**Домашняя работа**

**студента РКИ 01 2025**

**Желаева Дениса**

**Задача ОЗ-1**

**Дано:**

Объект, дискретно перемещающийся вдоль оси х. Время дискретно, измеряется в тактах.  
Положение объекта на оси так же дискретно.   
За один такт объект может:  
либо остаться на месте с вероятностью p0;  
либо сдвинуться влево (т.е. на Δх = -1) с вероятностью p1;  
либо сдвинуться вправо (т.е. на Δх = 1) с вероятностью p2.

**Требуется:**

Для момента времени t0  
определить параметры распределения  
случайной величины X(t0): положение объекта в момент t0: математическое ожидание, дисперсия

**Решение:**

1. M[ΔX] = (d0×po) + (d1×p1) + (d2×p2) = 0 – p1 + p2 = p2 – p1

(мат ожидание на один такт)

1. M[X] = M[ΔX] × t0 = (p2 – p1) × to

(мат ожидание на момент t0)

1. D[ΔX] = M[ΔX2] – (M[ΔX])2 = [(d02×po) + (d12×p1) + (d22×p2)] - (p2 – p1)2 = р1 + р2 - (p2 – p1)2  
   (дисперсия на один такт)
2. D [X] = D[ΔX] × t0 = [р1 + р2 - (p2 – p1)2] × to

(дисперсия на момент t0)

**Задача ОЗ-2**

**Дано:**

Пусть параметры, описывающие объекты в некотором множестве - двумерные векторы.

Заданы два кластера А и В. Каждый кластер порождается двумерной нормально распределённой случайной величиной с параметрами. Задана точка Q.

|  |  |  |  |
| --- | --- | --- | --- |
| Кластер | Математическое ожидание | Ковариационная матрица | Объем выборки |
| А | MA | ΣA | n |
| В | MB | ΣВ | m |

*Таблица из файла*

Есть некоторое множество Ω объектов.  
У каждого объекта есть параметры, и этих параметров два. Кластер представляет собой множество объектов или, что тоже самое, - множество точек на плоскости. С другой стороны, эти точки мы можем представить, как реализации двумерной случайной величины.

**Требуется:**

Найти расстояние от точки Q до кластера А (и до кластера В).  
Сделать вывод о том, к какому кластеру точка Q находится ближе.

**Решение:**

1. vA = q - MA = [q1 - mA1, q2 - mA2]  
   (Вектор, направленный из центра кластера A в точку q)
2. detA = (ծA11 × ծA22) – (ծA12 × ծA12)  
   (Определитель матрицы ΣA)  
   Поскольку σA12 = σA21, произведение σA12 × σA21 равно σA12²)
3. ΣA⁻¹ = detA-1 × [[ծA22, -ծA12], [-ծA12, ծA11]]  
   (Обратная матрица к ΣA. Делим на detA каждый элемент матрицы)
4. dA² = (vA)ᵀ × ΣA⁻¹ × vA  
   (Квадрат расстояния Махаланобиса.)
5. d(q, A) = √(dA²)  
   (Расстояние от точки q до кластера A)
6. Аналогично находим расстояние до кластера B
7. Сравниваем полученные значения, если d(q, A) < d(q, B), то ближе к А, если нет то в В, если равно – то на равном статистическом расстоянии.

**Задача ОЗ-3**

**Дано:**

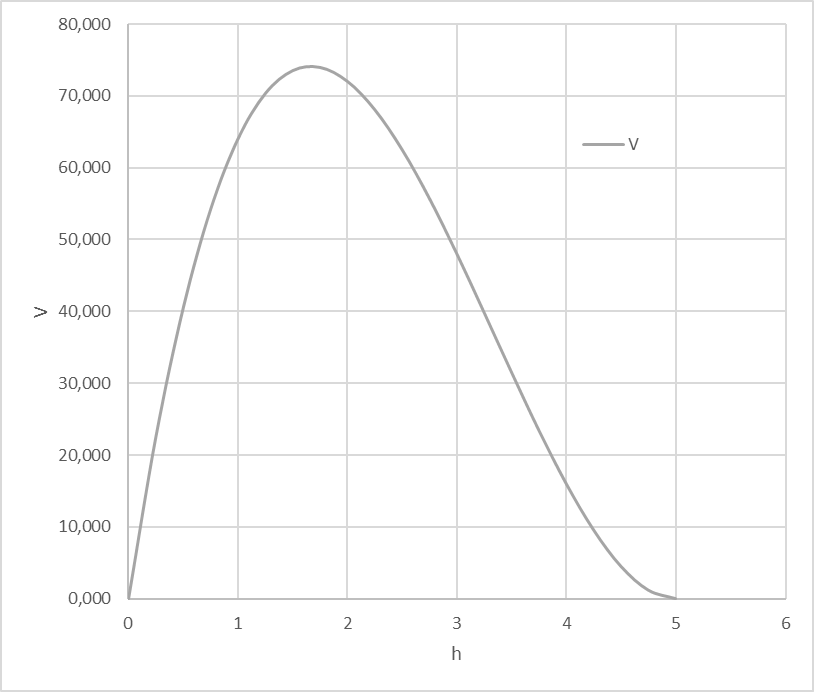
Квадратный лист бумаги 10 на 10 см. Имеются ножницы и клей.

Два листа бумаги можно склеить если сложить их в нахлест на 1 см.

Дополнительные материалы сравнения соотношений сторон в четырёхугольной призме

|  |  |  |
| --- | --- | --- |
| h | b | V |
| 0 | 10 | 0,000 |
| 0,25 | 9,5 | 22,563 |
| 0,5 | 9 | 40,500 |
| 0,75 | 8,5 | 54,188 |
| 1 | 8 | 64,000 |
| 1,25 | 7,5 | 70,313 |
| 1,5 | 7 | 73,500 |
| 1,75 | 6,5 | 73,938 |
| 2 | 6 | 72,000 |
| 2,25 | 5,5 | 68,063 |
| 2,5 | 5 | 62,500 |
| 2,75 | 4,5 | 55,688 |
| 3 | 4 | 48,000 |
| 3,25 | 3,5 | 39,813 |
| 3,5 | 3 | 31,500 |
| 3,75 | 2,5 | 23,438 |
| 4 | 2 | 16,000 |
| 4,25 | 1,5 | 9,563 |
| 4,5 | 1 | 4,500 |
| 4,75 | 0,5 | 1,188 |
| 5 | 0 | 0,000 |

*Таблица сравнения соотношений стороны дна(b) и высоты стенки(h) с итоговым объёмом.*



*График к таблице сравнения соотношений.*

**Требуется:**

Сконструировать из данного листа бумаги коробку без крышки максимального объема.

Объем коробки считаем, как площадь основания ⨯ высоту одинаковых стенок (коробка - призма).

