Национальный исследовательский Нижегородский государственный университет имени Н. И. Лобачевского.

Кафедра «Информатики и автоматизации научных исследований»

Отчёт по лабораторной работе № 10.

Выполнил:

Студент группы 381807-1

Суслов Никита Сергеевич

Руководитель:

Неймарк Е.А.

2020 год.

Содержание.

1. Постановка задачи………………………………………………………………………….3
2. Описание реализованного алгоритма……………………………………………4
3. Описание эксперимента…………………………………………………………………6
4. Результаты эксперимента в виде таблицы…………………………………….7
5. Выводы по результатам эксперимента………………………………………….8
6. Приложения…………………………………………………………………………………….9
7. Постановка задачи.

Математическая:

Дано N предметов, W — вместимость рюкзака, w={w1,w2,…,wN} — соответствующий ему набор положительных целых весов, p={p1,p2,…,pN} — соответствующий ему набор положительных целых стоимостей. Нужно найти набор бинарных величин B={b1,b2,…,bN}, где bi=1, если предмет ni включен в набор; bi=0, если предмет ni не включен, и такой что:

b1w1+⋯+bNwN ⩽ W

b1p1+⋯+bNpN максимальна.

Словесная:

Из заданного множества предметов со свойствами «стоимость» и «вес», требуется отобрать некое число предметов таким образом, чтобы получить максимальную суммарную стоимость при одновременном соблюдении ограничения на суммарный вес.

1. Описание реализованного алгоритма

Способ кодирования: Бинарное кодирование

Методы формирования начальной популяции:

1. Случайный с контролем ограничений. Случайно генерируется кандидат, если он является решением, то помещается в начальную популяцию, иначе кандидат генерируется заново.
2. С использованием жадной эвристики. Случайно кладётся один из предметов. Потом кладутся самые ценные предметы, пока они вмещаются.

Способ выбора родительской пары:

1. Случайный. Равновероятностно выбираются оба родителя.
2. Выбираются две лучших особи.

Операторы кроссовера:

1. Одноточечный. Первую часть (до случайно выбранного бита) берется от первого родителя, остальное – от второго.
2. Однородный. Генерируется случайное число 0 или 1. Если 0, то бит берется от первого родителя, иначе от второго. Так для всех битов в кодировке.

Операторы мутации:

1. Макромутация. Сальтация. Значения двух случайных битов меняются местами.
2. Хромосомная. Дополнение. Значение каждого бита кодировки меняется на противоположенное.

Стратегия формирования следующего поколения:

1. Стратегия перекрывающихся поколений. Элитарная. Особи копируются не только из созданных потомков, но и из множества родителей. При этом лучшая особь точно переходит в следующее поколение.

Операторы селекции:

1. B-турнир. Формируется турнир из b участников, самый приспособленный из них переходит в следующее поколение. Так повторяется, пока следующее поколение не будет заполнено.
2. Универсальный выбор. Генерируются равномерно распределённые случайные величины, столько, сколько должно быть особей в следующем поколении. На каждую случайную величину выбирается первая из особей, для которой сумма отношений приспособленности к сумме всех приспособленностей предыдущий особей и текущей особи больше этой величины, эта особь и проходит в следующее поколение.

Условия остановки: В начале программы задаётся число итераций алгоритма.

1. Описание эксперимента.

Основная задача алгоритма – найти оптимальное решение или максимально приблизиться к нему.

Цель:

1. Найти набор операторов селекции и выбора начальной популяции, наиболее эффективных для малого количества итераций.
2. Найти набор операторов селекции и выбора начальной популяции получающих оптимальное значение при очень большом количестве итераций количестве итераций.
3. Совпадают ли полученные наборы?

Параметры алгоритма:

Количество запусков для каждого набора параметров – 10. Размер популяции – 15. Способ выбора родительской пары - Случайный. Оператор кроссовера - Одноточечный. Оператор мутации – Сальтация. Стратегия формирования следующего поколения - Стратегия перекрывающихся поколений, Элитарная.

