Rješavanje problema premještanja kontejnera pomoću The Lowest Point prioritetnog pravila

Matasović, Andrea

Sveučilište Josipa Jurja Strossmayera u Osijeku

Fakultet primijenjene matematike i informatike

Sveučilišni prijediplomski studij Matematika i računarstvo

Rješavanje problema premještanja kontejnera pomoću *The Lowest Point* prioritetnog pravila

Završni rad

Mentor:

doc. dr. sc. Mateja Đumić

Student:

Andrea Matasović

Osijek, 2024

Sadržaj

1 | Uvod

Problem premještanja kontejnera (engl. *Container Relocation Problem*, CRP) je problem koji se susreće kod skladištenja kontejnera u lukama. Zbog ograničenog prostora, kontejneri se u skladištima slažu jedan na drugi i jedan pored drugog, formirajući redove i stupce. Kada je potrebno dohvatiti kontejner koji nije na vrhu, svi kontejneri iznad njega moraju se najprije premjestiti, što usporava proces i smanjuje učinkovitost rada.

CRP se bavi optimizacijom ovog procesa, odnosno pronalaženjem redoslijeda operacija koje minimiziraju broj dodatnih premještanja kontejnera prilikom dohvaćanja kontejnera u zadanom redoslijedu. U ovom radu za rješavanje CRP-a primijenit će se prioritetno pravilo *The Lowest Point* (TLP).

U drugom poglavlju opisan je CRP. Treće poglavlje daje pregled metoda rješavanja s posebnim naglaskom na prioritetna pravila i sheme za rješavanje CRP-a, dok četvrto poglavlje opisuje implementaciju rješenja problema i primjenu prioritetnog pravila TLP u programskom jeziku Python, uz razvoj grafičkog sučelja za simulaciju.

2 | Problem relokacije kontejnera

Značajan dio robe u međunarodnoj trgovini otprema se u kontejnerima, stoga su luke od velike važnosti za prijevoz robe. Kontejneri prevezeni u kontejnerski terminal pohranjuju se u skladištima tako da se postavljaju jedan uz drugi i jedan na drugi, formirajući redove i stupce kako je prikazano na Slici 2.1. Ako se kontejner koji nije na vrhu reda mora dohvatiti, svi kontejneri koji se nalaze iznad njega moraju biti premješteni na neki drugi stupac na kojem ima mjesta. Ova dodatna premještanja kontejnera usporavaju cijeli proces dohvaćanja. Problem premještanja kontejnera predstavlja optimizacijski problem koji uključuje pronalaženje optimalnog slijeda operacija za njihov dohvat iz skladišta u zadanom redoslijedu, najčešće minimizirajući dodatna premještanja kontejnera koji blokiraju one kontejnere koje treba dohvatiti prije njih samih [6].

Slika 2.1: Raspored kontejnera u skladištu. Izvor: [3].

CRP je od iznimne važnosti za učinkovito upravljanje kontejnerskim terminalima jer efikasno rješavanje ovog problema direktno utječe na smanjenje vremena i

troškova operacija, optimizirano korištenje skladišnog prostora, bržu obradu kontejnera te na povećanje ukupne učinkovitosti logističkog sustava.

Sculli i Hui su 1988. godine predstavili CRP za skladištenje kontejnera istih dimenzija u jednom skladišnom prostoru (engl. *single bay*). Njihova studija pokazala je da broj različitih vrsta kontejnera u sustavu značajno utječe na učinkovitost skladištenja, pri čemu veći broj vrsta smanjuje iskorištavanje prostora i povećava nepotrebne manipulacije. Također su naglasili važnost pažljivog oblikovanja strategija slaganja kako bi se optimiziralo korištenje prostora i smanjio broj manipulacija [5].

Matematički, CRP se prikazuje kao dvodimenzionalni niz podataka veličine (T, S), gdje T označava maksimalnu visinu, a S broj stupaca. Redci se numeriraju od dna (1) prema vrhu (T), a stupci s lijeva (1) nadesno (S). Takav niz nazivamo konfiguracijom. Početna konfiguracija ima C kontejnera. Da bi rješenje problema uvijek bilo izvedivo, trojka (T, S, C) mora zadovoljavati sljedeće: $0 \leq C \leq ST - (T-1)$. Naime, potrebno je $T-1$ praznih mjesta kako bi se maksimalno $T-1$ kontejnera koji blokiraju kontejner koji se dohvaća mogli premjestiti [7].

Svaki kontejner označen je jedinstvenim cijelim brojem koji označava njegov redoslijed dohvaćanja. Tijekom procesa dohvaćanja, cilj je dohvatiti kontejner koji u trenutnoj konfiguraciji ima najmanji broj. Kontejner c se smatra *blokirajućim* ako postoji kontejner d u nižem retku istog stupca, takav da je $d < c$.

Kontejner se može dohvatiti samo ako je na vrhu svog stupca, to jest ako nijedan drugi kontejner ne blokira njegovo dohvaćanje.

