Druga domaća zadaća

Igra Connect4

1. Implementacija

Upute: U ovom odjeljku potrebno je opisati ključne dijelove funkcionalnosti koristeći isječke programa i snimke zaslona. Obavezno uključite sljedeće elemente s odgovarajućim komentarima:

- Isječak programa koji prikazuje pripremu poslova na glavnom (master) procesu.

```
68  def generiraj_poslove(board, broj_razina, current_path=[]): #rekurzivno generiranje svih poslova
69      if not broj_razina:
70          return [tuple(current_path)]
71
72      jobs = []
73      for column in range(COLUMNS):
74          if board[0][column] == EMPTY:
75              next_board = board.copy()
76              player = AI
77              if len(current_path) % 2 == 1:
78                  player = PLAYER
79              row, column = odigraj_potez(next_board, column, player)
80              if row is not None and column is not None:
81                  jobs += generiraj_poslove(next_board, broj_razina - 1, current_path + [column])
82      return jobs
```

Ova funckija poziva se na sljedeći način:

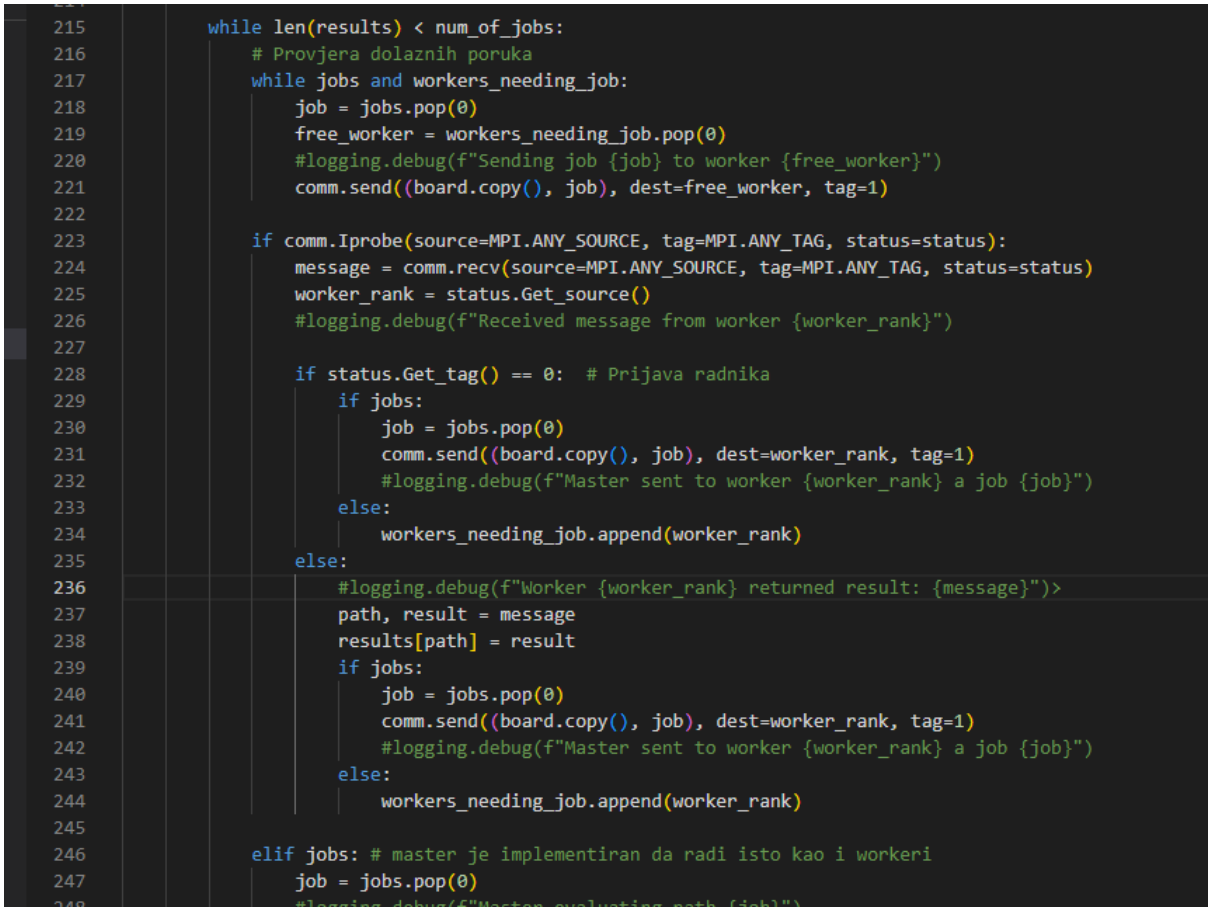
jobs = generiraj_poslove(board, DUBINA)

