|  |  |
| --- | --- |
| **TECHNICKÁ UNIVERZITA V KOŠICIACH**  **FAKULTA ELEKTROTECHNIKY A INFORMATIKY** | |
| HOP-Zadanie 2  Optimalizácia úradu | |
|  | |
| **2024** | **Tobiáš Jutka,Oliver Frič,Adam Broško,Soňa Korchňáková** |

Obsah

[1. Analýza problému 3](#_Toc184555629)

[Typ optimalizačného problému: 3](#_Toc184555630)

[1.1. Možné riešenia: 3](#_Toc184555631)

[1.1.1. Riešenie1 3](#_Toc184555632)

[1.1.2. Riešenie2 3](#_Toc184555633)

[1.1.3. Riešenie 3 4](#_Toc184555634)

[1.2. Komponenty: 4](#_Toc184555635)

[1.3. Skladba kandidáta: 4](#_Toc184555636)

[1.4. Podmienky pri optimalizácii: 5](#_Toc184555637)

[1.5. kvalita riešenia závisí od 5](#_Toc184555638)

[2. Návrh modelu problému 6](#_Toc184555639)

[2.1. Model 1 6](#_Toc184555640)

[2.2. Model2 7](#_Toc184555641)

[3. Programová implementácia 9](#_Toc184555642)

[3.1. Nastavenie prostredia 9](#_Toc184555643)

[3.2. Popis programovej implementácie: 9](#_Toc184555644)

[3.2.1. Vstup 9](#_Toc184555645)

[3.2.2. Výstup programu: 9](#_Toc184555646)

[3.2.3. Kód programu: 9](#_Toc184555647)

1. Analýza problému

Úlohou je optimalizovať rozdelenie rôznych typov žiadostí medzi viacerých zamestnancov s cieľom minimalizovať celkový čas spracovania všetkých žiadostí(najpomalší zamestnanec má čo najmenší čas) a zároveň minimalizovať náklady na mzdy. Každý zamestnanec má rôznu rýchlosť spracovania jednotlivých typov žiadostí, pričom opakované spracovanie rovnakého typu žiadosti umožňuje zvýšiť efektivitu.

Typ optimalizačného problému: kombinatorický minimalizačný problém- minimal makespan problem

* 1. Možné riešenia:
     1. Riešenie1

R1a. - Dá sa použiť veľmi upravený TSP problém a to tak, že dokumenty budú ako keby mestá a úradníci budú náš traveling salesman person- každému úradníkovi dáme štartovací bod , dokument ktorý vie urobiť najrýchlejšie zo všetkých dokumentov a vzdialenosť medzi mestami bude iná pre každého úradníka a to jeho rýchlosť pri robení daného dokumentu. Jasne budeme sledovať či už je dokument navštívený úradníkom potom ten dokument už nebude prístupný pre žiadneho úradníka.

R1b. - Môžeme medzi vybratých úradníkov rozdeliť dokumenty náhodne a malými zmenami ako napr. zmenením úradníka pre jednotlivý dokument sa dostať k optimálnemu riešeniu – Iteračné vylepšovanie.

R1(bonus). - Pri oboch týchto riešeniach ešte môžeme skontrolovať či nejaký úradník neprechádza tesne nad hodinu a mohli by sme ten dokument dať niekomu inému, kto napr. svoju hodinu práve začal, túto kontrolu môžeme robiť či už po rozdelení dokumentov alebo počas rozdeľovania.

* + 1. Riešenie2

.Mohli by sme to riešiť zmiešaným celočíselným programovaním kde by sme si definovali minimalizačnú funkciu a tá by nám definovala optimálny výsledok, ale to by bolo dosť neefektívne časovo náročné.

xij = 1 ak účtovník i roby dokument j inak 0

tij = rýchlosť zamestnanca i na dokumente j

tkij = skrátený čas o 5%pri opakovanom spracovaní

Jasne ešte obe časti ohodnocovanej funkcie vynásobí potrebnou váhou ktorú si určíme.

