PERBANDINGAN IMPLEMENTASI ALGORITMA-ALGORITMA PAGERANK PADA SATU MESIN KOMPUTER

Skripsi

Disusun untuk memenuhi salah satu syarat memperoleh gelar Sarjana Komputer

Oleh:
Farhan Herdian Pradana
1313618030

PROGRAM STUDI ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI JAKARTA

2023

Lembar Pernyataan

Saya menyatakan dengan sesungguhnya bahwa skripsi dengan judul **Perbandingan Implementasi Algoritma-algoritma Pagerank dalam Satu Mesin Komputer** yang disusun sebagai syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Ilmu Komputer Universitas Negeri Jakarta adalah karya ilmiah saya dengan arahan dari dosen pembimbing.

Sumber informasi yang diperoleh dari penulis lain yang telah dipublikasikan yang disebutkan dalam teks skripsi ini, telah dicantumkan dalam daftar pustaka sesuai dengan norma, kaidah, dan etika penulisan ilmiah.

Jika dikemudian hari ditemukan sebagian besar skripsi ini bukan hasil karya saya sendiri dalam bagian-bagian tertentu, saya bersedia menerima sanksi pencabutan gelar akademik yang saya sanding dan sanksi-sanksi lainnya sesuai dengan peraturan perundang-undangan yang berlaku.

Jakarta, 10 Oktober 2023

Farhan Herdian Pradana

HALAMAN PERSEMBAHAN

Untuk Almh. Kak Irna

KATA PENGANTAR

Puji syukur terhadap ke hadirat tuhan Yang Maha Esa, yang telah memberikan nikmat, baik berupa kesehatan, waktu, maupun finansial sehingga dapat menyelesaikan skripsi yang berjudul **"Perbandingan Implementasi Algoritma-algoritma Pagerank pada Satu Mesin Komputer"**.

Selesainya skripsi ini tidak hanya berkat Penulis seorang melainkan juga ada pihak - pihak lain yang dengan tulus mau membantu Penulis dalam menyusun skripsi ini baik secara langsung dengan memberikan bimbingan, maupun dukungan finansial dan moril. Untuk itu izinkan Penulis mengucapkan banyak terima kasih kepada pihak-pihak berikut:

1. Yth. Ibu Dr. Ria Arafiyah, M.Si selaku Koordinator Program Studi Ilmu Komputer.
2. Yth. Bapak Muhammad Eka Suryana, M.Kom selaku Dosen Pembimbing I.
3. Yth. Bapak Med Irzal, M.Kom selaku Dosen Pembimbing II.
4. Seluruh staf dan dosen Program Studi Ilmu Komputer.
5. Keluarga penulis.
6. Teman-teman Program Studi Ilmu Komputer 2018.

Skripsi ini masih jauh dari kata sempurna dan Penulis sangat terbuka terhadap kritik dan masukan dari pembaca sehingga dapat menjadi bahan evaluasi Penulis dalam menulis skripsi dan tulisan sejenisnya. Semoga skripsi ini dapat memberikan manfaat bagi pembaca.

Jakarta, 10 Oktober 2023

Farhan Herdian Pradana

ABSTRAK

FARHAN HERDIAN PRADANA. Perbandingan Implementasi Algoritma-algoritma Pagerank pada Satu Mesin Komputer. Skripsi. Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta. 2023. Di bawah bimbingan Muhammad Eka Suryana, M. Kom, dan Med Irzal, M. Kom.