Параметры эксперимента:

1. Оператор выбора начальной популяции – Случайный. Оператор селекции – B-турнир. Количество итераций – 5.
2. Оператор выбора начальной популяции – Случайный. Оператор селекции – Универсальный. Количество итераций – 5.
3. Оператор выбора начальной популяции – Жадная эвристика. Оператор селекции – B-турнир. Количество итераций – 5.
4. Оператор выбора начальной популяции – Жадная эвристика. Оператор селекции – Универсальный. Количество итераций – 5.
5. Оператор выбора начальной популяции – Случайный. Оператор селекции – B-турнир. Количество итераций – 1000.
6. Оператор выбора начальной популяции – Случайный. Оператор селекции – Универсальный. Количество итераций – 1000.
7. Оператор выбора начальной популяции – Жадная эвристика. Оператор селекции – B-турнир. Количество итераций – 1000.
8. Оператор выбора начальной популяции – Жадная эвристика. Оператор селекции – Универсальный. Количество итераций – 1000.

4. Результаты эксперимента в виде таблицы.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Случайный  B-турнир  5 | Случайный  Универ -сальный  5 | Жадный  B-турнир  5 | Жадный  Универ-сальный  5 | Случайный  B-турнир  1000 | Случайный  Универ-сальный  1000 | Жадный  B-турнир  1000 | Жадный  Универ-сальный  1000 |
| Лучшее решение  (больше – лучше) | 178 | 176 | 178 | **178** | 178 | **178** | 178 | **178** |
| Вероятность получения оптимума  (больше – лучше) | 0.1 | 0 | 0.9 | **1** | 0.4 | **1** | 0.9 | **1** |
| Среднее получаемое решение  (больше – лучше) | 161.9 | 158.6 | **177.5** | **178** | 170.5 | **178** | **177.5** | **178** |
| Отклонение среднего от оптимума  (меньше – лучше) | 16.1 | 19.4 | **2.5** | **0** | 7.5 | **0** | **2.5** | **0** |
| Максимальное отклонение от оптимума  (меньше – лучше) | 34 | 34 | **5** | **0** | 28 | **0** | **5** | **0** |
| Был ли хоть раз получен оптимум? | Да | Нет | Да | Да | Да | Да | Да | Да |

Оптимальным решением задачи является 000000111010111 с ценностью 178 или 101000110100111 с ценностью 178.

Отклонение среднего от оптимума равно разности оптимума и среднеарифметического из результатов серии.

Среднее получаемое решение равно среднеарифметическому из результатов серии.

5. Выводы по результатам эксперимента.

Наилучшим набором для малого количества итераций оказался (Жадный, Универсальный)

При большом количестве итераций (Случайный, Универсальный) и (Жадный, Универсальный) получили оптимальное решение с вероятностью 1.

Исходя из этого (Жадный, Универсальный) дают наилучшее решение на малом и большом количестве итераций.

При этом Универсальный оператор селекции при большом количестве итераций даёт оптимальное решение с вероятностью 1 вне зависимости от оператора выбора начальной популяции.

Жадный оператор выбора начальной популяции даёт наибольшее среднее решение по сравнению со случайным вне зависимости от количества итераций и оператора селекции.

6. Приложения.

Серия 1: Оператор выбора начальной популяции – Случайный. Оператор селекции – B-турнир. Количество итераций – 5.

|  |  |  |
| --- | --- | --- |
| Запуск | Кодировка | Ценность |
| 1 | 000001111100110 | 158 |
| 2 | 000000111100111 | 172 |
| 3 | 001100101000111 | 144 |
| 4 | 100001110010111 | 173 |
| 5 | 100010111010010 | 148 |
| 6 | 101000111100011 | 170 |
| 7 | 101000011010110 | 160 |
| 8 | 101000110100111 | 178 |
| 9 | 101000111010010 | 156 |
| 10 | 101001110000111 | 160 |

Серия 2: Оператор выбора начальной популяции – Случайный. Оператор селекции – Универсальный. Количество итераций – 5.