gdje varijabla DUBINA određuje dubinu pretraživanja, a board trenutnu konfiguraciju ploče. Poslovi su zapravo polja koja sadrže putanju, a ta putanja se sastoji od poteza. Drugim riječima poslovi su putanje po stablu dubine DUBINA. Zato u ovoj funkciji imamo varijablu broj_razina koja se kroz rekurzivne pozive smanjuje. Generiramo poslova onoliko koliko stablo pretraživanja ima listova. U jednom rekurzivnom pozivu prolazimo po svim stupcima i provjeravalo je li potez uopće moguć, u slučaju da nije, tada za tu putanju ne generiramo posao. Ako je moguć, određujemo koji igrač povlači taj potez te pozivamo ponovno generiraj_poslove ali sada na trenutnu putanju dodajemo malo prije spomenuti potez. Tako ćemo u jednom trenutku kroz rekurziju doći do lista stabla i time smo generirali jedan validni posao. Funkcija odigraj_potez samo dodaje potez na ploču.

Dodatno bih napomenuo da je korijen stabla na dubini 0, pa za stablo koje ima jednu razinu ispod korijena potrebno je definirati DUBINA = 1. DUBINA = 0 nije validna vrijednost.

Ova funkcija generiraj_poslove koristi se i kod generiranja poslova koje master šalje radnicima, i kada radnici generiraju svoje slijedne poslove do dubine pretraživanja.

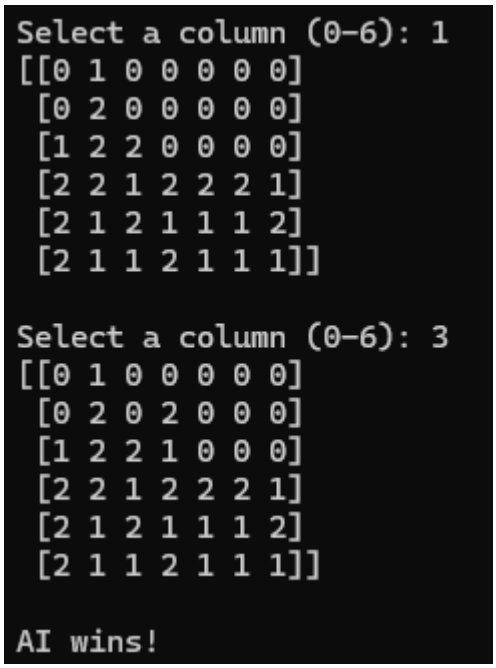
- Isječke programa koji pokazuju kako se zadaci prenose s glavnog (master) procesa na radničke (worker) procese.



```
215         while len(results) < num_of_jobs:
216             # Provjera dolaznih poruka
217             while jobs and workers_needing_job:
218                 job = jobs.pop(0)
219                 free_worker = workers_needing_job.pop(0)
220                 #logging.debug(f"Sending job {job} to worker {free_worker}")
221                 comm.send((board.copy(), job), dest=free_worker, tag=1)
222 
223             if comm.Iprobe(source=MPI.ANY_SOURCE, tag=MPI.ANY_TAG, status=status):
224                 message = comm.recv(source=MPI.ANY_SOURCE, tag=MPI.ANY_TAG, status=status)
225                 worker_rank = status.Get_source()
226                 #logging.debug(f"Received message from worker {worker_rank}")
227 
228                 if status.Get_tag() == 0:  # Prijava radnika
229                     if jobs:
230                         job = jobs.pop(0)
231                         comm.send((board.copy(), job), dest=worker_rank, tag=1)
232                         #logging.debug(f"Master sent to worker {worker_rank} a job {job}")
233                     else:
234                         workers_needing_job.append(worker_rank)
235                 else:
236                     #logging.debug(f"Worker {worker_rank} returned result: {message}")>
237                     path, result = message
238                     results[path] = result
239                     if jobs:
240                         job = jobs.pop(0)
241                         comm.send((board.copy(), job), dest=worker_rank, tag=1)
242                         #logging.debug(f"Master sent to worker {worker_rank} a job {job}")
243                     else:
244                         workers_needing_job.append(worker_rank)
245 
246             elif jobs: # master je implementiran da radi isto kao i workeri
247                 job = jobs.pop(0)
```

