

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждениевысшего образования"МИРЭА - Российский технологический университет"РТУ МИРЭА |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА №1**

**Метод Парето**

**по дисциплине**

**«Теория принятия решений»**

Студент группы: ИКБО-04-19 Сюртуков З.А *(Фамилия студента)*

Руководитель работы Железняк Л.М.\_

*(Фамилия преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2021

**Постановка задачи**

**Задача:** выбор оптимальной планеты для проживания

**Варианты:** Меркурий, Венера, Земля, Марс, Юпитер, Сатурн, Уран, Нептун, Плутон

**Критерии:**

Температура (-) – чем ниже температура, тем лучше (до нижней границы).

Плотность (+) – чем выше плотность планеты, тем лучше.

Вода (+) – вода необходимый для жизни элемент. Чем больше, тем лучше.

Вулканы (-) – вулканы мешают жизни. Чем меньше, тем лучше.

Атмосфера (+) – атмосфера необходима для жизни. Чем больше атмосферы, тем лучше (до верхней границы).

**Выбор планеты для проживания по методу Парето**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Вариант | Температура  (-) | Плотность  (+) | Вода  (+) | Вулканы  (-) | Атмосфера  (+) |
| 1 | Меркурий | 427 | 5.43 | 0 | 423 | 0 |
| 2 | Венера | 600 | 5.42 | 0 | 1023 | 90 |
| 3 | Земля | 20 | 5.5 | 5000 | 56 | 1 |
| 4 | Марс | -40 | 5.4 | 5 | 100 | 2 |
| 5 | Юпитер | -108 | 0.6 | 300 | 0 | 3000 |
| 6 | Сатурн | -109 | 0.7 | 300 | 0 | 3000 |
| 7 | Уран | -100 | 0.8 | 100 | 0 | 2000 |
| 8 | Нептун | -150 | 0.8 | 200 | 0 | 2500 |
| 9 | Плутон | -150 | 5.1 | 0 | 40 | 0.5 |

**Метод Парето**

Пусть X1, X2 ∈ D . Если для всех критериев f1, f2, ..., fk имеют место неравенства fk(x2) >= fk(x1), k =1,2,...,K, причем хотя бы одно неравенство строгое, то говорят, что решение X2 предпочтительнее решения X1 . Условие предпочтительности принято обозначать в виде X2 > X1.

Определение (оптимальность по Парето). В задаче МКО точка X0 ∈ D называется оптимальной по Парето, если не существует другой точки X ∈ D , которая была бы предпочтительнее, чем X0 .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | X | X | X | X | X | X | X | X | X |
| 2 | H | X | X | X | X | X | X | X | X |
| 3 | A3 | H | X | X | X | X | X | X | X |
| 4 | H | H | H | X | X | X | X | X | X |
| 5 | H | H | H | H | X | X | X | X | X |
| 6 | H | H | H | H | A6 | X | X | X | X |
| 7 | H | H | H | H | H | H | X | X | X |
| 8 | H | H | H | H | H | H | A8 | X | X |
| 9 | H | H | H | H | H | H | H | H | X |