Algoritma Pagerank merupakan algoritma mengurutkan halaman web pada *search engine* Google. Masalah pada Algoritma Pagerank adalah memerlukan memori utama yang cukup besar, dan tidak mungkin dilakukan pada satu mesin komputer dengan memori utama yang terbatas. Akan dicari algoritma alternatif dari Algoritma Pagerank Original buatan Google dengan membandingkannya pada algoritma-algoritma Pagerank dari penelitian lainnya dengan membandingkan kecepatan, penggunaan memori utama, dan kemiripan hasil. Penelitian dilakukan dengan melakukan pengkodean terhadap algoritma Pagerank Original, algoritma Distributed Pagerank Computation (DPC), algoritma Modified DPC (MDPC), dan algoritma Random Walker. Semua kode program dijalankan pada dataset dan dibandingkan kecepatan, penggunaan memori utama, dan kemiripan hasil akhirnya. Khusus hasil akhir, hasil dari algoritma Random Walker dijadikan acuan karena dasar dari Algoritma Pagerank adalah Random Walker. Hasilnya Algoritma Pagerank Original unggul dari sisi kecepatan dan hasil yang mirip dengan hasil Algoritma Random Walker. Sedangkan algoritma DPC dan MDPC unggul di penggunaan memori utama yang lebih hemat, sehingga cocok untuk satu mesin komputer yang memiliki memori utama yang terbatas, tetapi dengan catatan mengorbankan kecepatan yang lebih lambat dan hasil yang tidak mirip terhadap Random Walker.

Kata Kunci: *Search engine*, Google, Pagerank, Distributed Pagerank Computation.

ABSTRACT

FARHAN HERDIAN PRADANA. *Implementation Comparisson of Pagerank Algorithms in Single Machine Computer. Thesis. Faculty of Mathematic and Natural Sciences, State University of Jakarta. 2023. Under supervision of* Muhammad Eka Suryana, M. Kom, *and* Med Irzal, M. Kom.

Pagerank Algorithm is an algorithm used for calculating web page ranking in Google search engine. Problem arises for Pagerank Algorithm due to big main memory usage, thus make it impossible to run in single machine computer with limited main memory. Alternative algorithms will be proposed by comparing the alternative algorithms from other studies with the Original Google Pagerank in terms of speed, main memory usage, and their result similarity. In this study, the Orignal Pagerank, Distributed Pagerank Computation (DPC), Modified DPC, and Random Walker algorithms will be implemented. The implemented algorithms will be run with datasets, and their speed, main memory usage, and result similarity will be compared. For result similarity, Random Walker's result will be used as a benchmark, since it has been used as base concept of Pagerank. It is concluded that the Original Pagerank is faster and has very similar result with Random Walker, while DPC and MDPC have significantly smaller main memory usage, thus very suitable for single machine computer with limited main memory, but run slower and sacrificing result similarity.

Keywords: *Search engine*, Google, Pagerank, Distributed Pagerank Computation.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

PENDAHULUAN

A. Latar Belakang Masalah

Internet adalah jaringan luas yang membuat jaringan komputer seluruh dunia yang dijalankan oleh perusahaan, pemerintahan, universitas, dan organisasi lainnya untuk dapat saling berkomunikasi (Sample, 2018). Internet memiliki banyak kegunaan. Di bidang komunikasi, Internet melahirkan *Voice over Internet Protocol* (VoIP) dan surat elektronik (*email*). Di bidang pengiriman data, internet memungkinkan pengguna untuk dapat mengunggah berkas ke *file server* untuk dibagikan ke orang lain atau supaya bisa diakses di mana saja. Yang paling populer, selain dari dua bidang tersebut, internet melahirkan World Wide Web (WWW) yang memungkinkan situs web atau biasa disebut *website* untuk bisa diakses oleh semua orang.

Website adalah sekumpulan halaman web yang saling terkait dan berada pada nama domain yang sama. Website dapat dibuat dan dipelihara oleh seorang individu, grup, perusahaan, atau organisasi lain dengan berbagai macam tujuan (Techopedia, 2020). Dengan adanya *website*, kegunaan internet menjadi semakin lebih luas, lebih berkualitas, dan lebih mudah digunakan. *Website* yang menggunakan protokol HTTP, memungkinkan untuk mengirim berkas HTML, CSS, dan JavaScript sehingga internet dapat menampilkan visual yang lebih ramah terhadap pengguna dan menjadi media hiburan baru. Tidak heran, *website* terus tumbuh pesat. Pada tahun 1992 hanya terdapat sepuluh *website*, lalu pada tahun 1994 angka ini bertumbuh menjadi 3.000 *website*, dan semakin bertumbuh pesat pada tahun 2021 menjadi kurang lebih 1,88 miliar *website* (Amstrong, 2021).