**Решение:**

1. Рассматриваем вариант 4-угольной призмы, опираясь на таблицу из дано.
2. Из соотношений в таблицу мы можем составить простую систему уравнений:

b2 × h = V, где V объем, который нужно максимизировать.

b + 2h = 10

1. Из системы уравнений выведем границы решения:  
   0 ≤ b ≥ 10  
   0 ≤ h ≥ 5
2. Из системы уравнений выразим переменную b через переменную h:  
   b = 10 – 2h
3. Подставим это значение в первое уравнение:   
   (10 – 2h)2 × h = V
4. Развернём получившееся уравнение и возьмём от него производную для поиска точек экстремума:

V(h)´ = {(10 – 2h)2 × h}´ = {h(100 – 40h + 4b2)} ´ = {4h3 – 40h2 +100h} ´ =   
 = 12h2 – 80h + 100.

1. Получившуюся производную приравняем к нулю и решим получившееся квадратное уравнение через дискриминант:  
   12h2 – 80h + 100 = 0 | ÷ 4  
   3h2 – 20h + 25 = 0  
   D = (-20)2 – 4 × 3 × 25 = 400 – 300 = 100  
   h1 = (20 - √100) ÷ (2 × 3) = 5/3  
   h2 = (20 + √100) ÷ (2 × 3) = 5
2. Как мы видим, получившийся h2 совпадает с приграничным значением. Проверим пограничные значения:  
   102 × 0 = 0 //не подходит, так как нам нужен максимальный объём  
   02 × 5 = 0 //не подходит, так как нам нужен максимальный объём
3. Проверим получившееся значение:  
   При h = 5/3,  
   b = 10 – (2 × 5/3) = 10 - 10/3 = 30/3 – 10/3 = 20/3  
    =>  
    подставляем получившееся значения в первое уравнение из системы  
    =>  
   b2 × h = (20/3)2 × 5/3 = 400/9 × 5/3 = 2000/27 = 74,074

74,074 – итоговый максимальный объём, при высоте стенок 5/3 и ширине дна 20/3.

**Задача ОЗ-4**

**Дано:**

Робин Гуд стреляет на дальность. Начальная скорость стрелы постоянна и равна v0. Сопротивление воздуха пропорционально скорости стрелы: R = - a v( t ). Действие происходит на Земле: g = 9,8 м/с². Расстояние от земли до плеч Р. Гуда h = 1,5 м.

**Требуется:**

Определить угол наклона лука.

**Решение:**

* + - 1. О сопротивлении воздуха не говорится явно, поэтому мы им пренебрегаем.
      2. Находим уравнение траектории:  
         y(x) = y0 + x × tan(θ) − (g × x²) ÷ (2 × v0² × cos²(θ)),  
         где y(x) — высота стрелы над землёй в точке с координатой x,   
         y0 — начальная высота = h = 1.5 м,   
         θ — угол выстрела от горизонта,   
         g = 9.8 м/с² — ускорение свободного падения,  
         v0 — начальная скорость.
      3. Подставим значения:  
         Δy = L × tan(θ) − (g × L²) ÷ (2 × v0² × cos²(θ)),  
         где Δy = yT − y0 — разность высот;   
         предположим, что Δy = 0
      4. Используем 1 ÷ cos²(θ) = 1 + tan²(θ).   
         Тогда  
         0 = L × t − (g × L²) ÷ (2 × v0²) × (1 + t²),  
         где t = tan(θ).
      5. Приведём к виду A × t² + B × t + C = 0, где  
         A = (g × L²) ÷ (2 × v0²),  
         B = −L,  
         C = (g × L²) ÷ (2 × v0²).
      6. Решение квадратичного уравнения даёт (после упрощения) стандартную форму:  
         t = [v0² ± sqrt( v0^4 − g × ( g × L² + 2 × Δy × v0² ) ) ] ÷ (g × L),
      7. при Δy = 0 формула упрощается до   
         t = [v0² ± sqrt(v0^4 − g² × L² ) ] ÷ (g × L).)
      8. упростим и приведём к виду  
         t = [v0² ± sqrt(v0^4 − g² × L² ) ] ÷ (g × L).)
      9. Подставим значения:  
         t = [ 3600 ± sqrt(12 960 000 − 240 100) ] ÷ 490 = [ 3600 ± sqrt(12 719 900) ] ÷ 490   
         t₁ ≈ 14.62550383815101  
         t2 ≈ 0.06837371286939686  
         =>  
         θ₁ = arctan(14.62550383815101) ≈ 1.50233 рад ≈ 86.09°.  
         θ₂ = arctan(0.06837371286939686) ≈ 0.06825 рад ≈ 3.91°.

Вывод: оба корня подходят по значению, но скорее всего Робин Губ стрелял по менее крутой траектории, а значит под углом ≈ 3.91 ≈ 4.

**Задача ТЗ-1.1**

**Дано:**

Команда экспертов состоит из N человек. Каждый эксперт выбирает целое число из множества {1, 2, 3, …, 99, 100}.

Эксперты не имеют возможности предварительно договориться, каждый из них оценивает других независимо.

|  |  |  |
| --- | --- | --- |
| № эксперта | оценка | Ошибка |
| 1 | 90 | 9 |
| 2 | 100 | 1 |
| 3 | 100 | 1 |
| 4 | 100 | 1 |
| 5 | 100 | 1 |
| 6 | 100 | 1 |
| 7 | 100 | 1 |
| 8 | 100 | 1 |
| 9 | 100 | 1 |
| 10 | 100 | 1 |
| Среднее | 99 |  |

*Таблица из дано*

**Требуется:**

Задача эксперта выбрать то число, которое будет как можно ближе к среднему, вычисленному по всем выбранным числам.

**Решение:**

1. Рассмотрим задачу внимательно: среднее число считается по формуле среднего арифметического, то есть чем меньше шаг(разброс) между значениями, тем выгоднее для каждого эксперта.
2. Дальше идёт развилка в предположениях:  
   Рассмотрим случай, где «эксперты» недостаточно экспертны и выбирают случайное число от 0 до 100, тогда рассчитаем среднее математическое ожидание:  
   М(Х) = (100 × 0,01) + (99 × 0,01) + … + (1 × 0.01)
3. Это классическая формула арифметической прогрессии, умноженной на 0,01, где - 0,01 - шанс выбора числа.  
   Как я писал ранее, в этом случае мы рассматриваем случайный выбор, а значит шансы у каждого числа равны и в сумме должны составлять 1. Из этого можно сделать простое уравнение 100p = 1 => p = 0,01
4. Найдём же среднее математическое по формуле:  
   S = n × (a₁ + aₙ) / 2, где   
   n = 100 (количество членов прогрессии)  
   a₁ = 1 (первый член прогрессии)

aₙ = 100 (последний член прогрессии)  
100 × (1 + 100) / 2 = 100 × 101 / 2 = 5050

1. И умножим на 0,01 => М(Х) = 5050 × 0,01 = 50,5.
2. В случае, если эксперты выбирают случайное число, среднее число будет равняться 50,5 в большинстве случаев.

2.1) Если же мы считаем, что эксперты думают наравне с нами, то они точно так же придут к такому же итогу как и мы, а значит так же выберут 50,5 как и мы.

**Дополнительный комментарий**:

Возможно стоит склониться к числу 50.25, так как если эксперты будут думать так же как и мы, но считать быстро, то они допустят ошибку, которую допустил я, а именно выберут число 50.