**Программный код на языке Java для реализации метода Парето**

Solution.java

import First.Alternative;  
import First.Pareto;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Solution {  
 public static void main(String[] args) {  
 List<Alternative> alternatives = new ArrayList<>();  
 List<String> names = new ArrayList<>();  
 List<Boolean> heights = new ArrayList<>();  
 List<String> importance = new ArrayList<>();  
 names.add("Температура");  
 names.add("Плотность");  
 names.add("Вода");  
 names.add("Вулканы");  
 names.add("Атмосфера");  
 importance.add("Температура");  
 importance.add("Атмосфера");  
 importance.add("Вода");  
 importance.add("Вулканы");  
 importance.add("Плотность");  
 List<Alternative.Criteria> narrowing = new ArrayList<>();  
 narrowing.add(new Alternative.Criteria("Температура", 100, true));  
 narrowing.add(new Alternative.Criteria("Вода", 3, false));  
 narrowing.add(new Alternative.Criteria("Атмосфера", 0, false));  
 heights.add(false);  
 heights.add(true);  
 heights.add(true);  
 heights.add(false);  
 heights.add(true);  
 List<Double> counts = new ArrayList<>();  
 counts.add(427.);  
 counts.add(5.43);  
 counts.add(0.);  
 counts.add(423.);  
 counts.add(0.);  
 alternatives.add(new Alternative("Меркурий", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(600.);  
 counts.add(5.42);  
 counts.add(0.);  
 counts.add(1023.);  
 counts.add(90.);  
 alternatives.add(new Alternative("Венера", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(20.);  
 counts.add(5.5);  
 counts.add(5000.);  
 counts.add(56.);  
 counts.add(1.);  
 alternatives.add(new Alternative("Земля", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-40.);  
 counts.add(5.4);  
 counts.add(5.);  
 counts.add(100.);  
 counts.add(2.);  
 alternatives.add(new Alternative("Марс", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-108.);  
 counts.add(0.6);  
 counts.add(300.);  
 counts.add(0.);  
 counts.add(3000.);  
 alternatives.add(new Alternative("Юпитер", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-109.);  
 counts.add(0.7);  
 counts.add(300.);  
 counts.add(0.);  
 counts.add(3000.);  
 alternatives.add(new Alternative("Сатурн", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-100.);  
 counts.add(0.8);  
 counts.add(100.);  
 counts.add(0.);  
 counts.add(2000.);  
 alternatives.add(new Alternative("Уран", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-150.);  
 counts.add(0.8);  
 counts.add(200.);  
 counts.add(0.);  
 counts.add(2500.);  
 alternatives.add(new Alternative("Нептун", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-150.);  
 counts.add(5.1);  
 counts.add(0.);  
 counts.add(40.);  
 counts.add(0.5);  
 alternatives.add(new Alternative("Плутон", names, heights, counts));  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*narrowing*(alternatives, narrowing);  
 System.*out*.println();  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*paretoMethod*(alternatives);  
 System.*out*.println();  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*importance*(alternatives, importance);  
 System.*out*.println();  
 System.*out*.println(alternatives);  
 }  
}

Alternative.java

package First;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Objects;  
  
public class Alternative implements Comparable<Alternative> {  
 private final String name;  
 private final List<Criteria> criterias = new ArrayList<>();  
  
 public String getName() {  
 return name;  
 }  
  
 public Alternative(String name, List<String> nCriterias, List<Boolean> bCriterias, List<Double> iCriterias) {  
 this.name = name;  
 for (int i = 0; i < nCriterias.size(); i++)  
 criterias.add(new Criteria(nCriterias.get(i), iCriterias.get(i), bCriterias.get(i)));  
 }  
  
 public boolean checkToNarrowing(List<Criteria> criterias2) {  
 for (Criteria criteria : criterias2) {  
 if (criteria.height && criterias.get(criterias.indexOf(criteria)).count > criteria.count)  
 return true;  
 if (!criteria.height && criterias.get(criterias.indexOf(criteria)).count < criteria.count)  
 return true;  
 }  
 return false;  
 }  
  
 public double getCount(String name) {  
 for (Criteria criteria : criterias)  
 if (criteria.criteriaName.equals(name))  
 return criteria.count;  
 return 0;  
 }  
  
 public boolean getHeight(String name) {  
 for (Criteria criteria : criterias)  
 if (criteria.criteriaName.equals(name))  
 return criteria.height;  
 return false;  
 }  
  
 public int compareTo(Alternative o) {  
 int res = 0;  
 for (int i = 0; i < criterias.size(); i++) {  
 if (criterias.get(i).height && criterias.get(i).count > o.criterias.get(i).count || !criterias.get(i).height && criterias.get(i).count < o.criterias.get(i).count) {  
 if (res < 0)  
 return 0;  
 res++;  
 } else if (criterias.get(i).height && criterias.get(i).count < o.criterias.get(i).count || !criterias.get(i).height && criterias.get(i).count > o.criterias.get(i).count)  
 if (res > 0)  
 return 0;  
 res--;  
 }  
 return res;  
 }  
  
 public String toString() {  
 StringBuilder builder = new StringBuilder("Вариант ").append(name).append(":");  
 for (Criteria criteria : criterias)  
 builder.append(", ").append(criteria.criteriaName).append(" = ").append(criteria.count);  
 return builder.toString();  
 }  
  
  
 public static class Criteria {  
 private final String criteriaName;  
 private final double count;  
 private final boolean height;  
  
  
 public Criteria(String criteriaName, double count, boolean height) {  
 this.criteriaName = criteriaName;  
 this.count = count;  
 this.height = height;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Criteria criteria = (Criteria) o;  
 return Objects.*equals*(criteriaName, criteria.criteriaName);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(criteriaName);  
 }  
  
