A blue and white logo

Description automatically generated

**ALGORITHM DESIGN & ANALYSIS**

GROUP – 11

TC01L - TT02L

**Members and contributions**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | Student ID | Student Name | Task Description | Contribution (%) |
| 1 | 1211310827 | KAWSAR | Dijkstra’s, Kruskal’s Algorithm and Visualization | 25 |
| 2 | 1221302092 | YOUSSEF FATHY | Knapsack Algorithm | 25 |
| 3 | 1211306604 | AL REFAI AL BARAA | Selection and Heap Sort Algorithm | 25 |
| 4 | 1221301987 | BIN AFEEF ABDULLAH | Dataset1 and Dataset2 Generation | 25 |

# Project Leader: KAWSAR

## Leader ID: 1211310827

# Overview

Overview sections goes here

# Datasets Generation by (BIN AFEEF ABDULLAH)

## Dataset1:

**Abdulllah will fuck himself here!!!!!!**

Related Code Section:

import json  
import random  
  
with open('config.json', 'r') as file:  
 config = json.load(file)  
 file.close()  
  
  
def generateDataset1():  
 dataset1Config = config["datasets"]["dataset1"]  
 seed = str(dataset1Config['seed'])  
 setSizes = dataset1Config['setSizes']  
  
 for index in range(len(setSizes)):  
 dataset = [] # Initiate the dataset for the current set size  
 for \_ in range(setSizes[index] + 1):  
 # Select random digits  
 numberString = []  
 for digit in range(random.randint(1, 6)):  
 numberString.append(random.choice(seed))  
  
 # Convert selected digits to integer type data and append to current dataset  
 dataset.append(int(''.join(numberString)))  
 with open(f'datasets/dataset1/set{index}.json', 'w') as outfile:  
 json.dump(dataset, outfile)  
 outfile.close()

Sample Output:

[2, 188, 221, 1, 7, 231121, 7, 1, 283, 33, 3, 1, 171, 132211, 27288, 11121, 12208, 22, 1, 81, 31, 1, 111, 31, 81, 113028, 11, 28, 171, 1011, 130, 81172, 18117, 70881, 107, 2173, 3, 78, 2, 1, 311, 21, 3, 2, 3221, 21311, 7, 1781, 22, 277138, 1, 137111, 13082, 12221, 1, 2318, 10227, 222, 13, 21, 12, 1870, 201, 83, 10, 378371, 8110, 8331, 211871, 2132, 3271, 1122, 211, 3, 1, 1, 32, 28311, 12, 80323, 18802, 81, 11, 3202, 822802, 3013, 71203, 18320, 21, 217181, 21, 131, 71, 30, 33381, 11708, 18, 71, 1720, 27, 22177]

## Star Cluster Class:

**Abdulllah will fuck himself here!!!!!!**

FYI Abdu, dataset 2 is being generated using the star cluster class. Consider the class is a blueprint for the data structure and types. The functions name beginning with “\_” is meant to be used by the constructor only.

Related code section:

import string  
import random  
import json  
  
class StarCluster:  
 def \_\_init\_\_(self, numberOfStars: int, edgesPerStar: int, seed: str):  
 self.stars: dict = {}  
 self.edges: list[list] = []  
 self.numberOfStars = numberOfStars  
 self.edgesPerStar = edgesPerStar  
 self.seed = str(seed)  
 self.\_generate\_stars()  
 self.\_generate\_edges()  
 self.\_connect\_edges()  
  
 def \_generate\_stars(self):  
 for i in range(self.numberOfStars):  
 self.stars.update({  
 f'{string.ascii\_uppercase[i]}': {  
 "coordinates": [self.\_random\_coordinate(), self.\_random\_coordinate(), self.\_random\_coordinate()],  
 "weight": self.\_random\_positive\_integer(2, 3),  
 "profit": self.\_random\_positive\_integer(2, 2),  
 "edges": {}  
 }  
 })  
  
 def \_generate\_edges(self):  
 # Ensure each star gets at least 3 edges  
 for star in self.stars.keys():  
 connected\_stars = set()  
 while len(connected\_stars) < 3:  
 v2 = random.choice(list(self.stars.keys()))  
 if v2 != star and v2 not in connected\_stars:  
 weight = self.\_random\_positive\_integer(1, 2)  
 self.edges.append([star, v2, weight])  
 connected\_stars.add(v2)  
  
