Титульник

**Содержание**

Глава 1.Теория………………………………………………………….…………3

* 1. Вступление……………………………………..……………………….3
  2. Доминирование и LP-доминирование………..……………………….3
  3. Итоговый алгоритм и верхние границы...…………………..…………6

Глава 2.Реализация………………………………………………………..………7

2.1 Решение…………………………………………………………………7  
2.2 Тест………………………………………………………………..……13

Заключение……………………………………………………………………….16

Приложения……………………………………………………………………...17

Список Литературы……………………………………………………………...24

**Глава 1. Теория**

Условие Задачи:

Рюкзак с мультивыбором (The Multiple-Choice Knapsack Problem):

Пусть имеется набор предметов, каждый из которых имеет два параметра: вес и ценность. Также имеется рюкзак определённой вместимости. Нужно собрать рюкзак с максимальной ценностью предметов внутри, соблюдая при этом ограничение рюкзака на суммарный вес. Предметы разделены на группы, из каждой группы требуется выбрать только один предмет.

1. **Вступление**

Пусть у нас есть m непересекающихся классов **N1, N2,…,Nm** с предметами и рюкзак с вместимостью **с**. Каждый предмет **j ∈ Ni**, имеет ценность **pij** и вес **wij**. Наша задача выбрать ровно 1 предмет из каждого класса таким образом, чтобы сумма прибыли была максимальная без превышения вместимости рюкзака **с**.

Если ввести бинарные переменные **xij**, которые принимают значение 1 тогда и только тогда, когда предмет **j** выбран в классе **Ni**, наша задача будет формулироваться так:

Maximize ,

При условии: ,

i = 1,…, m.

∈ {0, 1}, i = 1,…, m, j ∈ Ni

Будем считать, что все и **c** — неотрицательные целые числа, причём **Ni** имеет размер **ni**.

1. **Доминирование и LP-доминирование**

**Определение:**Если предметы **j** и **k** в классе **Ni** удовлетворяют условиям и , мы говорим, что **j** доминирует над **k**.

То есть, предмет имеет больший вес и меньшую ценность, чем какой-либо другой предмет, то он нас не интересует, так как его выбор является хуже по всем параметрам, следовательно, его мы рассматривать далее не будем.

Отсюда сразу вытекает следующее **Утверждение:**

Даны **j,k ∈ Ni**. Если **j** доминирует над **k**, то оптимальное решение существует при **xik = 0**.

**Определение:**  
Если для элементов **j**, **k,** **l ∈ Ni**выполняются следующие неравенства:

**и** , то мы говорим, что происходит LP-доминирование(Linear Programming) элементов **j,l** над элементом **k**.

Также данное условие можно переписать так:

**(\*)**

В данном примере видим, что вес предмета 4 выше, чем у 1, а стоимость такая же. Значит предмет 1 доминирует над предметом 4. Касаемо объектов 1, 2 и 3, видим, что данные объекты удовлетворяют формуле (\*), следовательно, 1 и 3 LP-доминирует над 2 и мы его тоже не рассматриваем.

У LP-доминирования есть как +, так и -. Положительным является то, что мы уменьшаем количество рассматриваемых нами объектов в классе и тем самым ускоряем наш алгоритм. Плохим является то, что образуется небольшая погрешность в подсчётах(в частных случаях выгоднее будет выбрать именно элемент, который мы убрали из-за LP-доминации).

Для решения нашей задачи, необходимо упростить классы, то есть оставить в них только интересующие нас предметы. Концепция доминации будет выглядеть, как показано на рисунке 1.

Над объектами 3, 7 LP-доминируют объекты 2,4 и 4,9 соответственно. Над объектами 11, 10, 8, 6, 5 доминируют 3 и 7. Таким образом образуется своего рода границы выбора объектов.