Dengan meledaknya jumlah halaman web, memunculkan tantangan baru dalam memperoleh informasi dari web. Pengguna biasanya menelusuri web dengan mengunjungi graf *link* yang terdapat pada halaman web, biasanya dimulai pada situs kumpulan index halaman web berkualitas tinggi yang dipelihara oleh manusia seperti Yahoo.com, atau menggunakan *search engine* (Brin dan Page, 1998). Seiring perkembangan zaman, *search engine* Google menjadi *search engine* teratas dengan pengguna terbesar di dunia dengan penguasaan pasar sebesar 91% (GlobalStatCounter, 2022). Google awalnya merupakan proyek Sergey Brin dan Larry Page saat mereka mengambil gelar Ph.D di Universitas Stanford dengan tujuan membuat *search engine* yang lebih berkualitas dibandingkan *search engine* lain (Brin dan Page, 1998). Kunci kesuksesan dari Google terletak pada Pagerank. Pagerank merangking halaman web berdasarkan kepentingan relatif (*relative importance*) suatu halaman web berdasarkan graf tautan web (Page dkk., 1999). Sebelum adanya Pagerank, *search engine* lain biasanya merangking suatu halaman web dengan menghitung banyaknya *backlink* yang merujuk halaman tersebut (Page dkk., 1999). Metode tersebut akan menjadi mudah untuk dimanipulasi pemilik halaman web yang ingin mendapatkan *ranking* teratas dengan membuat halaman web lain yang berisi *link* yang menunjuk pada halaman web tujuan sebanyak-banyaknya. Pagerank menjawab permasalahan tersebut dengan melakukan normalisasi pada jumlah *backlink* suatu halaman web (Brin dan Page, 1998). Hal ini yang membuat hasil pencarian Google menjadi lebih relevan dibandingkan hasil pencarian *search engine* lain.

Telah dilakukan penelitian mengenai *search engine*, Chen dkk. (2006) melakukan sebuah penelitian berjudul "*Efficient Query Processing in Geographic Web Search Engines*". Pada penelitian tersebut diajukan algoritma pemrosesan kueri yang lebih efisien daripada algoritma kueri yang biasa dipakai pada *geographic*

search engine. Algoritma yang diajukan bersifat *low-level*, karena membuat struktur data internal tersendiri pada *hard drive* tanpa melalui lapisan komunikasi pada *database*, sehingga lebih efisien. Terdapat tiga algoritma yang diajukan: *K-Sweep*, *Tile Index*, dan *Space-Filling Inverted Index*, setelah dilakukan pengujian, menunjukkan performa algoritma tersebut jauh lebih baik daripada algoritma *Text-First* atau *Geo-First* yang merupakan algoritma pemrosesan kueri yang biasa dipakai pada *geographic search engine*, bahkan mendekati performa *search engine* teks biasa (Chen dkk., 2006). Selanjutnya Allah dkk. (2021) melakukan tinjauan literatur tentang desain *user interface* (UI) *web search* untuk lansia yang berjudul "*Designing web search UI for the elderly community: a systematic literature review*". Dari tinjauan literatur tersebut diberikan saran improvisasi dari UI *web search* yang sudah ada agar lebih ramah lansia seperti: Tampilan visual yang jelas dan mudah dibedakan, penjelasan singkat apa yang akan terjadi ketika menekan tombol dialog, halaman hasil pencarian yang muncul pada jendela atau tab baru, besaran karakter dan jarak yang bisa dikustomisasi pada kolom pencarian, dan lain-lain (Allah dkk., 2021). penelitian Allah dkk. (2021) hanya membahas *search engine* dari segi UI dan menyasar pada demografi tertentu.