|  |  |  |
| --- | --- | --- |
| Запуск | Кодировка | Ценность |
| 1 | 100001100010111 | 144 |
| 2 | 000000111010011 | 148 |
| 3 | 001001111000111 | 171 |
| 4 | 101000111010010 | 156 |
| 5 | 101000111000111 | 176 |
| 6 | 100011111000110 | 154 |
| 7 | 000000111110010 | 152 |
| 8 | 100100100010111 | 146 |
| 9 | 001000011100111 | 163 |
| 10 | 101000111000111 | 176 |

Серия 3: Оператор выбора начальной популяции – Жадная эвристика. Оператор селекции – B-турнир. Количество итераций – 5.

|  |  |  |
| --- | --- | --- |
| Запуск | Кодировка | Ценность |
| 1 | 000000111010111 | 178 |
| 2 | 000000111010111 | 178 |
| 3 | 000000111010111 | 178 |
| 4 | 000000111010111 | 178 |
| 5 | 001000110010111 | 173 |
| 6 | 000000111010111 | 178 |
| 7 | 000000111010111 | 178 |
| 8 | 000000111010111 | 178 |
| 9 | 000000111010111 | 178 |
| 10 | 000000111010111 | 178 |

Серия 4: Оператор выбора начальной популяции – Жадная эвристика. Оператор селекции – Универсальный. Количество итераций – 5.

|  |  |  |
| --- | --- | --- |
| Запуск | Кодировка | Ценность |
| 1 | 000000111010111 | 178 |
| 2 | 000000111010111 | 178 |
| 3 | 000000111010111 | 178 |
| 4 | 000000111010111 | 178 |
| 5 | 000000111010111 | 178 |
| 6 | 000000111010111 | 178 |
| 7 | 000000111010111 | 178 |
| 8 | 000000111010111 | 178 |
| 9 | 000000111010111 | 178 |
| 10 | 000000111010111 | 178 |

Серия 5: Оператор выбора начальной популяции – Случайный. Оператор селекции – B-турнир. Количество итераций – 1000.

|  |  |  |
| --- | --- | --- |
| Запуск | Кодировка | Ценность |
| 1 | 000000111010111 | 178 |
| 2 | 100001110010111 | 173 |
| 3 | 100000010110111 | 165 |
| 4 | 101000110100111 | 178 |
| 5 | 101000110100111 | 150 |
| 6 | 000000111010111 | 178 |
| 7 | 000100111010011 | 156 |
| 8 | 101000110100111 | 178 |
| 9 | 100001110010111 | 173 |
| 10 | 101000111000111 | 176 |

Серия 6: Оператор выбора начальной популяции – Случайный. Оператор селекции – Универсальный. Количество итераций – 1000.

|  |  |  |
| --- | --- | --- |
| Запуск | Кодировка | Ценность |
| 1 | 000000111010111 | 178 |
| 2 | 000000111010111 | 178 |
| 3 | 101000110100111 | 178 |
| 4 | 000000111010111 | 178 |
| 5 | 101000110100111 | 178 |
| 6 | 101000110100111 | 178 |
| 7 | 101000110100111 | 178 |
| 8 | 101000110100111 | 178 |
| 9 | 101000110100111 | 178 |
| 10 | 000000111010111 | 178 |

Серия 7: Оператор выбора начальной популяции – Жадная эвристика. Оператор селекции – B-турнир. Количество итераций – 1000.

|  |  |  |
| --- | --- | --- |
| Запуск | Кодировка | Ценность |
| 1 | 000000111010111 | 178 |
| 2 | 000000111010111 | 178 |
| 3 | 000000111010111 | 178 |
| 4 | 000000111010111 | 178 |
| 5 | 001000110010111 | 173 |
| 6 | 000000111010111 | 178 |
| 7 | 000000111010111 | 178 |
| 8 | 000000111010111 | 178 |
| 9 | 000000111010111 | 178 |
| 10 | 000000111010111 | 178 |

Серия 8: Оператор выбора начальной популяции – Жадная эвристика. Оператор селекции – Универсальный. Количество итераций – 1000.

|  |  |  |
| --- | --- | --- |
| Запуск | Кодировка | Ценность |
| 1 | 000000111010111 | 178 |
| 2 | 000000111010111 | 178 |
| 3 | 000000111010111 | 178 |
| 4 | 000000111010111 | 178 |
| 5 | 000000111010111 | 178 |
| 6 | 000000111010111 | 178 |
| 7 | 000000111010111 | 178 |
| 8 | 000000111010111 | 178 |
| 9 | 000000111010111 | 178 |
| 10 | 000000111010111 | 178 |

Программный код.