Prije ove petlje generiraju se poslovi. Na početku se provjerava postoje li neke dolazne poruke, te ako da onda se primaju. Radnici inicijalno šalju prijavu za posao kako bi se javili masteru, te im on daje posao ako ga ima. Drugi tip poruke koju master može primiti je rezultat računanja posla, a ta poruka ima tag=1, kada master primi rezultat također im šalje posao ako ga ima. Na taj način se smanjuje komunikacije umjesto da se ponovno šalje prijava. Jednom kada IProbe vrati False, odnosno kada nema više dolaznih poruka i ako nakon toga još ima poslova za odradit, master uzima jedan posao. Iz job varijable i trenutne konfiguracije ploče pronalazi stanje ploče koje treba evaluirati. Na gotovo isti način ploču evaluiraju i radnici kada prime posao. Kada master odradi taj jedan posao vraća se na početak petlje i prima sve poruke koje su došle u međuvremenu. Kada ponestane poslova, znači da se analiziralo cijelo stablo. Tada radnici kao i obično šalju zahtjeve za poslom, ali ne dobivaju odgovor. Master ih stavlja u listu workers_needing_job. Kada master primi sve rezultate određuje se sljedeći potez (što nije na slici) tako da se računaju vrijednosti čvorova na razini iznad listova, zatim na temelju toga vrijednosti čvorova iznad i tako dalje do razine 1. Kada se odabere sljedeći potez, geneiraju se novi poslovi koji se mogu pridijeliti radnicima od kojih neki u workers_needing_job čekaju još od prošle iteracije. Iz tog razloga imamo listu workers_needing_job.

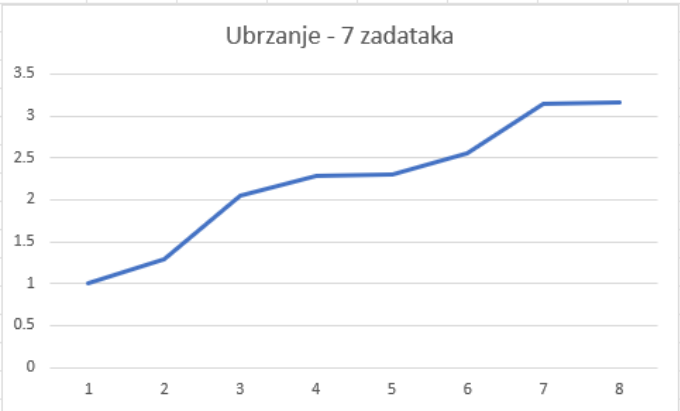
Snimku zaslona koja prikazuje posljednja dva koraka igre u kojoj računalo pobjeđuje.



Potezi računala označeni su brojem 2, a potezi igrača brojem 1. Igra je pokrenuta s dubinom pretraživanja 4.