В таком случае шаг составит 0,5. Я считаю что нужно подвинуть значение в сторону 50.25 чтобы снизить шаг до 0,25 вне зависимости от получившегося сценария – если все будут выбирать случайно, то среднее число будет 50,5 (от нашего шаг 0,25), если осознанно – выберут либо 50,5 (если посчитали правильно, шаг так же 0,25), либо 50 (если посчитали быстро навскидку, и тогда шаг так же 0,25).

Предложенное число снижает шаг до фиксированной величины и уменьшает количество рисков, но предполагает, что в большинстве случаев мы не угадаем число точно и минимальный шаг будет присутствовать.

**Задача ТЗ-1.2**

**Дано:**

Команда экспертов состоит из N человек. Каждый эксперт выбирает целое число из множества {1, 2, 3, …, 99, 100}.

Эксперты не имеют возможности предварительно договориться, каждый из них оценивает других независимо.

|  |  |  |
| --- | --- | --- |
| № эксперта | оценка | Ошибка |
| 1 | 90 | 28,81 |
| 2 | 100 | 38.81 |
| 3 | 100 | 38.81 |
| 4 | 100 | 38.81 |
| 5 | 100 | 38.81 |
| 6 | 100 | 38.81 |
| 7 | 100 | 38.81 |
| 8 | 100 | 38.81 |
| 9 | 100 | 38.81 |
| 10 | 100 | 38.81 |
| Среднее | 99 |  |
| Зол. Сечение | 61.99 |  |

*Таблица из дано*

**Требуется:**

Задача эксперта выбрать то число, которое будет как можно ближе к золотому сечению от среднего, вычисленного по всем выбранным числам.

**Решение:**

Данная задача схожа с предыдущей, но имеет ключевое отличие: сейчас всем экспертам НЕ выгодно выбирать одинаковое значение. Условно поделим возможные сюжеты на отдельные итерации:

Итерация 0:

В данном сценарии эксперты не задумываясь выбирают случайное число. Как мы выяснили в предыдущей задаче, при таком сценарии среднее число будет 50,5 => нам нужно выбрать золотое сечение от него, то есть 50,5 × 0.618 ≈ 31,2

Итерация 1:

В данном случае мы предполагаем, что большинство экспертов реализовало нулевую итерацию и выбрало число 31,2. Из этого следует, что нужно вычислить сечение уже от этого значения, то есть 31,2 × 0.618 ≈ 19,2

Итерация 2:

В данном случае мы предполагаем, что большинство экспертов реализовало первую итерацию и выбрало число 19,2. Из этого следует, что нужно вычислить сечение уже от этого значения, то есть 19,2 × 0.618 ≈ 11,8

Итерация 3:

В данном случае мы предполагаем, что большинство экспертов реализовало первую итерацию и выбрало число 11,8. Из этого следует, что нужно вычислить сечение уже от этого значения, то есть 11,8 × 0.618 ≈ 7,2

Итерация n:

В данном случае мы предполагаем, что большинство экспертов реализовало n - 1 итерацию и выбрало число F(n-1). Из этого следует, что нужно вычислить сечение уже от этого значения, то есть F(n-1) × 0.618 ≈ F(n)

Итог:

Как видно с каждой итерацией предполагаемый ответ снижается. Я предполагаю, что оптимальный вариант ответа - итерация 1, так как она равноудалена от 0 итерации (шаг = 12), так и от последующих (от 2 итерации шаг 9, от 3 – 11). Следовательно при стратегии итерации 1 даже если мы и не угадаем точно, разброс будет стабильным в пределах 12.

**Задача ТЗ-2**

**Дано:**

Пусть параметры, описывающие объекты в некотором множестве - двумерные векторы.

Заданы два кластера А и В. Каждый кластер порождается двумерной нормально распределённой случайной величиной с параметрами. Задана точка Q.

|  |  |  |  |
| --- | --- | --- | --- |
| Кластер | Математическое ожидание | Ковариационная матрица | Объем выборки |
| А | MA | ΣA | n |
| В | MB | ΣВ | m |

*Таблица из дано*

Есть некоторое множество Ω объектов.  
У каждого объекта есть параметры, и этих параметров два. Кластер представляет собой множество объектов или, что тоже самое, - множество точек на плоскости. С другой стороны, эти точки мы можем представить, как реализации двумерной случайной величины.

**Требуется:**

Требуется составить алгоритм, позволяющий найти расстояние между кластерами А и В.

**Решение:**

Алгоритм вычисления расстояния Хаусдорфа между кластерами A и B. Расстояние Хаусдорфа является каноническим расстоянием между множествами в метрических пространствах. В отличие от расстояний, основанных на параметрах распределений, оно:

* Работает с конкретными множествами точек, а не с параметрами распределений
* Учитывает геометрическую форму кластеров, а не только их статистические характеристики
* Является метрикой (удовлетворяет аксиомам тождества, симметрии и неравенству треугольника)

Детальный алгоритм:

* + - 1. Генерируем множества точек:
  1. Генерируем n точек из двумерного нормального распределения с параметрами MA и ΣA

Объем выборки n соответствует таблице данных  
SA = {a₁, a₂, ..., aₙ}, где aᵢ ∼ N(MA, ΣA)

* 1. Генерируем m точек из двумерного нормального

распределения с параметрами MB и ΣB

SB = {b₁, b₂, ..., bₘ}, где bⱼ ∼ N(MB, ΣB)

* + - 1. Вычисляем направленные расстояния Хаусдорфа, которые состоят из двух компонентов, показывающих "наибольшее минимальное расстояние" от одного множества до другого.  
         2.1 Для каждой точки a ∈ A находим расстояние до ближайшей точки в B и среди всех этих "ближайших расстояний" находим максимальное:

h(A→B) = max[min[d(a,b) для всех b ∈ SB] для всех a ∈ SA],

где d(a,b) = √[(a₁-b₁)² + (a₂-b₂)²] - евклидово расстояние  
2.4 Расстояние от B к A находим аналогично по формуле:  
h(B→A) = max[min[d(b,a) для всех a ∈ SA] для всех b ∈ SB]

* + - 1. Находим расстояние Хаусдорфа - это наибольшее из двух направленных расстояний, что гарантирует симметричность:  
         d\_H(A,B) = max[h(A→B), h(B→A)]
      2. Результат:
* d\_H(A,B) = 0 => кластеры идентичны (совпадают как множества)
* Малое d\_H => кластеры близки друг к другу
* Большое d\_H => кластеры далеки друг от друга
* Если d\_H < (характерные размеры кластеров) => кластеры пересекаются или близко расположены

**Задача ТЗ-3**

**Дано:**

В процессе обучения студент получает оценки по предметам. Предметов в зачетной книжке N. По результатам обучения требуется составить список студентов, ранжированный по успеваемости.

