Univerzitet Crne Gore

Prirodno-matematički fakultet

Smjer: Računarske nauke

Predmet: Paralelni algoritmi

Izvještaj za projekat na temu

“Paralelni Bellman-Ford algoritam

za najkraće puteve iz jednog čvora”

Profesor: Dr Igor Jovančević Studentkinje: Itana Radičević 8/24,

Dejana Vukčević 1/24

Podgorica, 14.05.2025.

Sadržaj

[1. Uvod 3](#_Toc198015255)

[2. Opis projekta 3](#_Toc198015256)

[2.1 Implementacija četiri verzije Bellman-Ford algoritma 3](#_Toc198015257)

[**2.1.1****Sekvencijalna verzija (bellmanFordSequential)** 3](#_Toc198015258)

[**2.1.2****Paralelna verzija 1 (bellmanFordParallelV1)** 4](#_Toc198015259)

[**2.1.3****Paralelna verzija 2 (bellmanFordParallelV2)** 4](#_Toc198015260)

[**2.1.4****Paralelna verzija 3 (bellmanFordParallelV3)** 5](#_Toc198015261)

[2.2 Generisanje grafova 5](#_Toc198015262)

[2.3 Mjerenje performansi 6](#_Toc198015263)

[2.4 Verifikacija rezultata 6](#_Toc198015264)

[2.5 Keširanje grafova sa LRU (Least Recently Used) policy 7](#_Toc198015265)

[2.6 Provjera rezultata 7](#_Toc198015266)

[2.7 Testiranje i logovanje 8](#_Toc198015267)

# **Uvod**

U ovom projektu smo implementirale sekvencijalnu i tri različite verzije paralelnog Bellman-Ford algoritma i uporedile performanse svih njih u C++ na nasumično generisanim potpunim grafovima. Bellman-Ford algoritam pronalazi najkraće puteve iz jednog čvora do svih ostalih čvorova u usmjerenom grafu sa težinama koje mogu imati i negativnu vrijednost. Za razliku od Dijkstrinog algoritma, on može detektovati negativne cikluse. Zbog grafovia velikih dimenzija potrebna je optimizacija performansi algoritma. U ovom projektu smo koristile **OpenMP** biblioteku za implementaciju paralelizacije kako bismo iskoristile višejezgarnu arhitekturu modernih procesora.

# **Opis projekta**

Ovaj projekat ima za cilj praćenje i analizu performansi različitih verzija Bellman-Ford algoritma kroz implementaciju i testiranje u jeziku C++. Implementacija se sastoji iz četiri cjeline.

## **Implementacija četiri verzije Bellman-Ford algoritma**

U okviru projekta razvijene su sljedeće verzije algoritma:

### **Sekvencijalna verzija (bellmanFordSequential)**

Klasična iterativna implementacija koja za svaki od n-1 koraka prolazi kroz sve grane i pokušava da ažurira najkraće rastojanje. Ako u nekoj iteraciji ne dođe do promjena, algoritam se prekida ranije.

### **Paralelna verzija 1 (bellmanFordParallelV1)**

Implementira paralelizaciju na nivou grana - to znači da više niti istovremeno prolazi kroz različite grane grafa i pokušava da ažurira rastojanja. Problem nastaje kada dvije ili više niti pokušaju istovremeno da promijene istu lokaciju u **dist** vektoru tj. kada različite grane vode ka istom čvoru, pa više niti pokuša da upiše novu vrijednost za isti odredišni čvor. Ovo može dovesti do tzv. **race condition**, gdje ishod zavisi od toga koja nit stigne prva, što može dati pogrešan rezultat. Da bi se to izbjeglo prilikom pisanja u zajednički vektor dist, koristi se direktiva #pragma omp critical. Ona označava da je naredni blok koda kritična sekcija – tj. u svakom trenutku samo jedna nit može da izvršava tu sekciju. Ostale niti moraju da čekaju svoj red. Iako **critical** rješava problem konkurentnog pristupa, on može usporiti izvršavanje ako se prečesto koristi, jer se uvodi čekanje između niti.

### **Paralelna verzija 2 (bellmanFordParallelV2)**

U ovoj verziji Bellman-Ford algoritma koristi se paralelizacija po čvorovima, a ne po granama. Ideja je izbjegavanje direktne konkurencije više niti nad zajedničkim resursima – konkretno nad vektorom **dist**, koji čuva trenuto najkraće rastojanje do svakog čvora. Ovdje se koristi lokalna kopija udaljenosti - vektora (newDist) unutar svake iteracije. Umjesto da niti direktno pišu u **dist**, svaka nit radi sa lokalnom kopijom – vektorom **newDist**. Taj vektor čuva ažurirane vrijednosti za trenutnu iteraciju. Svaka nit vrši ažuriranja isključivo nad sopstvenom kopijom **newDist**, i upoređuje trenutno poznato rastojanje sa mogućim kraćim rastojanjem kroz određeni čvor. Nakon iteracije, newDist se kopira nazad u dist koji se koristi za sljedeću iteraciju. Ovakav pristup povećava efikasnost.