Rješenje problema premještanja kontejnera sastoji se od pronalaženja optimalnog niza poteza za dohvaćanje kontejnera $1, 2, \ldots, C$ (poštujući zadani redoslijed) uz minimalan broj premještanja. Problem se može pojaviti u dvije varijante: ograničenoj (engl. *restricted*) i neograničenoj (engl. *unrestricted*).

Kod ograničenog CRP-a, kontejner se smije premjestiti samo ako blokira kontejner koji se trenutno mora dohvatiti.

Neograničeni CRP omogućuje premještanje kontejnera bez takvih ograničenja, što znači da se kontejneri mogu premještati i ako ne blokiraju kontejner koji se mora dohvatiti. Ova varijanta pruža veću fleksibilnost, ali može dovesti do većeg broja premještanja u usporedbi s ograničenim CRP-om.

Problemi poput CRP-a često su povezani s klasom NP, koja uključuje probleme za koje trenutno ne znamo pronaći rješenje u polinomnom vremenu, što znači da ne postoji poznat algoritam za njihovo rješavanje u polinomnom vremenu. Ipak, rješenja za ove probleme mogu se provjeriti u polinomnom vremenu. Unutar klase NP, NP-potpunost označava najteže probleme. Dodatno, zbog definicije ovakvih problema, ako bismo pronašli algoritam za rješavanje bilo kojeg NP-potpunog problema u polinomnom vremenu, svi problemi u NP klasi mogli bi biti riješeni u polinomnom vremenu [8].

Avriel, Penn i Shpirer [1] su pokazali da je problem minimizacije premještanja kontejnera NP-potpun, što znači da je rješavanje ovog problema izuzetno složeno. Njihovo istraživanje povezuje ovaj problem s NP-potpunim problemom bojanja kružnih grafova, dokazujući da ne postoji brz algoritam za njegovo rješavanje

osim ako se ne riješi pitanje "P = NP?". Pitanje "P = NP?" odnosi se na to postoji li polinomni algoritam za rješavanje svih problema čija se rješenja mogu provjeriti u polinomnom vremenu (NP problemi). Ako se pokaže da je P = NP, to bi značilo da svi NP-problemi mogu biti riješeni u polinomnom vremenu. Čak i uz ograničenja kapaciteta redova, CRP ostaje NP-potpun, što znači da se za pronalaženje rješenja unutar prihvatljivog vremena moramo osloniti na heurističke metode i aproksimacijske algoritme, iako ta rješenja možda neće biti optimalna.

3 | Metode rješavanja

Metode rješavanja problema relokacije kontejnera se obično dijele na egzaktne i heurističke.

Egzaktne metode traže optimalna rješenja i jamče optimalnost pronađenog rješenja za zadani problem. Međutim, zbog složenosti i NP-potpunosti problema premještanja kontejnera, ove metode često postaju nepraktične za veće instance zbog dugog vremena obrade i velike računalne složenosti. Neki primjeri egzaktnih metoda uključuju *Branch and Bound*, dinamičko programiranje (engl. *dynamic programming*) i linearno programiranje (engl. *linear programming*).

Heurističke metode ne jamče pronalaženje optimalnog rješenja, ali često nude za praksu dovoljno dobra rješenja u razumnom vremenskom okviru, što ih čini prikladnijima za velike i složene probleme poput CRP-a. Heurističke metode uključuju genetičke algoritme (engl. *genetic algorithms*), simulirano kaljenje (engl. *simulated annealing*), prioritetna pravila i druge.

Prioritetna pravila koriste se zajedno sa shemom za generiranje rasporeda, koja pruža opći okvir za generiranje rasporeda. Na temelju prioriteta koje dodijele pravila, shema osigurava da svi zadaci budu usklađeni s postavljenim ograničenjima. U nastavku će biti opisana shema, neka od prioritetnih pravila, njihova primjena i način na koji doprinose rješavanju problema premještanja kontejnera.

3.1 Shema za premještanje kontejnera

Kako bi pronašli rješenje ograničenog CRP-a pomoću prioritetnog pravila trebamo definirati shemu za premještanje kontejnera. Ova shema na osnovu prioriteta koje pridjelujuje prioritetno pravilo donosi odluke o tome koje premještanje je potrebno napraviti.

Shema za premještanje kontejnera za ograničeni CRP dana je Algoritmom 1. Ako je ciljni kontejner (kontejner koji se treba dohvatiti sljedeći) na vrhu svog stupca, po ovoj shemi on će se odmah ukloniti iz skladišta. Ako se, međutim, drugi kontejneri nalaze na vrhu ciljnog kontejnera, potrebno je premjestiti te kontejnere na druge stupce kako bi ciljni kontejner bio dostupan. Izvorni stupac (engl. *origin stack*) je onaj na kojem se trenutno nalazi ciljni kontejner, dok su mogući odredišni stupci (engl. *destionation stacks*) svi drugi stupci koji nisu puni. Za svaki od tih stupaca izračunava se prioritet prema prioritetnom pravilu, a kontejner na vrhu izvornog stupca premješta se na stupac s najnižom prioritetnom vrijednošću.