* + 1. Riešenie 3

Simulované žíhanie – kde by sme mohli každému dokumentu priradili náhodného úradníka a ako zmenu vymeníme nejakému dokumentu úradníka a zase vypočítame našu ohodnocovaniu funkciu a pomocou toho sa rozhodneme sa či zmeníme výsledok.(úprimne nie som si istý ako sa to popasuje s tým že platíme za každú hodinu ale to zistime až v programe :)

R**4** Generačné algoritmy – kde by sme si vytvorili populáciu z napr. z 10 rôznych kandidátov vypočítali by sme si výsledok ohodnocovanou funkciou vybrali 2 z nich asi jedných z najlepších mohli by sme tam pridať ešte nejakú mutáciu a pridáme ich do populácie.

* 1. Komponenty:

Úradníci

Žiadosti

Efektivita pri opakovaných formulároch

Mzdy za hodinu

* 1. Skladba kandidáta:

Kandidátom je konkrétne priradenie všetkých žiadostí k úradníkom.

(z toho vieme vyčítať celkovú sumu koľko zaplatíme aj najpomalšieho úradníka)

* 1. Podmienky pri optimalizácii:

1. Každá žiadosť musí byť pridelená práve jednému úradníkovi.
2. Efektivita opakovaného spracovania rovnakého typu žiadosti musí byť zohľadnená.
3. Náklady na prácu úradníkov musia byť minimalizované.
4. Najpomalší úradník musí mať čo najkratší čas
   1. kvalita riešenia závisí od
5. Minimálneho celkového času na spracovania všetkých dokumentov.
6. Času práce najpomalšieho úradníka
7. Minimálnych celkových mzdových nákladov.

Kde pri ohodnocovacej funkii môžeme meniť váhy podľa potreby našej optimalizáciie. My prcujeme s váhami 1x čas najpomalšieho pracovníka + 0,1x celkový čas preto aby sa priradili dokumenty dobrým pracovníkom ale nezhoršil sa največší čas + 10x počet hodín všetkých pracovníkov.

1. Návrh modelu problému

Popíšem tu 2 modely

* 1. Model 1

Účtovníka by sme reprezentovali objektom typu uct:

Ktorý by mal parametre

* Dict dokumenty = {document: čas}
* Int ID = poradie vo vstupe
* List dokuments = [zatial spravené dokumenty]
* Int value = hodnota pre inštanciu

Dokument by sme reprezentovali objektom typu doc:

Ktorý by mal parametre

* úradník inštanciu typu účtovník – alebo null pokiaľ neje nikomu zatiaľ priradený

V prvom rade by sme si ohodnotili každého úradníka ohodnocovanou funkciou ktorá zodpovedá tomu ako rýchly sú v daných dokumentoch. A odstránime tých ktorý sú pomalý alebo nespĺňajú isté podmienky.

Týmto si vieme zmenšiť počet účtovníkov keďže odstránime tých ktorý nám nijako nemôžu pomôcť s prácou.

* **Inicializácia:**
  + Nastav aktuálne riešenie s na náhodné riešenie pomocou funkcie [R1a](#R1a)
* **Iteratívna časť:**
  + **While:**
    - Dokiaľ nie sme v lokálnom optime (čo by mohlo trvať v niektorých prípadoch veľmi dlho )alebo nie je dosiahnutý určitý počet iterácii(môžeme nastaviť napr.celkový počet formulárov\*počet dokumentov\*2)
  + **Výber lepšieho suseda:**
    - Vymeníme 2 úradníkov.
    - Predtým než robíme ohodnocovaniu funkciu u každého úradníka zoskupíme dokumenty aby sme využili efektivitu práce pri robení viac rovnakých dokumentov.
    - Pokiaľ ohodnocovania funkcia je nižšia tak ho vyberieme inak nie.
  1. Model2

Druhý model je vyriešenie pomocou simulovaného žíhania.