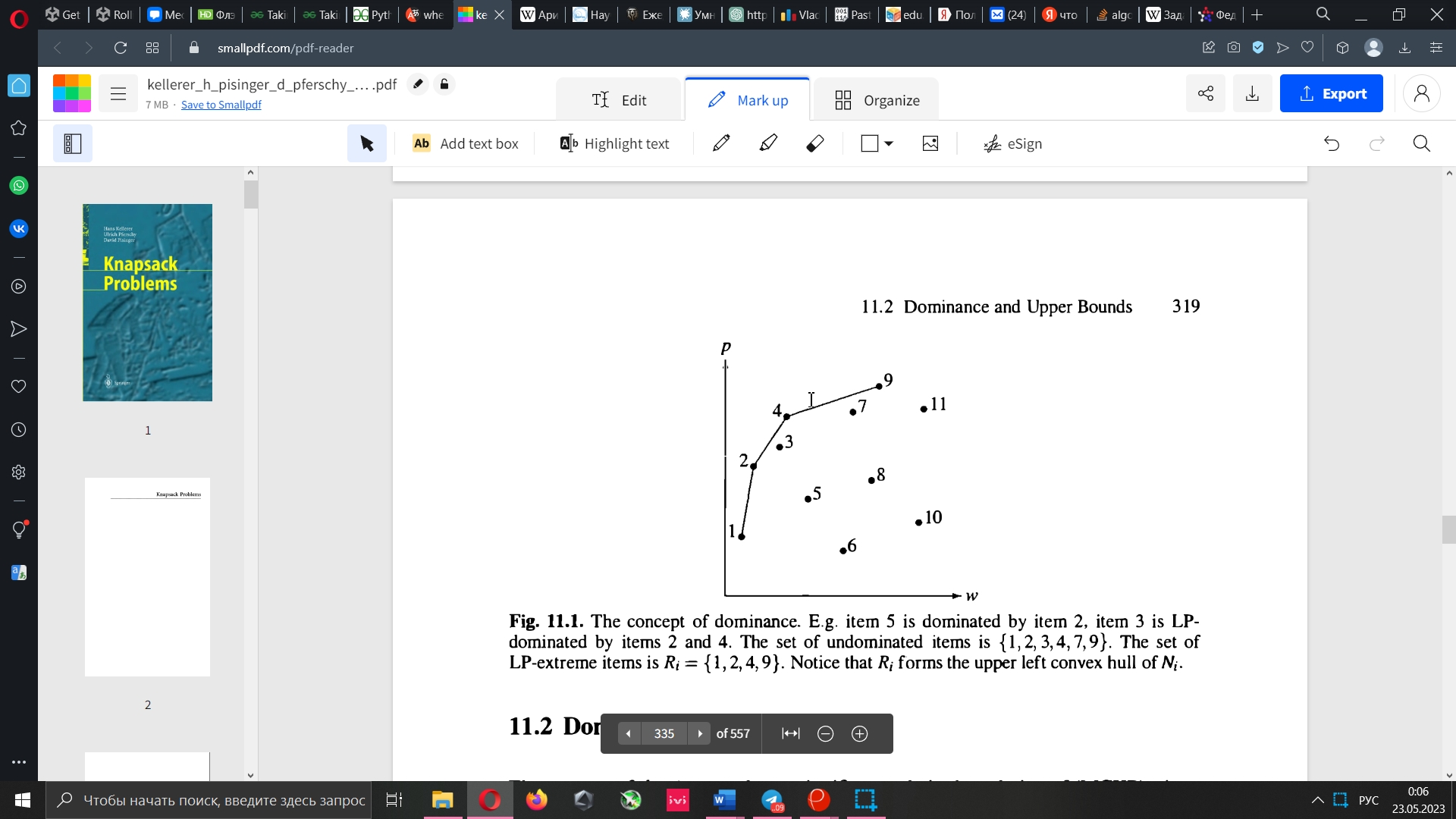


Рисунок 1. Графическое представление упрощения класса

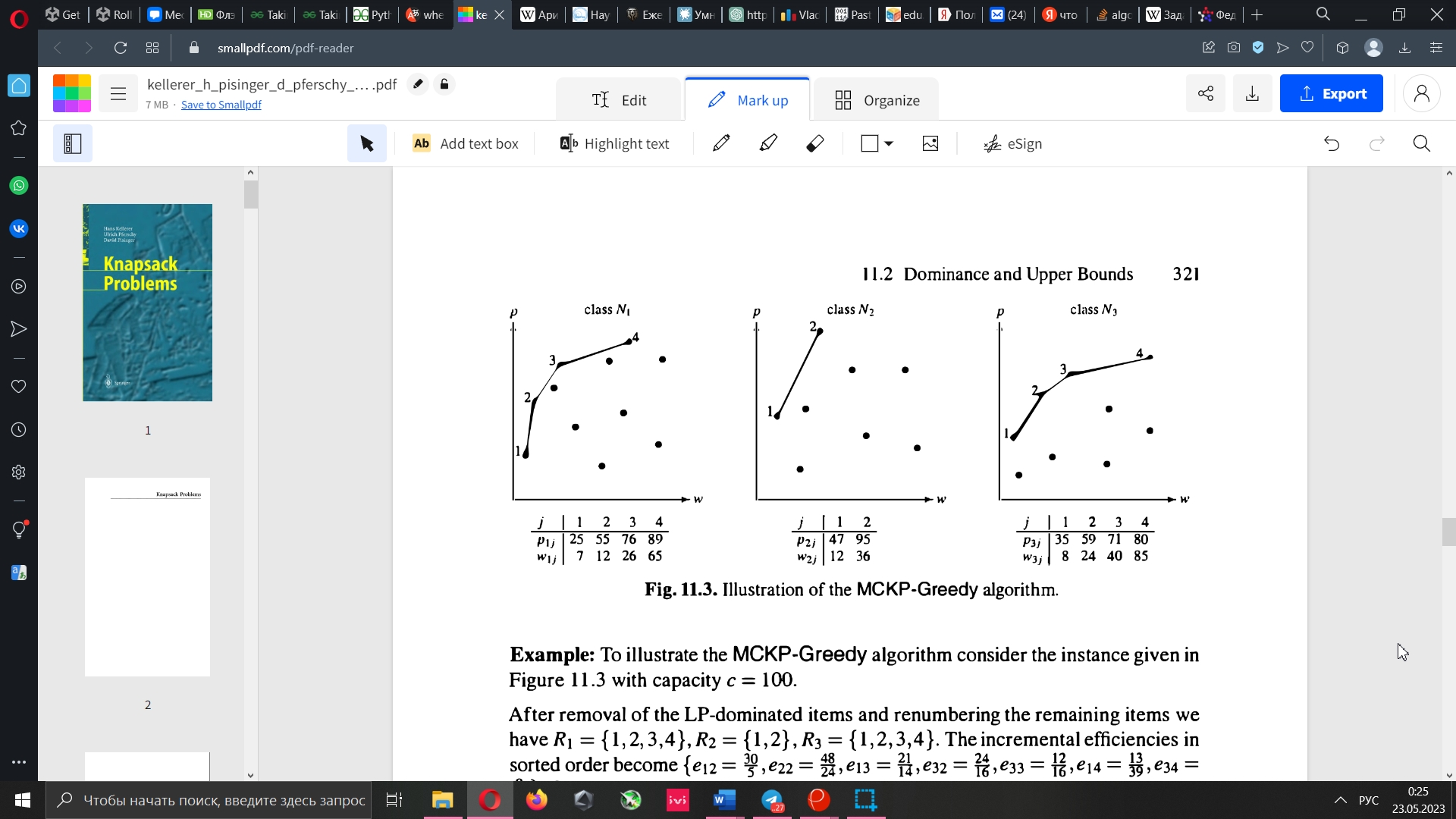
После упрощения классов и сортировки новых значений, на выходе получаем новые классы Ri. Для них введём понятие “эффективности” предмета.

**Определение:** Пусть дан объект класса **j ∈ Ri**. Эффективностью объекта **j** называется отношение улучшенной стоимости к улучшенному весу, т.е.

**, j>0;** **eij =0, j=0.**

Теперь наша задача состоит в сортировке объектов по убыванию эффективности и последующему их перебору(замена предмета в рюкзаке из класса **Ni** на предмет с эффективностью **eij**).

Пример работы упаковки рюкзака с входными данными в виде классов Ri.

****

Изначально в рюкзак с вместимостью **c = 100** помещаются предметы (25,7), (47,12), (35,8) из классов N1, N2 и N3 соответственно. Вместимость рюкзака становится **c’ =c – w11 – w21 – w32 = 73** Дальше оставшимся предметам присваиваются значения эффективности и сортируются по убыванию:

{**e12 = , e22 = , e13 = , e32 = , e33 = , e14 = , e34 =** }.