Nakon što se izvrše sva potrebna premještanja, ciljni kontejner će biti na vrhu i može se preuzeti. Ovaj postupak se ponavlja sve dok skladište ne postane prazno [2].

Algoritam 1 Shema za premještanje kontejnera

```
1:  dok vrijedi kontejnerski terminal nije prazan radi
2:      C = UzmiCiljniKontejner();
3:      S = UzmiStupacUKojemJeCiljniKontejner();
4:      dok vrijedi kontejner C nije na vrhu stupca S radi
5:          za svaki stupac st, st != S i st nije pun radi
6:              π_st = IzračunajPrioritet(st);
7:          kraj
8:          Relociraj kontejner s vrha stupca S na stupac s min(π_st);
9:      kraj
10:     Dohvati kontejner C;
11: kraj
```

3.2 Prioritetna pravila

Prioritetna pravila su heuristička metoda koja se zbog svoje jednostavnosti i brzine često koristi u praksi. Ona omogućava brzo donošenje odluka kroz dodjeljivanje prioriteta. Ova metoda je fleksibilna i može se prilagoditi različitim vrstama problema, čime se omogućuje učinkovito rješavanje bez složenih izračuna.

Kada se kontejneri moraju premjestiti kako bi se omogućilo pristup ciljnom kontejneru, potrebno je odabrati optimalan stupac za privremeno skladištenje kontejnera koji blokiraju one kontejnere koje treba dohvatiti prije njih samih. Razvijena su različita prioritetna pravila koja pomažu u donošenju odluke koji od dostupnih stupaca odabrati prilikom premještanja kontejnera. M. Rotteveel je u svom radu *Effect of Introducing a Time Factor in the Container Relocation Problem* [4] opisano tri pravila koja se često koriste za ovaj problem.

3.2.1 *The Lowest Point* (TLP)

Prioritetno pravilo TLP za rješavanje problema premještanja kontejnera za svaki stupac određuje broj kontejnera koji se u njemu nalaze. Zatim TLP odabire stupac s najmanjim brojem kontejnera kao odredište za premještanje blokirajućeg kontejnera. Ako postoji više stupaca s istim najmanjim brojem kontejnera, nasumično se bira na koji od tih stupaca će biti premješten kontejner. Na kraju, broj kontejnera u oba stupca se ažurira i proces se ponavlja dok je to potrebno.

3.2.2 *Reshuffle Index* (RI)

Ključna ideja algoritma je izračunati *reshuffle indeks* za svaki stupac, što predstavlja broj kontejnera u tom stupcu koji imaju viši prioritet od blokirajućeg kontejnera, tj.

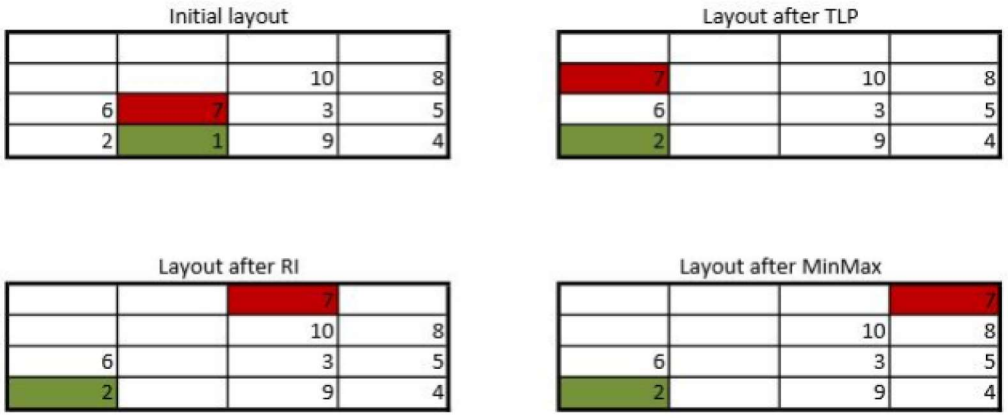
kontejnera koji trenutno blokira pristup ciljnom kontejneru. Algoritam zatim odabire stupac s najnižim *reshuffle indeksom* za premještanje blokirajućeg kontejnera, s ciljem smanjenja budućih premještanja. Na taj način se minimizira vjerojatnost da će premješteni kontejner ponovo blokirati pristup nekom drugom kontejneru u budućnosti.

3.2.3 *MinMax*

Prioritetno pravilo MinMax funkcionira tako da prioritizira stupce na način da za svaki stupac odredi kontejner s najvišim prioritetom koji se u njemu nalazi, i među njima odabere onaj koji ima najmanji takav prioritet.

Na taj način, MinMax osigurava da kontejner koji blokira druge kontejnere bude premješten tamo gdje će najkasnije prouzrokovati blokiranje nekog od kontejnera koji se već tamo nalaze.

Na Slici 3.1 prikazan je prvi korak premještanja kontejnera uz primjenu prethodno opisanih prioritetnih pravila.