Každý kandidát bude predstavovať iné možné rozdelenie úloh medzi zamestnancov.

**V programe budeme mať na vyriešenie tieto Triedy:**

Zamestnanec - bude obsahovať id zamestnanca (parameter pri vytváraní objektu)a dictionary formulárov kde klúč bude názov formulára a hodnota bude rýchlosť s akou náš zamestnanec sa vie s daným formulárom vysporiadať

pričom obsahuje metódy na výpočet celkového času spracovania formulárov ,generovanie zoznamu formulárov na spracovanie a manipuláciu s počtom formulárov (`plus\_doc`, `minus\_doc`).

Job- má vlastnosti názov a pravedpodobnosť vybratia formulára zamestancom

Solution-má paramater list zamestnancov,v danej triede vymieňame prácu zamestnancov a evalujeme list zamestnancov

Asigment2 – v nej budeme simulovať ich prácu,pracovať s objektmi Zamestnanec,Job a Solution zabezpečovať načítavanie zo vstupu csv,simulovať a vyhodnocovať možné riešenia až kým neklessne tepolota pod nami určenú hranicu a to bude naše riešenie

**Inicializácia**Naše prvotné riešenie dostaneme tak ze rozdelíme zamestnancom dokumenty tak ze každý dokument dostane pravdepodobnost ku ktorému zamestnancovy ho priradíme a podľa nej sa už priradí.

**Vyhodnocovacia funkcia:**

f(xcurrent​) = λ⋅ (čas všetkých zamestnancov) + *+* λ*\* čas najpomalsieho zametnanca*

kde:

λ - váha pre danú časť

mzda všetkých zamestnacov - súčet mzdy každého zamestnanca

výpočet mzdy jedného zamestnanca - jeho celkový čas (zaokrúhlený na celé hodiny )\* mzda za hodinu(v našom prípade 30 eur)

čas všetkých zamestnancov - súčet času každého zamestnanca

čas jedného zamestnanca - podľa toho koľko mu trval jeden formulár (kde je zarátané aj

poprípade skrátenie času v prípade viacerých formulárov)\*počet kusov daného formulára,kt. spracoval

**Generovanie nového stavu**:

* Vytvoríme nový stav xnew​ miernou zmenou aktuálneho priradenia. Môže to byť napríklad výmena pridelenia niektorej žiadosti medzi dvoma zamestnancami.

Následne vypočítame f(xnew​) pomocou toho istého vzorca ako f(xcurrent​)

a následne aplikujeme kontrolu spĺňania podmienok:

**Kontrola spĺňania podmienok:**

Spočíva v porovnaní výsledkov predchádzajúcej a aktuálnej vyhodnocovacej funkcie

Ak 𝑓(𝑥new)<𝑓(𝑥current), nový stav automaticky prijímame, pretože zlepšuje hodnotu cieľovej funkcie následne aktualizujeme Node a uložíme aktuálny výsledok.

Ak 𝑓(𝑥new)>𝑓(𝑥current) tak už podľa momentálnej teploty a náhodnosti sa nový stav príjme alebo nie.

1. Programová implementácia
   1. Nastavenie prostredia

v našom riešení máme implementované tieto knižnice :

json,csv,math,random,copy  
Tento kód bol robený na verzii python 3.11.9 a s touto verziou odporúčame pracovať

Pokyny pre spustenie programu aj celý program nájdete na: <https://github.com/tobotobitobo/HOP-As2?tab=readme-ov-file>

pre spustenie programu použite command:

python .\Asigment2.py

* 1. Popis programovej implementácie:
     1. Vstup

**Na vstupe máme : 2 json súbory**

**1.obsahuje zamestnacov a jeho rýchlosti pre daný formulár napr.takto vyzeral ukážkový vstup**

**Obrázok, na ktorom je text, snímka obrazovky, písmo, číslo

Automaticky generovaný popis**

**2.obsahuje názvy formulárov a ich celkový počet napr.takto vyzeral ukážkový vstup**

**Obrázok, na ktorom je text, písmo, snímka obrazovky, dizajn

Automaticky generovaný popis**

* + 1. Výstup programu:

Obsah csv:pričom prvé je id úradníka a potom zoznam formulárov ,ktoré vybavoval

0,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451  
1,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843  
4,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843  
5,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,IRV-843  
6,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451  
7,OPT-451,OPT-451,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843,IRV-843  
11,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989  
13,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989,CLI-989  
18,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451,OPT-451

**Výpis v konzole**

{'zam\_id': '0', 'zamestnanec': {'CLI-989': 4, 'OPT-451': 4, 'EBC-794': 2}, 'celkovy cas': 34.99999999999999}

{'zam\_id': '1', 'zamestnanec': {'IRV-843': 2, 'HCG-289': 3, 'AIS-362': 4}, 'celkovy cas': 35.49999999999999}

{'zam\_id': '4', 'zamestnanec': {'IRV-843': 4, 'OOR-173': 2, 'IXL-672': 4, 'ALZ-178': 4, 'IME-920': 2}, 'celkovy cas': 34.99999999999999}

{'zam\_id': '5', 'zamestnanec': {'IXL-672': 3, 'AIS-362': 4, 'MRW-171': 3, 'CLI-989': 3, 'YNA-597': 2}, 'celkovy cas': 34.24999999999999}

{'zam\_id': '6', 'zamestnanec': {'IME-920': 3, 'OPT-451': 4, 'OOR-173': 2, 'EBC-794': 3}, 'celkovy cas': 34.99999999999999}

{'zam\_id': '7', 'zamestnanec': {'FIO-135': 2, 'SJJ-954': 4, 'EBC-794': 2, 'OPT-451': 3, 'IRV-843': 2}, 'celkovy cas': 35.349999999999994}

{'zam\_id': '11', 'zamestnanec': {'CLI-989': 2, 'IXL-672': 4, 'SJJ-954': 3, 'YOV-950': 4, 'YNA-597': 2}, 'celkovy cas': 35.49999999999999}

{'zam\_id': '13', 'zamestnanec': {'AIS-362': 2, 'YNA-597': 2, 'CLI-989': 3, 'OOR-173': 3, 'EBC-794': 3}, 'celkovy cas': 30.749999999999993}

{'zam\_id': '18', 'zamestnanec': {'OPT-451': 4, 'EBC-794': 4, 'FIO-135': 2, 'AIS-362': 3, 'CLI-989': 2}, 'celkovy cas': 34.99999999999999}

zaplatil si 270

najpomalsi cas je 35.49999999999999

celkovy cas je311.34999999999997

* + 1. Kód programu:

**Trieda Zamestnanec(predstavuje jedného nášho úradníka)**

import json  
import random  
  
class Zamestnanec:  
 *#v konštrukture načítavame vstupy-zamestnacov a formulárov* def \_\_init\_\_(self,ID):  
 self.dictdoc = {}  
 inputformulare="formulare\_todo.json"  
 try:  
 with open(inputformulare) as json1:  
 formulate\_json = json1.read()  
 formulate\_todo = json.loads(formulate\_json)  
 for entry,value in formulate\_todo.items():  
 self.dictdoc[entry] = 0  
 except FileNotFoundError:  
 print(f"Súbor {inputformulare} nebol nájdený.")  
 self.ID = ID  
  
 inputzamestanci="possible\_zamestnanci.json"  
 try:  
 with open(inputzamestanci) as json1:  
 possible\_zamestnanci\_json = json1.read()  
 possible\_zamestnanci = json.loads(possible\_zamestnanci\_json)  
 except FileNotFoundError:  
 print(f"Súbor {inputzamestanci} nebol nájdený.")  
 self.speed = possible\_zamestnanci[ID]  
 self.celkovy\_cas = 0  
  