Оценки ставятся в 4-балльной системе оценивания:

d>c>b>a

|  |  |  |
| --- | --- | --- |
| Описание | Традиционная | Из задачи |
| Отлично | 5 | D |
| Хорошо | 4 | C |
| Удовлетворительно | 3 | B |
| Неудовлетворительно | 2 | A |

Примеры расчётов оценок:

|  |  |  |
| --- | --- | --- |
| D | 5 | 12 |
| D | 5 | 12 |
| B | 3 | 0 |
| C | 4 | 10 |
|  | 4,25 | 8,5 |

*Таблица 1, студент 1*

|  |  |  |
| --- | --- | --- |
| D | 5 | 12 |
| C | 4 | 10 |
| C | 4 | 10 |
| C | 4 | 10 |
|  | 4,25 | 10.5 |

*Таблица 2, студент 2*

|  |  |  |
| --- | --- | --- |
| D | 5 | 12 |
| D | 5 | 12 |
| D | 5 | 12 |
| A | 3 | 0 |
|  | 4,5 | 9 |

*Таблица 3, студент 3*

**Требуется:**

Требуется найти алгоритм, не зависящий от конкретных значений a, b, c, d.

**Решение :**

1. Если мы принципиально хотим сделать независимым от конкретных значений, можно представить оценки студента как двумерный массив[4][4], где в первом векторе у нас будут все варианты оценок (отсортированные уникальные числа), а во втором массиве количество повторов оценок. (можно сделать и обычным массивом, но так нагляднее)

Абстрактный пример:

StudentX[4][4] = [d][c][b][a]

[nd][ nc][ nb][na], где nх это количество оценок типа х

Пример из таблиц 1,2,3:

Student1[4][4] = [d][c][b][a]

[2][1][1][0]

Student2[4][4] = [d][c][b][a]

[1][3][0][0]

Student3[4][4] = [d][c][b][a]

[3][0][0][1]

1. При создании и сортировки списка мы обращаемся не к общему коэффициенту оценок студента (который представлен средним арифметическим), а к лексикографическому сравнению – сравниваем среднее количество оценок среди всех и на основе этого сравнения мы продвигаем студента вперёд или назад.

**Частные случаи:**

Частный случай 1:

StudentA[4][4] = [d][c][b][a]

[1][0][0][3]

StudentB[4][4] = [d][c][b][a]

[0][4][0][0]

При стандартном лексикографическом поиске студент А получит более высокий приоритет, игнорируя факт, что студент А не проходит требованиям успеваемости (при которых хотя бы одна оценка «а» означает, что студент не справился с заданием и не может быть допущен к зачёту\экзамену.

Частный случай 1.1:

StudentF[4][4] = [d][c][b][a]

[1][0][3][0]

StudentE[4][4] = [d][c][b][a]

[0][4][0][0]

При стандартном лексикографическом поиске студент F получит более высокий приоритет, игнорируя факт, что в среднем оценки лучше у студента E. Это является подвидом частного случая 1, когда выстроенная система сравнения нарушает общий смысл приоритетов.

Частный случай 1.2:

StudentG[4][4] = [d][c][b][a]

[3][0][5][0]

StudentH[4][4] = [d][c][b][a]

[2][4][0][0]

При стандартном лексикографическом поиске студент G получит более высокий приоритет, игнорируя факт, что в среднем оценки лучше у студента H. Это так же является подвидом проблемы 1, при нестандартном условии – у студентов разное общее количество оценок (у студента G 8 оценок, у студента H - 6).

**Решение частных случаев:**

Есть несколько вариантов модификации для решения данных случаев: обратный поиск(1.1), комбинированный поиск(1.2), поиск на основе соотношений(1.3) и многоуровневый поиск с приоритезацией(1.4).

Рассмотрим решения внимательнее:

1.1 Обратный поиск

При таком варианте мы просто сравниваем в обратном порядке (начиная с а), отдавая приоритет студентам без плохих оценок. Соответственно таким образом в топе списков будут студенты гарантированно без плохих оценок и получившие зачёт, при этом мы не привязаны жёсткого к числовым значениям, что удовлетворяет требованиям задачи и требованиям по успеваемости, решая частный случай 1.

1.2 Комбинированный поиск

При таком варианте мы будет сравнивать по сумме первых двух параметров (d, c), при их большем числе мы будем двигать студента вперёд. При равенстве будем рассматривать отдельно по каждому параметру сохраняя заданный приоритет. Данный вариант решает проблему частного случая 1.1.

1.3 Поиск на основе соотношений

При таком варианте мы будем использовать стандартный лексикографический поиск, но в качестве аргументов передавать не просто количество оценок, а соотношение количества оценок к общему количеству оценок, что позволяет точнее оценивать успеваемость студентов в случае, если у студентов разная общее количество оценок.

1.4 Многоуровневый поиск с приоритезацией

При таком варианте мы объединяем предыдущие решения в одно: сначала мы отсеиваем по наличию оценок типа а.  
Это решает проблему частного случая 1.  
Потом мы начинаем сравнивать по соотношению суммы хороших оценок (d + c) к общему количеству оценок.  
Это решает проблему частного случая 1.1 и 1.2.  
В случае, если получившиеся значения равны мы сравниваем по количеству оценок типа d.

Вывод: классический лексикографический поиск с поэтапным направленным сравнением типа LR (Left-to-Right) или RL(Right-to-Left) не подходит для реализации этой задачи.

Учитывая описанные модификации итоговый алгоритм ближе к многоуровневой компарации с дополнительной приоритезацией. Перейдём к конкретной реализации данного алгоритма.

**Реализация итогового алгоритма:**

Алгоритм сравнения элементов

* + - 1. Для каждого студента создаем вектор из четырех элементов, содержащий количество оценок каждого типа в порядке от высшей к низшей: [nd, nc, nb, na].
      2. Основной алгоритм сравнения двух студентов:

2.1 Сравниваем количество неудовлетворительных оценок («a»). Студент с меньшим количеством оценок «a» считается более успевающим.

2.2 Если количество оценок a одинаково, сравниваем соотношение суммы оценок d и с к общей сумме оценок. Студент с большим соотношением лучших оценок к остальным считается более успевающим.

2.3 Если соотношение суммы оценок d и с к общей сумме оценок одинаково, сравниваем количество оценок d. Студент с большим числом отличных оценок считается более успевающим.

2.5 Если общее количество оценок d равно, то студенты имеют одинаковую успеваемость. Если нам нужна дополнительная логика при данном соответствии (например ранжирование по id или по алфавитному порядку), её можно вынести в проверку в следующий пункт или оставить как есть.

3. Для сортировки можно использовать любой алгоритм, основанный на сравнении элементов. Единственное условие это стабильность алгоритма сортировки, так как в задании не прописаны действия при полном совпадении оценок у двух сравниваемых студентов. В таком случае я считаю, что студенты являются равными и не нуждаются в дополнительном перемещении, следовательно необходим стабильный алгоритм, который:

* Сохранит изначально заданный порядок при равенстве оценок
* Сэкономит вычислительные мощности
* Путем незначительного увеличения трудоёмкости позволит стабильно оценивать трудоёмкость и сделать её зависимой только от объема данных.

По описанным причинам я выбрал стабильный алгоритм сортировки слиянием, описанный ниже.