Setiap *search engine* memiliki arsitektur berbeda-beda. Arsitektur Google dipilih dan dijadikan acuan dalam topik penelitian peningkatan arsitektur *search engine* yang merupakan penelitian induk dari judul penelitian ini karena keunggulannya dibandingkan *search engine* lain. Arsitektur Google dapat dilihat pada gambar 2.3. Modul *crawler* dan pendukungnya yang ditandai dengan warna biru muda sudah dibuat pada penelitian sebelumnya yang berjudul "Perancangan *Crawler* Sebagai Pendukung Pada Search Engine" oleh Qoriiba (2021). Pada penelitian tersebut digunakan algoritma *modified similarity based crawling* dan selanjutnya hasil dari *crawling* disimpan kedalam *database* MySQL (Qoriiba, 2021).

Selanjutnya penelitian berjudul "Perancangan Arsitektur *Search Engine* dengan Mengintegrasikan *Web Crawler*, Algoritma Page *ranking*, dan Dokumen *ranking*" oleh Khatulistiwa (2022). Pada penelitian tersebut digabungkan modul *crawler* dari penelitian Qoriiba (2021), Pagerank, dan *searcher* pada penelitian lain sebelumnya menjadi *search engine* berbasis konsol (Khatulistiwa, 2022). Pada modul *indexer* dilakukan penelitian oleh Pratama (2022) berjudul "Perancangan Modul Pengindeks pada *Search Engine* Berupa *Induced Generalized Suffix Tree* untuk Keperluan Perangkingan Dokumen" dan Zalghornain (2022) berjudul "Rancang Bangun Sistem Pencarian Teks dengan Menggunakan Model *Continuous-Bag-of-Words* dan Model *Continuous Skip-Gram* pada Koleksi Dokumen".

Dalam melakukan perangkingan halaman web, Pagerank dapat didefinisikan pada persamaan 1.1. Dimana u adalah halaman web. F_u adalah himpunan halaman u yang menunjuk halaman lain dan B_u adalah himpunan halaman yang menunjuk ke u. $C_u = |F_u|$ adalah jumlah *link* dari u dan c adalah faktor yang digunakan untuk normalisasi (sehingga jumlah total *ranking* semua halaman web adalah konstan) dan $c < 1$. $E(u)$ adalah vektor yang berkorespondensi dengan *ranking* halaman web. $||\pi'||_1 = 1$. Iterasi perhitungan terus dilakukan sampai konvergen. Jika diubah kedalam persamaan matriks, maka persamaan 1.1 dapat diubah menjadi persamaan 1.2. Dimana X adalah matriks persegi yang baris dan kolomnya berkorespondensi dengan halaman web, dengan elemen $X_{u,v} = \frac{1}{C_u}$ jika terdapat *link* pada halaman u ke halaman v atau $X_{u,v} = 0$ jika tidak ada.

$$\pi'(u) = c \sum_{v \in B_u} \frac{\pi'(v)}{C_v} + cE(u) \tag{1.1}$$

$$\pi' = c(X\pi' + E) \tag{1.2}$$

Dasar intuitif dari persamaan 1.1 adalah *random walks* pada sebuah graf. Anggap pengguna internet sebagai *"random surfer"* yang terus meng-klik *link* selanjutnya secara acak. Namun, jika pengguna terjebak pada sebuah lingkaran halaman web (*link* yang diklik terus menampilkan halaman web yang pernah dikunjungi sebelumnya), tidak mungkin pengguna akan terus mengikuti *link* tersebut, melainkan pengguna akan langsung pindah ke halaman lain. Oleh sebab itu faktor E dipakai untuk memodelkan perilaku ini (Pengguna akan bosan dan langsung lompat ke halaman web lain berdasarkan distribusi pada E) (Page dkk., 1999). E dapat didefinisikan oleh pengguna (*user-defined parameter*) dan nilai elemennya dapat diisi dengan nilai yang seragam atau berbeda-beda. Menariknya, jika nilai elemen E dibuat berbeda-beda, maka dapat membuat Pagerank yang dipersonalisasi (Page dkk., 1999).

