УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ИСПИТНИ РАД

Кандидат: **Петар Петровић**
Број индекса: **12345**

Предмет: **Системска програмска подршка у реалном времену II**
Тема рада: **<наслов рада>**

Ментор рада: **др Миодраг Ђукић**

Нови Сад, мај, 2020.

SADRŽAJ

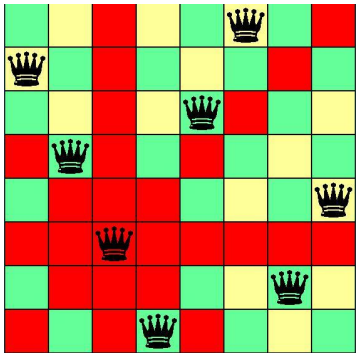
SPISAK SLIKA

SPISAK TABELA

1. Uvod

1.1 Problem N-kraljica

Problem N-kraljica je šahovski problem koji se može formulisati ovako: Na šahovskoj tabli N x N postaviti N kraljica tako da se međusobno ne napadaju. U originalnoj postavci, problem se naravno odnosio na pravu šahovsku tablu 8 x 8, i zvao se „Problem 8 kraljica“. Jedno rešenje prikazuje Slika 1.



Slika 1 Jedno rešenje problema 8 kraljica

Za određenu veličinu table postoji više od jednog rešenja. Pitanje na koje je potrebno naći odgovor glasi: Koliko rešenja postoji za dato N? Tabela 1 prikazuje broj rešenja za veličine table do 15.

Tabela 1 Broj rešenja problema N kraljica za N = 1 do 15

N	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
broj rešenja	1	0	0	2	10	4	40	92	352	724	2.680	14.200	73.712	365.596	2.279.184

1.1.1 Pronalaženje svih rešenja

Postoji niz alogritama koji pobrojavaju rešenja. Najtrivijalniji, ali i najneefikasniji je pretraživanje grubom silom celokupnog prostora svih mogućih postavljanja N kraljica, to jest generisanje svih mogućih postavljanja N kraljica na tablu, da bi se slučajevi koji ne zadovoljavaju uslove eliminisali. Unapređenji algoritam pretraživanja bazira se na postavljanju jedne po jedne kraljice na tablu. Prvo se postavi jedna kraljica u prvu kolonu, zatim u drugu, i tako redom. Međutim, kraljica se u određenu kolonu postavlja samo na ispravnu poziciju, tj. samo na polje koje ne napadaju već postavljene kraljice. Dva su kraja ovog postupka: 1. postavljena je kraljica u poslednju kolonu, i 2. u kolonu koja je na redu ne može se postaviti ni jedna kraljica. U prvom slučaju nađeno je još jedno rešenje, dok drugi slučaj znači da se pozicije prethodno postavljenih kraljica moraju menjati. No, u oba slučaja algoritam se nastavlja tako što se vraća na poslednju kolonu u koju je kraljica ispravno postavljena i pomera je na sledeće validno mesto.

1.1.2 Pronalaženje jedinstvenih rešenja

Od svih rešenja ovog problema za neko N, mnoga rešenja se rotacijom i preslikavanjem preko horizontalne ili vertikalne ose svode na isto rešenje. Ako rešenja koja se pomenutim transformacijama mogu svesti jedna na drugo smatramo jednakim onda je broj rešenja problema manji. Broj takvih rešenja se naziva brojem jedinstvenih rešenja. Jedan način da se prebroje samo jedinstvena rešenja je da se svaki put kada se nađe jedno rešenje najpre proveri da li je ekvivalentno rešenje već nađeno, pa se broj rešenja povećava za jedan samo u slučaju da nije.

Svako pojedinačno rešenje problema N kraljica se može predstaviti permutacijom indeksa redova table. Jasno je da pronalaženje rešenja pretragom koja je gore opisana, teče po rastućem leksikografskom poretku permutacija. U tom smislu, prvi primerak rešenja koji se pronađe od jedne klase ekvivalencije je upravo onaj primerak koji je leksikografski najmanji. Dakle, kada se nađe neko rešenje, dovoljno je proveriti u kakvoj je ono relaciji sa rešenjima koja se dobijaju njenim transformisanjem. Ako neka od transformacija daje rešenje koje je leksikografski manje od trenutnog, onda to znači da je klasa ekvivalencije kojoj pripada trenutno rešenje već ubrojana.