### **Paralelna verzija 3 (bellmanFordParallelV3)**

Ova verzija Bellman-Ford algoritma predstavlja hibridni pristup koji kombinuje prednosti prethodnih verzija, sa nekim dodatnim optimizacijama u podjeli posla i upotrebi memorije. Za razliku od verzije 1 i verzije 2 ova verzija koristi statičku raspodjelu uz chunking. To znači da se zadac unaprijed ravnomjerno dijele u komadiće koji se dodjeljuju nitima tako da se izbjegne preklapanje i obezbijedi ujednačeno opterećenje. Ova tehnika raspodjele poboljšava iskorišćenost CPU resursa i smanjuje vrijeme čekanja među nitima, što je posebno važno kod velikih i neravnomjerno raspodijeljenih grafova. Chunking doprinosi boljem keširanju podataka, jer se rad na susjednim čvorovima često odnosi na slične memorijske lokacije, što povećava šanse da su potrebni podaci već u kešu procesora. Kao i kod verzije 2, koristi se lokalna kopija distance-vektora (newDist) unutar svake iteracije. Svaka nit radi na svojoj verziji newDist, izbjegavajući konkurentno pisanje u zajednički dist vektor. Nakon što sve niti završe sa obradom svog dijela čvorova, newDist se kopira nazad u dist, čime se završava iteracija. Ovaj hibridni pristup kao rezultat ima brže izvršavanje.

## **Generisanje grafova**

Metodom **Graph::generateCompleteGraph**, generiše se potpuno povezan graf sa zadatim brojem čvorova. Svaki čvor je povezan sa svim ostalim čvorovima osim sa samim sobom, a težine grana se dodjeljuju random unutar zadatog intervala (minWeight, maxWeight). Težine mogu biti i negativne, čime se simuliraju grafovi za koje Bellman-Ford ima prednost nad Dijkstrinim algoritmom.

## **Mjerenje performansi**

Za mjerenje vremena izvršavanja koristi se:

**auto start = chrono::high\_resolution\_clock::now();**

**auto end = chrono::high\_resolution\_clock::now();**

**chrono::duration<double> duration = end - start;**

Vrijeme se mjeri za svaku verziju posebno, a rezultati se upisuju u CSV fajl kako bi se mogli kasnije analizirati i prikazati vizuelno.

## **Verifikacija rezultata**

Za svaku paralelnu verziju vrši se **verifikacija tačnosti rezultata** poređenjem sa sekvencijalnom verzijom:

bool verifyResults(const BFResult& seq, const BFResult& par);

Upoređuju se: niz distances (udaljenosti od izvora do svih čvorova), niz predecessors (putanja) i informacija o tome da li graf sadrži negativan ciklus. Ukoliko postoji bilo kakvo odstupanje, ispisuje se poruka o neuspjehu verifikacije.

## **Keširanje grafova sa LRU (Least Recently Used) policy**

Da bi se izbjeglo višestruko generisanje istih grafova prilikom testiranja, implementirano je keširanje grafova pomoću LRU (Least Recently Used) algoritma. Mehanizam se sastoji iz:

* unordered\_map<int, Graph> graphCache: čuva generisane grafove po ključu (npr. broju čvorova),
* vector<int> cacheAccessOrder: prati redosljed pristupa grafovima u kešu.

Ograničenje: najviše 5 grafova se može čuvati istovremeno. Kada se u keš doda novi graf, a keš je pun, najstariji (najmanje korišćen) se uklanja. Ovaj mehanizam smanjuje vrijeme generisanja grafova pri više testiranja sa istim parametrima i čini sistem boljim.

## **Provjera rezultata**

Funkcija verifyResults ima ključnu ulogu u potvrđivanju ispravnosti paralelnih verzija Bellman-Ford algoritma. Njena osnovna svrha je da uporedi rezultate dobijene paralelnim implementacijama sa onima iz sekvencijalne verzije. U okviru ove funkcije porede se vektori rastojanja (dist) koje svaka verzija generiše. Ukoliko se pronađe bilo kakvo odstupanje između vrijednosti dobijenih u sekvencijalnoj i nekoj od paralelnih verzija, funkcija prijavljuje grešku i daje koji čvor ima pogrešnu udaljenost. Ovakva verifikacija je ključna jer omogućava otkrivanje potencijalnih logičkih grešaka izazvanih paralelnim pisanjem, race conditions ili nekonzistentnim ažuriranjem podataka.

## **Testiranje i logovanje**

Funkcija runTests predstavlja centralni dio za automatsko testiranje performansi svih verzija algoritma. Ona se koristi za testiranje implementacija na grafovima različitih veličina – od manjih (npr. 50 čvorova) do većih (2000 čvorova). Za svaki graf, izvršavaju se sljedeći koraci:

* Generisanje grafa – Graf se generiše sa slučajnim težinama ivica. Ako je graf iste veličine već generisan ranije, koristi se keširana verzija zahvaljujući LRU (Least Recently Used) keširanju.
* Pokretanje algoritama – Izvršavaju se sve četiri verzije Bellman-Ford algoritma: sekvencijalna i tri paralelne.
* Mjerenje vremena izvršavanja – Za svaku verziju bilježi se vrijeme pomoću std::chrono, kako bi se precizno uporedile performanse.
* Verifikacija rezultata – Poziva se funkcija verifyResults kako bi se osiguralo da rezultati svih paralelnih verzija odgovaraju onima iz sekvencijalne verzije.

Zapis rezultata – Svi podaci se zapisuju u CSV fajl koji se koristi za dalju analizu, kreiranje grafova ili statističko izvještavanje. Ovakav pristup omogućava pouzdano testiranje i objektivnu procjenu performansi svakog algoritma. Zapis rezultata u CSV format dodatno olakšava upotrebu podataka u prezentacijama ili izvještajima.

Reference: [Microsoft Word - COMS 4995 report.docx](https://www.cs.columbia.edu/~sedwards/classes/2020/4995-fall/reports/Bellman-Ford.pdf)