**main.cpp**

#include "matrix.h"

#include "code.h"

#include <vector>

#include <random>

/// Варинат 7

const int LENGTH\_OF\_CODE = 15; // Длина кодировка (количество предметов)

const int Bin::Lsize = LENGTH\_OF\_CODE;

// Возвращает вес кодировки

int Weight(const Bin& b, const Matrix<int>& m)

{

int sum = 0;

for (int i = 0; i < LENGTH\_OF\_CODE; ++i)

{

sum += b[i] \* m(i, 2); // 2 это столбец весов

}

return sum;

}

// Возвращает ценность кодировки

int Cost(const Bin& b, const Matrix<int>& m)

{

int sum = 0;

for (int i = 0; i < LENGTH\_OF\_CODE; ++i)

{

sum += b[i] \* m(i, 1); // 1 это столбец цен

}

return sum;

}

// Случайный алгоритм генерации начальной популяции с контролем ограничений

std::vector< Bin > CreateNewPopulationRand(const int size, const Matrix<int>& m, const int bagCapacity)

{

std::vector< Bin > Result;

int CurrentCapacity;

while (Result.size() < size)

{

Bin temp( rand() % static\_cast<int>(pow(2, Bin::Lsize)) );

if (Weight(temp, m) <= bagCapacity)

{

Result.push\_back(temp);

}

}

return Result;

}

// Жадный алгоритм генерации начальной популяции

std::vector< Bin > CreateNewPopulationGreedy(const int size, const Matrix<int>& m, const int bagCapacity)

{

std::vector< Bin > Result;

int CurrentCapacity, i, MaxPrice, j, NumberMax;

while (Result.size() < size)

{

Bin temp;

CurrentCapacity = 0;

i = rand() % LENGTH\_OF\_CODE;

temp[i] = 1;

CurrentCapacity += temp[i] \* m(i, 2); // 2 - это столбец весов в матрице

i = 1;

while ( (i < LENGTH\_OF\_CODE) )

{

MaxPrice = -1;

for (j = 0; j < LENGTH\_OF\_CODE; ++j)

{

if ((MaxPrice < m(j, 1)) and (temp[j] == 0) and (CurrentCapacity + m(j, 2) <= bagCapacity)) // 1 это столбец цен в матрице, 2 - это столбец весов в матрице

{

MaxPrice = m(j, 1);

NumberMax = j;

}

}

if (MaxPrice == -1)

{

break;

}

temp[NumberMax] = 1;

CurrentCapacity += m(NumberMax, 2);

++i;

}

Result.push\_back(temp);

}

return Result;

}

// Одноточечный кроссовер со случайным выбором родителей

void SinglePointReproduction(const int size, const Matrix<int>& m, const int bagCapacity, std::vector< Bin >& population)

{

if (population.size() > 40)

{

srand(time(0));

}

int p1, p2, point, n, l;

l = population.size();

n = population.size() \* 2 + rand() % (5 \* size) - size / 4;

while (population.size() < n)

{

Bin temp;

p1 = rand() % l;

do

{

p2 = rand() % l;

} while (p1 == p2);

point = rand() % (LENGTH\_OF\_CODE - 1) + 1; // от 1 до 14 <

for (int j = 0; j < point; ++j)

{

temp[j] = population.at(p1)[j];

}

for (int j = point; j < LENGTH\_OF\_CODE; ++j)

{

temp[j] = population.at(p2)[j];

}

if (Weight(temp, m) <= bagCapacity)

{

population.push\_back(temp);

}

}

}

// Однородный кроссовер с наиболее приспособленными родителями

void HomogeneousReproduction(const int size, const Matrix<int>& m, const int bagCapacity, std::vector< Bin >& population)

{

if (population.size() > 40)

{

srand(time(0));

}

Bin Best1, Best2;

int max = -1;

for (int i = 0; i < population.size(); ++i)

{

if (max < Cost(population[i], m))