Postupak dobijana ekvivalentnih rešenja je sledeći:

Tri puta rotirati tablu za 90 stepeni (u proizvoljnom smeru, ali sva tri puta u istom),

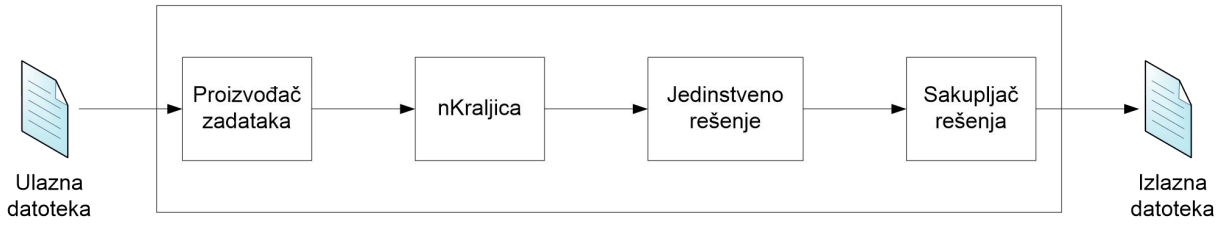
Preslikati jednom oko horizontalne ose (mada može i oko vertikalne) i

Tako preslikanu tablu ponovo rotirati tri puta za 90 stepeni (sva tri puta u istom smeru).

Na taj način se dobijaju 7 (3+1+3) ekvivalentnih rešenja. Ako je bar jedno od njih leksikografski manje od originalnog onda to znači da to originalno rešenje ne treba ubrojati.

1.2 Zadatak

Realizovati sekvencijalan program koji obuhvata prikazane blokove, Slika 2. Kod sekvencijalnog rešenja, neki blokovi, odnosno veze između njih, mogu se zanemariti.



Slika 2 Blok dijagram obrade

Proizvođač zadataka čita ulaznu datoteku koja sadrži u svakom redu po jedan broj N. Za svako N pokreće se blok za traženje broja rešenja za postavljanje N kraljica na tablu veličine NxN, *nKraljica*. Blok *nKraljica* svako pronađeno rešenje prosleđuje bloku *Jedinstveno rešenje*, kao i poruku o završetku pronalaženja rešenja. Blok *Jedinstveno rešenje* određuje broj jedinstvenih rešenja od svih prosleđenih i prosleđuje rezultat bloku *Sakupljač rezultata*. Blok *Sakupljač rezultata* dodaje nov red u izlaznu datoteku i u njega ispisuje rezultat u sledećem formatu:

N: R J,

gde je N broj kraljica, odnosno veličina table, a R broj rešenja, a J broj jedinstvenih rešenja.

Izvršiti testiranje sekvencijalnog programa *unit-test* mehanizmom, korišćenjem CPPUnit biblioteke, tako da se sekvencijalan kod može proglasiti za referentni.

Upotrebom TBB biblioteke izvršiti paralelizaciju referentnog koda. Dozvoljeno je postojanje više instanci istog bloka, ako za to ima potrebe.

Izvršiti testiranje paralelnog programa *unit-test* mehanizmom, korišćenjem CPPUnit biblioteke, poređenjem sa referentnim kodom.

2. Analiza problema

Ako se zanemari algoritamski deo zadatka, glavni problem ostaje paralelizacija sekvencijalnog koda i verifikacija oba programa.

S obzirom da će se nakon sekvencijalnog programa praviti paralelni, sekvencijalan kod treba da bude modularan i sa jasno odvojenim blokovima obrade, tako da se kasnije može što lakše paralelizovati.