 }  
}

Pareto.java

package First;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Pareto {  
 public static class ParetoHash {  
 public List<Alternative.Criteria> criteriaToNarrowing;  
 public List<String> importance;  
 public List<Alternative> alternatives;  
  
 public ParetoHash(List<Alternative.Criteria> criteriaToNarrowing, List<String> importance, List<Alternative> alternatives) {  
 this.criteriaToNarrowing = criteriaToNarrowing;  
 this.importance = importance;  
 this.alternatives = alternatives;  
 }  
 }  
  
 public static ParetoHash in() throws IOException {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
 List<Alternative> alternatives = new ArrayList<>();  
 System.*out*.println("Введите количество критериев");  
 int n = Integer.*parseInt*(reader.readLine());  
 List<String> names = new ArrayList<>();  
 List<Boolean> heights = new ArrayList<>();  
 List<Alternative.Criteria> criteriaToNarrowing = new ArrayList<>();  
 List<String> importance = new ArrayList<>();  
 for (int i = 0; i < n; i++) {  
 System.*out*.println("Введите имя для критерия");  
 names.add(reader.readLine());  
 System.*out*.println("Критерий положительный или отрицательный(true/false)?");  
 heights.add(Boolean.*parseBoolean*(reader.readLine()));  
 System.*out*.println("Хотите задать верхнюю границу для критерия(1-да, 2-нет)?");  
 if (Integer.*parseInt*(reader.readLine()) == 1) {  
 System.*out*.println("Задайте значение верхней границы");  
 criteriaToNarrowing.add(new Alternative.Criteria(names.get(names.size() - 1), Integer.*parseInt*(reader.readLine()), true));  
 }  
 System.*out*.println("Хотите задать нижнюю границу для критерия(1-да, 2-нет)?");  
 if (Integer.*parseInt*(reader.readLine()) == 1) {  
 System.*out*.println("Задайте значение нижней границы");  
 criteriaToNarrowing.add(new Alternative.Criteria(names.get(names.size() - 1), Integer.*parseInt*(reader.readLine()), false));  
 }  
 }  
 System.*out*.println("Напишите название критерия для добавления его в приоритетность или exit для продолжения");  
 String impStr = reader.readLine();  
 while (!impStr.equals("exit")) {  
 importance.add(impStr);  
 impStr = reader.readLine();  
 }  
 System.*out*.println("Введите количество альтернатив");  
 n = Integer.*parseInt*(reader.readLine());  
 for (int i = 0; i < n; i++) {  
 System.*out*.println("Введите название варианта");  
 String name = reader.readLine();  
 List<Double> counts = new ArrayList<>();  
 for (String na : names) {  
 System.*out*.println("Введите значение для критерия " + na);  
 counts.add(Double.*parseDouble*(reader.readLine()));  
 }  
 alternatives.add(new Alternative(name, names, heights, counts));  
 }  
 return new ParetoHash(criteriaToNarrowing, importance, alternatives);  
 }  
  
 public static void paretoMethod(List<Alternative> alternatives) {  
 label:  
 for (int i = 0; i < alternatives.size(); ) {  
 for (int j = i + 1; j < alternatives.size(); ) {  
 int res = alternatives.get(i).compareTo(alternatives.get(j));  
 if (res < 0) {  
 System.*out*.println(alternatives.get(j).getName() + " доминирует над " + alternatives.get(i).getName());  
 alternatives.remove(i);  
 continue label;  
 } else if (res > 0) {  
 System.*out*.println(alternatives.get(i).getName() + " доминирует над " + alternatives.get(j).getName());  
 alternatives.remove(j);  
 continue;  
 }  
 j++;  
 }  
 i++;  
 }  
 }  
  
 public static void narrowing(List<Alternative> alternatives, List<Alternative.Criteria> criteries) {  
 for (int i = 0; i < alternatives.size(); ) {  
 if (alternatives.get(i).checkToNarrowing(criteries)) {  
 System.*out*.println("Вариант " + alternatives.get(i).getName() + " исключается в результате сужения");  
 alternatives.remove(alternatives.get(i));  
 } else  
 i++;  
 }  
 }  
  
