
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1362	1362	1362	1362	1362	1362	1362	1362	1400	1400	1400	1400	1400	1400	1402	1402	1402	1402	1402	1402
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1362	1362	1362	1362	1362	1362	1362	1362	1400	1400	1400	1400	1400	1400	1402	1402	1402	1402	1402	1402
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1362	1362	1362	1362	1362	1362	1362	1362	1400	1400	1400	1400	1400	1400	1402	1402	1402	1402	1402	1402

Segment [260– 279):

	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	330	330	330	330	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	392	392	392	392	392	392	392	392	392	392	392	392	392	392	392	438	438	438	438	438
{a,b,c,d,e}	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608
{a,b,c,d,e,f}	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631
{a,b,c,d,e,f,g}	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646
{a,b,c,d,e,f,g,h}	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686
{a,b,c,d,e,f,g,h,i}	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957
{a,b,c,d,e,f,g,h,i,j}	1203	1203	1203	1203	1203	1203	1203	1203	1203	1203	1203	1203	1203	1203	1203	1203	1203	1203	1203	1203
{a,b,c,d,e,f,g,h,i,j,k}	1203	1203	1203	1203	1203	1203	1203	1203	1221	1221	1221	1221	1221	1221	1221	1221	1221	1221	1283	1283
{a,b,c,d,e,f,g,h,i,j,k,l}	1220	1220	1220	1220	1220	1220	1220	1221	1235	1235	1243	1243	1243	1243	1243	1243	1243	1243	1283	1283
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1220	1220	1220	1220	1220	1220	1220	1221	1235	1235	1243	1243	1243	1243	1243	1243	1243	1243	1283	1283
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1220	1220	1220	1220	1220	1220	1220	1221	1235	1235	1243	1243	1243	1243	1243	1243	1243	1243	1283	1283
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1400	1400	1400	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1400	1400	1400	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1402	1402	1402	1440	1470	1470	1470	1470	1470	1470	1470	1470	1470	1470	1470	1532	1532	1532	1532	1532
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1402	1402	1402	1440	1470	1470	1470	1470	1470	1470	1470	1470	1470	1470	1470	1532	1532	1532	1532	1532
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1402	1402	1402	1440	1470	1470	1470	1470	1470	1470	1470	1470	1470	1470	1470	1532	1532	1532	1532	1532
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1402	1402	1402	1440	1470	1470	1470	1470	1470	1470	1470	1470	1470	1470	1470	1532	1532	1532	1532	1532

Segment [280– 299):

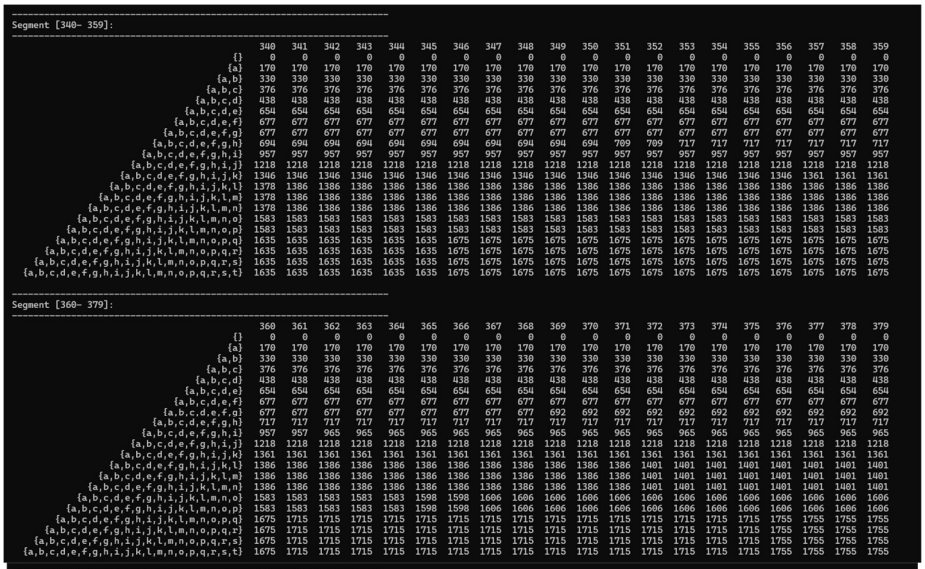
	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608	608
{a,b,c,d,e,f}	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631	631
{a,b,c,d,e,f,g}	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646	646
{a,b,c,d,e,f,g,h}	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686
{a,b,c,d,e,f,g,h,i}	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957
{a,b,c,d,e,f,g,h,i,j}	1203	1203	1203	1203	1203	1203	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218
{a,b,c,d,e,f,g,h,i,j,k}	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1323	1323	1323	1323	1323	1323	1323
{a,b,c,d,e,f,g,h,i,j,k,l}	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1323	1323	1323	1323	1323	1323	1323
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1323	1323	1323	1323	1323	1323	1323
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1283	1323	1323	1323	1323	1323	1323	1323
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1441	1455	1455	1463	1463	1463	1463
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440	1441	1455	1455	1463	1463	1463	1463
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1532	1532	1532	1532	1532	1532	1532	1532	1532	1532	1572	1572	1572	1572	1572	1572	1572	1572	1572	1572
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1532	1532	1532	1532	1532	1532	1532	1532	1532	1532	1572	1572	1572	1572	1572	1572	1572	1572	1572	1572
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1532	1532	1532	1532	1532	1532	1532	1532	1532	1532	1572	1572	1572	1572	1572	1572	1572	1572	1572	1572
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1532	1532	1532	1532	1532	1532	1532	1532	1532	1532	1572	1572	1572	1572	1572	1572	1572	1572	1572	1572