{

max = Cost(population[i], m);

Best1 = population[i];

}

}

max = -1;

for (int i = 0; i < population.size(); ++i)

{

if ( (max < Cost(population[i], m)) and (population[i] != Best1) )

{

max = Cost(population[i], m);

Best2 = population[i];

}

}

int n;

n = population.size() \* 2 + rand() % (5 \* size) - size / 4;

while (population.size() < n)

{

Bin temp;

for (int i = 0; i < LENGTH\_OF\_CODE; ++i)

{

max = rand() % 2;

if (max == 0)

{

temp[i] = Best1[i];

}

else

{

temp[i] = Best2[i];

}

}

if (Weight(temp, m) <= bagCapacity)

{

population.push\_back(temp);

}

}

}

// Мутация Сальтация (где два случайных бита меняются местами)

void MutationSaltation(const int size, const Matrix<int>& m, const int bagCapacity, std::vector< Bin >& population)

{

if (population.size() > 40)

{

srand(time(0));

}

int p, g1, g2;

Bin temp;

for (int i = 0; i < population.size(); ++i)

{

p = rand() % 5; // мутация с вероятностью 0.2

if (p == 0)

{

g1 = rand() % LENGTH\_OF\_CODE;

do

{

g2 = rand() % LENGTH\_OF\_CODE;

} while (g1 == g2);

temp = population[i];

p = temp[g1];

temp[g1] = temp[g2];

temp[g2] = p;

if (Weight(temp, m) <= bagCapacity)

{

population.push\_back(temp);

}

}

}

}

// Мутация Дополнение (где каждый бит меняется на противоположенный)

void MutationAddition(const int size, const Matrix<int>& m, const int bagCapacity, std::vector< Bin >& population)

{

if (population.size() > 40)

{

srand(time(0));

}

int p;

Bin temp;

for (int i = 0; i < population.size(); ++i)

{

p = rand() % 10; // мутация с вероятностью 0.1

if (p == 0)

{

temp = population[i];

for (int j = 0; j < LENGTH\_OF\_CODE; ++j)

{

if (temp[j] == 0)

{

temp[j] = 1;

}

else

{

temp[j] = 0;

}

}

if (Weight(temp, m) <= bagCapacity)

{

population.push\_back(temp);

}

}

}

}

// Оператор селекции. Стратегия перекрывающихся поколений. Элитарная. B-турнир

std::vector< Bin > SelectionBTournament(const int size, const Matrix<int>& m, const int bagCapacity, std::vector< Bin >& population)

{

if (population.size() > 40)

{

srand(time(0));

}

std::vector< Bin > NextGen;

Bin temp;

int max = -1;

for (int i = 0; i < population.size(); ++i)

{

if (max < Cost(population[i], m))

{

max = Cost(population[i], m);

temp = population[i];

}

}

NextGen.push\_back(temp); // лучший проходит сразу

int b, i, num;

b = 2; // размер турнира

std::vector< Bin > B;

i = 0;

for (int j = 1; j < size; ++j)

{

B.clear();

while (B.size() < b)

{

num = rand() % size;

if (num == 0)

{

B.push\_back(population[i % population.size()]);

}

++i;

}

max = -1;

for (int k = 0; k < b; ++k)

{

if (max < Cost(B[k], m))

{

max = Cost(B[k], m);

temp = B[k];

}

}

NextGen.push\_back(temp);

}

return NextGen;

}

// Оператор селекции. Стратегия перекрывающихся поколений. Элитарная. Универсальный выбор

std::vector< Bin > SelectionUniversaChoice(const int size, const Matrix<int>& m, const int bagCapacity, std::vector< Bin >& population)

{

if (population.size() > 40)

{

srand(time(0));

}

std::vector< Bin > NextGen;

Bin temp;

int max = -1;

for (int i = 0; i < population.size(); ++i)

{

if (max < Cost(population[i], m))

{

max = Cost(population[i], m);

temp = population[i];

}

}

NextGen.push\_back(temp); // лучший проходит сразу

std::vector< double > RandomVariables;

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_real\_distribution<> dis(0.0, 1.0); // равномерное распределение

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

{

RandomVariables.push\_back(dis(gen));