Алгоритм сортировки слиянием:

1. Если массив содержит один элемент или меньше, он уже отсортирован
2. Разделить массив на две примерно равные части
3. Рекурсивно отсортировать каждую из половин
4. Объединить (слить) две отсортированные половины в один отсортированный массив

Процесс слияния:

1. Создать временный массив для хранения результата
2. Сравнивать элементы из двух половин по очереди
3. Выбирать меньший элемент и помещать его во временный массив
4. Продолжать пока не будут обработаны все элементы обеих половин
5. Скопировать отсортированный временный массив обратно в исходный

Оценка трудоемкости:

* Худший случай: O (n log n)
* Лучший случай: O (n log n)
* Средний случай: O (n log n)

То есть трудоёмкость фиксирована и не зависит от расположения элементов и их значения, а только от их количества.

Оценка ресурсоемкости:

* Память: O(n) для временного массива
* Рекурсивные вызовы: O (log n) уровней стека

Стоит отметить, что стабильные алгоритмы сортировки имеют постоянную и независимую от сортируемых данных трудоёмкость и ресурсоёмкость, пусть и завышенную, если сравнивать с лучшими случаями для нестабильных алгоритмов сортировок. Далее рассмотрим крайне популярный нестабильный алгоритм Быстрой Сортировки для анализа преимуществ и недостатков.

Алгоритм сортировки Хоара (Быстрая сортировка):

1. Выбрать из массива опорный элемент.
2. Разделить массив на три части: элементы меньше опорного, равные опорному, и больше опорного.
3. Рекурсивно применить алгоритм к подмассиву элементов меньше опорного и подмассиву элементов больше опорного.
4. Рекурсия: массив из одного элемента или пустой массив считается отсортированным.

Процесс разделения (на примере схемы Ломуто или Хоара):

1. После выбора опорного элемента (Хоар: последнего или среднего; Ломуто: последнего; Хоар эффективнее, Ломуто проще для реализации), массив переупорядочивается так, чтобы все элементы, меньше опорного, оказались слева от него, а все элементы, больше опорного - справа.
2. Опорный элемент занимает свою окончательную позицию в отсортированном массиве.

Оценка трудоемкости:

* Худший случай: O(n²). Возникает, когда выбранный опорный элемент постоянно оказывается минимальным или максимальным в текущем подмассиве (например, при попытке отсортировать уже отсортированный массив и выборе в качестве опоры первого или последнего элемента). Это приводит к максимально несбалансированному разбиению, когда одна часть содержит n-1 элемент, а другая — 0.
* Лучший случай: O(n log n). Возникает, когда опорный элемент каждый раз делит массив на две примерно равные части. Глубина рекурсии составляет O(log n), а работа на каждом уровне — O(n).
* Средний случай: O(n log n). На практике при случайных данных или рандомизированном выборе опора алгоритм демонстрирует именно такую сложность.

Оценка ресурсоемкости:

* Память: O(1) дополнительной памяти (сортировка на месте), но используется стек вызовов.
* Рекурсивные вызовы: Глубина стека напрямую зависит от сбалансированности разбиений. В худшем случае O(n), в лучшем и среднем случае O (log n).

В задаче не было описано, какой язык использовать для написания программы, однако было требование к детальности алгоритма, поэтому я выбрал самый низкоуровневый язык, который я знаю, а именно ассемблер от Майкрософт, как язык, позволяющий работать непосредственно на уровне побитовых операций.

**Код для алгоритмов лексикографического комбинированного сравнения обратного порядка и стабильной сортировки вставками на языке MASM x64 (Macro Assembler от Microsoft x64):**

; student\_ranking.asm

extrn printf:proc

extrn exit:proc

.data

; Форматы вывода

fmt\_student db "Student %d: d=%d, c=%d, b=%d, a=%d", 0Ah, 0

; Каждый студент представлен как [nd, nc, nb, na]

students dword 2, 1, 1, 0 ; Student 1: [2,1,1,0]

dword 1, 3, 0, 0 ; Student 2: [1,3,0,0]

dword 3, 0, 0, 1 ; Student 3: [3,0,0,1]

dword 1, 0, 0, 3 ; Student 4: [1,0,0,3]

dword 0, 4, 0, 0 ; Student 5: [0,4,0,0]

dword 3, 0, 5, 0 ; Student 6: [3,0,5,0]

student\_count equ 6

student\_size equ 16 ; 4 dword \* 4 bytes = 16 bytes

; Временный массив для сортировки слиянием

temp\_students dword student\_count \* 4 dup(0)

.code

; Структура для передачи параметров студента

Student struct

nd dword ?

nc dword ?

nb dword ?

na dword ?

Student ends

; Функция сравнения

; Вход: RCX = указатель на студента1, RDX = указатель на студента2

; Выход: RAX = -1 (студент1 лучше), 0 (равны), 1 (студент2 лучше)

compare\_students proc

push rsi

push rdi

push rbx

push r12

push r13

push r14

push r15

mov rsi, rcx ; студент1

mov rdi, rdx ; студент2

; сравниваем по а

mov eax, [rsi + Student.na]

mov ebx, [rdi + Student.na]

cmp eax, ebx

jl student1\_better\_step1

jg student2\_better\_step1

; сравниваем по соотношению

; сумма 1

mov eax, [rsi + Student.nd]

mov ebx, [rsi + Student.nc]

mov ecx, [rsi + Student.nb]

mov edx, [rsi + Student.na]

add eax, ebx

add eax, ecx

add eax, edx

mov r12d, eax ; суммируем

; сумма 2

mov eax, [rdi + Student.nd]

mov ebx, [rdi + Student.nc]

mov ecx, [rdi + Student.nb]

mov edx, [rdi + Student.na]

add eax, ebx

add eax, ecx

add eax, edx

mov r13d, eax ; суммируем

; соотношение 1

mov eax, [rsi + Student.nd]

mov ebx, [rsi + Student.nc]

add eax, ebx

imul eax, r13d

mov r14d, eax ; соотношение1

; соотношение 2

mov eax, [rdi + Student.nd]

mov ebx, [rdi + Student.nc]

add eax, ebx

imul eax, r12d

mov r15d, eax ; соотношение2

; Сравниваем соотношения

cmp r14d, r15d

jg student1\_better\_step2

jl student2\_better\_step2

; по оценкам d считаем

mov eax, [rsi + Student.nd]

mov ebx, [rdi + Student.nd]

cmp eax, ebx

jg student1\_better\_step3

jl student2\_better\_step3

; Студенты равны

xor eax, eax

jmp compare\_done

student1\_better\_step1:

mov eax, -1

jmp compare\_done

student2\_better\_step1:

mov eax, 1

jmp compare\_done

student1\_better\_step2:

mov eax, -1

jmp compare\_done

student2\_better\_step2:

mov eax, 1

jmp compare\_done

student1\_better\_step3:

mov eax, -1

jmp compare\_done

student2\_better\_step3:

mov eax, 1

compare\_done:

pop r15

pop r14

pop r13

pop r12

pop rbx

pop rdi

pop rsi

ret

compare\_students endp

; Процедура слияния

; Вход: RCX = массив студентов, RDX = левый индекс, R8 = средний индекс, R9 = правый индекс

merge proc

push rbp

mov rbp, rsp

sub rsp, 40h

mov [rbp+10h], rcx ; массив

mov [rbp+18h], rdx ; левый

mov [rbp+20h], r8 ; центральный

mov [rbp+28h], r9 ; правый

; i = левый, j = центральный+1, k = левый

mov eax, edx

mov [rbp+30h], eax ; i

mov eax, r8d

inc eax

mov [rbp+38h], eax ; j

mov eax, edx

mov [rbp+40h], eax ; k

merge\_loop:

; while i <= j-1 AND j <= right

mov eax, [rbp+30h] ; i

cmp eax, [rbp+20h] ; j-1

jg merge\_copy\_remaining\_left

mov eax, [rbp+38h] ; j

cmp eax, [rbp+28h] ; k

jg merge\_copy\_remaining\_left

; Сравниваем students[i] и students[j]

mov rcx, [rbp+10h] ; массив

mov eax, [rbp+30h] ; i

imul eax, student\_size

lea rcx, [rcx + rax] ; &students[j]

mov rdx, [rbp+10h] ; массив

mov eax, [rbp+38h] ; j

imul eax, student\_size

lea rdx, [rdx + rax] ; &students[j]

call compare\_students

; Если students[i] <= students[j]

cmp eax, 0

jg use\_right\_element

use\_left\_element:

; temp[k] = students[i]

mov rcx, [rbp+10h] ; массив

mov eax, [rbp+30h] ; i

imul eax, student\_size

lea rsi, [rcx + rax] ; &students[i]

mov rdi, offset temp\_students

mov eax, [rbp+40h] ; k

imul eax, student\_size

lea rdi, [rdi + rax] ; &temp[k]

; Копируем

mov rax, [rsi]

mov [rdi], rax

mov rax, [rsi+8]

mov [rdi+8], rax

inc dword ptr [rbp+30h] ; i++

jmp merge\_increment\_k

use\_right\_element:

; temp[k] = students[j]

mov rcx, [rbp+10h] ; массив

mov eax, [rbp+38h] ; j

imul eax, student\_size

lea rsi, [rcx + rax] ; &students[j]

mov rdi, offset temp\_students

mov eax, [rbp+40h] ; k

imul eax, student\_size

lea rdi, [rdi + rax] ; &temp[k]

; Копируем 16 байт (4 dword)

mov rax, [rsi]

mov [rdi], rax

mov rax, [rsi+8]

mov [rdi+8], rax

inc dword ptr [rbp+38h] ; j++

merge\_increment\_k:

inc dword ptr [rbp+40h] ; k++

jmp merge\_loop

merge\_copy\_remaining\_left:

; Копируем оставшиеся элементы из левой части

mov eax, [rbp+30h] ; i

cmp eax, [rbp+20h] ; mid

jg merge\_copy\_remaining\_right

copy\_left\_loop:

mov rcx, [rbp+10h] ; массив

mov eax, [rbp+30h] ; i

imul eax, student\_size

lea rsi, [rcx + rax] ; &students[i]

mov rdi, offset temp\_students

mov eax, [rbp+40h] ; k

imul eax, student\_size

lea rdi, [rdi + rax] ; &temp[k]

mov rax, [rsi]

mov [rdi], rax

mov rax, [rsi+8]

mov [rdi+8], rax

inc dword ptr [rbp+30h] ; i++

inc dword ptr [rbp+40h] ; k++

mov eax, [rbp+30h] ; i

cmp eax, [rbp+20h] ; mid

jle copy\_left\_loop

merge\_copy\_remaining\_right:

; Копируем оставшиеся элементы из правой части

mov eax, [rbp+38h] ; j

cmp eax, [rbp+28h] ; right

jg merge\_copy\_back

copy\_right\_loop:

mov rcx, [rbp+10h] ; массив

mov eax, [rbp+38h] ; j

imul eax, student\_size

lea rsi, [rcx + rax] ; &students[j]

mov rdi, offset temp\_students

mov eax, [rbp+40h] ; k

imul eax, student\_size

lea rdi, [rdi + rax] ; &temp[k]

mov rax, [rsi]

mov [rdi], rax

mov rax, [rsi+8]

mov [rdi+8], rax

inc dword ptr [rbp+38h] ; j++

inc dword ptr [rbp+40h] ; k++

mov eax, [rbp+38h] ; j

cmp eax, [rbp+28h] ; right

jle copy\_right\_loop

merge\_copy\_back:

; Копируем из temp обратно в students

mov eax, [rbp+18h] ; left

mov [rbp+30h], eax ; k = left

copy\_back\_loop:

mov eax, [rbp+30h] ; k

cmp eax, [rbp+28h] ; right

jg merge\_done

mov rsi, offset temp\_students

mov eax, [rbp+30h] ; k

imul eax, student\_size

lea rsi, [rsi + rax] ; &temp[k]

mov rdi, [rbp+10h] ; массив

mov eax, [rbp+30h] ; k

imul eax, student\_size

lea rdi, [rdi + rax] ; &students[k]

mov rax, [rsi]

mov [rdi], rax

mov rax, [rsi+8]

mov [rdi+8], rax

inc dword ptr [rbp+30h] ; k++

jmp copy\_back\_loop

merge\_done:

mov rsp, rbp

pop rbp

ret

merge endp

; Рекурсивная сортировка

; Вход: RCX = массив студентов, RDX = левый индекс, R8 = правый индекс

merge\_sort proc

push rbp

mov rbp, rsp

sub rsp, 20h

mov [rbp+10h], rcx ; массив

mov [rbp+18h], edx ; left

mov [rbp+20h], r8d ; right

; if left >= right, return

mov eax, edx

cmp eax, r8d

jge merge\_sort\_done

; mid = (left + right) / 2

mov eax, edx

add eax, r8d

shr eax, 1

mov [rbp+28h], eax ; mid

; merge\_sort(students, left, mid)

mov rcx, [rbp+10h] ; массив

mov edx, [rbp+18h] ; left

mov r8d, [rbp+28h] ; mid

call merge\_sort

; merge\_sort(students, mid+1, right)

mov rcx, [rbp+10h] ; массив

mov edx, [rbp+28h] ; центральный

inc edx

mov r8d, [rbp+20h] ; правый

call merge\_sort

; merge(students, left, mid, right)

mov rcx, [rbp+10h] ; массив

mov edx, [rbp+18h] ; левый

mov r8d, [rbp+28h] ; центральный

mov r9d, [rbp+20h] ; правый

call merge

merge\_sort\_done:

mov rsp, rbp

pop rbp

ret

merge\_sort endp

; Процедура для демонстрации работы

; Вход: RCX = указатель на массив студентов, RDX = количество студентов

print\_students proc

push rsi

push rdi

push rbx

push r12

mov rsi, rcx ; массив студентов

mov r12d, edx ; количество студентов

xor edi, edi ; индекс

print\_loop:

cmp edi, r12d

jge print\_done

; Вычисляем указатель на текущего студента

mov eax, edi

imul eax, student\_size

lea rbx, [rsi + rax]