 # Generate remaining edges randomly  
 total\_edges\_needed = self.numberOfStars \* self.edgesPerStar - len(self.edges)  
 for \_ in range(total\_edges\_needed):  
 v1, v2 = random.sample(list(self.stars.keys()), 2)  
 weight = self.\_random\_positive\_integer(1, 2)  
 self.edges.append([v1, v2, weight])  
  
 def \_connect\_edges(self):  
 for edge in self.edges:  
 self.stars[edge[0]]["edges"].update({edge[1]: self.stars[edge[1]]["coordinates"]})  
  
 def \_random\_positive\_integer(self, min\_digit: int, max\_digit: int) -> int:  
 weightStr = ""  
 weight = 0  
 while int(weight) <= 0:  
 for \_ in range(random.randint(min\_digit, max\_digit)):  
 weightStr += random.choice(self.seed)  
 weight = int(weightStr)  
 weightStr = ""  
 return weight  
  
 def \_random\_coordinate(self):  
 coordinate = ""  
 for \_ in range(2):  
 coordinate += random.choice(self.seed)  
 return int(coordinate)  
  
 def save\_cluster(self):  
 cluster = {"stars": self.stars, "edges": self.edges, "shortestPaths": [], "mst": [], "conquered": []}  
 with open("datasets/dataset2/dataset2.json", 'w') as outfile:  
 json.dump(cluster, outfile)  
 outfile.close()  
  
 def get\_stars(self):  
 return self.stars  
  
 def get\_edges(self):  
 return self.edges  
  
 def depict(self):  
 print(f"Total stars: {len(self.stars)}")  
 print(f"Stars: {self.stars}")  
 print(f"Total edges: {len(self.edges)}")  
 print(f"Edges: {self.edges}")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 cluster = StarCluster(20, 3, "1211310827")

## Dataset2:

**Abdulllah will fuck himself here!!!!!!**

Related Code Section:

import json  
import pickle  
import random  
from starCluster import StarCluster  
  
with open('config.json', 'r') as file:  
 config = json.load(file)  
 file.close()

def generateDataset2():  
 dataset2Config = config["datasets"]["dataset2"]  
 seeds = dataset2Config['seeds']  
 seed = str(seeds["member1"] + seeds["member2"] + seeds["member3"])  
 numOfStars = dataset2Config['numOfStars']  
 edgePerStar = dataset2Config['edgePerStar']  
  
 # Generate and save the star cluster using the class  
 starCluster = StarCluster(numOfStars, edgePerStar, seed)  
 starCluster.save\_cluster()  
  
 # Save the cluster object for later use  
 with open("objs/dataset2.pkl", 'wb') as outfile:  
 pickle.dump(starCluster, outfile)  
 outfile.close()

Sample Output:

{"stars": {"A": {"coordinates": [56, 13, 90], "weight": 61, "profit": 59, "edges": {"D": [33, 91, 96], "C": [16, 31, 33], "B": [80, 38, 83]}}, "B": {"coordinates": [80, 38, 83], "weight": 136, "profit": 66, "edges": {"D": [33, 91, 96], "C": [16, 31, 33], "A": [56, 13, 90]}}, "C": {"coordinates": [16, 31, 33], "weight": 93, "profit": 31, "edges": {"D": [33, 91, 96], "A": [56, 13, 90], "B": [80, 38, 83]}}, "D": {"coordinates": [33, 91, 96], "weight": 38, "profit": 35, "edges": {"C": [16, 31, 33], "B": [80, 38, 83], "A": [56, 13, 90]}}}, "edges": [["A", "D", 53], ["A", "C", 3], ["A", "B", 90], ["B", "D", 6], ["B", "C", 35], ["B", "A", 5], ["C", "D", 3], ["C", "A", 3], ["C", "B", 5], ["D", "C", 96], ["D", "B", 33], ["D", "A", 3]], "shortestPaths": [], "mst": [], "conquered": []}

# Selection Sort Algorithm by (AL BARAA, AL REFAI)

**Siriyans will fuck themselves here!!!!!!**

Related Code section:

import json  
import time  
from helper import files\_in\_dir  
from tqdm import tqdm  
  
  
def selection\_sort(arr):  
 n = len(arr)  
 for i in tqdm(range(n - 1)):  
 # Subproblem: Find the minimum element in arr[i:n]  
 min\_index = i  
 for j in range(i + 1, n):  
 if arr[j] < arr[min\_index]:  
 min\_index = j  
 # Subproblem: Swap the found minimum element with arr[i]  
 arr[i], arr[min\_index] = arr[min\_index], arr[i]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 datasets = files\_in\_dir('../datasets/dataset1')  
 datasetNum = 0  
  