Segment [300– 319):

	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	608	608	608	608	608	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	631	631	631	631	631	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f,g}	646	646	646	646	646	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f,g,h}	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686
{a,b,c,d,e,f,g,h,i}	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957
{a,b,c,d,e,f,g,h,i,j}	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218
{a,b,c,d,e,f,g,h,i,j,k}	1323	1323	1323	1323	1323	1323	1323	1323	1323	1323	1323	1323	1323	1323	1323	1323	1323	1323	1323	1323
{a,b,c,d,e,f,g,h,i,j,k,l}	1323	1323	1323	1323	1323	1323	1323	1323	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1323	1323	1323	1323	1323	1323	1323	1323	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1463	1463	1463	1463	1463	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1463	1463	1463	1463	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503	1503	1543
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1572	1572	1572	1572	1572	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1572	1572	1572	1572	1572	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1572	1572	1572	1572	1572	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1572	1572	1572	1572	1572	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1572	1572	1572	1572	1572	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612

Segment [320– 339):

	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	650	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	677	677
{a,b,c,d,e,f,g}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	669	669	677
{a,b,c,d,e,f,g,h}	694	694	694	694	694	694	694	694	694	694	694	694	694	694	694	694	694	694	694	694
{a,b,c,d,e,f,g,h,i}	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957
{a,b,c,d,e,f,g,h,i,j}	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218
{a,b,c,d,e,f,g,h,i,j,k}	1323	1323	1323	1323	1338	1338	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346
{a,b,c,d,e,f,g,h,i,j,k,l}	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1378
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1378
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1363	1378
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1583	1583	1583	1583	1583	1583
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1583	1583	1583	1583	1583	1583
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1613	1627	1627	1635	1635
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1613	1627	1627	1635	1635
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1613	1627	1627	1635	1635
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1612	1613	1627	1627	1635	1635



Segment [340– 359]:

	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g,h}	694	694	694	694	694	694	694	694	694	694	694	709	709	717	717	717	717	717	717	717
{a,b,c,d,e,f,g,h,i}	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957	957
{a,b,c,d,e,f,g,h,i,j}	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218
{a,b,c,d,e,f,g,h,i,j,k}	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1346	1361	1361	1361
{a,b,c,d,e,f,g,h,i,j,k,l}	1378	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1378	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1378	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386	1386
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583	1583
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1635	1635	1635	1635	1635	1635	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1635	1635	1635	1635	1635	1635	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1635	1635	1635	1635	1635	1635	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1635	1635	1635	1635	1635	1635	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675	1675

Segment [360– 379]:

	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692
{a,b,c,d,e,f,g,h}	717	717	717	717	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732
{a,b,c,d,e,f,g,h,i}	965	965	965	965	965	965	965	965	965	965	965	965	965	980	980	980	980	980	988	988
{a,b,c,d,e,f,g,h,i,j}	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218
{a,b,c,d,e,f,g,h,i,j,k}	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361
{a,b,c,d,e,f,g,h,i,j,k,l}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1621	1621
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1606	1621	1621
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755	1755

Segment [380– 399]:

	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692
{a,b,c,d,e,f,g,h}	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732
{a,b,c,d,e,f,g,h,i}	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988
{a,b,c,d,e,f,g,h,i,j}	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218
{a,b,c,d,e,f,g,h,i,j,k}	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361
{a,b,c,d,e,f,g,h,i,j,k,l}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1755	1755	1755	1755	1755	1755	1755	1770	1770	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1755	1755	1755	1755	1755	1755	1755	1770	1770	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1755	1755	1755	1755	1755	1755	1755	1770	1770	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1755	1755	1755	1755	1755	1755	1755	1770	1770	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778

Segment [400– 419]:

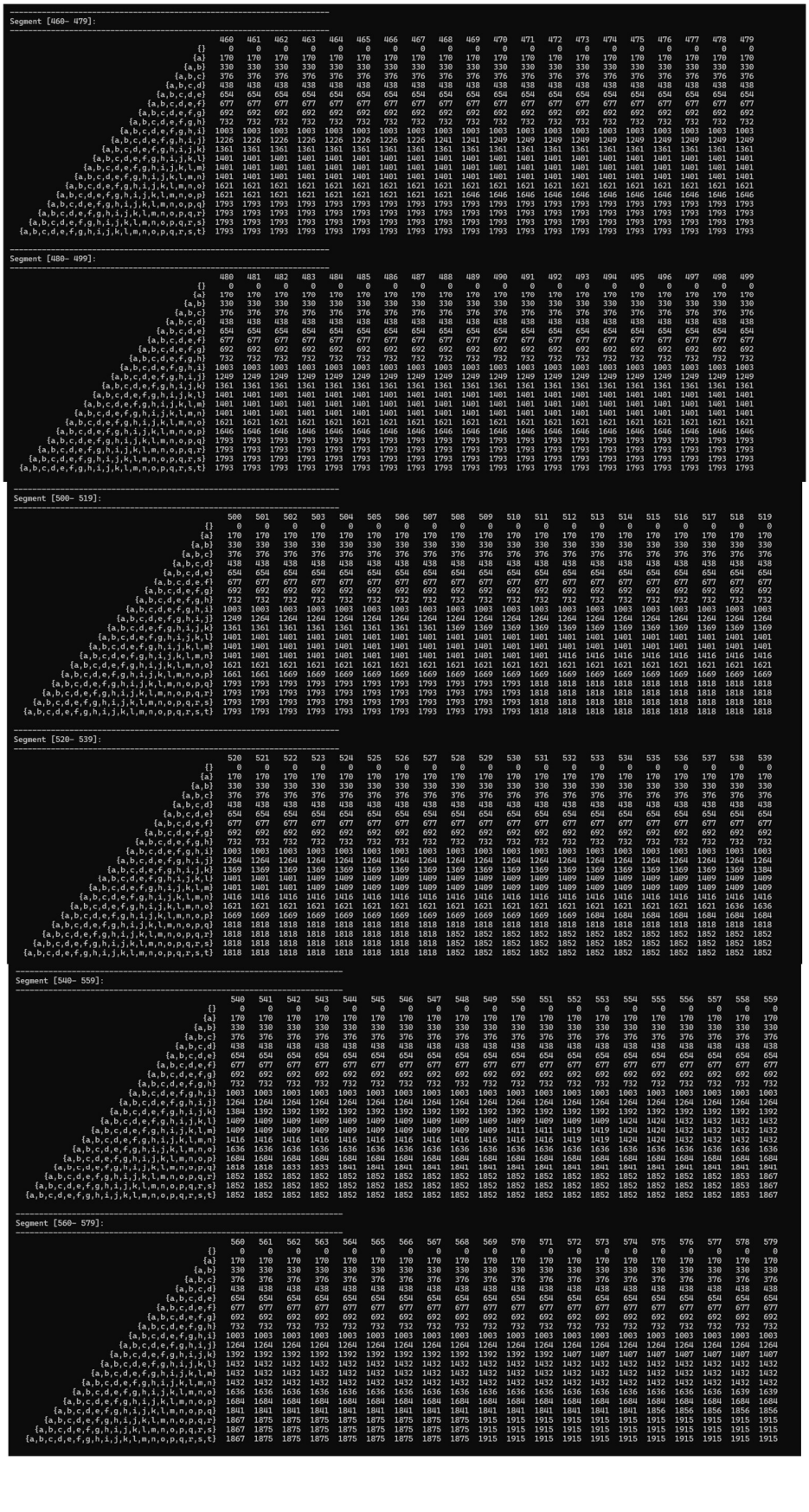
	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692
{a,b,c,d,e,f,g,h}	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732
{a,b,c,d,e,f,g,h,i}	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988
{a,b,c,d,e,f,g,h,i,j}	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218
{a,b,c,d,e,f,g,h,i,j,k}	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361
{a,b,c,d,e,f,g,h,i,j,k,l}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1755	1755	1755	1755	1755	1755	1755	1770	1770	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1755	1755	1755	1755	1755	1755	1755	1770	1770	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1755	1755	1755	1755	1755	1755	1755	1770	1770	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1755	1755	1755	1755	1755	1755	1755	1770	1770	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778

Segment [420– 439]:

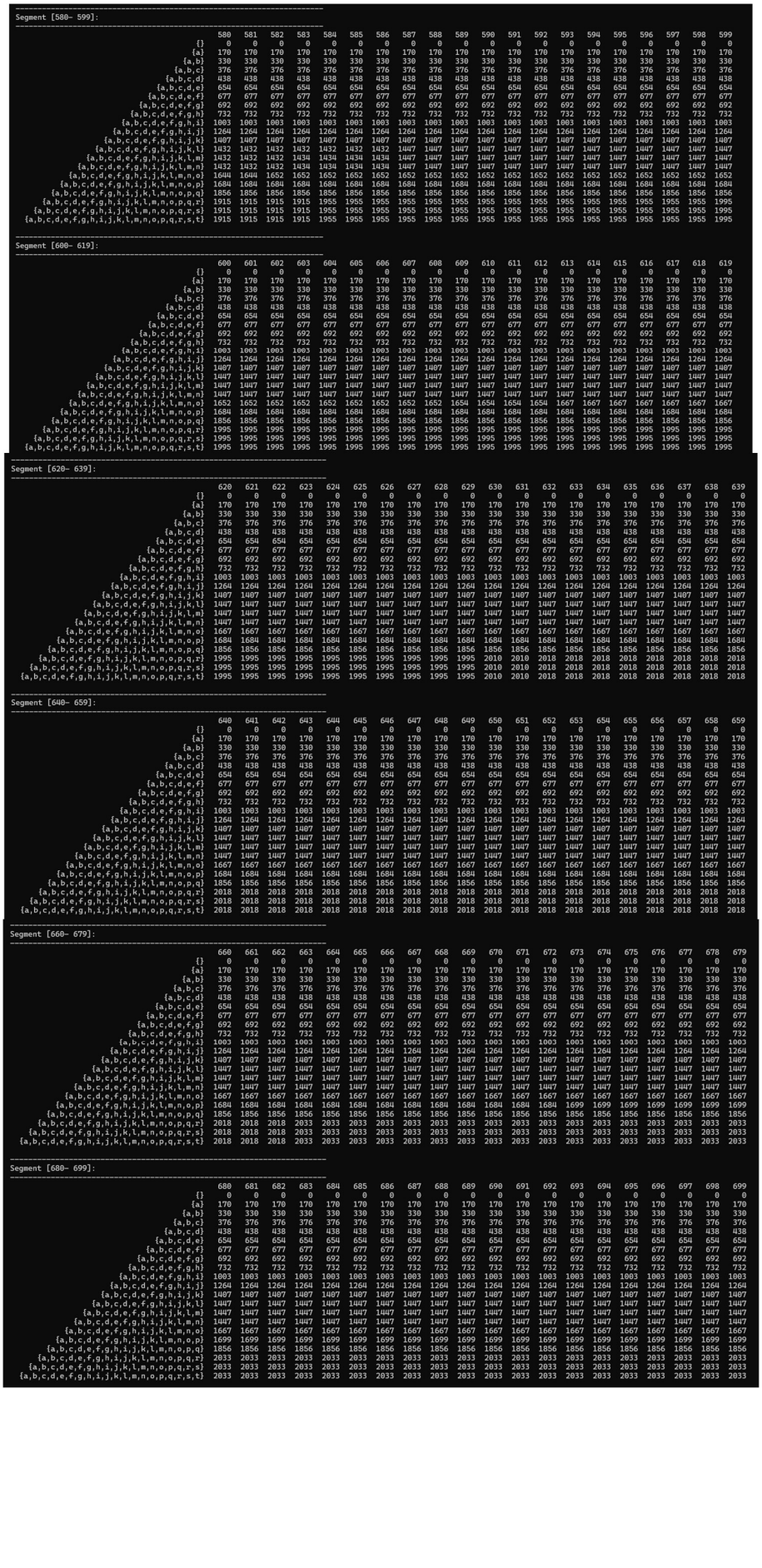
	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692
{a,b,c,d,e,f,g,h}	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732
{a,b,c,d,e,f,g,h,i}	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988
{a,b,c,d,e,f,g,h,i,j}	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1218	1226	1226	1226
{a,b,c,d,e,f,g,h,i,j,k}	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361
{a,b,c,d,e,f,g,h,i,j,k,l}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778	1778

Segment [440– 459]:

	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692
{a,b,c,d,e,f,g,h}	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732
{a,b,c,d,e,f,g,h,i}	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003
{a,b,c,d,e,f,g,h,i,j}	1226	1226	1226	1226	1226	1226	1226	1226	1226	1226	1226	1226	1226	1226	1226	1226	1226	1226	1226	1226
{a,b,c,d,e,f,g,h,i,j,k}	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361	1361
{a,b,c,d,e,f,g,h,i,j,k,l}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401	1401
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621	1621
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793	1793



```
------------------------------------------------------------
Segment [460- 479]:
------------------------------------------------------------
                                                460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479
                                              {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                                             {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                                            {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                                           {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                                         {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                                       {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                                     {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                                   {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                                 {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                               {a,b,c,d,e,f,g,h,i} 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                             {a,b,c,d,e,f,g,h,i,j} 1226 1226 1226 1226 1226 1226 1226 1226 1241 1241 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249
                           {a,b,c,d,e,f,g,h,i,j,k} 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361
                         {a,b,c,d,e,f,g,h,i,j,k,l} 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401
                       {a,b,c,d,e,f,g,h,i,j,k,l,m} 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401
                     {a,b,c,d,e,f,g,h,i,j,k,l,m,n} 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401
                   {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o} 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p} 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793
           {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793
         {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793

------------------------------------------------------------
Segment [480- 499]:
------------------------------------------------------------
                                                480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499
                                              {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                                             {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                                            {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                                           {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                                         {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                                       {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                                     {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                                   {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                                 {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                               {a,b,c,d,e,f,g,h,i} 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                             {a,b,c,d,e,f,g,h,i,j} 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249 1249
                           {a,b,c,d,e,f,g,h,i,j,k} 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 1361
                         {a,b,c,d,e,f,g,h,i,j,k,l} 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401
                       {a,b,c,d,e,f,g,h,i,j,k,l,m} 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401
                     {a,b,c,d,e,f,g,h,i,j,k,l,m,n} 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401
                   {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o} 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p} 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646 1646
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793
           {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793
         {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793

------------------------------------------------------------
Segment [500- 519]:
------------------------------------------------------------
                                                500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519
                                              {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                                             {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                                            {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                                           {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                                         {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                                       {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                                     {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                                   {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                                 {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                               {a,b,c,d,e,f,g,h,i} 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                             {a,b,c,d,e,f,g,h,i,j} 1249 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264
                           {a,b,c,d,e,f,g,h,i,j,k} 1361 1361 1361 1361 1361 1361 1361 1361 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369
                         {a,b,c,d,e,f,g,h,i,j,k,l} 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401
                       {a,b,c,d,e,f,g,h,i,j,k,l,m} 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401
                     {a,b,c,d,e,f,g,h,i,j,k,l,m,n} 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1401 1416 1416 1416 1416 1416 1416 1416 1416
                   {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o} 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p} 1661 1661 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1818 1818 1818 1818 1818 1818 1818 1818
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1818 1818 1818 1818 1818 1818 1818 1818
           {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1818 1818 1818 1818 1818 1818 1818 1818
         {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t} 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1793 1818 1818 1818 1818 1818 1818 1818 1818

------------------------------------------------------------
Segment [520- 539]:
------------------------------------------------------------
                                                520  521  522  523  524  525  526  527  528  529  530  531  532  533  534  535  536  537  538  539
                                              {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                                             {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                                            {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                                           {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                                         {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                                       {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                                     {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                                   {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                                 {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                               {a,b,c,d,e,f,g,h,i} 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                             {a,b,c,d,e,f,g,h,i,j} 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264
                           {a,b,c,d,e,f,g,h,i,j,k} 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1369 1384
                         {a,b,c,d,e,f,g,h,i,j,k,l} 1401 1401 1401 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409
                       {a,b,c,d,e,f,g,h,i,j,k,l,m} 1401 1401 1401 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409
                     {a,b,c,d,e,f,g,h,i,j,k,l,m,n} 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416
                   {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o} 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1621 1636 1636 1636
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p} 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1669 1684 1684 1684 1684 1684 1684 1684
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q} 1818 1818 1818 1818 1818 1818 1818 1818 1818 1818 1818 1818 1818 1818 1818 1818 1818 1818 1818 1818
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r} 1818 1818 1818 1818 1818 1818 1818 1818 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852
           {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s} 1818 1818 1818 1818 1818 1818 1818 1818 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852
         {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t} 1818 1818 1818 1818 1818 1818 1818 1818 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852

------------------------------------------------------------
Segment [540- 559]:
------------------------------------------------------------
                                                540  541  542  543  544  545  546  547  548  549  550  551  552  553  554  555  556  557  558  559
                                              {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                                             {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                                            {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                                           {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                                         {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                                       {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                                     {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                                   {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                                 {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                               {a,b,c,d,e,f,g,h,i} 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                             {a,b,c,d,e,f,g,h,i,j} 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264
                           {a,b,c,d,e,f,g,h,i,j,k} 1384 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392
                         {a,b,c,d,e,f,g,h,i,j,k,l} 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1424 1424 1432 1432 1432 1432
                       {a,b,c,d,e,f,g,h,i,j,k,l,m} 1409 1409 1409 1409 1409 1409 1409 1409 1409 1409 1411 1411 1419 1419 1424 1424 1432 1432 1432 1432
                     {a,b,c,d,e,f,g,h,i,j,k,l,m,n} 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1416 1419 1419 1424 1424 1432 1432 1432 1432
                   {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o} 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p} 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q} 1818 1818 1833 1833 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r} 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1853 1867
           {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s} 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1853 1867
         {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t} 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1852 1853 1867

------------------------------------------------------------
Segment [560- 579]:
------------------------------------------------------------