; Подготавливаем параметры для printf

sub rsp, 40h

; printf("Student %d: d=%d, c=%d, b=%d, a=%d\n",

; index+1, nd, nc, nb, na)

lea rcx, fmt\_student

mov edx, edi

inc edx ; index+1

mov r8d, [rbx + Student.nd]

mov r9d, [rbx + Student.nc]

mov eax, [rbx + Student.nb]

mov [rsp+20h], eax

mov eax, [rbx + Student.na]

mov [rsp+28h], eax

call printf

add rsp, 40h ; восстанавливаем стек

inc edi

jmp print\_loop

; чистим память

print\_done:

pop r12

pop rbx

pop rdi

pop rsi

ret

print\_students endp

; Основная функция

main proc

sub rsp, 28h ; динамический буфер

; Выводим заголовок

lea rcx, fmt\_separator

call printf

; Выводим исходный массив студентов

lea rcx, offset students

mov edx, student\_count

call print\_students

; Сортируем студентов

lea rcx, offset students

xor edx, edx ; left = 0

mov r8d, student\_count - 1

call merge\_sort

; Выводим заголовок рейтинга

lea rcx, fmt\_ranking

call printf

; Выводим отсортированный массив студентов

lea rcx, offset students

mov edx, student\_count

call print\_students

; Завершаем программу

xor ecx, ecx

call exit

main endp

end

**Дополнительный комментарий:**

Стоит отметить, что мы можем немного оптимизировать ресурсоёмкость данной программы, однако предлагаемое решение сильно зависит от контекста:

В случае, если мы реализуем отдельное ПО которое получает данные, парсит и структурирует их, при этом у нас нет обязательного требования выводить в ответ сами оценки, то нам выгоднее вынести процесс рассчётов отношений суммы к общему количеству и производить один раз, а не при каждом сравнении, в таком случае предложенную структуру [nd ; nc ; nb ; na] стоит заменить на структуру [-na; ((nd+nc)/ Σn) ; nd]. Это позволит нам снизить количество операций для сравнения, что снизить нагрузку на вычислительную мощность аппаратуры и позволит **сэкономить целых n × 16 бит!**

Однако если мы разрабатываем только модуль к конкретному приложению которое уже реализует парсинг данных и передаёт их уже в данной структуре, то данный метод оптимизации неэффективен, поскольку требует больше ресурсов для преобразования одной структуры данных с сохранением ненужных нам фрагментов в другую временную структуры, а так же не интуитивной сортировкой неиспользуемых фрагментов данных на основе главного массива, что увеличивает кодовую базу, усложняет общий алгоритм работы программы и увеличивает ресурсоёмкость, так как нам необходимо хранить неиспользуемые данные отдельно в памяти, перемещать их, а потом имплементировать обратно в отсортированный список с типом временной структуры, возвращая данные к первоначальной структуре.

**Задача ТЗ-4**

**Дано:**

Элайнмент – характеристика персонажа.

В ходе игры элайнмент ограничивает возможности персонажа принять те или иные решения.

Проблема заключается в том, что отдельное решение может быть никак не связано с элайнментом.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Good | Neutral | Evil |
| Lawful |  |  |  |
| Neutral |  |  |  |
| Chaotic |  |  |  |

*Dangeons & Dragons*

**Требуется:**

Найти способ, ограничивающий не отдельные решения, но их совокупности (стратегии).

**Решение:**

1. Уточним условия: необходимо написать условную математическую модель, которая учитывает не только отдельные решения персонажа, но и их совокупность (стратегию).
2. Дополним условия: время условно дискретно, чтобы не создавать игрокам тайм-гейт. Поступки так же дискретны и оказывают влияние в процессе. Помимо влияния поступков нужно так же учитывать текущий эллаймент.
3. Переведём таблицу в двумерное представление по двум осям:  
   оG – ось Добра-Зла от -100 до 100   
   оL – ось Порядка-Хаоса от – 100 до 100
4. Введём понятие фиксированных референсных точек (ФРТ) для каждой оси – это «идеальные» точки проявления эллаймента.

4.1. Персонаж при старте игры имеет свою точку поведения (Pt), которая изначально совпадает с ФРТ – мировоззрением выбранном в начале игры. Данная фрт считается активной (Afrt).

4.2. Каждая ФРТ имеет некоторую зону толерантности, заданную жестко изначально. Если Pt находится не конкретно в ФРТ, но в её зоне толерантности, точка ФРТ считается активной(Afrt). Это позволяет нам определять мировоззрение персонажа, добавляя стабильности системе и не меняя мировоззрение персонажа из-за одного нестандартного\нетипичного для его эллаймента события\поступка.  
4.3 Список ФРТ:

* (-100, 100) - Законопослушный-добрый
* (0, 100) - Законопослушный-нейтральный
* (100, 100) - Законопослушный-злой
* (-100, 0) - Нейтральный-добрый
* (0, 0) - Истинно-нейтральный
* (100, 0) - Нейтральный-злой
* (-100, -100) - Хаотичный-добрый
* (0, -100) - Хаотичный-нейтральный
* (100, -100) - Хаотичный-злой

Зона толерантности: Радиус r\_tol = 15 единиц

Активной точкой (Afrt) может быть только одна точка.

5. Игрок может совершать действия, которые будут влиять на координаты его мировоззрения. Последствия действий для персонажа будут жёсткого заданы заранее\задаваться в моменте мастером игры (в зависимости от контекста). Таким образом на точку будет оказывать влияния вектор Dv (Decision Vector) – вектор принятия решений.

Примеры:

* Мелкий поступок: Dv = (1, 0)
* Крупный поступок: Dv = (15, 10)
* Эпический поступок: Dv = (30, 20)

6. Для того, чтобы сделать систему более плавной и логичной, введём два дополнительных вектора: вектор Mv (Memory Vector) и вектор Rv (Resistance Vector):

6.1 Rv (Resistance Vector) – направлен от текущей точке к активной ФРТ (единичный вектор в этом направлении) умноженный на константу R (сила сопротивления).

Rv = R × (Afrt - Pt) / ||Afrt - Pt||, где R - константа.

6.2 Mv (Memory Vector) направлен также к Afrt, но длина обратно пропорциональна расстоянию. Введем коэффициент M (сила памяти) и зависимость от расстояния:  
Mv = M × (1 / (1 + || Afrt - Pt ||)) × (Afrt - Pt) / || Afrt - Pt ||

Здесь M - константа, а 1/(1+d) - убывающая функция от расстояния d.

7. Тогда изменение позиции будет рассчитываться по формуле: ΔP = Dv + Rv + Mv  
Из которой следует формула следующей позиции:

P[t+1] = Pt + ΔP = Pt + Dv + Rv + Mv = Pt + At + (R × [Afrt - Pt] / [|| Afrt - Pt ||]) + (M × 1/[1 + || Afrt - Pt ||] × [Afrt - Pt] / [|| Afrt - Pt ||]), где:  
Pt = (xt, yt) - текущая позиция персонажа  
Afrt - активная ФРТ в момент времени t  
At - вектор решения (Dv)  
R = 0.3 - константа сопротивления  
M = 1.0 - константа памяти  
|| Afrt - Pt || - евклидово расстояние до активной ФРТ

7. Уточним логику алгоритма:

Персонаж на старте имеет ПРТ, его текущее состояние, которое совпадает с одним из ФРТ. В зависимости от ПРТ персонаж может выбирать ролевые взаимодействия.