 public static void importance(List<Alternative> alternatives, List<String> criteries) {  
 for (String criteria : criteries) {  
 if (alternatives.size() == 1)  
 return;  
 boolean height = alternatives.get(0).getHeight(criteria);  
 List<Alternative> best = new ArrayList<>();  
 for (int i = 0; i < alternatives.size(); ) {  
 Alternative alternative = alternatives.get(i);  
 if (best.isEmpty() || (height && alternative.getCount(criteria) > best.get(0).getCount(criteria)) || (!height && alternative.getCount(criteria) < best.get(0).getCount(criteria))) {  
 alternatives.removeAll(best);  
 if (!best.isEmpty()) {  
 StringBuilder builder = new StringBuilder("Варианты ");  
 for (Alternative alternative1 : best) {  
 builder.append(alternative1.getName()).append(", ");  
 alternatives.remove(alternative1);  
 }  
 builder.append(" проигрывают варианту ").append(alternative.getName()).append(" по критерию ").append(criteria);  
 System.*out*.println(builder);  
 }  
 best = new ArrayList<>();  
 best.add(alternative);  
 i = 1;  
 } else if ((height && alternative.getCount(criteria) < best.get(0).getCount(criteria)) || (!height && alternative.getCount(criteria) > best.get(0).getCount(criteria))) {  
 System.*out*.println("Вариант " + alternative.getName() + " проигрывает варианту " + best.get(0).getName());  
 alternatives.remove(alternative);  
 }  
 else {  
 best.add(alternative);  
 i++;  
 }  
 }  
 }  
 }  
  
 public static void out(List<Alternative> alternatives) {  
 for (Alternative alternative : alternatives)  
 System.*out*.println(alternative);  
 }  
}

**Скриншот результата работы программы**

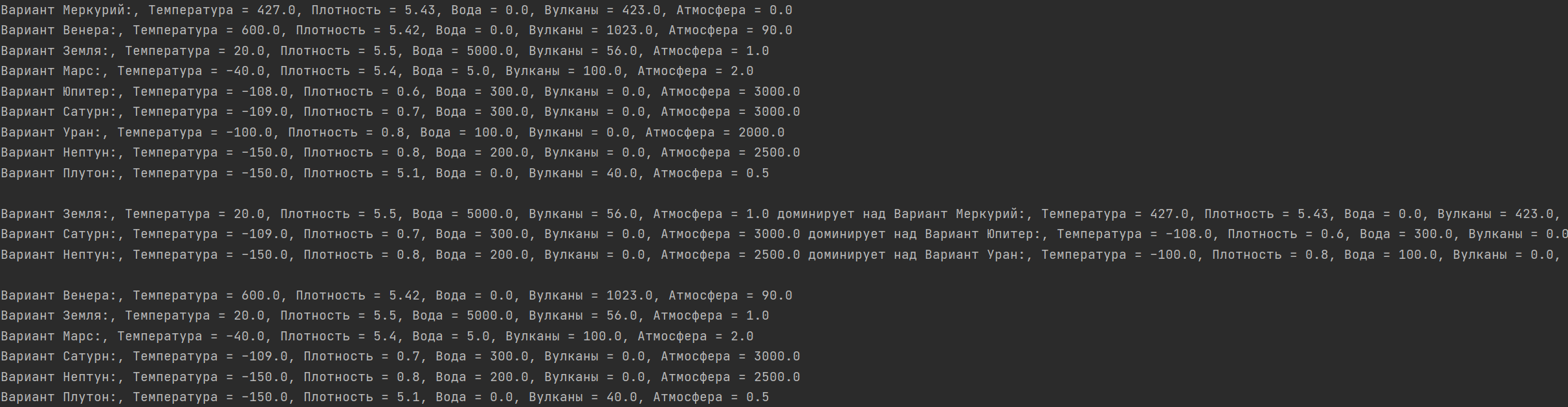
****

Рисунок 1. Вывод программы “метод Парето”



|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждениевысшего образования"МИРЭА - Российский технологический университет"РТУ МИРЭА |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА №2**

**Метод Парето**

**по дисциплине**

**«Теория принятия решений»**

Студент группы: ИКБО-04-19 Сюртуков З.А *(Фамилия студента)*

Руководитель работы Железняк Л.М.\_

*(Фамилия преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2021

**Сужение Парето**

Сужение можно осуществлять с помощью указания нижних и верхних границ критериев. Если в каком-то варианте какой-то критерий в указанные границы не попадает, то данные вариант из множества Парето исключается. Укажем, например, для предыдущей практической верхнюю границу для температуры в 100 градусов и нижнюю для воды в 3 условных единицы и для атмосферы в 0 условных единиц. Получаем:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Вариант | Температура  (-) | Плотность  (+) | Вода  (+) | Вулканы  (-) | Атмосфера  (+) |
| 1 | ~~Меркурий~~ | 427 | 5.43 | 0 | 423 | 0 |
| 2 | ~~Венера~~ | 600 | 5.42 | 0 | 1023 | 90 |
| 3 | Земля | 20 | 5.5 | 5000 | 56 | 1 |
| 4 | Марс | -40 | 5.4 | 5 | 100 | 2 |
| 5 | Юпитер | -108 | 0.6 | 300 | 0 | 3000 |
| 6 | Сатурн | -109 | 0.7 | 300 | 0 | 3000 |
| 7 | Уран | -100 | 0.8 | 100 | 0 | 2000 |
| 8 | Нептун | -150 | 0.8 | 200 | 0 | 2500 |
| 9 | ~~Плутон~~ | -150 | 5.1 | 0 | 40 | 0.5 |

Меркурий на проходит по температуре, атмосфере и воде, Венера не проходит по температуре и воде, Плутон не проходит по воде.