Slika 3.1: Relokacije nakon prvog koraka TLP, RI i *MinMax*. Zelenom bojom je označen ciljni kontejner, a crvenom kontejner koji ga blokira. Izvor: [4].

4 | Primjena TLP pravila na problem premještanja kontejnera

U sklopu praktičnog dijela ovog završnog rada izrađeno je grafičko sučelje te implementirano TLP prioritetno pravilo. Programski kod i prateća dokumentacija dostupni su na: https://github.com/andreamatasovic/ZPP.

4.1 Grafičko sučelje

Kao prvi korak u implementaciji, potrebno je implementirati grafičko sučelje koje prikazuje dohvaćanje kontejnera koristeći prioritetno pravilo TLP. Za to nam je potreban paket *tkinter* koji omogućuje prikaz prozora aplikacije te dodavanje kontrola kao što su gumbi i izbornici na kojima korisnici mogu vizualno pratiti premještanje kontejnera. *Filedialog* omogućuje učitavanje podataka o luci i kontejnerima iz JSON datoteka. *Messagebox* prikazuje obavijesti o uspjehu ili greškama.

JSONDecodeError omogućuje prepoznavanje i upravljanje greškama pri parsiranju JSON datoteka. *Randint* i *shuffle* su funkcije koje se koriste za generiranje nasumičnih brojeva i nasumično miješanje elemenata, što je korisno za simuliranje rasporeda kontejnera u luci. Prikaz korištenih biblioteka može se vidjeti u Programskom kodu 4.1.

```
1  import tkinter as tk
2  from tkinter import ttk, filedialog, messagebox
3  from random import shuffle, randint
4  from json import JSONDecodeError
```

Programski kod 4.1: Biblioteke koje su potrebne za implementaciju.

Klasa `Container` predstavlja kontejner u aplikaciji (vidi Programski kod 4.2). Ona sadrži osnovne informacije kao što su pozicija kontejnera, *reshuffle index*, *lookahead cost* i jedinstveni identifikator (id). Ova klasa omogućava upravljanje i praćenje stanja svakog kontejnera unutar simulacije.

```
class Container:
    def __init__(self, container_id, position, reshuffle_index, lookahead_cost=0):
        self.id = container_id
        self.position = position
        self.reshuffle_index = reshuffle_index
        self.lookahead_cost = lookahead_cost

    def __repr__(self):
        return f"Container(id={self.id}, position={self.position}, reshuffle_index={self.reshuffle_index},
        lookahead_cost={self.lookahead_cost})"
```

Programski kod 4.2: Klasa `Container`.

Klasa `ContainerRelocationAuto` je središnji dio aplikacije i odgovorna je za sve operacije vezane za korisničko sučelje i upravljanje simulacijom premještaja kontejnera te sadrži nekoliko metoda.

Metoda `__init__` stvara osnovne karakteristike aplikacije i inicijalizira korisničko sučelje. To uključuje konfiguraciju početnih postavki, kreiranje okvira za kontrole, postavljanje statusne trake te dodavanje gumba za pokretanje simulacije i učitavanje JSON datoteka. Metoda `setup_ui` dizajnira grafičko sučelje prikazano na Slici 4.3.

Slika 4.3: Prikaz grafičkog sučelja aplikacije.