 # Store the time statistics in a dictionary  
 stats = {'algorithm': 'selectionSort'}  
  
 sortedArray: list[int] = []  
  
 # Load the datasets one by one, sort and save them into the ouput directory  
 for dataset in datasets:  
 # Load the datasets into json from the file  
 with open(dataset, 'r') as file:  
 arr = json.load(file)  
  
 startTime = time.time()  
 selection\_sort(arr)  
 endTime = time.time()  
  
 timeElapsed = (endTime - startTime) \* 1000 # Convert the time into milliseconds  
  
 stats[f"{len(arr)}"] = timeElapsed  
  
 # Save the sorted set as a json list into the correct file  
 with open(f"sorted/set{datasetNum}.json", 'w') as outputFile:  
 json.dump(arr, outputFile)  
 outputFile.close()  
  
 datasetNum += 1  
  
 # Print the sets having less than 500 values into the console  
 if len(arr) < 500:  
 sortedArray = arr  
 file.close()  
  
 print(f"Selection Sort (size < 500): {sortedArray}")  
  
 with open("../statistics/selection\_stats.json", 'w') as statsOut:  
 json.dump(stats, statsOut)  
 print("Selection sort statistics")  
 for key, value in stats.items():  
 if key == 'algorithm':  
 continue  
 print(f"Time taken for array size {key}: {value} milliseconds")

# Heap Sort Algorithm by (AL BARAA, AL REFAI)

**Siriyans will fuck themselves here!!!!!!**

Related Code section:

import json  
import time  
from tqdm import tqdm  
from helper import files\_in\_dir  
  
def heapify(array: list[int], n: int, i: int):  
 largestIndex = i # Initialize largest as root  
 leftIndex = 2 \* i + 1 # left = 2\*i + 1  
 rightIndex = 2 \* i + 2 # right = 2\*i + 2  
  
 # If left child exists and is larger than it's parent  
 if leftIndex < n and array[leftIndex] > array[largestIndex]:  
 largestIndex = leftIndex  
  
 # If right child exists and is larger than it's parent  
 if rightIndex < n and array[rightIndex] > array[largestIndex]:  
 largestIndex = rightIndex  
  
 # If largest is not the parent of the current subtree  
 if largestIndex != i:  
 array[i], array[largestIndex] = array[largestIndex], array[i] # swap the largest child with the current parent  
 # Recursively heapify the affected sub-tree  
 heapify(array, n, largestIndex)  
 else:  
 return  
  
def heapSort(array: list[int]) -> None:  
 n: int = len(array)

# Build a maxheap and iterate through each node that has children  
 print("Generating initial heap: ")  
 for i in tqdm(range(n // 2 - 1, -1, -1)):  
 heapify(array, n, i)  
  
 # One by one extract the root and store it at the end of the array while keeping the heap as max-heap  
 print("Generating final heap: ")  
 for i in tqdm(range(n - 1, 0, -1)):  
 array[i], array[0] = array[0], array[i] # swap  
 heapify(array, i, 0)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 datasets = files\_in\_dir('../datasets/dataset1')  
 datasetNum = 0  
  
 # Store the time statistics in a dictionary  
 stats = {'algorithm': 'heapSort'}  
  
 # Load the datasets one by one, sort and save them into the output directory  
 for dataset in datasets:  
 # Load the datasets into json from the file  
 with open(dataset, 'r') as file:  
 arr = json.load(file)  
  
 startTime = time.time()  
 heapSort(arr)  
 endTime = time.time()  
  
 timeElapsed = (endTime - startTime) \* 1000 # Convert the time into milliseconds  
  
 stats[f"{len(arr)}"] = timeElapsed  
  
 # Save the sorted set as a json list into the correct file  
 with open(f"sorted/set{datasetNum}.json", 'w') as outputFile:  
 json.dump(arr, outputFile)  
 outputFile.close()  
  
 datasetNum += 1  
  
 # Print the sets having less than 500 values into the console  
 if len(arr) < 500:  
 print(f"Heap Sort (size < 500): {arr}")  
 file.close()

with open("../statistics/heap\_stats.json", 'w') as statsOut:  
 json.dump(stats, statsOut)  
 statsOut.close()  
 print("Heap Sort Statistics")  
 for key, value in stats.items():  
 if key == 'algorithm':  
 continue  
 print(f"Time taken for array size {key}: {value} milliseconds")