Жадный алгоритм сначала выбирает предмет (1,2) и уменьшает вместимость до 68, затем выбирает (2,2), вместимость теперь равна 44, (1,3), **с’=30**, и наконец предмет (3,2), **c’=14**. При попытке выбрать предмет (3,3) вместимость становится **c’=-2**, значит рюкзак переполнен и данный предмет мы пропускаем. Аналогично для предметов (1,4) и (3,4). Итоговая стоимость **z=76 + 95 + 59 = 230**.

1. **Итоговый алгоритм и верхние границы**

Получаем в итоге разновидность алгоритма “жадный алгоритм”.

Пошаговый жадный алгоритм для нашего случая:

1. Для каждого класса **Ni** делаем упрощение и получаем класс **Ri**. Сортируем объекты класса **Ri** по возрастающему весу.
2. Создать рюкзак и наполнить его самыми лёгкими предметами **eik** по одному из каждого класса Ri.
3. Отсортировать оставшиеся элементы по убыванию эффективности **eij = pij / wij**.
4. Поочерёдный перебор всех **eij**. При выборе **eij**(если это возможно) элемент **eik** заменяется на **eij**, ёмкость рюкзака увеличивается на (pij - **pik**).
5. Возвращаем ответ.

Общая сложность алгоритма составляет O( , где первый член берётся из шага 1, а второй член берётся из шага 3.