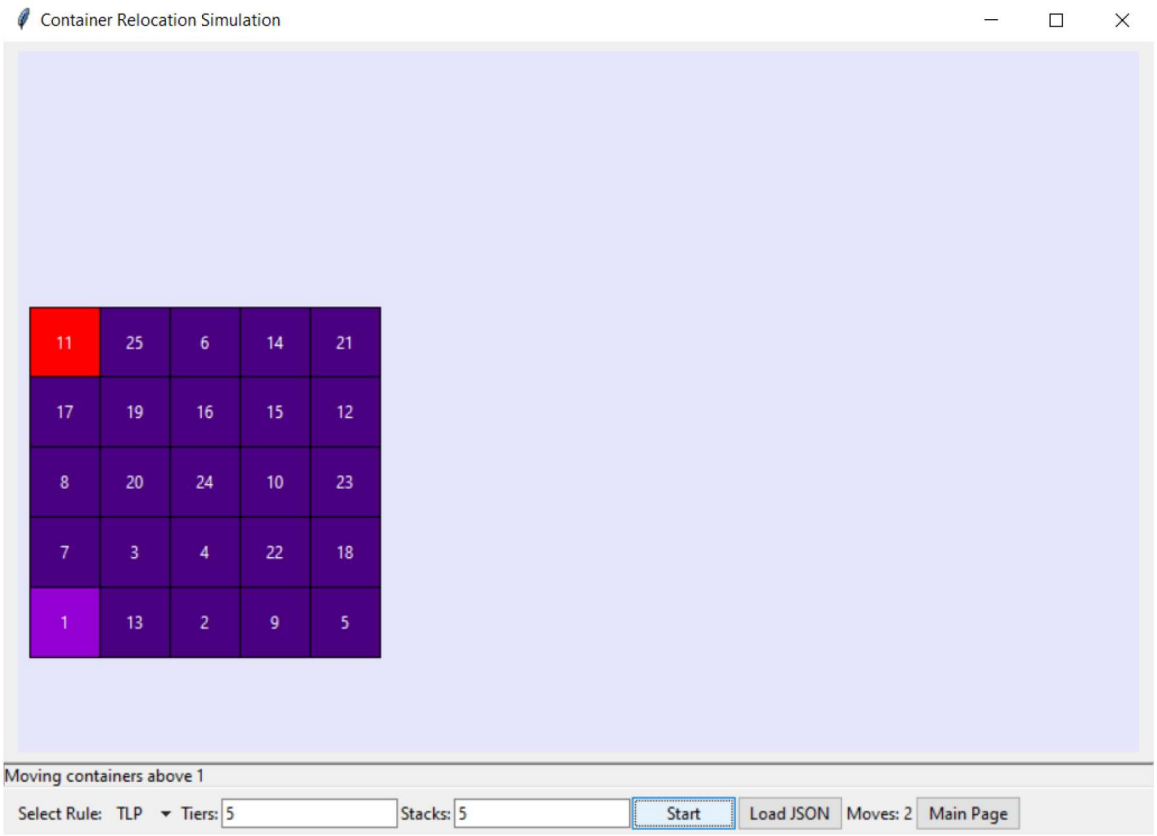
Korisniku je omogućeno da odabere broj redaka i stupaca, čime se generira odgovarajući broj kontejnera metodama `start_simulation` (vidi Programski kod 4.4), `initial_stack` (vidi Programski kod 4.5) i `draw_canvas`, pri čemu su im nasumično dodijeljeni brojevi koji označavaju prioritet, odnosno redoslijed u kojem ih je potrebno dohvatiti (prikazano na Slici 4.6). Korisnik može, umjesto nasumično generiranih kontejnera, učitati prethodno spremljene podatke iz JSON datoteka.

```
78      def start_simulation(self):
79          self.move_count = 0
80          try:
81              tiers = int(self.tiers_entry.get())
82              stacks = int(self.stacks_entry.get())
83              self.stacks = self.initial_stacks(tiers, stacks)
84              self.containers = [container for stack in self.stacks for container in stack]
85              self.containers.sort(key=lambda c: c.position)
86              self.selected_container_index = 0
87              self.current_rule = self.get_selected_rule()
88              self.draw_canvas()
89              self.status_var.set("Status: Simulation started")
90              self.next_step()
91          except ValueError:
92              messagebox.showerror("Error", "Invalid input.
93                                   Please enter numeric values for tiers and stacks.")
```

Programski kod 4.4: Metoda `start_simulation`.

```
94      def initial_stacks(self, tiers, stacks):
95          all_containers = []
96          container_id = 1
97          for _ in range(stacks * tiers):
98              container = Container(container_id, container_id)
99              all_containers.append(container)
100             container_id += 1
101
102         shuffle(all_containers)
103
104         stacks_list = [[] for _ in range(stacks)]
105         for i, container in enumerate(all_containers):
106             stack_index = i % stacks
107             stacks_list[stack_index].append(container)
108
109         return stacks_list
```

Programski kod 4.5: Metoda `initial_stack`.



Slika 4.6: Prikaz nasumično generiranih kontejnera razmještenih u 5 redaka i 5 stupaca.

4.2 Implementacija TLP prioritetnog pravila

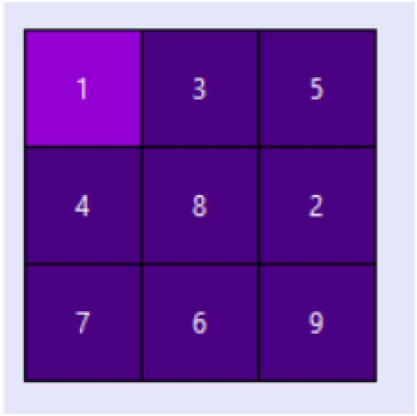
Kao što je opisano u potpoglavlju 3.2.1, pravilo TLP dodjeluje prioritet svakom stupcu na koji se može premjestiti kontejner, na temelju broja kontejnera koji se nalaze u tom stupcu. Zatim odabire stupac s najmanjim brojem kontejnera kao odredište za premještanje. Ako više stupaca ima isti najmanji broj, odabir se vrši nasumično. Kada se kontejner premjesti, broj kontejnera u stupcima se ažurira, a proces se može ponoviti prema potrebi. Implementacija prioritetnog pravila prikazana je u Programskom kodu 4.7.

```
3   def TLP(stacks):
4       selected_containers = []
5       min_position = float('inf')
6
7       for stack in stacks:
8           if stack:
9               top_container = stack[-1]
10              if top_container.position < min_position:
11                  min_position = top_container.position
12                  selected_containers = [top_container]
13              elif top_container.position == min_position:
14                  selected_containers.append(top_container)
15
16      if len(selected_containers) > 1:
17          selected_container = random.choice(selected_containers)
18
19      return selected container
```

Programski kod 4.7: Implementacija prioritetnog pravila `TLP`.