                                                560  561  562  563  564  565  566  567  568  569  570  571  572  573  574  575  576  577  578  579
                                              {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                                             {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                                            {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                                           {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                                         {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                                       {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                                     {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                                   {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                                 {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                               {a,b,c,d,e,f,g,h,i} 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                             {a,b,c,d,e,f,g,h,i,j} 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264
                           {a,b,c,d,e,f,g,h,i,j,k} 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1392 1407 1407 1407 1407 1407 1407 1407 1407
                         {a,b,c,d,e,f,g,h,i,j,k,l} 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432
                       {a,b,c,d,e,f,g,h,i,j,k,l,m} 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432
                     {a,b,c,d,e,f,g,h,i,j,k,l,m,n} 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432 1432
                   {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o} 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1636 1639 1639
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p} 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q} 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1841 1856 1856 1856 1856 1856
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r} 1867 1875 1875 1875 1875 1875 1875 1875 1875 1875 1915 1915 1915 1915 1915 1915 1915 1915 1915 1915
           {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s} 1867 1875 1875 1875 1875 1875 1875 1875 1875 1875 1915 1915 1915 1915 1915 1915 1915 1915 1915 1915
         {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t} 1867 1875 1875 1875 1875 1875 1875 1875 1875 1875 1915 1915 1915 1915 1915 1915 1915 1915 1915 1915
```



```
Segment [580- 599]:
------------------------------------------------------------
                                580  581  582  583  584  585  586  587  588  589  590  591  592  593  594  595  596  597  598  599
                                {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                            {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                           {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                          {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                         {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                        {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                       {a,b,c,d,e,f,g,h,i}  1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                      {a,b,c,d,e,f,g,h,i,j}  1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264
                     {a,b,c,d,e,f,g,h,i,j,k}  1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407
                    {a,b,c,d,e,f,g,h,i,j,k,l}  1432 1432 1432 1432 1432 1432 1432 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  1432 1432 1432 1434 1434 1434 1434 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  1432 1432 1432 1434 1434 1434 1434 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  1644 1644 1652 1652 1652 1652 1652 1652 1652 1652 1652 1652 1652 1652 1652 1652 1652 1652 1652 1652
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}  1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  1915 1915 1915 1915 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1995
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  1915 1915 1915 1915 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1995
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  1915 1915 1915 1915 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1955 1995

------------------------------------------------------------
Segment [600- 619]:
------------------------------------------------------------
                                600  601  602  603  604  605  606  607  608  609  610  611  612  613  614  615  616  617  618  619
                                {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                            {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                           {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                          {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                         {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                        {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                       {a,b,c,d,e,f,g,h,i}  1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                      {a,b,c,d,e,f,g,h,i,j}  1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264
                     {a,b,c,d,e,f,g,h,i,j,k}  1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407
                    {a,b,c,d,e,f,g,h,i,j,k,l}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  1652 1652 1652 1652 1652 1652 1652 1652 1652 1654 1654 1654 1654 1667 1667 1667 1667 1667 1667 1667
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}  1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995

------------------------------------------------------------
Segment [620- 639]:
------------------------------------------------------------
                                620  621  622  623  624  625  626  627  628  629  630  631  632  633  634  635  636  637  638  639
                                {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                            {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                           {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                          {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                         {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                        {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                       {a,b,c,d,e,f,g,h,i}  1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                      {a,b,c,d,e,f,g,h,i,j}  1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264
                     {a,b,c,d,e,f,g,h,i,j,k}  1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407
                    {a,b,c,d,e,f,g,h,i,j,k,l}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}  1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 2010 2010 2018 2018 2018 2018 2018 2018 2018 2018
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 2010 2010 2018 2018 2018 2018 2018 2018 2018 2018
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 2010 2010 2018 2018 2018 2018 2018 2018 2018 2018

------------------------------------------------------------
Segment [640- 659]:
------------------------------------------------------------
                                640  641  642  643  644  645  646  647  648  649  650  651  652  653  654  655  656  657  658  659
                                {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                            {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                           {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                          {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                         {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                        {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                       {a,b,c,d,e,f,g,h,i}  1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                      {a,b,c,d,e,f,g,h,i,j}  1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264
                     {a,b,c,d,e,f,g,h,i,j,k}  1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407
                    {a,b,c,d,e,f,g,h,i,j,k,l}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}  1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018 2018

------------------------------------------------------------
Segment [660- 679]:
------------------------------------------------------------
                                660  661  662  663  664  665  666  667  668  669  670  671  672  673  674  675  676  677  678  679
                                {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                            {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                           {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                          {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                         {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                        {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                       {a,b,c,d,e,f,g,h,i}  1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                      {a,b,c,d,e,f,g,h,i,j}  1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264
                     {a,b,c,d,e,f,g,h,i,j,k}  1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407
                    {a,b,c,d,e,f,g,h,i,j,k,l}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1684 1699 1699 1699 1699 1699 1699 1699
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}  1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  2018 2018 2018 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  2018 2018 2018 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  2018 2018 2018 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033

------------------------------------------------------------
Segment [680- 699]:
------------------------------------------------------------
                                680  681  682  683  684  685  686  687  688  689  690  691  692  693  694  695  696  697  698  699
                                {}   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                               {a}  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170  170
                              {a,b}  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330  330
                             {a,b,c}  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376  376
                            {a,b,c,d}  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438  438
                           {a,b,c,d,e}  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654  654
                          {a,b,c,d,e,f}  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677  677
                         {a,b,c,d,e,f,g}  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692  692
                        {a,b,c,d,e,f,g,h}  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732  732
                       {a,b,c,d,e,f,g,h,i}  1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003 1003
                      {a,b,c,d,e,f,g,h,i,j}  1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264 1264
                     {a,b,c,d,e,f,g,h,i,j,k}  1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407
                    {a,b,c,d,e,f,g,h,i,j,k,l}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                   {a,b,c,d,e,f,g,h,i,j,k,l,m}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                  {a,b,c,d,e,f,g,h,i,j,k,l,m,n}  1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447 1447
                 {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}  1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667 1667
                {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}  1699 1699 1699 1699 1699 1699 1699 1699 1699 1699 1699 1699 1699 1699 1699 1699 1699 1699 1699 1699
               {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}  1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856 1856
              {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}  2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033
             {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}  2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033
            {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}  2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033
```