 def getspeed(self):  
 return self.speed  
   
 *#vracia list názvov formulárov* def listofnames(self):  
 names = []  
 for entry,value in self.dictdoc.items():  
 for i in range(0,value):  
 names.append(entry)  
 return names  
  
 def gettotalspeed(self):  
 *#táto metóda nám vyrátava rýchlosť spracovania žiadosti  
 #buď sa berie rýchlosť z súboru,alebo je defaultne nastavená na 5  
 #taktiež sa do nášho výpočtu zarátava efektívnosť,ak úradník má po sebe  
 #spracovávať formuláre rovnakého typu, dokáže ešte viac zefektívniť proces, a to o 5% pri každom následnom spracovaní* efficiency = 1  
 self.celkovy\_cas = 0  
 for doc, value in self.dictdoc.items():  
 efficiency = 1  
 for i in range(0,value):  
 if(i != 0):  
 efficiency -= 0.05 if efficiency>0.5 else 0  
 if doc in self.speed:  
 self.celkovy\_cas += self.speed[doc] \* efficiency  
 else:  
 self.celkovy\_cas += 5 \* efficiency  
 return self.celkovy\_cas  
  
 def getRandomKey(self):  
 *#vybratie náhodného formulára* return random.choice([job for job, count in self.dictdoc.items() if count > 0])  
   
 def sum\_of\_docs(self):  
 *#vráti celkový počet formulárov na spracovanie* a = 0  
 for doc, value in self.dictdoc.items():  
 a += value  
 return a  
  
 *#metóda plus\_doc pridáva danému zamestnancovi formulár  
 #metóda minus\_doc ju naopak odoberá* def plus\_doc(self,keyname):  
 self.dictdoc[keyname] += 1  
 def minus\_doc(self,keyname):  
 self.dictdoc[keyname] -= 1

**Trieda Job**(predstavuje náš dokument)  
class Job:  
 def \_\_init\_\_(self, name):  
 self.name = name  
 self.propability = {}  
 self.totalpropability = 0  
   
 def nameofdoc(self):  
 return self.name  
   
 def getPropability(self):  
 return self.propability  
   
  
 def setpropability(self,zamestnanci):  
 *#vypočítame ako prispeje zamestnanec na tento dokument* for zam in zamestnanci:  
 self.propability[zam] = 5 - zam.getspeed()[self.nameofdoc()] if self.nameofdoc() in zam.getspeed() else 0.0001  
 *#pokiaľ nevie vôbec robiť ten dokument, dáme mu minimálnu pravdepodobnosť kvôli tomu, žeby vždy tam bola nejaká pravdepodobnosť* self.totalpropability += self.propability[zam]  
   
 *#prechádza zamestnancov a každému dá pravdepodobnosť, čím rýchlejší úradník, tým väčšia pravdepodobnosť* for zam in zamestnanci:  
 self.propability[zam] = self.propability[zam] / self.totalpropability

**Trieda Solution**-predstavuje jedno naše možné riešenie

from zamestnanec import Zamestnanec  
import random  
import math  
  
class Solution:  
 def \_\_init\_\_(self, zamestnanci):  
 self.zamestnanci = zamestnanci  
  
 def get\_zamestnanci(self):  
 return self.zamestnanci  
   
 def get\_neighbour(self):  
 *# v tejto metóde najprv vyberieme náhodneho zamestanca  
 #pozrieme či spracovával nejaký dokument  
 #tak sa vyberie druhy náhodny zamestnanec  
 # prvému sa odoberie dokument a druhému sa pridá* targeted\_zamestnanec = random.choice(self.zamestnanci)  
 if targeted\_zamestnanec.sum\_of\_docs() != 0:  
 job\_to\_switch = targeted\_zamestnanec.getRandomKey()  
 targeted\_zamestnanec.minus\_doc(job\_to\_switch)  
 *#zabezpečenie aby sa sa nevygeneroval ten istý zamestanec/úradník* targeted\_zamestnanec2 = random.choice(self.zamestnanci)  
 while(targeted\_zamestnanec==targeted\_zamestnanec2):  
 targeted\_zamestnanec2 = random.choice(self.zamestnanci)  
 if targeted\_zamestnanec!=targeted\_zamestnanec2:  
 targeted\_zamestnanec2.plus\_doc(job\_to\_switch)  
 return self  
   