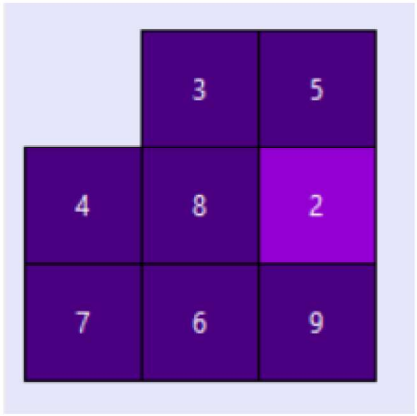
Pritiskom na gumb *Start*, aplikacija automatski premješta kontejnere prema prioritetnom pravilu, ažurira prikaz i prikazuje broj premještaja. Cijeli proces premještanja kontinuirano se opisuje u statusnoj traci, pružajući korisniku ažurirane informacije o trenutnom stanju.

Na Slikama od 4.8 do 4.20 prikazan je primjer simulacije dohvaćanja kontejnera pomoću ovog prioritetnog pravila. Tamnoljubičasti kontejneri prikazuju kontejnere koji još nisu dohvaćeni, a svjetloljubičasti su oni koji su na redu za dohvaćanje.



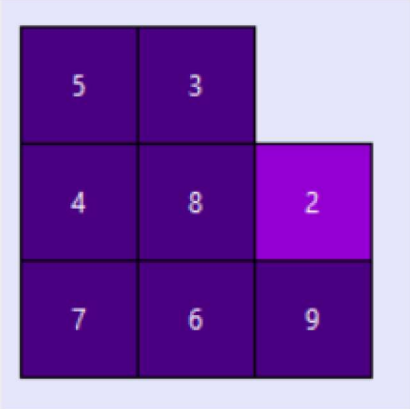
Slika 4.8: Prikaz nasumično generiranih kontejnera razmještenih u 3 retka i 3 stupca.

Prvo se dohvaća kontejner broj 1 koji je već na vrhu stupca (vidi Sliku 4.9).



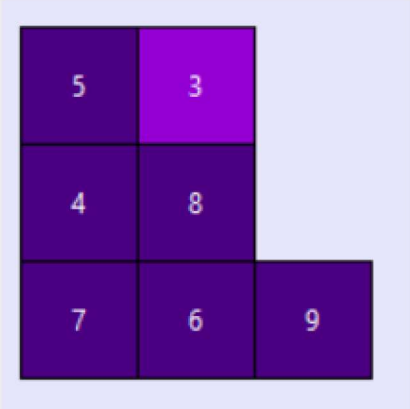
Slika 4.9: Dohvaćanje prvog kontejnera i označavanje drugog kao ciljnog.

Kontejner broj 5 sada blokira kontejner broj 2 koji se sljedeći mora dohvatiti. Tada se kontejner broj 5 premješta na stupac s najmanjim brojem kontejnera (vidi Sliku 4.10).

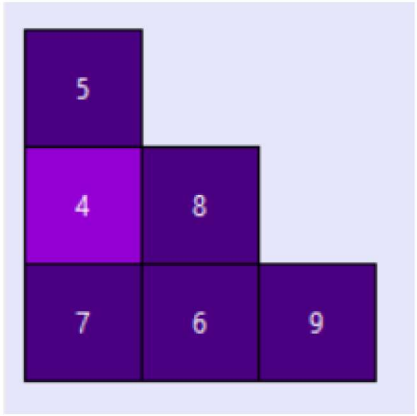


Slika 4.10: Premještanje kontejnera broj 5.

Sada kada je kontejner broj 2 na vrhu svog stupca, aplikacija ga dohvaća (vidi Sliku 4.11). Sljedeći kontejner koji se mora dohvatiti je s brojem 3 koji je već na vrhu stupca, pa se odmah dohvaća (vidi Sliku 4.12).

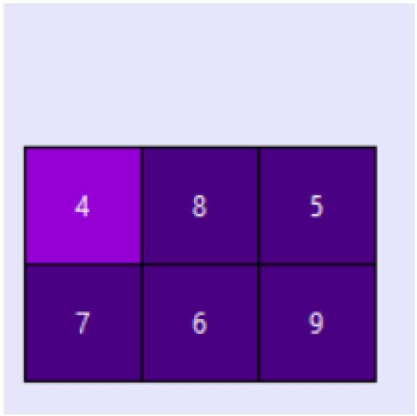


Slika 4.11: Dohvaćanje kontejnera broj 2.



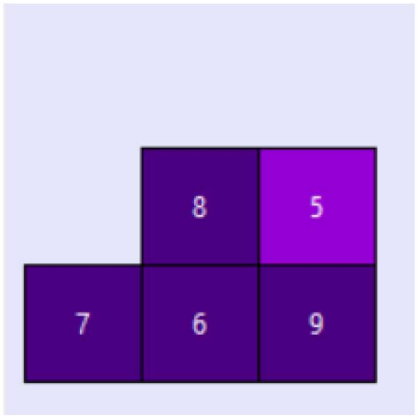
Slika 4.12: Dohvaćanje kontejnera broj 3.