Также мы можем использовать выделение приоритетности некоторых критериев над другими. Допустим, если мы скажем, что из оставшегося множества нас больше всего волнует критерий “Вода”, то мы получим:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Вариант | Температура  (-) | Плотность  (+) | Вода  (+) | Вулканы  (-) | Атмосфера  (+) |
| 1 | Земля | 20 | 5.5 | 5000 | 56 | 1 |
| 2 | ~~Марс~~ | -40 | 5.4 | 5 | 100 | 2 |
| 3 | ~~Юпитер~~ | -108 | 0.6 | 300 | 0 | 3000 |
| 4 | ~~Сатурн~~ | -109 | 0.7 | 300 | 0 | 3000 |
| 5 | ~~Уран~~ | -100 | 0.8 | 100 | 0 | 2000 |
| 6 | ~~Нептун~~ | -150 | 0.8 | 200 | 0 | 2500 |

А если же нас больше всего интересует критерий “Температура”, то выходит:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Вариант | Температура  (-) | Плотность  (+) | Вода  (+) | Вулканы  (-) | Атмосфера  (+) |
| 1 | ~~Земля~~ | 20 | 5.5 | 5000 | 56 | 1 |
| 2 | ~~Марс~~ | -40 | 5.4 | 5 | 100 | 2 |
| 3 | ~~Юпитер~~ | -108 | 0.6 | 300 | 0 | 3000 |
| 4 | ~~Сатурн~~ | -109 | 0.7 | 300 | 0 | 3000 |
| 5 | ~~Уран~~ | -100 | 0.8 | 100 | 0 | 2000 |
| 6 | Нептун | -150 | 0.8 | 200 | 0 | 2500 |

**Программный код для сужения множества Парето на языке Java**

**Solution.java**

import First.Alternative;  
import First.Pareto;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Solution {  
 public static void main(String[] args) {  
 List<Alternative> alternatives = new ArrayList<>();  
 List<String> names = new ArrayList<>();  
 List<Boolean> heights = new ArrayList<>();  
 List<String> importance = new ArrayList<>();  
 names.add("Температура");  
 names.add("Плотность");  
 names.add("Вода");  
 names.add("Вулканы");  
 names.add("Атмосфера");  
 importance.add("Температура");  
 importance.add("Атмосфера");  
 importance.add("Вода");  
 importance.add("Вулканы");  
 importance.add("Плотность");  
 List<Alternative.Criteria> narrowing = new ArrayList<>();  
 narrowing.add(new Alternative.Criteria("Температура", 100, true));  
 narrowing.add(new Alternative.Criteria("Вода", 3, false));  
 narrowing.add(new Alternative.Criteria("Атмосфера", 0, false));  
 heights.add(false);  
 heights.add(true);  
 heights.add(true);  
 heights.add(false);  
 heights.add(true);  
 List<Double> counts = new ArrayList<>();  
 counts.add(427.);  
 counts.add(5.43);  
 counts.add(0.);  
 counts.add(423.);  
 counts.add(0.);  
 alternatives.add(new Alternative("Меркурий", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(600.);  
 counts.add(5.42);  
 counts.add(0.);  
 counts.add(1023.);  
 counts.add(90.);  
 alternatives.add(new Alternative("Венера", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(20.);  
 counts.add(5.5);  
 counts.add(5000.);  
 counts.add(56.);  
 counts.add(1.);  
 alternatives.add(new Alternative("Земля", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-40.);  
 counts.add(5.4);  
 counts.add(5.);  
 counts.add(100.);  
 counts.add(2.);  
 alternatives.add(new Alternative("Марс", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-108.);  
 counts.add(0.6);  
 counts.add(300.);  
 counts.add(0.);  
 counts.add(3000.);  
 alternatives.add(new Alternative("Юпитер", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-109.);  
 counts.add(0.7);  
 counts.add(300.);  
 counts.add(0.);  
 counts.add(3000.);  
 alternatives.add(new Alternative("Сатурн", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-100.);  
 counts.add(0.8);  
 counts.add(100.);  
 counts.add(0.);  
 counts.add(2000.);  
 alternatives.add(new Alternative("Уран", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-150.);  
 counts.add(0.8);  
 counts.add(200.);  
 counts.add(0.);  
 counts.add(2500.);  
 alternatives.add(new Alternative("Нептун", names, heights, counts));  
 counts = new ArrayList<>();  
 counts.add(-150.);  
 counts.add(5.1);  
 counts.add(0.);  
 counts.add(40.);  
 counts.add(0.5);  
 alternatives.add(new Alternative("Плутон", names, heights, counts));  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*narrowing*(alternatives, narrowing);  
 System.*out*.println();  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*paretoMethod*(alternatives);  
 System.*out*.println();  
 Pareto.*out*(alternatives);  
 System.*out*.println();  
 Pareto.*importance*(alternatives, importance);  
 System.*out*.println();  
 System.*out*.println(alternatives);  
 }  
}