**Глава 2. Реализация.**

**Решение:**

Решать данную задачу будем на языке С++. В ходе решения использовалось авторское пространство имён Random с генератором случайных чисел, для заполнения групп различными объектами:

//Генерация случайных чисел

namespace Random

{

//Генератор

std::mt19937 mt{ std::random\_device{}() };

//Получение значения в диапазоне [min,max]

int get(int min, int max)

{

std::uniform\_int\_distribution die{ min, max };

return die(mt);

}

}

Листинг 1. То-то то-то

Создадим класс *Group* и добавим класс *Object* в качестве приватного члена класса *Group*. В классе *Object* полями являются вес, цена и последующие значения после преобразований, необходимые нам для дальнейшей упаковки рюкзака:

//Группы

class Group

{

//Объекты

class Object

{

public:

//Вес объекта

int weight;

//Стоимость объекта

int price;

//Оценка объекта, после чистки класса

double second\_value{};

//Конструктор по умолчанию

Object()

{

generator();

}

//Базовый конструктор

Object(pair<int, int> values)

{

weight = values.first;

price = values.second;

}

//Генератор предметов

void generator()

{

weight = Random::get(1, 1000);

price = Random::get(1, 1000);

}

//Оператор сравнения

bool operator >(const Object& x) const

{

return (this->weight <= x.weight) && (this->price >= x.price);

}

//Оператор равенства

bool operator ==(const Object& x) const

{

return (this->weight == x.weight) && (this->price == x.price);

}

//Оператор присваивания

Object& operator=(pair<int,int> values)

{

weight = values.first;

price = values.second;

return \*this;

}

void print() const

{

cout << price << ' ' << weight << ' ' << second\_value;

}

};

// Элементы нашей группы

vector<Object> elements;

// Нулевой элемент

Object null\_el{make\_pair(0,0)};

public:

//Конструктор по умолчанию

Group()

:elements(10)

{}

// Базовый конструктор

Group(const vector<pair<int, int>>& vec)

{

elements.resize(vec.size());

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

elements[i] = { vec[i].first,vec[i].second };

}

//Геттеры

//Количество элементов

int get\_count() const

{

return static\_cast<int>(elements.size());

}

//Вес элемента с индексом index

int get\_weight(const int index) const

{

return elements[index].weight;

}

//Цена элемента с индексом index

int get\_price(const int index) const

{

return elements[index].price;

}

//Новое значение элемента с индексом index

double get\_value(const int index) const

{

return elements[index].second\_value;

}

//Пара(вес,цена) элемента с индексом index

pair<int, int> get\_object(const int index) const

{

return make\_pair(elements[index].price, elements[index].weight);

}

//Убираем ненужные нам элементы

void reduction()

{

//Убираем все элементы,над которыми доминируют

for (int i = 0; i < elements.size() - 1; ++i)

{

for (int g = i + 1; g < elements.size(); ++g)

{

if (elements[i] > elements[g] && !(elements[g] == null\_el))

elements[g] = null\_el;

else if (elements[g] > elements[i] && !(elements[i] == null\_el))

elements[i] = null\_el;

}

}

clear();

sorting();

//Убираем элементы, над которыми имеется LP-доминация

if (elements.size() > 2)

{

for (int i = 0; i < elements.size()-2; ++i)

{

for (int g = i + 2; g < elements.size(); ++g)

{

for (int k = i + 1; k < g; ++k)

{

if (!(elements[i] == null\_el) && !(elements[g] == null\_el) && !(elements[k] == null\_el))

if (double(elements[g].price - elements[k].price) / double(elements[g].price - elements[k].price) >=

double(elements[k].price - elements[i].price) / double(elements[k].weight - elements[i].weight))

elements[k] = null\_el;

}

}

}

}

clear();

sorting();

// Теперь получаем новые значения в нашей группе

creating\_new\_values();

}

//Сортировка элементов группы по весу

void sorting()

{

sort(elements.begin(), elements.end(),[&](const auto x, const auto y)

{ return y.weight > x.weight; } );

}

//Очистка мусора из группы

void clear()

{

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

{

if (elements[i] == null\_el) {

elements.erase(elements.begin() + i);

--i;

}

}

}

//Создание новых значений, после чистки группы

void creating\_new\_values()

{

if (elements.size() < 2)

return;

for (int i = 1; i < elements.size(); ++i)

elements[i].second\_value = static\_cast<double>(elements[i].price - elements[i - 1].price) / static\_cast<double>(elements[i].weight - elements[i - 1].weight);

}

void print() const

{

cout << "\nGroup:\n";

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

elements[i].print();

cout << '\n';

}

}

};

Соответствующие сортировки группы по доминации и LP-доминации находятся в методе *reduction().* После его вызова, группа очищена и отсортирована, осталось придать новые значения элементам группы. Для этого воспользуемся методом *creating\_new\_values()*. Теперь наша группа полностью отсортирована и готова к упаковке. Повторим данные действия для остальных групп.