 def evaluatefromlist(self,slowest\_weight,totaltime\_weight,hours\_weight):  
 *#táto funkcia vyhodnocuje efektivitu skupiny zamestnancov na základe času* slowest = 0  
 num\_of\_hours = 0  
 celkovy\_cas = 0  
  
 for zamestnanec in self.zamestnanci:  
 cas = zamestnanec.gettotalspeed()  
 if cas > slowest:  
 slowest = cas  
 celkovy\_cas += cas  
 num\_of\_hours += math.ceil(cas/60)  
  
 n = 0  
 for a in self.zamestnanci:  
 if (a.sum\_of\_docs() > 0):  
 n+=1  
  
 return slowest\*slowest\_weight + celkovy\_cas/n\*totaltime\_weight +num\_of\_hours\*n\*hours\_weight

**Asigment2-hlavná trieda**

import json  
import csv  
import math  
from zamestnanec import Zamestnanec  
from job import Job  
from solution import Solution  
import random  
import copy  
  
  
def evaluate(filename):  
 *#vďaka tejto metóde okrem csv outputu vypíšeme info k nášmu riešeniu do konzoly* with open(filename) as csvfile:  
 output\_to\_check = csv.reader(csvfile)  
 slowest = 0  
 num\_of\_hours = 0  
 cas = 0  
  
 for output\_line in output\_to\_check:  
 tipek = {"zam\_id": output\_line[0],  
 "zamestnanec": possible\_zamestnanci[int(output\_line[0])],  
 "celkovy cas": 0}  
   
 *# efficiency = 1* for i, formular in enumerate(output\_line[1:], 1):  
 if output\_line[i-1] == formular:  
 efficiency -= 0.05 if efficiency>0.5 else 0  
 else:  
 efficiency = 1  
 *# vzdy nastavi efficiency na 1 na zaciatku formularu  
 # lebo output\_line[0] nikdy nebude same ako formular* if formular in tipek['zamestnanec']:  
 tipek['celkovy cas'] += tipek['zamestnanec'][formular] \* efficiency  
 else:  
 tipek['celkovy cas'] += 5 \* efficiency  
  
 print(tipek)  
 cas += tipek['celkovy cas']  
 if tipek['celkovy cas'] > slowest:  
 slowest = tipek['celkovy cas']  
  
 num\_of\_hours += math.ceil(tipek['celkovy cas']/60)  
 print("")  
 print("zaplatil si " + str(num\_of\_hours \* 30))  
 print("najpomalsi cas je " + str(slowest) + " ")  
 print("celkovy cas je" + str(cas) + " ")  
  
 return num\_of\_hours\*30, slowest, cas  
  
  
  
def selectWorkers(possible\_zamestnanci):  
 *#vyberie vhodných zamestancov a vracia ich ako list* zamesnanci = []  
 for i, entry in enumerate(possible\_zamestnanci):  
 zamesnanci.append(Zamestnanec(i))  
 if entry and any(key in formulare\_todo for key in entry):  
 zamesnanci[i].usefull = True  
  
 return zamesnanci  
  
def notRandomSelect(formulare\_todo,possible\_zamestnanci):  
 *#vyberú sa zamestnanci* zamesnanci = selectWorkers(possible\_zamestnanci)  
 joblist = []  
 *#inicializácia každého jobu  
 #+ vytvorenie pravdepodobností s ktorými sa budú dávať zamestnancom* for entry,value in formulare\_todo.items():  
 for i in range(0,value):  
 job = Job(entry)  
 job.setpropability(zamesnanci)  
 joblist.append(job)  
  