Idući ciljni kontejner je kontejner broj 4, ali njegovo dohvaćanje blokira kontejner broj 5. Zbog toga se kontejner broj 5 premješta na stupac s najmanje kontejnera (vidi Sliku 4.13).

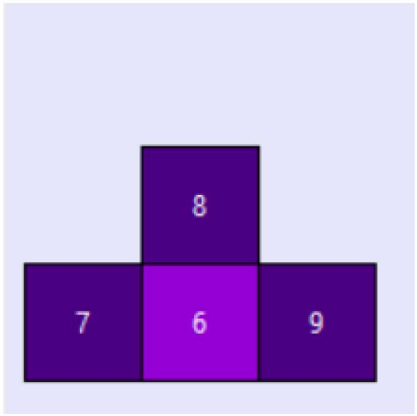


Slika 4.13: Premještanje kontejnera 5 koji blokira kontejner 4.

Sada je kontejner broj 4 na vrhu stupca i dohvaća se (Slika 4.14), a isto tako se dohvaća i kontejner broj 5 (Slika 4.15).

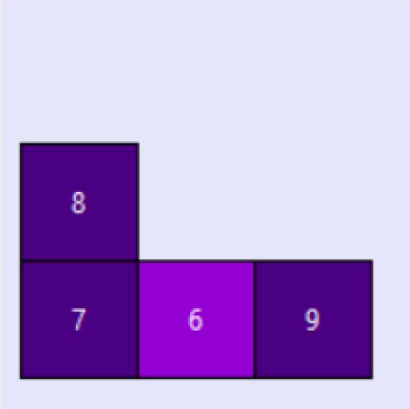


Slika 4.14: Dohvaćanje kontejnera broj 4.



Slika 4.15: Dohvaćanje kontejnera broj 5.

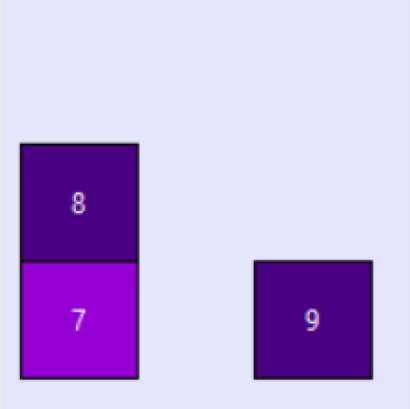
Naredni kontejner za dohvaćanje je broj 6 kojeg blokira kontejner broj 8, te se on premješta na stupac s najmanjim brojem kontejnera. Postoje dva takva stupca, pa se kontejner broj 8 premješta na nasumično odabran stupac (vidi Sliku 4.16).



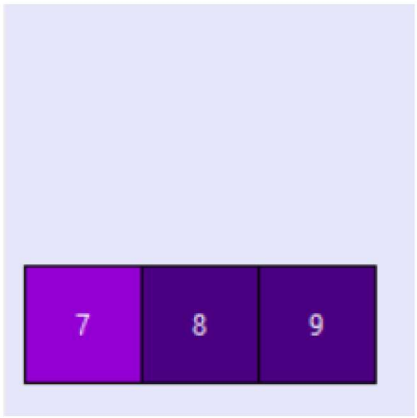
Slika 4.16: Premještanje kontejnera 8.

Sada se dohvaća kontejner 6 (vidi Sliku 4.17).

Kontejner 8, koji blokira kontejner broj 7, se premješta na stupac koji ima najmanje kontejnera (u ovom slučaju u prazan stupac) (vidi Sliku 4.18).

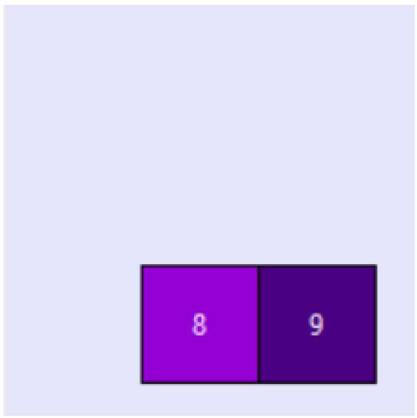


Slika 4.17: Dohvaćanje kontejnera 6.

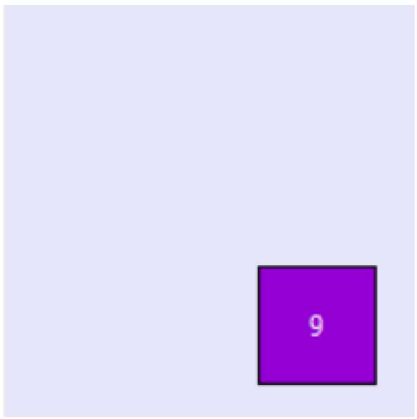


Slika 4.18: Premještanje kontejnera 8 koji blokira ciljni kontejner 7.