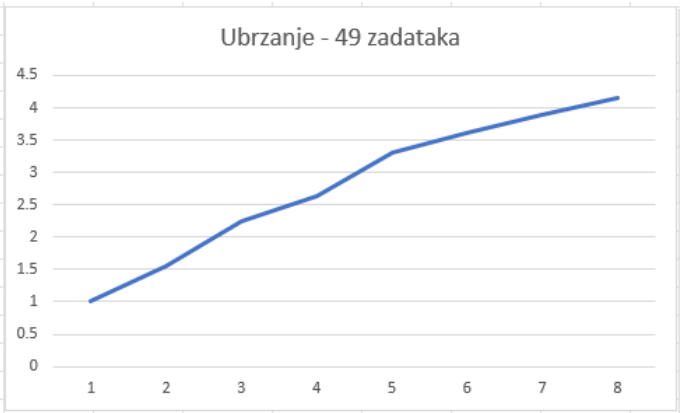
2. Kvantitativna analiza

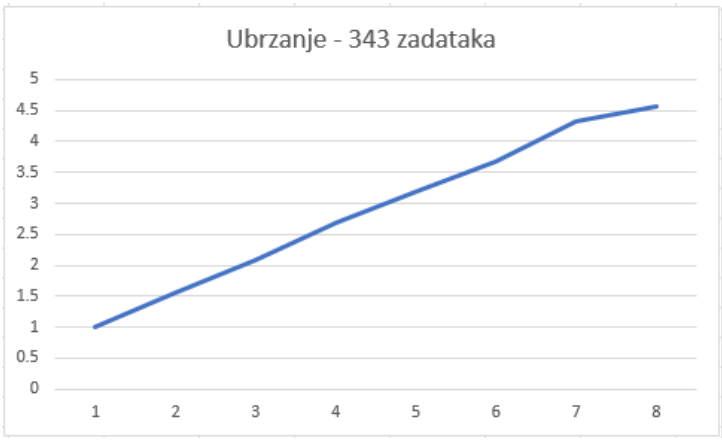
*Upute: U ovom dijelu potrebno je priložiti **tablice s rezultatima mjerenja te grafove ubrzanja i učinkovitosti** za tri različita scenarija: kada paralelni algoritam ima 7, 49 i 343 zadatka (uz aglomeraciju na dubini 1, 2 i 3). Mjerenja treba provesti tako da je najmanje mjereno trajanje (za 8 procesora) reda veličine barem **nekoliko sekundi** (definirajte potrebnu dubinu pretraživanja). Uz grafove, dodajte kratki komentar koji opisuje kako broj zadataka utječe na ubrzanje i učinkovitost (uzevši u obzir utjecaj zrnatosti zadataka, komunikacijskog overhead-a, te udjela programa koji se ne može paralelizirati).*



Za sva ova mjerenja korištena je dubina pretraživanja 6. Kada algoritam vrtimo s 7 zadataka, to znači da svaki zadatak vrlo dugo traje jer se u svakom pretražuje stablo dubine 5. Zadatci su tada krupnozrnati te je otežana mogućnost za ujednačavanjem. Na grafu ubrzanje možemo vidjeti stagnaciju na srednjem dijelu. Razlog tomu je što imamo mali broj zadataka s obzirom na procesore. Kako god da podijelimo 7 zadataka na 4,5 ili 6 procesora, uvijek će na neki procesori imati zadatak više, pa će ostali morati čekati. A to čekanje će biti dugo jer su i sami zadatci dugi za rješavanje. Kod povećanja na 7 procesora možemo uočiti značajno ubrzanje jer se sada svaki zadatak može računati na posebnom procesoru. Kod povećanja na 8 nema nikakve razlike, već samo jedan procesor čeka jer za njega nema zadatka.

Ako algoritam pokrenemo s 49 zadataka riješit ćemo ovaj problem jer su zadatci sada manje krupnozrnasti. Ovo stvara dodatni trošak komunikacije, ali zbog boljeg ujednačavanja vidimo kako dobivamo bolje ubrzanje. Sada dodavanjem više procesora ostvarujemo bolji rad algoritma.





Ako povećamo broj zadataka na 343, odnosno na dubinu zadataka 3, ubrzanje i učinkovitost ostaju gotovo isti. Razlikuju se uglavnom zbog pogreške mjerenja. Ali ako bismo dubinu zadataka povećali na recimo 5, tada uočavamo porast u duljini izvođenja, odnosno pad ubrzanja. Tada utjecaj komunikacije postaje značajniji. Zadatci su tada presitnozrnati te se obavlja jako puno komunikacija u odnosu na računanje. Zbog toga dolazi do pada performansi.
Dodatno treba napomenuti da udio programa koji se može paralelizirati ima bitan utjecaj na performanse. Što je taj udio veći, više toga se može izvoditi istovremeno pa tim i brže.