}

int denominator = 0;

for (int i = 0; i < population.size(); ++i)

{

denominator += Cost(population[i], m);

}

double x;

int i;

for (int k = 0; k < size; ++k)

{

x = 0;

for (i = 0; i < population.size(); ++i)

{

x += static\_cast< double >(Cost(population[i], m)) / static\_cast<double>(denominator);

if (x >= RandomVariables[k])

{

break;

}

}

NextGen.push\_back(population[i]);

}

return NextGen;

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(0));

Matrix<int> Input = {

{1, 11, 6},

{2, 1, 20},

{3, 17, 16},

{4, 8, 17},

{5, 9, 19},

{6, 6, 12},

{7, 26, 12},

{8, 29, 4},

{9, 22, 19},

{10, 24, 25},

{11, 30, 30},

{12, 17, 29},

{13, 30, 17},

{14, 21, 3},

{15, 20, 10}

};

int BagCapacity = 95;

int Answer[5] = {0,0,0,0,0}, PopulationSize, NumberOfIterations;

//std::cout << Input << std::endl;

std::cout << "Введите размер популяции: ";

std::cin >> PopulationSize;

std::cout << "Введите количество итераций алгоритма: ";

std::cin >> NumberOfIterations;

std::cout << std::endl;

std::vector<Bin> Population;

////////////////////////////////////////////////////////////////////////

///// диалог выбора других операторов

std::cout << "Алгоритм по умолчанию: " << std::endl;

while (true)

{

if (Answer[1] == 0) // алгоритм создания начальной популяции

{

std::cout << "1. Формирование новой популяции: Случайный алгоритм с контролем ограничений" << std::endl;

}

else

{

std::cout << "1. Формирование новой популяции: Жадный алгоритм" << std::endl;

}

if (Answer[2] == 0)

{

std::cout << "2. Оператор кроссовера: Одноточечный кроссовер со случайным выбором родителей" << std::endl;

}

else

{

std::cout << "2. Оператор кроссовера: Однородный кроссовер с наиболее приспособленными родителями" << std::endl;

}

if (Answer[3] == 0)

{

std::cout << "3. Оператор мутации: Сальтация (два случайных бита меняются местами)" << std::endl;

}

else

{

std::cout << "3. Оператор мутации: Дополнение (каждый бит меняется на противоположенный)" << std::endl;

}

if (Answer[4] == 0)

{

std::cout << "4. Оператор селекции: Стратегия перекрывающихся поколений. Элитарная. B-турнир" << std::endl;

}

else

{

std::cout << "4. Оператор селекции: Стратегия перекрывающихся поколений. Элитарная. Универсальный выбор" << std::endl;

}

// остальные операторы

std::cout << "Изменить алгоритм (выбрать другие операторы). 0 или 1? ";

std::cin >> Answer[0];

std::cout << std::endl;

if (Answer[0] == 0)

{

break;

}

std::cout << "Пункт алгоритма: ";

std::cin >> Answer[0];

switch (Answer[0])

{

case 1:

std::cout << "0 - Случайный алгоритм с контролем ограничений" << std::endl;

std::cout << "1 - Жадный алгоритм" << std::endl;

std::cout << "Варинат: ";

std::cin >> Answer[1];

break;

case 2:

std::cout << "0 - Одноточечный кроссовер со случайным выбором родителей" << std::endl;

std::cout << "1 - Однородный кроссовер с наиболее приспособленными родителями" << std::endl;

std::cout << "Вариант: ";

std::cin >> Answer[2];

break;

case 3:

std::cout << "0 - Сальтация (два случайных бита меняются местами)" << std::endl;

std::cout << "1 - Дополнение (каждый бит меняется на противоположенный)" << std::endl;

std::cout << "Вариант: ";

std::cin >> Answer[3];

break;

case 4:

std::cout << "0 - B-турнир. Стратегия перекрывающихся поколений. Элитарная." << std::endl;

std::cout << "1 - Универсальный выбор. Стратегия перекрывающихся поколений. Элитарная." << std::endl;

std::cout << "Вариант: ";

std::cin >> Answer[4];

break;

default:

break;