Nakon toga se dohvaćaju kontejner i 8 (vidi Sliku 4.19) i 9 (vidi Sliku 4.20) koji su na vrhu stupaca.



Slika 4.19: Dohvaćanje kontejnera 7.



Slika 4.20: Dohvaćanje kontejnera 8 i 9.

Iz prethodnog primjera možemo zaključiti da prioritetno pravilo TLP optimizira premještanje kontejnera prema trenutnoj konfiguraciji stupaca čime pokušava smanjiti broj dodatnih premještanja kontejnera. Ovo pravilo odabire stupac za premještanje kontejnera na temelju broja kontejnera prisutnih u svakom stupcu. S obzirom da je maksimalni broj mogućih dodatnih premještanja jednak broju kontejnera koji se nalaze u stupcu na koji se premješta, ovakvim odabirom ta gornja granica se minimizira. Iako ovako dobiveno rješenje nije nužno i optimalno, do njega se dolazi brzo. Tako se osigurava da su svi kontejneri dostupni za dohvaćanje u najkraćem mogućem vremenu, što povećava brzinu i efikasnost procesa skladištenja.

Literatura

[1] M. Avriel, M. Penn, N. Shpirer, *Container ship stowage problem: complexity and connection to the coloring of circle graphs*, Discrete Appl. Math **103** (2000) 271–279, https://doi.org/10.1016/S0166-218X(99)00245-0.

[2] M.Đurasević, M.Đumić, *Automated design of heuristics for the container relocation problem using genetic programming*, Appl. Soft Comput. **130** (2022) 109696, https://doi.org/10.1016/j.asoc.2022.109696.

[3] Y. Lee, Y. J. Lee, *A heuristic for retrieving containers from a yard*, Comput. Oper. Res. **37** (2010), 1139–1147, https://doi.org/10.1016/j.cor.2009.10.005.

[4] M. Rotteeveel, *Effect of Introducing a Time Factor in the Container Relocation Problem*, Faculty of exact sciences Vrije Universiteit van Amsterdam, (2018).

[5] D. Sculli, C. Hui, *Three dimensional stacking of containers*, Omega **16** (1988), 585–594, https://doi.org/10.1016/0305-0483(88)90032-1.

[6] S. Tanaka, S. Voss, *An exact approach to the restricted block relocation problem based on a new integer programming formulation*, European Journal of Operational Research **296** (2022), 485–503, https://doi.org/10.1016/j.ejor.2021.03.062.

[7] Y.-w. Wan, J. Liu, P.-C. Tsai, *The Assignment of Storage Locations to Containers for a Container Stack*, Naval Research Logistics (NRL) **56** (2009) 699–713.

[8] *Web stranica Europski institut za certificiranje informacijskih tehnologija*, dostupno na https://hr.eitca.org/".

Sažetak

Problem relokacije kontejnera (engl. *Container Relocation Problem*, CRP) predstavlja značajan izazov u skladištima luka gdje se kontejneri slažu jedna pored drugoga i jedan na drugi zbog ograničenog prostora. Zbog načina slaganja često je potrebno raditi premještaje kontejnera kako bi se moglo doći do onog koji je potrebno dohvatiti. Cilj CRP-a je minimizirati broj nepotrebnih premještanja kako bi se poboljšala učinkovitost.

U ovom radu za rješavanje CRP problema koristi se prioritetno pravilo *The Lowest Point* (TLP). U praktičnom dijelu ovog rada napravljena je implementacija ovog pravila te grafičko sučelje u kojem je moguće simulirati proces dohvaćanja kontejnera koristeći TLP pravilo.

Ključne riječi

Container Relocation Problem, prioritetna pravila, TLP pravilo, kontejneri, relokacija

Solving the Container Relocation Problem Using *The Lowest Point* Rule

Summary

Container Relocation Problem (CRP) represents a significant challenge in port warehouses, where containers are stacked side by side and on top of each other due to limited storage space. Due to the stacking method, containers are often relocated to access the one that needs to be retrieved. The goal of the CRP is to minimize the number of unnecessary relocations to improve efficiency.

In this paper, the priority rule *The Lowest Point* (TLP) is used to solve the CRP. In the practical part of this work, an implementation of this rule has been developed, along with a graphical interface that allows simulation of the container retrieval process using the TLP rule.

Keywords

Container Relocation Problem, priority rules, TLP rule, containers, relocation

Životopis

Rođena sam 13. listopada 2002. godine u Zagrebu. Završila sam Osnovnu školu Bogoslav Šulek i Osnovnu glazbenu školu u Slavonskom Brodu. Nakon toga upisujem Gimnaziju Matija Mesić u Slavonskom Brodu, prirodoslovno-matematički smjer. Po završetku srednje škole, upisujem Sveučilišni preddiplomski studij Matematika i računarstvo na Fakultetu primijenjene matematike i informatike (tadašnji Odjel za matematiku) Sveučilišta Josipa Jurja Strossmayera u Osijeku.