Prilikom paralelizacije, potrebno je uočiti regione koji se mogu paralelizovati, kao i odgovarajući tip paralelizacije (podaci, zadaci, i sl.). Nakon toga, treba odabrati način podele problema na zadatke, odrediti način kako će se zadaci formirati i povezati, odnosno kako će izgledati graf tih zadataka. Prilikom izdvajanja zadataka, nastaje problem sinhronizacije i komunikacije između zadataka, koje treba rešiti nekom od dostupnih metoda.

Neophodno je obezbediti isključiv pristup svim deljenim promenljivama, kako između zadataka, tako i na mestima gde se koristi paralelizam podataka, a zavisnost između nekih podataka postoji.

Poseban problem predstavlja signalizacija o završetku rada pojedinih blokova i celog programa.

3. Koncept rešenja

Da bi se ostvarilo što veće ubrzanje prilikom paralelizacije, pored paralelizacije algoritma za određivanje boja mogućih rešenja za postaljanje N kraljica na tablu veličine NxN, potrebno je paralelizovati i ostatak obrade koji se nalazi oko samog algoritma.

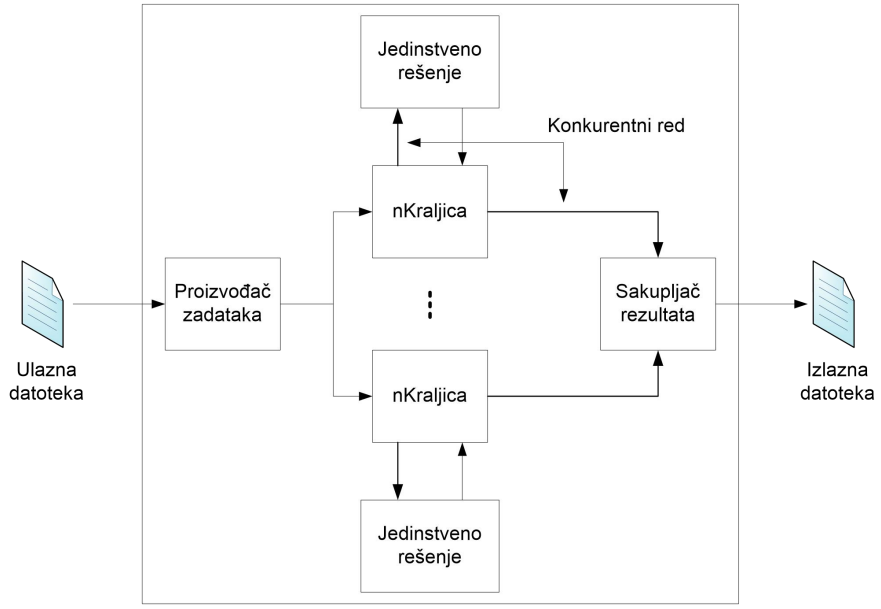
Pošto je, osim samog algoritma, sve zasnovano na blokovskoj obradi, dobro se može predstaviti zadacima. Kako problem ne odgovara nekom od postojeći tipova krafova zadataka, može se koristiti uopšten aciklični tip. Svaki blok obrade može predstaviti zadatkom (čvor grafa). Veze između zadataka su sinhronizacija, odnosno komunikacioni kanali preko kojih se prenose podaci. Sinhronizacija se kod opšteg acikličnog tipa grafa postiže kontrolom *ref_count* polja i pokretanjem, odnosno čekanjem na završetak izvršenja zadataka.

Kako je po tekstu zadatka dozvoljeno, a potencijalno može uneti dodatno ubrzanje, blok *nKraljica* se može umnožiti tako da se omogući paralelna obrada više zadataka koje proizvede blok *Proizvođač zadataka*. *Proizvođač zadataka* je takođe zadatak i on može za svaki parametar koji pročita iz ulazne datoteke da stvori i pokrene (izmresti) novi zadatak *nKraljica* i prosledi mu pročitani parametar. Zadatak *nKraljica* dalje poziva blok *Jedinstveno rešenje* odnosno komunicira sa njim. Više zadataka *nKraljica* postoje i izvršavaju se istovremeno, tako da bi postojanje samo jednog bloka (zadatka) *Jedinstveno rešenje* predstavljalo usko grlo jer bi komunicirao sa svakim od zadataka *nKraljica*. Takođe, morala bi da postoji kopija konteksta za svaki zadatak *nKraljica*. Stoga se nameće i umnožavanje zadataka *Jedinstveno rešenje*, tako da svaki zadatak *nKraljica* pravi i pokreće po jedan zadatak *Jedinstveno rešenje* i prosleđuje mu svako pronađeno rešenje, a zadatak *Jedinstveno rešenje* ažurira broj jedinstvenih rešenja u zavisnosti na osnovu provere opisane u tekstu zadatka. Svaki zadatak *nKraljica* i odgovarajući zadatak *Jedinstveno rešenje* komuniciraju preko konkurentnog reda (prosleđuje se novo, pronađeno rešenje i proverava se da li je ono jedinstveno).