**Alternative.java**

package First;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Objects;  
  
public class Alternative implements Comparable<Alternative> {  
 private final String name;  
 private final List<Criteria> criterias = new ArrayList<>();  
  
 public String getName() {  
 return name;  
 }  
  
 public Alternative(String name, List<String> nCriterias, List<Boolean> bCriterias, List<Double> iCriterias) {  
 this.name = name;  
 for (int i = 0; i < nCriterias.size(); i++)  
 criterias.add(new Criteria(nCriterias.get(i), iCriterias.get(i), bCriterias.get(i)));  
 }  
  
 public boolean checkToNarrowing(List<Criteria> criterias2) {  
 for (Criteria criteria : criterias2) {  
 if (criteria.height && criterias.get(criterias.indexOf(criteria)).count > criteria.count)  
 return true;  
 if (!criteria.height && criterias.get(criterias.indexOf(criteria)).count < criteria.count)  
 return true;  
 }  
 return false;  
 }  
  
 public double getCount(String name) {  
 for (Criteria criteria : criterias)  
 if (criteria.criteriaName.equals(name))  
 return criteria.count;  
 return 0;  
 }  
  
 public boolean getHeight(String name) {  
 for (Criteria criteria : criterias)  
 if (criteria.criteriaName.equals(name))  
 return criteria.height;  
 return false;  
 }  
  
 public int compareTo(Alternative o) {  
 int res = 0;  
 for (int i = 0; i < criterias.size(); i++) {  
 if (criterias.get(i).height && criterias.get(i).count > o.criterias.get(i).count || !criterias.get(i).height && criterias.get(i).count < o.criterias.get(i).count) {  
 if (res < 0)  
 return 0;  
 res++;  
 } else if (criterias.get(i).height && criterias.get(i).count < o.criterias.get(i).count || !criterias.get(i).height && criterias.get(i).count > o.criterias.get(i).count)  
 if (res > 0)  
 return 0;  
 res--;  
 }  
 return res;  
 }  
  
 public String toString() {  
 StringBuilder builder = new StringBuilder("Вариант ").append(name).append(":");  
 for (Criteria criteria : criterias)  
 builder.append(", ").append(criteria.criteriaName).append(" = ").append(criteria.count);  
 return builder.toString();  
 }  
  
  
 public static class Criteria {  
 private final String criteriaName;  
 private final double count;  
 private final boolean height;  
  
  
 public Criteria(String criteriaName, double count, boolean height) {  
 this.criteriaName = criteriaName;  
 this.count = count;  
 this.height = height;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Criteria criteria = (Criteria) o;  
 return Objects.*equals*(criteriaName, criteria.criteriaName);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(criteriaName);  
 }  
  
 }  
}

**Pareto.java**

package First;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Pareto {  
 public static class ParetoHash {  
 public List<Alternative.Criteria> criteriaToNarrowing;  
 public List<String> importance;  
 public List<Alternative> alternatives;  
  
 public ParetoHash(List<Alternative.Criteria> criteriaToNarrowing, List<String> importance, List<Alternative> alternatives) {  
 this.criteriaToNarrowing = criteriaToNarrowing;  
 this.importance = importance;  
 this.alternatives = alternatives;  
 }  
 }  
  