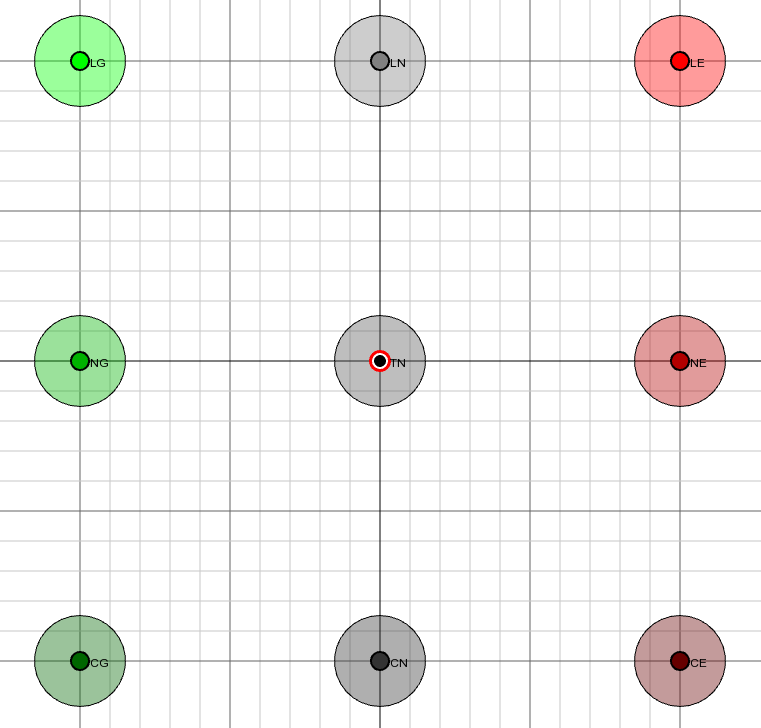
Все действия оказывают влияние на персонажа, при этом учитывается его текущий характер (коэффициент K\_res, зависящий от ПРТ) и его человеческая натура (K\_mem), которая мешает персонажу выбраться из зоны комфорта (зоны толерантности). Это позволяет избежать ситуации, когда один поступок полностью меняет мировоззрение персонажа.

При этом чем дальше персонаж в своих поступках уходит от своего ПРТ тем слабее сопротивление со стороны памяти и его старого мировоззрения – таким образом персонаж делая продолжительное время нехарактерные для него поступки создаёт тенденцию к изменению.

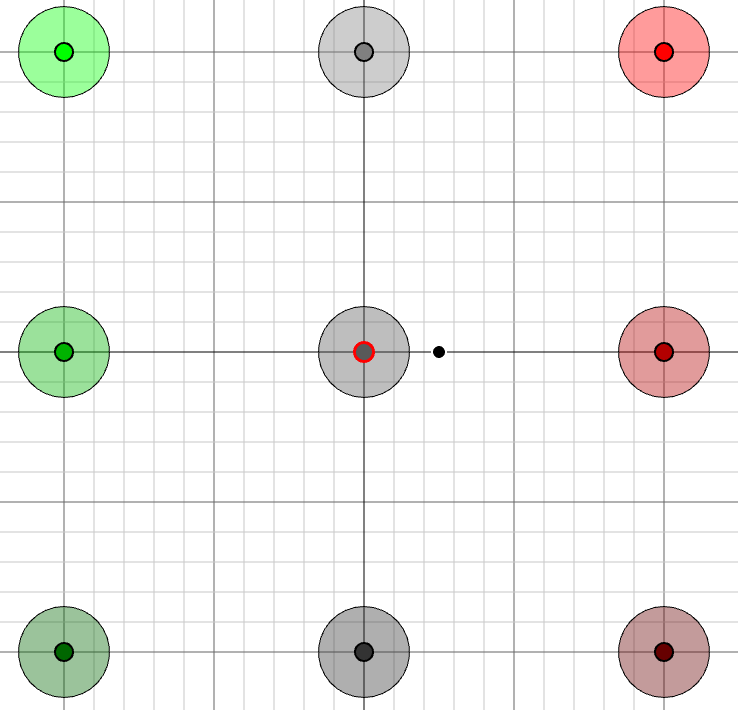
Если же нехарактерные для него поступки редки – они будут сильно ослаблены, а оставшиеся последствия будут нивелированы K\_mem. Таким образом персонаж со временем придёт в свою идеальную референсную позицию.

Стоит отметить, что K\_mem так же может помочь достичь этой позиции со временем, если персонаж добрался до зоны комфорта(толерантности).

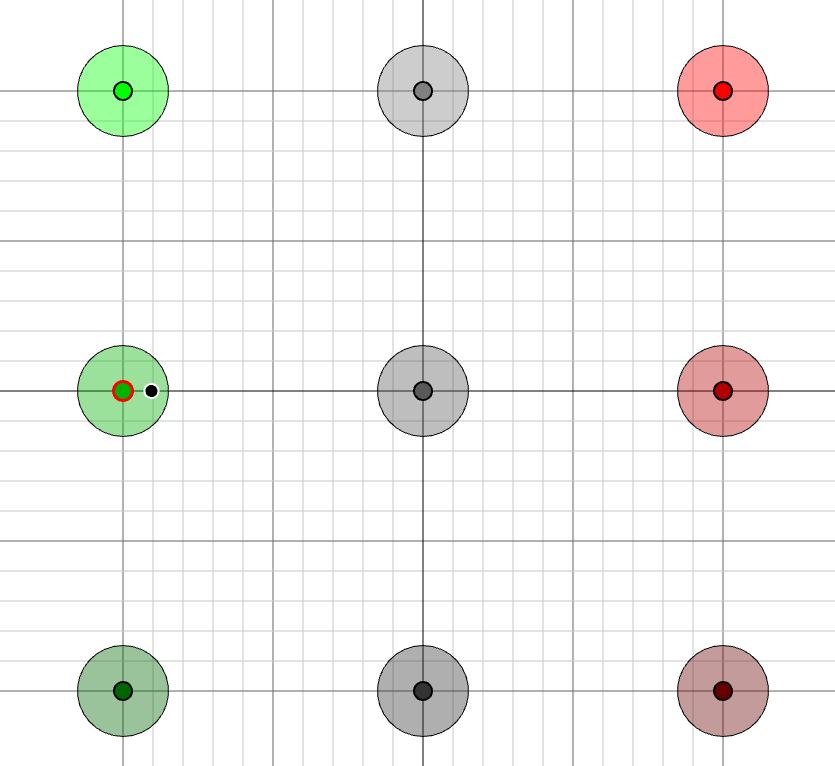
**Пример визуализации:**

****

*Примерная таблица мировоззрений (цвет отвечает за добро, яркость – за законопослушность)*



*Активная Afrt помечена красным, сохранятся при перемещении. Она сохраняется при передвижении точки героя*



*При достижении толерантной зоны – Afrt обновляется, отмечается визуально и выводит оповещение в консоль.*



*Пример вывода в консоль*

Ознакомиться подробнее с программой, визуализирующей данный подход вы можете в гит-репозитории:  
https://github.com/Denis-Zhelaev/Ranepa-SSPP-2025