Sva pronađena rešenja se prosleđuju bloku (zadatku) *Sakupljač rezultata*. Pošto je jedna izlazna datoteka u koju se upisuju rezultati, potreban je samo jedan zadatak *Sakupljač rezultata*. Prosleđivanje pronađenih rešenja od zadataka *nKraljica* do zadatka *Sakupljač rezultata* se može izvesti korišćenjem konkurentnog reda iz TBB biblioteke.

Konkurentni redovi između zadataka omogućavaju konkurentan pristup toj strukturi koji je neophodan jer joj više zadataka pristupa u jednom trenutku radi pisanja i, dodatno, postoje zadaci koji joj pristupaju radi čitanja.

Opisan koncept prikazujeSlika 3.



Slika 3 Koncept rešenja paralelnog programa

Da bi se program regularno završio, što podrazumeva regularno zaustavljanje svih zadataka, potrebo je svakom zadatku javiti da su se stekli uslovi za završetak njihovog rada. Proizvođač zadataka završava sa radom kada se završe svi zadaci *nKraljica* koje je on pokrenuo, a potom i zadatak *Sakupljač rezultata* koji takođe on može da pokrene. Zadatak *nKraljica* može da se završi kada se završi zadatak *Jedinstveno rešenje* koji je on pokrenuo. Da bi se zadatak *Jedinstveno rešenje* završio, potrebno je da zna da je njegov zadatak sledbenik pronašao sva rešenja. To je moguće izvesti upisom specifične poruke u konkurentni red (npr. Poruke sa NULL umesto pokazivača na nađeno rešenje). Zadatak *Sakupljač rezultata* može da se završi kada više nema zadataka, odnosno neće biti pronađeno više rešenja. To mu može signalizirati *Proizvođač zadataka*, kada je došao do kraja ulazne datoteke (nema više zadataka-zahteva) i kada su svi zahtevi obrađeni (svi zadaci *nKraljica* završeni). Signalizacija se takođe može izvesti upisom specifične poruke u konkurentni red ka *Sakupljaču rezultata* (upisom NULL umesto pokazivača na rezultat).

4. Opis rešenja

4.1 Sekvencijalan program – moduli i osnovne metode

4.1.1 Modul glavnog programa (MainProgram)

```
int mainSerial(int argc, char* argv[]);
```

Glavna funkcija programa. Prikazuje osnovne informacije o programu, pokreće i uvezuje sve blokove i meri i prikazuje vreme izvršenja programa.

4.1.2 Modul za čitanje ulazne datoteke (TaskProducer)

Modul za čitanje ulazne datoteke i pokretanje blokova NQueens.

4.1.2.1 Metoda za pokretanje modula (Run)

```
virtual int Run(void* param = NULL);
```

Čita ulaznu datoteku i pokreće *Run* metodu zadatka *task* za svaki pročitan parametar koji i proslećuje metodi Run.

Parametri:

- param – ne koristi ses.

Povratne vrednosti:

- 0 – uspešno izvršena,
- ostalo – neuspešno izvršena.

4.1.3 Modul algoritma (NQueens)

Modul (algoritam) za rešavanje problema N-kraljica.

4.1.3.1 Funkcija za pokretanje modula (Solve)

```
void Solve();
```

Postavlja parametre algoritma u početno stanje i pokreće algoritam.