 public static ParetoHash in() throws IOException {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
 List<Alternative> alternatives = new ArrayList<>();  
 System.*out*.println("Введите количество критериев");  
 int n = Integer.*parseInt*(reader.readLine());  
 List<String> names = new ArrayList<>();  
 List<Boolean> heights = new ArrayList<>();  
 List<Alternative.Criteria> criteriaToNarrowing = new ArrayList<>();  
 List<String> importance = new ArrayList<>();  
 for (int i = 0; i < n; i++) {  
 System.*out*.println("Введите имя для критерия");  
 names.add(reader.readLine());  
 System.*out*.println("Критерий положительный или отрицательный(true/false)?");  
 heights.add(Boolean.*parseBoolean*(reader.readLine()));  
 System.*out*.println("Хотите задать верхнюю границу для критерия(1-да, 2-нет)?");  
 if (Integer.*parseInt*(reader.readLine()) == 1) {  
 System.*out*.println("Задайте значение верхней границы");  
 criteriaToNarrowing.add(new Alternative.Criteria(names.get(names.size() - 1), Integer.*parseInt*(reader.readLine()), true));  
 }  
 System.*out*.println("Хотите задать нижнюю границу для критерия(1-да, 2-нет)?");  
 if (Integer.*parseInt*(reader.readLine()) == 1) {  
 System.*out*.println("Задайте значение нижней границы");  
 criteriaToNarrowing.add(new Alternative.Criteria(names.get(names.size() - 1), Integer.*parseInt*(reader.readLine()), false));  
 }  
 }  
 System.*out*.println("Напишите название критерия для добавления его в приоритетность или exit для продолжения");  
 String impStr = reader.readLine();  
 while (!impStr.equals("exit")) {  
 importance.add(impStr);  
 impStr = reader.readLine();  
 }  
 System.*out*.println("Введите количество альтернатив");  
 n = Integer.*parseInt*(reader.readLine());  
 for (int i = 0; i < n; i++) {  
 System.*out*.println("Введите название варианта");  
 String name = reader.readLine();  
 List<Double> counts = new ArrayList<>();  
 for (String na : names) {  
 System.*out*.println("Введите значение для критерия " + na);  
 counts.add(Double.*parseDouble*(reader.readLine()));  
 }  
 alternatives.add(new Alternative(name, names, heights, counts));  
 }  
 return new ParetoHash(criteriaToNarrowing, importance, alternatives);  
 }  
  
 public static void paretoMethod(List<Alternative> alternatives) {  
 label:  
 for (int i = 0; i < alternatives.size(); ) {  
 for (int j = i + 1; j < alternatives.size(); ) {  
 int res = alternatives.get(i).compareTo(alternatives.get(j));  
 if (res < 0) {  
 System.*out*.println(alternatives.get(j).getName() + " доминирует над " + alternatives.get(i).getName());  
 alternatives.remove(i);  
 continue label;  
 } else if (res > 0) {  
 System.*out*.println(alternatives.get(i).getName() + " доминирует над " + alternatives.get(j).getName());  
 alternatives.remove(j);  
 continue;  
 }  
 j++;  
 }  
 i++;  
 }  
 }  
  
 public static void narrowing(List<Alternative> alternatives, List<Alternative.Criteria> criteries) {  
 for (int i = 0; i < alternatives.size(); ) {  
 if (alternatives.get(i).checkToNarrowing(criteries)) {  
 System.*out*.println("Вариант " + alternatives.get(i).getName() + " исключается в результате сужения");  
 alternatives.remove(alternatives.get(i));  
 } else  
 i++;  
 }  
 }  
  
 public static void importance(List<Alternative> alternatives, List<String> criteries) {  
 for (String criteria : criteries) {  
 if (alternatives.size() == 1)  
 return;  
 boolean height = alternatives.get(0).getHeight(criteria);  
 List<Alternative> best = new ArrayList<>();  
 for (int i = 0; i < alternatives.size(); ) {  
 Alternative alternative = alternatives.get(i);  
 if (best.isEmpty() || (height && alternative.getCount(criteria) > best.get(0).getCount(criteria)) || (!height && alternative.getCount(criteria) < best.get(0).getCount(criteria))) {  
 alternatives.removeAll(best);  
 if (!best.isEmpty()) {  
 StringBuilder builder = new StringBuilder("Варианты ");  
 for (Alternative alternative1 : best) {  
 builder.append(alternative1.getName()).append(", ");  
 alternatives.remove(alternative1);  
 }  
 builder.append(" проигрывают варианту ").append(alternative.getName()).append(" по критерию ").append(criteria);  
 System.*out*.println(builder);  
 }  
 best = new ArrayList<>();  
 best.add(alternative);  
 i = 1;  
 } else if ((height && alternative.getCount(criteria) < best.get(0).getCount(criteria)) || (!height && alternative.getCount(criteria) > best.get(0).getCount(criteria))) {  
 System.*out*.println("Вариант " + alternative.getName() + " проигрывает варианту " + best.get(0).getName());  
 alternatives.remove(alternative);  
 }  
 else {  
 best.add(alternative);  
 i++;  
 }  
 }  
 }  
 }  
  
 public static void out(List<Alternative> alternatives) {  
 for (Alternative alternative : alternatives)  
 System.*out*.println(alternative);  
 }  
}