}

std::cout << std::endl;

}

////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////

if (Answer[1] == 0) // создание начальной популяции

{

Population = CreateNewPopulationRand(PopulationSize, Input, BagCapacity);

}

else

{

Population = CreateNewPopulationGreedy(PopulationSize, Input, BagCapacity);

}

int MaxCost = 0;

Bin CodeMaxCost;

std::cout << "Поколение 0:" << std::endl;

for (auto it = Population.begin(); it != Population.end(); ++it)

{

std::cout << (\*it) << ". Вес: " << Weight((\*it), Input) << ". Ценность: " << Cost((\*it), Input) << std::endl;

if (MaxCost < Cost((\*it), Input))

{

MaxCost = Cost((\*it), Input);

CodeMaxCost = (\*it);

}

}

std::cout << "Лучшая особь: " << CodeMaxCost << ". Ценность: " << Cost(CodeMaxCost, Input) << std::endl << std::endl;

for (int i = 0; i < NumberOfIterations; ++i) // основной цикл алгоритма

{

// Воспроизводство

if (Answer[2] == 0)

{

SinglePointReproduction(PopulationSize, Input, BagCapacity, Population);

}

else

{

HomogeneousReproduction(PopulationSize, Input, BagCapacity, Population);

}

// Мутация

if (Answer[3] == 0)

{

MutationSaltation(PopulationSize, Input, BagCapacity, Population);

}

else

{

MutationAddition(PopulationSize, Input, BagCapacity, Population);

}

// Отбор

if (Answer[4] == 0)

{

Population = SelectionBTournament(PopulationSize, Input, BagCapacity, Population);

}

else

{

Population = SelectionUniversaChoice(PopulationSize, Input, BagCapacity, Population);

}

std::cout << "Поколение " << i+1 << ": " << std::endl;

for (auto it = Population.begin(); it != Population.end(); ++it)

{

std::cout << (\*it) << ". Вес: " << Weight((\*it), Input) << ". Ценность: " << Cost((\*it), Input) << std::endl;

if (MaxCost < Cost((\*it), Input))

{

MaxCost = Cost((\*it), Input);

CodeMaxCost = (\*it);

}

}

std::cout << "Лучшая особь: " << CodeMaxCost << ". Ценность: " << Cost(CodeMaxCost, Input) << std::endl << std::endl;

}

std::cout << "Итог работы: " << CodeMaxCost << ". Ценность: " << Cost(CodeMaxCost, Input) << std::endl << std::endl;

}

**code.h**

#pragma once

#include <string>

#include <cmath>

extern const int LENGTH\_OF\_CODE;

class Bin

{

private:

bool\* arr;

int numberD;

public:

static const int Lsize; // длина кодировки

Bin()

{

numberD = 0;

arr = new bool[Lsize];

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

{

arr[i] = 0;

}

}

Bin(const Bin& n) : numberD(n.numberD)

{

arr = new bool[Lsize];

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

{

arr[i] = n.arr[i];

}

}

Bin(const int n) : numberD(n)

{

int i, temp = numberD;

arr = new bool[Lsize];

for (i = Lsize - 1; i >= 0; --i)

{

arr[i] = temp % 2;

temp /= 2;

}

}

~Bin()

{

delete[] arr;

}

void operator= (const Bin& right)

{

numberD = right.numberD;

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

{

arr[i] = right.arr[i];

}

}

bool operator== (const Bin& right) const

{

return numberD == right.numberD;

}

bool operator!= (const Bin& right) const

{

return !( (\*this) == right );

}

bool& operator[] (const int num)

{

if (num >= Lsize)

{

throw ("num >= Lsize");

}

return arr[num];

}

bool operator[] (const int num) const

{

if (num >= Lsize)

{

throw ("num >= Lsize");

}

return arr[num];

}

friend bool operator< (const Bin& left, const Bin& right);

friend std::ostream& operator<< (std::ostream& out, const Bin& m);

};

bool operator< (const Bin& left, const Bin& right)

{

return left.numberD < right.numberD ? 1 : 0;

}

std::ostream& operator<< (std::ostream& out, const Bin& m)