4.1.3.2 Funkcija za proveru da li je kraljica napadnuta (IsAttacked)

```
bool IsAttacked(int i);
```

Proverava da li je kraljica u redu *i* napadnuta od bilo koje kraljice iz prethodnih redova.

Parametri:

- i – red u kojem je poslednja postavljena kraljica za koju se vrši provera.

Povratne vrednosti:

- true – kraljica je napadnuta,
- false – kraljica nije napadnuta.

4.1.3.3 Funkcija za postavljanje kraljice (TryPlace)

```
void TryPlace(int i);
```

Funkcija pokušava da postavi kraljicu u *i*-ti red, tražeći prvo slobodno mesto, a na osnovu napada od drugih kraljica (korišćenjem funkcije *IsAttacked*).

Parametri:

- i – red u kojem je poslednja postavljena kraljica za koju se vrši provera.

4.1.3.4 Metoda za pokretanje modula (Run)

```
virtual int Run(void* param = NULL);
```

Pokreće NQueens za prosleđeno N.

Parametri:

- param – pokazivač na veličinu table (N), odnosno broj kraljica koje treba postaviti.

Povratne vrednosti:

- 0 – uspešno izvršena,
- ostalo – neuspešno izvršena.

4.1.4 Modul za određivanje jedinstvenog rešenja (UniqueSolution)

Modul proverava da li je prosleđeno rešenje jedinstveno u skladu sa svim rotacijama i simetriji prosleđenog rešenja.

4.1.4.1 Metoda za određivanje simetričnog rešenja (Symetre)

```
void Symetre(int* out, int* in, int n);
```

Određuje vertikalnu simetriju prosleđenog rešenja.

Parametri:

- out – pokazivač na izlazno rešenje (simetrično u odnosu na ulazno),
- in – pokazivač na ulazno rešenje čiju simetriju treba odrediti,

- n – veličina table.

4.1.4.2 Metoda za određivanje rotacije rešenja (Rotate)

```
void Rotate(int* out, int* in, int n);
```

Određuje rotaciju prosleđenog rešenja.

Parametri:

- out – pokazivač na izlazno rešenje (rotirano u odnosu na ulazno),
- in – pokazivač na ulazno rešenje čiju rotaciju treba odrediti,
- n – veličina table.

4.1.4.3 Metoda za pokretanje modula (Run)

```
virtual int Run(void *param = NULL);
```

Pokreće UniqueSolution za prosleđeno rešenje.

Parametri:

- param – pokazivač na strukturu koja predstavlja rešenje.

Povratne vrednosti:

- 0 – uspešno izvršena,
- ostalo – neuspešno izvršena.

4.1.5 Modul za upis rezultata u izlaznu datoteku (ResultsCollector)

Modul prima kao parametar metode Run novi rezultat i upisuje ga u izlaznu datoteku, polje outFile.

4.1.5.1 Metoda za pokretanje modula (Run)

```
virtual int Run(void *param = NULL);
```

Pokreće ResultsCollector za prosleđen rezultat.

Parametri:

- param – pokazivač na strukturu koja predstavlja rezultat.

Povratne vrednosti:

- 0 – uspešno izvršena,
- ostalo – neuspešno izvršena.

4.2 Paralelni program – moduli i osnovne metode

4.2.1 Modul glavnog programa (MainProgramP)

```
int mainParallel (int argc, char* argv[]);
```

Glavna funkcija programa. Prikazuje osnovne informacije o programu, pokreće i uvezuje sve blokove i meri i prikazuje vreme izvršenja programa.

4.2.2 Modul za čitanje ulazne datoteke (TaskProducerP)

Modul za čitanje ulazne datoteke i pokretanje zadataka CNQueensP i zadatka za prikupljanje rezultata CResultsCollectorP.

4.2.2.1 Metoda za pokretanje zadatka (execute)

```
task* execute();
```

Čita ulaznu datoteku, stvara i mresti CResultsCollectorP zadatak za prikupljanje zadataka, stvara i mresti CNQueensP zadatak za svaki pročitan parametar koji mu i prosleđuje. Čeka da se završe svi CNQueensP zadaci. Šalje NULL u konkurentni niz rezultata i time javlja CResultsCollectorP zadatku da može da se završi, a potom još čeka na njegov završetak.