Теперь для решения нам понадобится класс *Knapsack*. Полем класса является его вместимость *weight* а также итоговая ценность рюкзака, которую мы будем подсчитывать после упаковки.

//Рюкзак

class Knapsack

{

//Вес рюкзака

int weight;

// Итоговая ценность рюкзака

long long get\_sum(const vector<pair<int, int>>& answer) const

{

long long ans{};

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

ans += answer[i].first;

return ans;

}

public:

//Конструктор по умолчанию

Knapsack(int \_weight = 1000)

{

weight = \_weight;

}

// Геттер

// Вес рюкзака

int get\_weight() const

{

return weight;

}

//Решение

void solve(vector<Group>& vec)

{

int curr\_weight = 0;

vector <pair<int, int>> answer(vec.size());

//Кладём минимальные по весу из каждой группы

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

{

answer[i] = vec[i].get\_object(0);

curr\_weight += vec[i].get\_weight(0);

}

//Вектор пара(новое значение, пара(номер группы,номер элемента в группе)

vector<pair<double, pair<int, int>>> it;

(vec, it);

//Если превысили — невозможно дать ответ

if (curr\_weight > weight) {

cout << "Ответ: невозможно\n";

return;

}

//Идеально уложились

else if (curr\_weight == weight)

{

sort(answer.begin(), answer.end(), [&](const auto x, const auto y)

{

if (x.first == y.first)

return x.second > y.second;

else return x.first < y.first;

});

print\_answer(answer,curr\_weight);

}

// Теперь будем брать самые лучшие значения среди новых элементов и поочердёно пихать их в рюкзак.

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

{

if (good\_choise(answer, it, i, vec,curr\_weight))

{

curr\_weight -= answer[it[i].second.second].second;

answer[it[i].second.first] = vec[it[i].second.first].get\_object(it[i].second.second);

curr\_weight += answer[it[i].second.first].second;

}

}

sort(answer.begin(), answer.end(), [&](const auto x, const auto y)

{

if (x.second == y.second)

return x.first < y.first;

else return x.second < y.second;

});

//Ответ

print\_answer(answer, curr\_weight);

}

// Проверяем, можем ли мы заменить элемент на новый

bool good\_choise(const vector < pair<int, int>> answer, const vector<pair<double,pair<int,int>>>& it,const int index, const vector<Group>& vec,

const int curr\_weight) const

{

int buff\_weight = curr\_weight;

buff\_weight -= answer[it[index].second.first].second;

buff\_weight += vec[it[index].second.first].get\_weight(it[index].second.second);

return (buff\_weight <= weight);

}

void print\_answer(const vector<pair<int, int>>& answer, const int curr\_weight) const

{

cout << "Ответ:\n(цена, вес)\n";

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

cout << answer[i].first << '\t' << answer[i].second << '\n';

cout << "Итого занято: " << curr\_weight << "\nЦенность:" << get\_sum(answer) << '\n';

}

};

Решением рюкзака является метод *solve()* класса *Knapsack*. Он принимает все наши группы, затем пихает в рюкзак по одному элементу из каждой группы самого маленького веса. Если нам не хватило вместимости рюкзака, значит задачу решить невозможно. Если мы уложились идеально, то выводим сразу ответ(потому что найти элемент получше в группе невозможно, из-за введённой нами доминацией и LP-доминацией). В противном случае введём новый вектор it, в который положим все наши объекты, которые мы можем засунуть на замену другим, уже находящимся в рюкзаке. Каждый элемент вектора будет представлен в виде ценности каждого предмета по отношению к предыдущему в группе, номеру группы и индексу данного элемента в его группе. Сделаем это с помощью функции *push\_new\_values():*

// Пихаем новые значения в вектор и сортируем

void push\_new\_values(const vector<Group>& vec, vector < pair<double,pair<int,int>>>& it)

{

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

{

if (vec[i].get\_count() < 2)

continue;

for (int g = 1; g < vec[i].get\_count(); ++g)

it.push\_back(make\_pair(vec[i].get\_value(1), make\_pair(i, g)));

}

//Сортируем по значениям

sort(it.begin(), it.end(), [&](const auto x, const auto y)

{

return x.first > y.first;

});

}

Теперь сортируем наш вектор и перебираем все значения. Если мы сможем вместить какой-то другой предмет, то заменяем его, иначе оставляем всё как есть. И выводим результат в методе print\_answer.

Полный код предоставлен в приложении 1.

**Тест:**

Теперь возьмём тестовую выборку данных, для убеждения, что наш алгоритм работает правильно. Данные представлены ниже в таблице 1.

