|  |  |
| --- | --- |
| **NAME** | **Ms. Neerja Doshi** |
| **UID** | **2021300029** |
| **DIV** | **CE - SE – A DIV (B BATCH)** |
| **DAA EXPT - 5** | |

|  |  |
| --- | --- |
| **AIM** | Greedy approach - Fractional knapsack problem |
| **THEORY** | 1. The fractional knapsack problem involves selecting a subset of items to place into a knapsack, subject to the constraint that the total weight of the selected items cannot exceed the capacity of the knapsack. 2. Each item has a weight and a value, and the goal is to maximize the total value of the selected items while having a greedy approach towards both- weight and profit 3. The key difference between the fractional knapsack problem and the classical knapsack problem is that in the fractional version, items can be selected partially, i.e., fractions of an item can be placed into the knapsack. 4. In the classical version, items must be either included entirely or excluded entirely. 5. **ALGORITHM :**    1. **Start.**    2. **Take in the input structure array of items from the user.**    3. **Sort the array in descending order of the profit by weight ratio.**    4. **Check if weight of the item is less than or equal to the capacity of knapsack.**    5. **If less than or equal to, include the complete profit of that item in the overall profit and update the capacity.**    6. **If not, take the required fraction of the item and update the capacity to 0**    7. **Repeat steps 4 to 6 till the capacity of knapsack becomes zero.**    8. **End.** 6. **Time Complexity :**     * 1. O(nlogn), where n is the number of items, due to the need to sort the items by value-to-weight ratio. |
| **CODE** | // Knapsack  #include <stdio.h>  #include <stdlib.h>  // structure for an item in the knapsack problem  *struct* item  {  *int* itemId;  *double* weight;  *double* profitVal;  *double* profitByWeightRatio;  };  // function to solve the knapsack problem  *int* solveKnapsack(*struct* item *items*[], *int* *numOfItems*, *double* *capacity*)  {      // sorting the array  *struct* item tempItem;      for (*int* i = 0; i < *numOfItems* - 1; i++)      {          if (*items*[i].profitByWeightRatio < *items*[i + 1].profitByWeightRatio)          {              tempItem = *items*[i];  *items*[i] = *items*[i + 1];  *items*[i + 1] = tempItem;              i = -1;          }      }      printf("\nThe rearranged items based on their profit to weight ratio are as follows : \n\n");      printf("Item\tProfit\tWeight\tProfit to Weight Ratio\n");      for (*int* i = 0; i < *numOfItems*; i++)      {          printf("I%d\t%.0lf\t%.0lf\t%.2lf\n", *items*[i].itemId, *items*[i].profitVal, *items*[i].weight, *items*[i].profitByWeightRatio);      }      // selecting the required items as per greedy approach  *int* index = 0;  *double* maxProfitVal = 0;      while (index < *numOfItems*)      {          if (*items*[index].weight <= *capacity*)          {              maxProfitVal += *items*[index].profitVal;  *capacity* -= *items*[index].weight;              index++;          }          else if (*capacity* > 0)          {              maxProfitVal += (*items*[index].profitVal) \* (*capacity* / *items*[index].weight);              break;          }      }      printf("\nThe max profit value as obtained(considering greedy approach towards both weight and profit) : %.2lf\n", maxProfitVal);      return index;  }  // main function  *void* main()  {      // taking user inputs  *int* numOfItems;  *double* capacity;      printf("\nEnter the number of items to be considered for knapsack : ");      scanf("%d", &numOfItems);      printf("Enter the capacity of knapsack : ");      scanf("%lf", &capacity);      // dynamically allocating the memory for array of items  *struct* item \*items = malloc(numOfItems \* sizeof(*struct* item \*));      // taking inputs regarding all the items      printf("\nEnter the Weight and Profit value of all the items-\n");      for (*int* i = 0; i < numOfItems; i++)      {          printf("Item - %d : ", i + 1);          items[i].itemId = i + 1;          scanf("%lf", &items[i].weight);          scanf("%lf", &items[i].profitVal);          items[i].profitByWeightRatio = (items[i].profitVal) / (items[i].weight);      }      // solving the knapsack problem  *int* index = solveKnapsack(items, numOfItems, capacity);  *double* weightSum = 0;      printf("Set of items to be included in the knapsack : {");      for (*int* i = 0; i < index; i++)      {          printf("I%d, ", items[i].itemId);          weightSum += items[i].weight;      }      if (weightSum < capacity)      {          printf("(%.0lf/%.0lf) of I%d}\n\n", capacity-weightSum, items[index].weight, items[index].itemId);      }      // deallocating the used memory      free(items);  } |
| **OUTPUT** |  |
| **CONCLUSION** | By performing the above experiment , I was able to implement Greedy approach for solving.  Ive succefully understood coding Fractional Knapsack problem and its Algorithm. |