Povratne vrednosti:

- NULL – ne pokreće ni jedan zadatak nakon završetka.

4.2.3 Modul algoritma (NQueensP)

Modul (algoritam) za rešavanje problema N-kraljica.

4.2.3.1 Funkcija za pokretanje modula (Solve)

```
void Solve();
```

Postavlja parametre algoritma u početno stanje i pokreće algoritam.

4.2.3.2 Funkcija za proveru da li je kraljica napadnuta (IsAttacked)

```
bool IsAttacked(int* column, int i, int n);
```

Proverava da li je kraljica u redu *i* niza *column* veličine *n* napadnuta od bilo koje kraljice iz prethodnih redova.

Parametri:

- column – memorijski niz sa postavljenim kraljicama,
- i – red u kojem je poslednja postavljena kraljica za koju se vrši provera,
- n – veličina table.

Povratne vrednosti:

- true – kraljica je napadnuta,
- false – kraljica nije napadnuta.

4.2.3.3 Funkcija za postavljanje kraljice (TryPlace)

```
void TryPlace(int* column, int i, int n, int* totalSolutionsNo, concurrent_queue<int*>* queue);
```

Funkcija pokušava da postavi kraljicu u *i*-ti red, tražeći prvo slobodno mesto, a na osnovu napada od drugih kraljica (korišćenjem funkcije *IsAttacked*). Nakon postavljanja N-te kraljice, konstatuje da je pronađeno novo rešenje i smešta ga u konkurentni red *queue*, tako da može da se eventualno uveća broj jedinstvenih rešenja.

Parametri:

- column – memorijski niz sa postavljenim kraljicama,
- i – red u kojem je poslednja postavljena kraljica za koju se vrši provera,
- n – veličina table,
- queue – konkurentni red sa pronađenim rešenjima.

4.2.3.4 Metoda za pokretanje zadatka (execute)

```
task* execute();
```

Stvara i mresti zadatak *CUniqueSolutionP* za određivanje broja jedinstvenih rešenja. Poziva metodu *Solve* i time pokreće algoritam. Dodaje NULL u konkurentni red *queue* i time signalizira zadatku *CUniqueSolutionP* da može da se završi. Čeka da se završi zadatak *CUniqueSolutionP* i prosleđuje rezultate zadatku *CResultsCollectorP* tako što ga smešta u konkurentni red *result_queue*.

Povratne vrednosti:

- NULL – ne pokreće ni jedan zadatak nakon završetka.

4.2.4 Modul za određivanje jedinstvenog rešenja (UniqueSolutionP)

Modul proverava da li je prosleđeno rešenje jedinstveno u skladu sa svim rotacijama i simetriji prosleđenog rešenja.

4.2.4.1 Metoda za određivanje simetričnog rešenja (Symetre)

```
void Symetre(int* out, int* in, int n);
```

Određuje vertikalnu simetriju prosleđenog rešenja.

Parametri:

- out – pokazivač na izlazno rešenje (simetrično u odnosu na ulazno),
- in – pokazivač na ulazno rešenje čiju simetriju treba odrediti,
- n – veličina table.

4.2.4.2 Metoda za određivanje rotacije rešenja (Rotate)

```
void Rotate(int* out, int* in, int n);
```

Određuje rotaciju prosleđenog rešenja.

Parametri:

- out – pokazivač na izlazno rešenje (rotirano u odnosu na ulazno),
- in – pokazivač na ulazno rešenje čiju rotaciju treba odrediti,
- n – veličina table.

4.2.4.3 Metoda za pokretanje modula (Run)

```
int Run(int* column);
```

Pravi celu klasu rešenja kojoj pripada i prosleđeno rešenje. Proverava da li je prosleđeno rešenje jedinstveno, odnosno leksikografski najmanje.

Parametri:

- column – pokazivač na memorijski niz koji predstavlja rešenje.

Povratne vrednosti:

- 0 – uspešno izvršena,
- ostalo – neuspešno izvršena.