Segment [700- 719]:

	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692
{a,b,c,d,e,f,g,h}	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732
{a,b,c,d,e,f,g,h,i}	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003
{a,b,c,d,e,f,g,h,i,j}	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264
{a,b,c,d,e,f,g,h,i,j,k}	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407
{a,b,c,d,e,f,g,h,i,j,k,l}	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1699	1699	1699	1699	1699	1699	1699	1699	1699	1699	1699	1699	1699	1702	1702	1707	1707	1715	1715	1715
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1856	1856	1856	1856	1856	1856	1856	1856	1856	1856	1856	1856	1856	1856	1856	1871	1871	1871	1871	1871
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033

Segment [720- 739]:

	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692
{a,b,c,d,e,f,g,h}	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732
{a,b,c,d,e,f,g,h,i}	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003
{a,b,c,d,e,f,g,h,i,j}	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264
{a,b,c,d,e,f,g,h,i,j,k}	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407
{a,b,c,d,e,f,g,h,i,j,k,l}	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1447	1447	1447	1449	1449	1449	1449	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667	1667
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1715	1715	1715	1715	1715	1715	1715	1715	1715	1715	1715	1715	1715	1715	1715	1715	1715	1715	1715	1715
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2058	2058	2058	2058	2058	2058
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2058	2058	2058	2058	2058	2058
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2033	2058	2058	2058	2058	2058	2058

Segment [740- 759]:

	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692
{a,b,c,d,e,f,g,h}	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732
{a,b,c,d,e,f,g,h,i}	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003
{a,b,c,d,e,f,g,h,i,j}	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264
{a,b,c,d,e,f,g,h,i,j,k}	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407
{a,b,c,d,e,f,g,h,i,j,k,l}	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462	1462
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1667	1667	1667	1667	1667	1667	1667	1667	1667	1669	1669	1669	1669	1682	1682	1682	1682	1682	1682	1682
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1715	1715	1715	1715	1717	1717	1717	1717	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1871	1874	1874	1879	1879	1887
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058	2058

Segment [760- 779]:

	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692
{a,b,c,d,e,f,g,h}	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732
{a,b,c,d,e,f,g,h,i}	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003
{a,b,c,d,e,f,g,h,i,j}	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264
{a,b,c,d,e,f,g,h,i,j,k}	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407
{a,b,c,d,e,f,g,h,i,j,k,l}	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1447	1447	1447	1447	1447	1457	1457	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1462	1462	1462	1462	1462	1462	1462	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1887	1887	1887	1887	1887	1887	1887	1887	1887	1887	1887	1887	1887	1887	1887	1887	1887	1887	1887	1887
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	2058	2058	2058	2058	2058	2073	2073	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	2058	2058	2058	2058	2058	2073	2073	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	2058	2058	2058	2058	2058	2073	2073	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081

Segment [780- 799]:

	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{a}	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170	170
{a,b}	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330	330
{a,b,c}	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376	376
{a,b,c,d}	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438	438
{a,b,c,d,e}	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654	654
{a,b,c,d,e,f}	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677	677
{a,b,c,d,e,f,g}	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692	692
{a,b,c,d,e,f,g,h}	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732	732
{a,b,c,d,e,f,g,h,i}	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003	1003
{a,b,c,d,e,f,g,h,i,j}	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264	1264
{a,b,c,d,e,f,g,h,i,j,k}	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407	1407
{a,b,c,d,e,f,g,h,i,j,k,l}	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447	1447
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1480	1480
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1465	1480	1480
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1682	1685	1685	1685	1685	1685	1685	1685
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730	1730
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1887	1887	1887	1887	1887	1887	1889	1889	1889	1889	1902	1902	1902	1902	1902	1902	1902	1902	1902	1902
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2096	2096
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2096	2096
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2081	2096	2096