Таблица 1. Тестовая выборка данных.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** |  |  |  |  |  |  |  |  |  |  |
| p | 666 | 527 | 716 | 148 | 659 | 626 | 255 | 94 | 510 | 351 |
| w | 839 | 300 | 630 | 708 | 547 | 411 | 68 | 389 | 272 | 727 |
| **2** |  |  |  |  |  |  |  |  |  |  |
| p | 189 | 7 | 89 | 461 | 431 | 539 | 960 | 242 | 67 | 910 |
| w | 736 | 448 | 150 | 778 | 484 | 831 | 523 | 668 | 480 | 495 |
| **3** |  |  |  |  |  |  |  |  |  |  |
| p | 68 | 722 | 917 | 417 | 99 | 442 | 482 | 772 | 25 | 125 |
| w | 173 | 628 | 265 | 421 | 952 | 379 | 168 | 335 | 646 | 236 |
| **4** |  |  |  |  |  |  |  |  |  |  |
| p | 479 | 915 | 984 | 922 | 638 | 813 | 927 | 985 | 821 | 839 |
| w | 841 | 782 | 395 | 614 | 723 | 634 | 589 | 791 | 624 | 36 |
| **5** |  |  |  |  |  |  |  |  |  |  |
| p | 805 | 640 | 521 | 730 | 823 | 30 | 461 | 391 | 894 | 141 |
| w | 10 | 833 | 336 | 172 | 904 | 44 | 686 | 579 | 793 | 921 |
| **6** |  |  |  |  |  |  |  |  |  |  |
| p | 640 | 341 | 227 | 499 | 671 | 302 | 927 | 246 | 955 | 515 |
| w | 802 | 62 | 798 | 622 | 625 | 900 | 789 | 196 | 4 | 119 |
| **7** |  |  |  |  |  |  |  |  |  |  |
| p | 269 | 461 | 913 | 851 | 153 | 482 | 897 | 850 | 175 | 671 |
| w | 439 | 222 | 656 | 82 | 611 | 918 | 858 | 755 | 98 | 59 |
| **8** |  |  |  |  |  |  |  |  |  |  |
| p | 404 | 988 | 333 | 9 | 453 | 142 | 257 | 855 | 372 | 337 |
| w | 1 | 582 | 221 | 528 | 839 | 982 | 341 | 353 | 385 | 563 |
| **9** |  |  |  |  |  |  |  |  |  |  |
| p | 947 | 577 | 502 | 934 | 118 | 693 | 619 | 157 | 55 | 597 |
| w | 828 | 991 | 566 | 101 | 611 | 601 | 685 | 462 | 540 | 686 |
| **10** |  |  |  |  |  |  |  |  |  |  |
| p | 351 | 899 | 241 | 663 | 494 | 78 | 53 | 743 | 966 | 132 |
| w | 551 | 816 | 381 | 527 | 269 | 13 | 213 | 346 | 26 | 142 |

Теперь перейдём непосредственно к нашей функции main:

Создаём вектор наших групп:

vector<Group> condition;

Считываем данные из файла “test.in”, записываем их поочерёдно в вектор пар и добавляем в вектор, пользуясь конструктором, принимающим вектор пар:

//Считываем

ifstream fin("test.in");

if (fin) //Елси файл открылся и всё ок

{

for (int index = 0; index < 10; ++index)

{

vector<pair<int, int>> input(10);

for (int i = 0; i < 10; ++i)

{

fin >> input[i].second >> input[i].first;

}

condition.push\_back({ input });

}

fin.close(); //Закрыть

}

Теперь вызываем метод упрощения группы для всех групп:

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

condition[i].reduction();

После упрощения наши группы теперь выглядят так:

Таблица 2. Объекты групп после их упрощения.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** |  |  |  |  | **6** |  |  |  |  |
| p | 255 | 510 | 716 |  | p | 955 |  |  |  |
| w | 68 | 272 | 630 |  | w | 4 |  |  |  |
| new | 0 | 1.25 | 0.575419 |  | new | 0 |  |  |  |
| **2** |  |  |  |  | **7** |  |  |  |  |
| p | 89 | 431 | 910 | 960 | p | 671 | 851 | 913 |  |
| w | 150 | 484 | 495 | 523 | w | 59 | 82 | 656 |  |
| new | 0 | 1.02395 | 43.5455 | 1.78571 | new | 0 | 7.82609 | 0.108014 |  |
| **3** |  |  |  |  | **8** |  |  |  |  |
| p | 482 | 917 |  |  | p | 404 | 855 | 988 |  |
| w | 168 | 265 |  |  | w | 1 | 353 | 582 |  |
| new | 0 | 4.48454 |  |  | new | 0 | 1.28125 | 0.580786 |  |
| **4** |  |  |  |  | **9** |  |  |  |  |
| p | 839 | 985 |  |  | p | 934 | 947 |  |  |
| w | 36 | 791 |  |  | w | 101 | 828 |  |  |
| new | 0 | 0.193377 |  |  | new | 0 | 0.0178817 |  |  |
| **5** |  |  |  |  | **10** |  |  |  |  |
| p | 805 | 894 |  |  | p | 78 | 966 |  |  |
| w | 10 | 793 |  |  | w | 13 | 26 |  |  |
| new | 0 | 0.113665 |  |  | new | 0 | 68.3077 |  |  |