4.2.4.4 Metoda za pokretanje zadatka (execute)

```
task* execute();
```

Zadatak koji preuzima rešenje iz konkurentnog reda *queue* i poziva metodu *Run* za preuzeto rešenje. Završava sa radom ukoliko je pročitani parametar jednak NULL.

Povratne vrednosti:

- NULL – ne pokreće ni jedan zadatak nakon završetka.

4.2.5 Modul za upis rezultata u izlaznu datoteku (ResultsCollectorP)

Modul prima kao parametre konstruktora ime izlazne datoteke *outFileName* i konkurentni red *queue* sa rezultatima programa. Kao zadatak, preuzima novi rezultat i upisuje ga u izlaznu datoteku *outFile*.

4.2.5.1 Metoda za pokretanje zadatka (execute)

```
task* execute();
```

Zadatak koji preuzima rezultat iz konkurentnog reda *queue* i ispisuje ga u izlaznu datoteku *outFile*. Završava sa radom ukoliko je pročitani parametar jednak NULL.

Povratne vrednosti:

- NULL – ne pokreće ni jedan zadatak nakon završetka.

5. Testiranje

Implementacija se testira mehanizmom *unit-test*, a korišćena je biblioteka CPPUNIT za testiranje. Napravljeni su posebni testni skupovi (skupovi testnih slučajeva) za svaki modul, a svaki testni skup sadrži testove (testne slučajeve, *test cases*) raznih segmenata tog modula – funkcija i struktura, kao i testiranje njihovih međusobnih odnosa.

5.1 Testni skupovi

5.1.1 TaskProducerTest

Skup testova za verifikaciju bloka (klase) *TaskProducer*.

5.1.1.1 TaskProducerRunTest

Test pokreće blok *TaskProducer* pozivom njegove metode *Run* sa prosleđenim imenom datoteke i zadatkom koji *TaskProducer* pokreće nakon svake pročitane linije (parametra N). „Podmeće“ se zadatak posebno napravljen u ovom skupu testova koji pri svakom pozivu proverava da li je pokrenut za odgovarajući parametar iz ulazne datoteke.

5.1.2 NQueensTest

Skup testova za proveru ispravnosti bloka *NQueens*, osnovnog bloka programskog rešenja.

5.1.2.1 NQueensIsAttackedTest

Test postavlja polja *n* i *column* već napravljenog objekta klase NQueens na početnu vrednost i poziva njenu metodu *IsAttacked* za različite parametre uz proveru valjanosti povratnih vrednosti (da li funkcija javlja da je kraljica napadnuta kada zaista jeste i obrnuto).

5.1.2.2 NQueensTryPlaceTest

Test postavlja polje *n* već napravljenog objekta klase NQueens na početnu vrednost, zauzima prostor za *column* i poziva njenu metodu *NQueens* klase *TryPlace* za red 0 uz proveru valjanosti povratne vrednosti (da li funkcija pronalazi ispravan broj rešenja).

5.1.2.3 NQueensSolveTest

Test postavlja polje *n* već napravljenog objekta klase NQueens na početnu vrednost, zauzima prostor za *column* i poziva njenu metodu *NQueens* klase *Solve* uz proveru valjanosti povratne vrednosti (da li funkcija pronalazi ispravan broj rešenja).

5.1.2.4 NQueensRunTest

Pokreće *Run* metodu klase *NQueens* tako da iz nje bude pokrenut zadatak definisan u ovom testnom skupu (zadatak je „podmetnut“ prilikom stvaranja objekta klase *NQueens*). Podmetnuti zadatak svaki put kada je pozvan proverava ispravnost rezultata koji mu je prosleđen (broj rešenja za dato N).

5.1.3 UniqueSolutionTest

Skup testova za verifikovanje bloka (klase) *UniqueSolution*.

5.1.3.1 UniqueSolutionRotateTest

Test formira jedan memorijski niz i poziva metodu *Rotate* klase *UniqueSolution* nad tim nizom. Proverava da li izlazni memorijski niz odgovara očekivanom (rotiranom u odnosu na ulazni).