# Dijkstra’s Algorithm by (KAWSAR)

**Kawsar will put his great steps here.**

# Kruskal’s Algorithm by (KAWSAR)

**Kawsar will put his great steps here.**

# Dynamic Programming- Knapsack Algorithm by (YOUSSEF)

Related Code Section:

import json  
from visualizeConquered import visualize  
  
def knapsack(profits, weights, capacity):  
 num\_items = len(profits) # Number of items  
 # Initialize the dp table with 0 values  
 dp = [[0 for \_ in range(capacity + 1)] for \_ in range(num\_items + 1)]  
  
 # Build the dp table in bottom-up manner  
 for item\_index in range(num\_items + 1):  
 for current\_capacity in range(capacity + 1):  
 if item\_index == 0 or current\_capacity == 0: # Base case: no items or knapsack capacity is 0  
 dp[item\_index][current\_capacity] = 0  
 elif weights[item\_index - 1] <= current\_capacity: # Current item can be included  
 dp[item\_index][current\_capacity] = max(  
 profits[item\_index - 1] + dp[item\_index - 1][current\_capacity - weights[item\_index - 1]],  
 dp[item\_index - 1][current\_capacity]  
 )  
 else: # Current item cannot be included  
 dp[item\_index][current\_capacity] = dp[item\_index - 1][current\_capacity]  
  
 # Printing the dp table  
 print("DP Table:")  
 print([[i for i in range(capacity + 1)]])  
 for row in dp:  
 print(row)  
  
 # Finding the items included in the knapsack for the maximum value  
 max\_value = dp[num\_items][capacity] # The maximum value in the knapsack  
 remaining\_capacity = capacity # Start with the full capacity  
 items\_included = [] # List to store the indices of included items  
  
 # Backtrack to find the included items  
 for item\_index in range(num\_items, 0, -1):  
 if max\_value <= 0: # If the result is zero, break the loop  
 break  
 if max\_value == dp[item\_index - 1][remaining\_capacity]: # Item is not included  
 continue  
 else: # Item is included  
 items\_included.append(item\_index - 1) # Add the index of the item to the list  
 max\_value -= profits[item\_index - 1] # Subtract the item's value from the result  
 remaining\_capacity -= weights[item\_index - 1] # Subtract the item's weight from the remaining capacity  
  
 return dp[num\_items][capacity], items\_included # Return the maximum value and items included  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 with open("../datasets/dataset2/dataset2.json", 'r') as dataFile:  
 data = json.load(dataFile)  
 stars = data["stars"]  
 dataFile.close()  
  
  
 starNames = [key for key, \_ in stars.items()]  
 starProfits = [profit["profit"] for \_, profit in stars.items()]  
 starWeights = [weight["weight"] for \_, weight in stars.items()]  
 knapsackCapacity = 800  
  
 maxProfit, visitedStarIndexes = knapsack(starProfits, starWeights, knapsackCapacity)  
 print(f"\nThe maximum value that can be obtained is {maxProfit}") # Print the maximum value  
 print(f"Items included: {[starNames[index] for index in visitedStarIndexes]}")  
  
 with open("../datasets/dataset2/dataset2.json", 'w') as outFile:  
 data['conquered'] = [starNames[index] for index in visitedStarIndexes]  
 json.dump(data, outFile)  
 outFile.close()  
  
 showVisual = input("1-> Visualize conquered stars: ")  
 if showVisual == '1':  
 visualize()

Sample Input:

Star Profits: [6, 33, 33, 66, 93, 39, 93, 3, 36, 1, 83, 50, 98, 31, 18, 61, 96, 66, 31, 38]

Star Weights: [635, 69, 69, 588, 83, 918, 803, 8, 13, 13, 38, 931, 38, 333, 893, 36, 33, 933, 39, 333]

Knapsack Capacity: 800

Sample Output:

Max Profit: 606

Visited Stars Index: [19, 18, 16, 15, 12, 10, 9, 8, 7, 4, 2, 1]

Visited Stars Names: ["T", "S", "Q", "P", "M", "K", "J", "I", "H", "E", "C", "B"]