Segment [800- 800]:

	800
{}	0
{a}	170
{a,b}	330
{a,b,c}	376
{a,b,c,d}	438
{a,b,c,d,e}	654
{a,b,c,d,e,f}	677
{a,b,c,d,e,f,g}	692
{a,b,c,d,e,f,g,h}	732
{a,b,c,d,e,f,g,h,i}	1003
{a,b,c,d,e,f,g,h,i,j}	1264
{a,b,c,d,e,f,g,h,i,j,k}	1407
{a,b,c,d,e,f,g,h,i,j,k,l}	1447
{a,b,c,d,e,f,g,h,i,j,k,l,m}	1480
{a,b,c,d,e,f,g,h,i,j,k,l,m,n}	1480
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o}	1685
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	1730
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q}	1902
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r}	2096
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s}	2096
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}	2096

Display of which stars are chosen to maximize the profit within the weight constraint of 800kg, resulting in a total profit of 2096

```
Star_R - Weight: 223kg, Profit: 240
Star_Q - Weight: 42kg, Profit: 172
Star_P - Weight: 135kg, Profit: 63
Star_O - Weight: 26kg, Profit: 220
Star_L - Weight: 15kg, Profit: 40
Star_K - Weight: 71kg, Profit: 143
Star_J - Weight: 75kg, Profit: 261
Star_I - Weight: 42kg, Profit: 271
Star_H - Weight: 15kg, Profit: 40
Star_G - Weight: 31kg, Profit: 15
Star_F - Weight: 33kg, Profit: 23
Star_E - Weight: 30kg, Profit: 216
Star_D - Weight: 11kg, Profit: 62
Star_B - Weight: 13kg, Profit: 160
Star_A - Weight: 36kg, Profit: 170
--------------------------------
Total Weight: 798kg
Total Profit: 2096
Results saved to ./Q4_AnswerOutput/knapsack_output.txt
```

These displays which include segmented views of the DP matrix and the final selection of stars, along with their total weight and profit are finally stored into the **knapsack_output.txt**.

5.4 Discussion

***Notation: n** is the number of stars/vertices and **W** is the maximum weight capacity.*

5.4.1 Time Complexity

1. **Initialization and reading input:** Reading the stars from the input file takes **O(n)** time.
2. **Filling the DP matrix:** The nested loops iterate over each star and each possible weight capacity up to the maximum weight. Each entry in the DP matrix is computed in constant time **O(1)**.

By combining these, the overall time complexity of this 0/1 Knapsack algorithm is **O(nW)**.

5.4.2 Space Complexity

1. **DP matrix:** The DP matrix is a 2D vector **dp** of size **(n+1) x (W+1)** and it is used to store the maximum profit for each subproblem.

Therefore, the overall space complexity of this 0/1 Knapsack algorithm is **O(nW)**.

5.4.3 Conclusion

The 0/1 Knapsack algorithm implemented using dynamic programming has both time and space complexity of **O(nW)**. This makes the algorithm efficient and practical for moderate values of **n** and **W**. However, for very large values of **n** or **W**, the space and time requirements can become substantial, which might necessitate alternative approaches or optimizations, such as using a space-optimized version of the DP algorithm that reduces the space complexity to **O(W)**.

6 References

Dijkstra's Shortest Path Algorithm: Brilliant math & science wiki. Brilliant. (n.d.). https://brilliant.org/wiki/dijkstras-short-path-finder/

Find shortest paths from source to all vertices using Dijkstra's algorithm. GeeksforGeeks. (2024, June 4). https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

Lecture Notes

Lab Materials

Priority queue in C++ Standard Template Library (STL). GeeksforGeeks. (2023, October 9). https://www.geeksforgeeks.org/priority-queue-in-cpp-stl/

Time complexity of Kruskals algorithm - javatpoint. www.javatpoint.com. (n.d.). https://www.javatpoint.com/time-complexity-of-kruskals-algorithm

W3schools.com. W3Schools Online Web Tutorials. (n.d.). https://www.w3schools.com/dsa/dsa_ref_knapsack.php

Yadav, P. (2022, January 25). *Disjoint set (union find algorithm)*. Scaler Topics. https://www.scaler.com/topics/data-structures/disjoint-set/

YouTube. (2015, June 14). *0/1 Knapsack Problem Dynamic Programming*. YouTube. https://www.youtube.com/watch?v=8LusJS5-AGo