5.1.3.2 UniqueSolutionSymetreTest

Test formira jedan memorijski niz i poziva metodu *Symetre* klase *UniqueSolution* nad tim nizom. Proverava da li izlazni memorijski niz odgovara očekivanom (simetričnim u odnosu na ulazni).

5.1.3.3 UniqueSolutionRunTest

Test prosleđuje metodi *Run* bloka *UniqueSolution* sva rešenja koja pripadaju istoj klasi rešenja za neko N. Očekuje se da se broj jedinstvenih rešenja uveća samo kada je prosleđeno leksikografski najmanje rešenje, a da u ostalim slučajevima broj jedinstvenih rešenja ostaje nepromenjen.

5.1.4 ResultsCollectorTest

Skup testova za verifikovanje bloka (klase) *ResultsCollector*.

5.1.4.1 ResultsCollectorRunTest

Test prosleđuje bloku *ResultsCollector* različite rezultate i proverava da li su svi rezultati, tim redom upisani u datoteku. Ime datoteke u koju *ResultsCollector* upisuje rezultate, prosleđena je prilikom stvaranja objekta ove klase.

5.1.5 MainTests

5.1.5.1 SerialAndParallelCmp

Test za poređenje sekvencijalnog (referentnog) i paralelnog programa. Pošto paralelni program može da izbaci rezultate u različitom redosledu od redosleda pravljenja zadataka, poređenje se radi pokretanjem programa za samo jedan parametar N i proverom jednakosti dobijenih rezultata.

5.2 Rezultati i analiza vremena izvršenja

Svi skupovi testova i testni slučajevi unutar njih su uspešno izvršeni, što ukazuje na ispravnost verifikovanog programa nad definisanim skupom testova. Pod pretpostavkom da definisani skup testova obuhvata sve kritične slučajeve, sekvencijalan kod možemo smatrati ispravnim i uzeti za referentni.

Poređenjem izlaza sekvencijalnog i paralelnog programa, utvđeno je da daju iste rezultate, odnosno da paralelni program isto kao i referentni, sekvencijalni.

U cilju merenja vremena izvršenja i poređenja sekvencijalnog i paralelnog programa, oba su izvršavana na dvojezgarnom procesoru *Intel Core2 Duo P8700 2.53 GHz*.

Izmerena vremena izvršavanja sekvencijalnog i paralelnog programa za određeni testni vektor i dva slučaja (sa/bez određivanja jedinstvenih rešenja i upisom rezultata) prikazuje Tabela 2.

Tabela 2 Vremena izvršenja sekvencijalnog i paralelnog programa

	Sa određivanjem broja jedinstvenih rešenja	**Bez određivanja broja jedinstvenih rešenja**
Serijski program	313.2	296.4
Paralelni program	327.7	190.1
Ubrzanje (serijski/paralelni)	0.96	1.56

6. Zaključak

Napravljen je jedan koncept za realizaciju datog problema. Napravljen je i verifikovan sekvencijalni i paralelni model koji obuhvata problem opisan u zadatku.

Verifikacija sekvencijalnog programa urađena je kroz pravljenje *unit-testova* pomoću *CPPUnit* biblioteke. Time je potvrđena funkcionalnost svakog pojedinačnog bloka, njihovih veza i programa u celini. Poređenjem izlaza paralelnog i sekvecijalnog programa potvrđeno je njihovo podudaranje, odnosno ispravnost paralelnog programa.

Serijski i paralelni program su pokretani za iste testne vektore na istom računaru u cilju merenja vremena i ubrzanja. Rezultati merenja (Tabela 2) pokazuju da se ubrzanje postiže kada se ispituje jedan deo rešenja (bez određivanja jedinstvenih rešenja i upisa rezultata). Ubrzanje se gubi (čak dolazi do usporenja) kada se pusti i ostatak programa u rad. Razlog za to je koncept sinhronizacije (komunikacije) između pojedinih zadataka koja je realizovana pomoću konkurentnih redova koji su spori (sporiji i od STL redova zbog dodatne sinhrnizacije pri čitanju/upisu). Promenom koncepta sinhronizacije moglo bi se postici ubrzanje i u ovim segmentima.