Создаём объект класса knapsack и запускаем метод, решающий задачу:

Knapsack knapsack;

cout << "\nВес рюкзака:" << knapsack.get\_weight() << '\n';

knapsack.solve(condition);

Вывод программы:

Вес рюкзака:1000  
Ответ:  
(цена, вес)  
404 1  
955 4  
805 10  
966 26  
839 36  
255 68  
934 101  
89 150  
482 168  
913 656  
Итого занято: 906  
Ценность:6642

Рюкзак справился с нашей задачей отлично.

**Заключение**

Было предложено и реализовано решение задачи о рюкзаке с мультивыбором. Данный алгоритм является линейным(работает за O(n log n ) ), имеет небольшую погрешность и сравнительно высокую скорость, по сравнению с другими возможными решениями данной задачи. Данный алгоритм также можно немного улучшить, введя некоторые новые правила, а также расширить и использовать в решении более сложных задач(например того же рюкзака, но уже с отрицательными значениями).

**Приложения**

Приложение 1. Полный код на языке С++ для решения задачи о рюкзаке с мультивыбором.

#include <iostream>

#include <vector>

#include <random>

#include <utility>

using namespace std;

//Генерация случайных чисел

namespace Random

{

//Генератор

std::mt19937 mt{ std::random\_device{}() };

//Получение значения в диапазоне [min,max]

int get(int min, int max)

{

std::uniform\_int\_distribution die{ min, max };

return die(mt);

}

}

//Группы

class Group

{

//Объекты

class Object

{

public:

//Вес объекта

int weight;

//Стоимость объекта

int price;

//Оценка объекта, после чистки класса

double second\_value{};

//Конструктор по умолчанию

Object()

{

generator();

}

//Базовый конструктор

Object(pair<int, int> values)

{

weight = values.first;

price = values.second;

}

//Генератор предметов

void generator()

{

weight = Random::get(1, 1000);

price = Random::get(1, 1000);

}

//Оператор сравнения

bool operator >(const Object& x) const

{

return (this->weight <= x.weight) && (this->price >= x.price);

}

//Оператор равенства

bool operator ==(const Object& x) const

{

return (this->weight == x.weight) && (this->price == x.price);

}

//Оператор присваивания

Object& operator=(pair<int,int> values)

{

weight = values.first;

price = values.second;

return \*this;

}

void print() const

{

cout << price << ' ' << weight << ' ' << second\_value;

}

};

// Элементы нашей группы

vector<Object> elements;

// Нулевой элемент

Object null\_el{make\_pair(0,0)};

public:

//Конструктор по умолчанию

Group()

:elements(10)

{}

// Базовый конструктор

Group(const vector<pair<int, int>>& vec)

{

elements.resize(vec.size());

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

elements[i] = { vec[i].first,vec[i].second };

}

//Геттеры

//Количество элементов

int get\_count() const

{

return static\_cast<int>(elements.size());

}

//Вес элемента с индексом index

int get\_weight(const int index) const

{

return elements[index].weight;

}

//Цена элемента с индексом index

int get\_price(const int index) const

{

return elements[index].price;

}

//Новое значение элемента с индексом index

double get\_value(const int index) const

{

return elements[index].second\_value;

}

//Пара(вес,цена) элемента с индексом index

pair<int, int> get\_object(const int index) const

{

return make\_pair(elements[index].price, elements[index].weight);

}

//Убираем ненужные нам элементы

void reduction()

{

//Убираем все элементы,над которыми доминируют

for (int i = 0; i < elements.size() - 1; ++i)

{

for (int g = i + 1; g < elements.size(); ++g)

{

if (elements[i] > elements[g] && !(elements[g] == null\_el))

elements[g] = null\_el;

else if (elements[g] > elements[i] && !(elements[i] == null\_el))

elements[i] = null\_el;

}

}

clear();

sorting();

//Убираем элементы, над которыми имеется LP-доминация

if (elements.size() > 2)

{

for (int i = 0; i < elements.size()-2; ++i)

{

for (int g = i + 2; g < elements.size(); ++g)

{

for (int k = i + 1; k < g; ++k)

{

if (!(elements[i] == null\_el) && !(elements[g] == null\_el) && !(elements[k] == null\_el))

if (double(elements[g].price - elements[k].price) / double(elements[g].price - elements[k].price) >=

double(elements[k].price - elements[i].price) / double(elements[k].weight - elements[i].weight))

elements[k] = null\_el;