 *#s danou pravdepodobnosťou sa dá job zamestnancovi* for job in joblist:  
 weight = list(job.getPropability().values())  
 zam = random.choices(zamesnanci,weights=weight, k=1)[0]  
 zam.dictdoc[job.nameofdoc()] += 1  
  
 return zamesnanci  
  
  
def writedoc(zamestnanci,filename):  
 *#zápis výstupu do csv súboru* try:  
 with open(filename, 'w', newline='') as csvfile:  
 writer = csv.writer(csvfile)  
 for zamestnanec in zamestnanci:  
 if(len(zamestnanec.listofnames()) > 0):  
 writer.writerow([zamestnanec.ID] + zamestnanec.listofnames())  
 except FileNotFoundError:  
 print(f"Súbor {filename} nebol nájdený.")  
  
def load\_data(possible\_zamestnanci,formulare\_todo):  
 *#načítanie dát z json súborov* try:  
 with open(possible\_zamestnanci) as json1:  
 possible\_zamestnanci\_json = json1.read()  
 except FileNotFoundError:  
 print(f"Súbor {possible\_zamestnanci} nebol nájdený.")  
 try:  
 with open(formulare\_todo) as json2:  
 formulare\_todo\_json = json2.read()  
 except FileNotFoundError:  
 print(f"Súbor {formulare\_todo} nebol nájdený.")  
 return json.loads(possible\_zamestnanci\_json), json.loads(formulare\_todo\_json)  
  
def simulated\_aneling(T,alpha,limit,output,possible\_zamestnanci,formulare\_todo,slowest\_weight,total\_time\_weight,hours\_weight):  
 *#vyberie jedno riešenie nie náhodne* solution = Solution(notRandomSelect(formulare\_todo,possible\_zamestnanci))  
 *#pokiaľ T je väčšie ako náš limit ,skúšame nové riešenie a vyhodnocujeme ho* while(T > limit):  
 newsolution = Solution(copy.deepcopy(solution.get\_zamestnanci()))  
 *#vytvarame nové riešenie pozmenením rozdelenia formulárov* newsolution = newsolution.get\_neighbour()  
 *#vyhodnocujeme nové riešenie či je lepšie ako predchádzajúce* if(newsolution.evaluatefromlist(slowest\_weight,total\_time\_weight,hours\_weight) <= solution.evaluatefromlist(slowest\_weight,total\_time\_weight,hours\_weight)):  
 solution = newsolution  
 continue  
 *#ak nové riešenie nie je lepšie o rozhodnutí či prijmeme nové riešenie  
 #rozhoduje náhodnosť a aktuálna "teplota"* ap = math.exp((solution.evaluatefromlist(slowest\_weight,total\_time\_weight,hours\_weight) - newsolution.evaluatefromlist(slowest\_weight,total\_time\_weight,hours\_weight))/T)  
 if(ap > random.uniform(0, 1)):  
 solution = newsolution  
  
 T \*= alpha  
 print(f'{T:0.4f}', solution.evaluatefromlist(slowest\_weight,total\_time\_weight,hours\_weight))  
 *#zavoláme metódu na zapísanie outputu, evalujeme vstup a zapíšeme  
 # aj do konzoly naše najlepšie riešenie,ktoré sme pomocou algoritmu našli* writedoc(solution.get\_zamestnanci(), output)  
 evaluate(output)  
  
*#nastavenie hodnôt a spustenie nášho algoritmu*slowest\_weight = 1  
total\_time\_weight = 1  
hours\_weight = 1  
possible\_zamestnanci,formulare\_todo = load\_data("possible\_zamestnanci.json", "formulare\_todo.json")  
simulated\_aneling(1,0.999,0.01,"output.csv",possible\_zamestnanci,formulare\_todo,slowest\_weight,total\_time\_weight,hours\_weight)