**Результат выполнения программы**

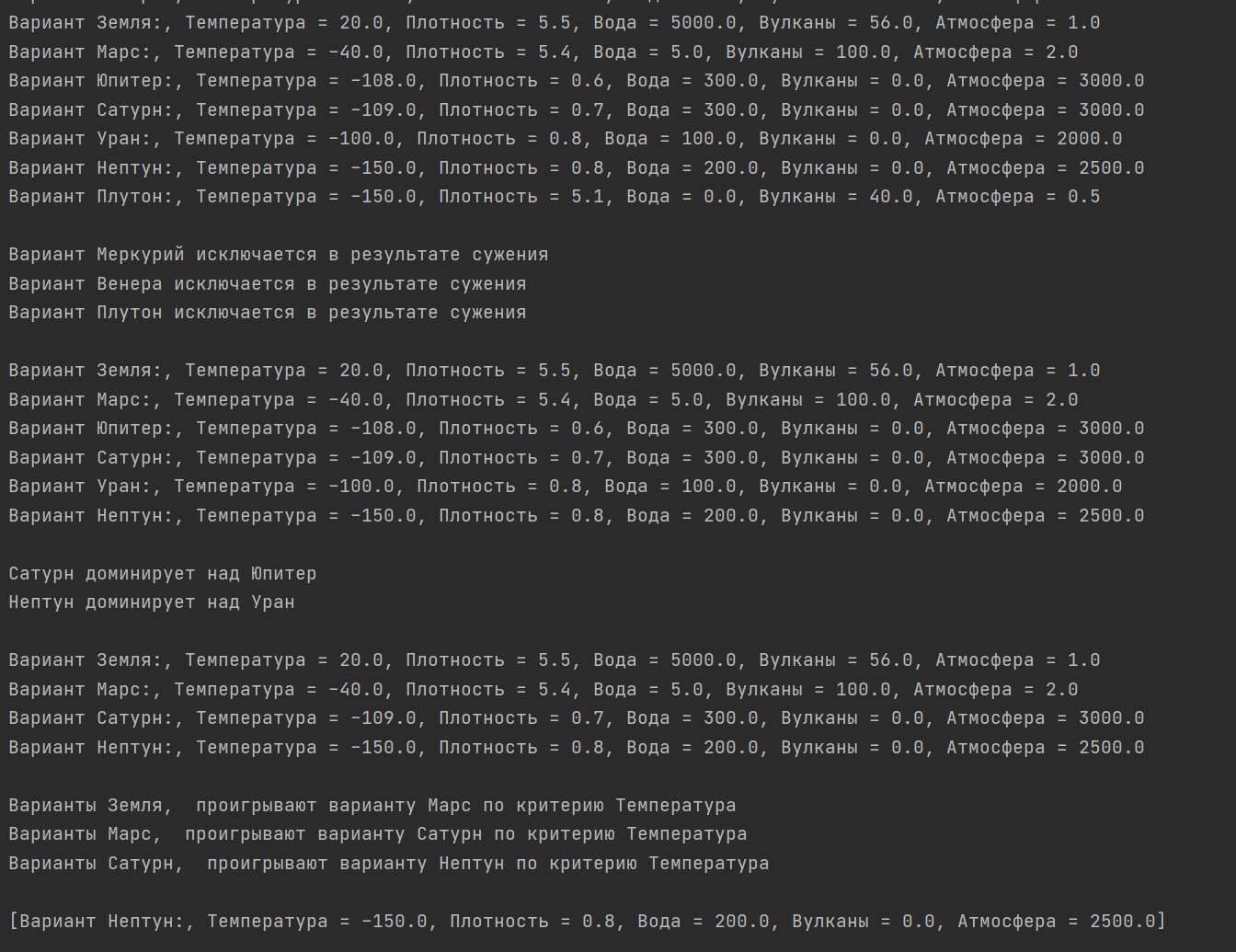
****

Рисунок 1. Вывод программы “сужение множества Парето”