{

for (int i = 0; i < Bin::Lsize; ++i)

{

out << m[i];

}

return out;

}

**matrix.h**

#pragma once

#include <string>

#include <fstream>

#include <iostream>

#include <ctime>

template <typename TypeMatrix>

class Matrix

{

public:

Matrix(const std::string fileName);

Matrix(const Matrix<TypeMatrix>& m);

Matrix();

Matrix(const std::initializer\_list< std::initializer\_list<TypeMatrix> > in);

~Matrix();

template <typename T>

friend std::ostream& operator<< (std::ostream& out, const Matrix<T>& m);

TypeMatrix& operator() (const int line, const int col) const;

int size() const;

int len() const;

private:

int height;

int width;

TypeMatrix\*\* arr = nullptr;

};

template <typename TypeMatrix>

Matrix<TypeMatrix>::Matrix()

{

std::cout << "Height = ";

std::cin >> height;

std::cout << "Width = ";

std::cin >> width;

arr = new TypeMatrix \* [height];

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

{

arr[i] = new TypeMatrix[width];

for (int j = 0; j < width; ++j)

{

std::cin >> arr[i][j];

}

}

}

template <typename TypeMatrix>

Matrix<TypeMatrix>::Matrix(const Matrix& m)

{

height = m.height;

width = m.width;

arr = new TypeMatrix \* [height];

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

{

arr[i] = new TypeMatrix[width];

for (int j = 0; j < width; ++j)

{

arr[i][j] = m.arr[i][j];

}

}

}

template <typename TypeMatrix>

Matrix<TypeMatrix>::Matrix(const std::string fileName)

{

std::ifstream iFile;

iFile.open(fileName);

if (!iFile.is\_open())

{

throw("Can't open file");

}

iFile >> height >> width;

arr = new TypeMatrix \* [height];

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

{

arr[i] = new TypeMatrix[width];

for (int j = 0; j < width; ++j)

{

iFile >> arr[i][j];

}

}

iFile.close();

}

template <typename TypeMatrix>

Matrix<TypeMatrix>::~Matrix()

{

if (arr != nullptr)

{

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

{

delete[] arr[i];

}

delete[] arr;

}

}

template <typename TypeMatrix>

std::ostream& operator<< (std::ostream& out, const Matrix< TypeMatrix >& m)

{

for (int i = 0; i < m.height; ++i)

{

for (int j = 0; j < m.width; ++j)

{

out.width(4);

if (j == m.width - 1)

{

out << m.arr[i][j];

}

else

{

out << m.arr[i][j] << " ";

}

}

out << "\n";

}

return out;

}

template <typename TypeMatrix>

TypeMatrix& Matrix<TypeMatrix>::operator() (const int line, const int col) const

{

if (line >= height)

{

throw("line >= height");

}

if (col >= width)

{

throw("col >= width");

}

return arr[line][col];

}

template <typename TypeMatrix>

Matrix< TypeMatrix >::Matrix(const std::initializer\_list< std::initializer\_list<TypeMatrix> > in)

{

arr = new TypeMatrix \* [in.size()];

int i = 0, j = 0;

for (auto it = in.begin(); it != in.end(); ++it)

{

arr[i] = new TypeMatrix[(\*it).size()];

j = 0;

for (auto jt = (\*it).begin(); jt != (\*it).end(); ++jt)

{

arr[i][j] = \*jt;

++j;

}

++i;

}

height = i;

width = j;

}

template <typename TypeMatrix>

int Matrix< TypeMatrix >::size() const

{

if (width != height)

{

throw ("width != height");

}

return width;

}

void createRandMatrix(const int height, const int width, std::string fileName)

{

std::ofstream oFile;

oFile.open(fileName);

if (!oFile.is\_open())

{

throw("Can't open file");

}

srand(time(0));

oFile << height << " " << width << std::endl;

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

{

for (int j = 0; j < width; ++j)

{

if (j == width - 1)

{

oFile << (rand() % 20);

}

else

{

oFile << (rand() % 20) << " ";

}

}

oFile << std::endl;

}

oFile.close();

}

template < typename TypeMatrix >

int Matrix< TypeMatrix >::len() const

{

return height;

}