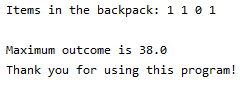
**Листинг практической работы № 16**

package ru.mirea.ikbo20.AOD.pr16;  
import java.util.\*;  
  
class Item {  
 *// Переменная для веса* float weight;  
  
 *// Переменная для значения* int value;  
  
 *// Переменная для индексов элементов в рюкзаке* int idx;  
 public Item(int value, float weight, int idx)  
 {  
 this.value = value;  
 this.weight = weight;  
 this.idx = idx;  
 }  
}  
  
class Node {  
 float ub, lb, tv, tw;  
 int level;  
 boolean flag;  
 public Node() {}  
 public Node(Node cpy)  
 {  
 this.tv = cpy.tv;  
 this.tw = cpy.tw;  
 this.ub = cpy.ub;  
 this.lb = cpy.lb;  
 this.level = cpy.level;  
 this.flag = cpy.flag;  
 }  
}  
  
*// Компаратор для сортировки по нижней границе*class sortByC implements Comparator<Node> {  
 public int compare(Node a, Node b)  
 {  
 boolean temp = a.lb > b.lb;  
 return temp ? 1 : -1;  
 }  
}  
  
class sortByRatio implements Comparator<Item> {  
 public int compare(Item a, Item b)  
 {  
 boolean temp = (float)a.value  
 / a.weight  
 > (float)b.value  
 / b.weight;  
 return temp ? -1 : 1;  
 }  
}  
  
class Backpack {  
 private static int *size*;  
 private static float *capacity*;  
  
 static float upperBound(float tv, float tw,  
 int idx, Item[] arr)  
 {  
 float value = tv;  
 float weight = tw;  
 for (int i = idx; i < *size*; i++) {  
 if (weight + arr[i].weight  
 <= *capacity*) {  
 weight += arr[i].weight;  
 value -= arr[i].value;  
 }  
 else {  
 value -= (float)(*capacity* - weight)  
 / arr[i].weight  
 \* arr[i].value;  
 break;  
 }  
 }  
 return value;  
 }  
  
 *//Вычислить нижнюю границу* static float lowerBound(float tv, float tw,  
 int idx, Item[] arr)  
 {  
 float value = tv;  
 float weight = tw;  
 for (int i = idx; i < *size*; i++) {  
 if (weight + arr[i].weight  
 <= *capacity*) {  
 weight += arr[i].weight;  
 value -= arr[i].value;  
 }  
 else {  
 break;  
 }  
 }  
 return value;  
 }  
  
 static void assign(Node a, float ub, float lb,  
 int level, boolean flag,  
 float tv, float tw)  
 {  
 a.ub = ub;  
 a.lb = lb;  
 a.level = level;  
 a.flag = flag;  
 a.tv = tv;  
 a.tw = tw;  
 }  
  
 public static void solve(Item[] arr)  
 {  
 *// Сортировка* Arrays.*sort*(arr, new sortByRatio());  
  
 Node current, left, right;  
 current = new Node();  
 left = new Node();  
 right = new Node();  
  
 float minLB = 0, finalLB  
 = Integer.*MAX\_VALUE*;  
 current.tv = current.tw = current.ub  
 = current.lb = 0;  
 current.level = 0;  
 current.flag = false;  
  
 PriorityQueue<Node> pq  
 = new PriorityQueue<Node>(  
 new sortByC());  
 pq.add(current);  
  
 boolean[] currPath = new boolean[*size*];  
 boolean[] finalPath = new boolean[*size*];  
  
 while (!pq.isEmpty()) {  
 current = pq.poll();  
 if (current.ub > minLB  
 || current.ub >= finalLB) {  
 continue;  
 }  
  
 if (current.level != 0)  
 currPath[current.level - 1]  
 = current.flag;  
  
 if (current.level == *size*) {  
 if (current.lb < finalLB) {  
 for (int i = 0; i < *size*; i++)  
 finalPath[arr[i].idx]  
 = currPath[i];  
 finalLB = current.lb;  
 }  
 continue;  
 }  
  
 int level = current.level;  
 *assign*(right, *upperBound*(current.tv,  
 current.tw,  
 level + 1, arr),  
 *lowerBound*(current.tv, current.tw,  
 level + 1, arr),  
 level + 1, false,  
 current.tv, current.tw);  
  
 if (current.tw + arr[current.level].weight  
 <= *capacity*) {  
 left.ub = *upperBound*(  
 current.tv  
 - arr[level].value,  
 current.tw  
 + arr[level].weight,  
 level + 1, arr);  
 left.lb = *lowerBound*(  
 current.tv  
 - arr[level].value,  
 current.tw  
 + arr[level].weight,  
 level + 1,  
 arr);  
 *assign*(left, left.ub, left.lb,  
 level + 1, true,  
 current.tv - arr[level].value,  
 current.tw  
 + arr[level].weight);  
 }  
 else {  
 left.ub = left.lb = 1;  
 }  
  
 minLB = Math.*min*(minLB, left.lb);  
 minLB = Math.*min*(minLB, right.lb);  
  
 if (minLB >= left.ub)  
 pq.add(new Node(left));  
 if (minLB >= right.ub)  
 pq.add(new Node(right));  
 }  
 System.*out*.print("Items in the backpack: ");  
 for (int i = 0; i < *size*; i++) {  
 if (finalPath[i])  
 System.*out*.print("1 ");  
 else  
 System.*out*.print("0 ");  
 }  
 System.*out*.println("\n\n" +  
 "Maximum outcome is "  
 + (-finalLB));  
 }  
 public static void main(String[] args)  
 {  
 *size* = 4;  
 *capacity* = 15;  
  
 Item[] arr = new Item[*size*];  
 arr[0] = new Item(10, 2, 0);  
 arr[1] = new Item(10, 4, 1);  
 arr[2] = new Item(12, 6, 2);  
 arr[3] = new Item(18, 9, 3);  
  
 *solve*(arr);  
 }  
}

**Демонстрация работы программы**

****