}

}

}

}

clear();

sorting();

// Теперь получаем новые значения в нашей группе

creating\_new\_values();

}

//Сортировка элементов группы по весу

void sorting()

{

sort(elements.begin(), elements.end(),[&](const auto x, const auto y)

{ return y.weight > x.weight; } );

}

//Очистка мусора из группы

void clear()

{

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

{

if (elements[i] == null\_el) {

elements.erase(elements.begin() + i);

--i;

}

}

}

//Создание новых значений, после чистки группы

void creating\_new\_values()

{

if (elements.size() < 2)

return;

for (int i = 1; i < elements.size(); ++i)

elements[i].second\_value = static\_cast<double>(elements[i].price - elements[i - 1].price) / static\_cast<double>(elements[i].weight - elements[i - 1].weight);

}

void print() const

{

cout << "\nGroup:\n";

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

elements[i].print();

cout << '\n';

}

}

};

// Пихаем новые значения в вектор и сортируем

void push\_new\_values(const vector<Group>& vec, vector < pair<double,pair<int,int>>>& it)

{

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

{

if (vec[i].get\_count() < 2)

continue;

for (int g = 1; g < vec[i].get\_count(); ++g)

it.push\_back(make\_pair(vec[i].get\_value(1), make\_pair(i, g)));

}

//Сортируем по значениям

sort(it.begin(), it.end(), [&](const auto x, const auto y)

{

return x.first > y.first;

});

}

//Рюкзак

class Knapsack

{

//Вес рюкзака

int weight;

// Итоговая ценность рюкзака

long long get\_sum(const vector<pair<int, int>>& answer) const

{

long long ans{};

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

ans += answer[i].first;

return ans;

}

public:

//Конструктор по умолчанию

Knapsack(int \_weight = 1000)

{

weight = \_weight;

}

// Геттер

// Вес рюкзака

int get\_weight() const

{

return weight;

}

//Решение

void solve(vector<Group>& vec)

{

int curr\_weight = 0;

vector <pair<int, int>> answer(vec.size());

//Кладём минимальные по весу из каждой группы

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

{

answer[i] = vec[i].get\_object(0);

curr\_weight += vec[i].get\_weight(0);

}

//Вектор пара(новое значение, пара(номер группы,номер элемента в группе)

vector<pair<double, pair<int, int>>> it;

push\_new\_values(vec, it);

//Если превысили — невозможно дать ответ

if (curr\_weight > weight) {

cout << "Ответ: невозможно\n";

return;

}

//Идеально уложились

else if (curr\_weight == weight)

{

sort(answer.begin(), answer.end(), [&](const auto x, const auto y)

{

if (x.first == y.first)

return x.second > y.second;

else return x.first < y.first;

});

print\_answer(answer,curr\_weight);

}

// Теперь будем брать самые лучшие значения среди новых элементов и поочердёно пихать их в рюкзак.

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

{

if (good\_choise(answer, it, i, vec,curr\_weight))

{

curr\_weight -= answer[it[i].second.second].second;

answer[it[i].second.first] = vec[it[i].second.first].get\_object(it[i].second.second);

curr\_weight += answer[it[i].second.first].second;

}

}

sort(answer.begin(), answer.end(), [&](const auto x, const auto y)

{

if (x.second == y.second)

return x.first < y.first;

else return x.second < y.second;

});

//Ответ

print\_answer(answer, curr\_weight);

}

// Проверяем, можем ли мы заменить элемент на новый

bool good\_choise(const vector < pair<int, int>> answer, const vector<pair<double,pair<int,int>>>& it,const int index, const vector<Group>& vec,

const int curr\_weight) const

{

int buff\_weight = curr\_weight;

buff\_weight -= answer[it[index].second.first].second;

buff\_weight += vec[it[index].second.first].get\_weight(it[index].second.second);

return (buff\_weight <= weight);

}

void print\_answer(const vector<pair<int, int>>& answer, const int curr\_weight) const

{

cout << "Ответ:\n(цена, вес)\n";

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

cout << answer[i].first << '\t' << answer[i].second << '\n';

cout << "Итого занято: " << curr\_weight << "\nЦенность:" << get\_sum(answer) << '\n';

}

};

int main() {

setlocale(LC\_ALL, "rus");

vector<Group> condition(10);

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

{

condition[i].print();

condition[i].reduction();

condition[i].print();

}

Knapsack;

cout << "\nВес рюкзака:" << knapsack.get\_weight() << '\n';

knapsack.solve(condition);

return 0;

}

**Список литературы**

1. Kellerer H., Pferschy U., Pisinger D. Knapsack Problems — Springer Science+Business Media, 2004. P.317
2. https://en.wikipedia.org/wiki/Knapsack\_problem