**############################################################**

**1. Fractional Knapsack**

Given *weights* and *values* of **N** items, we need to put these items in a knapsack of capacity **W** to get the *maximum* total value in the knapsack.  
**Note:** Unlike 0/1 knapsack, you are allowed to break the item.

**Example 1:**

**Input:**

N = 3, W = 50

values[] = {60,100,120}

weight[] = {10,20,30}

**Output:**

240.00

**Explanation:**Total maximum value of item

we can have is 240.00 from the given

capacity of sack.

**Example 2:**

**Input:**

N = 2, W = 50

values[] = {60,100}

weight[] = {10,20}

**Output:**

160.00

**Explanation:**

Total maximum value of item

we can have is 160.00 from the given

capacity of sack.

**Your Task** :  
Complete the function ***fractionalKnapsack****()* that receives maximum capacity , array of structure/class and size n and returns a double value representing the maximum value in knapsack.  
**Note:**The details of structure/class is defined in the comments above the given function.

**Expected Time Complexity** : O(NlogN)  
**Expected Auxilliary Space**: O(1)

**Constraints:**  
1 <= N <= 105  
1 <= W <= 105

static bool cmp(Item a, Item b)

{ return ((double)a.value/(double)a.weight) > ((double)b.value/(double)b.weight);

}

double fractionalKnapsack(int W, Item arr[], int n)

{ sort(arr, arr + n, cmp);

double sum = 0, val = 0;

int i = 0;

while(sum + arr[i].weight <= W && i < n)

{ sum += arr[i].weight;

val += arr[i].value;

i++;

}

if(W - sum == 0)

return val;

if(i != n)

val += (double)(arr[i].value \* (W - sum)) / (double)(arr[i].weight);

return val;

}

**############################################################**

**2. Largest number with given sum**

Geek lost the password of his super locker. He remembers the number of digits **N** as well as the sum **S** of all the digits of his password. He know that his password is the largest number of **N** digits that can be made with given sum **S**. As he is busy doing his homework, help him retrieving his password.

**Example 1:**

**Input:**

N = 5, S = 12

**Output:**

93000

**Explanation:**

Sum of elements is 12. Largest possible

5 digit number is 93000 with sum 12.

**Example 2:**

**Input:**

N = 3, S = 29

**Output:**

-1

**Explanation:**

There is no such three digit number

whose sum is 29.

**Your Task :**  
You don't need to read input or print anything. Your task is to complete the function l**argestNumber()** which takes 2 integers **N** and **S** as input parameters and returns the password in the form of string, else return "-1" in the form of string.

**Constraints:**  
1 ≤ N ≤ 104  
0 ≤ S ≤ 9\*104

**Expected Time Complexity** : O(N)  
**Expected Auxilliary Space**: O(1)

string largestNumber(int n, int sum)

{

if(sum > 9\*n)

return "-1";

string ans = "";

for(int i=0; i<n; i++)

{ if(sum >= 9)

{ ans.push\_back('9');

sum -= 9;

}

else if(sum > 0)

{ ans.append(to\_string(sum));

sum = 0;

}

else

ans.push\_back('0');

}

return ans;

}

**############################################################**

**3. Job Sequencing Problem**

Given a set of **N** jobs where each job *i* has a deadline and profit associated to it. Each job takes *1* unit of time to complete and only one job can be scheduled at a time. We earn the profit if and only if the job is completed by its deadline. The task is to find the **maximum profit** and the number of jobs done.

**Note:**Jobs will be given in the form (Job id, Deadline, Profit) associated to that Job.

**Example 1:**

**Input:**

N = 4

Jobs = (1,4,20)(2,1,10)(3,1,40)(4,1,30)

**Output:**

2 60

**Explanation:**

2 jobs can be done with

maximum profit of 60 (20+40).

**Example 2:**

**Input:**

N = 5

Jobs = (1,2,100)(2,1,19)(3,2,27)

(4,1,25)(5,1,15)

**Output:**

2 127

**Explanation:**

2 jobs can be done with

maximum profit of 127 (100+27).

**Your Task** :  
You don't need to read input or print anything. Your task is to complete the function **JobScheduling()** which takes an Integer N and an array of Jobs(Job id, Deadline, Profit) as input and returns the count of jobs and maximum profit.

**Expected Time Complexity**: O(NlogN)  
**Expected Auxilliary Space**: O(N)

**Constraints:**  
1 <= N <= 105  
1 <= Deadline <= 100  
1 <= Profit <= 500

static bool cmp(Job a, Job b)

{

return a.profit > b.profit;

}

vector<int> JobScheduling(Job arr[], int n)

{ vector<int> ans(n, -1);

sort(arr, arr+n, cmp);

int pro = 0, count = 0, j;

for(int i=0; i<n; i++)

{ j = (arr[i].dead - 1) > n-1? n-1: arr[i].dead - 1;

while(j >= 0)

{ if(ans[j] == -1)

{ ans[j] = arr[i].profit;

break;

}

j--;

}

}

for(int i=0; i<n; i++)

if(ans[i] != -1)

{ count++;

pro += ans[i];

}

return {count, pro};

}

**############################################################**

**4. Huffman Decoding**

Given a encoded binary string and a Huffman MinHeap tree, your task is to complete the function decodeHuffmanData(), which decodes the binary encoded string and return the original string.

**Note:** Each node of the min heap contains 2 data members, a character and an integer to denote its frequency. The character '$' is the special character used for internal nodes whose min heap node only need a integer field.

**Example 1:**

**Input :**

binaryString =

0000000000001100101010101011111111010101010

Min Heap Tree =

$(20)

/ \

/ \

$(8) \

/ \ \

$(3) \ $(12)

/ \ \ / \

B(1) D(2) E(5) C(6) A(6)

**Output:** AAAAAABCCCCCCDDEEEEE

**Explanation:**

The following chart can be made from the

given min heap tree.

**character frequency code**

A 6 00

B 1 110

C 6 01

D 2 111

E 5 10

**Example 2:**

**Input :**

binaryString =

01110100011111000101101011101000111

Min Heap Tree =

$(13)

/ \

/ \

/ \

$(5) \

/ \ \

/ \ \

$(3) \ $(8)

/ \ \ / \

$(2) \ \ $(4) \

/ \ \ \ / \ \

f(1) o(1) r(1) g(2) k(2) s(2) e(4)

**Output:** geeksforgeeks

**Explanation:**

The following chart can be made from the

given min heap tree.

**character frequency code**

f 1 0000

o 1 0001

r 1 001

g 2 01

k 2 100

s 2 101

e 4 11

**Your Task:**  
You dont need to read input or print anything. Complete the function **decodeHuffmanData()** which takes the root of the Huffman min heap tree and the encoded Binary String as input parameters and returns the decoded string.

**Expected Time Complexity:** O(N log N)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 ≤ N ≤ 10^3

string decodeHuffmanData(struct MinHeapNode\* root, string str)

{ int n = str.length(), i = 0;

string ans = "";

while(i < n)

{ MinHeapNode \*temp = root;

while(true)

{ if(temp->left == NULL && temp->right == NULL)

{ ans += temp->data;

break;

}

if(str[i] == '0')

{ temp = temp->left;

}

if(str[i] == '1')

{ temp = temp->right;

}

i++;

}

}

return ans;

}

**############################################################**