Walaupun persamaan Pagerank terlihat sederhana, terdapat masalah dari sisi ruang dan waktu. Dari sisi ruang, misal terdapat 1000 halaman web, maka akan terbentuk 1000x1000 matriks X. Misal tiap sel elemen X memerlukan memori 8 *Byte*, maka untuk membentuk 1000x1000 matriks X, tanpa menghitung alokasi *overhead* memori, memerlukan 8 *Mega Byte* (MB) memori utama (Lihat tabel 1.1). Di internet terdapat miliaran *website* dan setiap *website* dapat memiliki lebih dari 1 halaman, sehingga untuk bisa membentuk matriks X membutuhkan memori utama dengan kapasitas mencapai *Peta Byte* atau bahkan *Exa Byte*. Hal tersebut sangat tidak mungkin dilakukan pada komputer pribadi biasa yang memori utamanya hanya pada kisaran 4 GB sampai 32 GB, yang berarti ketika program dieksekusi langsung *crash* karena memori yang tidak cukup. Dari sisi waktu, proses *string matching* untuk mengakses nilai *ranking* suatu halaman web berdasarkan *string* URLnya juga memiliki kompleksitas waktu yang besar yakni O(NM), jika dilakukan dengan cara dicari satu-persatu. Beruntung *database* seperti MySQL menggunakan B-Tree

dalam mengindeks datanya (MySQLDoc, 2022). B-tree memiliki kompleksitas waktu kecil yakni hanya O(log(n)) (GeeksForGeeks, 2022).

Tabel 1.1: Tabel alokasi memori utama untuk membentuk matriks X

No	Jumlah Halaman Web	Dimensi Matriks	Alokasi Memori (*Byte*)
1	256	256 x 256	524416
2	512	512 x 512	2097280
3	1024	1024 x 1024	8388736
4	2048	2048 x 2048	33554560
5	4096	4096 x 4096	134217856
6	8192	8192 x 8192	536871040
7	16384	16384 x 16384	2147483776
8	32768	32768 x 32768	8589934720

Telah dilakukan penelitian tentang Pagerank yang terdistribusi dengan metode *iterative aggregation-disaggregation* (IAD) dengan *Block Jacobi smoothing* (Zhu dkk., 2005). Sederhananya, dilakukan *divide-and-conquer* dengan mengelompokan halaman web berdasarkan *domain*-nya lalu dihitung Pagerank lokalnya dan disatukan dengan metode komunikasi yang hemat memori dengan sebuah koordinator (Zhu dkk., 2005). Hasilnya, ditemukan sebuah metode Pagerank terdistribusi yang bisa dijalankan pada memori utama kecil dan lebih cepat konvergen sehingga menghemat waktu (Zhu dkk., 2005). Oleh karena itu, akan dicari algoritma Pagerank alternatif yang dapat dijalankan pada satu mesin komputer dengan memori utama terbatas, dengan cara melakukan perbandingan implementasi beberapa algoritma Pagerank pada satu mesin komputer.

B. Rumusan Masalah

Berdasarkan penjelasan sebelumnya, dapat ditemukan sebuah masalah, yaitu bagaimana perbandingan implementasi algoritma-algoritma Pagerank yang dijalankan dalam satu mesin komputer ?

C. Pembatasan Masalah

Pembatasan masalah dirumuskan agar penelitian menjadi lebih fokus dan tidak terlalu luas. Pembatasan masalah pada penelitian ini adalah hanya membandingkan implementasi algoritma-algoritma Pagerank ke dalam satu mesin komputer yang memiliki memori utama terbatas.

D. Tujuan Penelitian

Tujuan dari penelitian ini adalah mencari algoritma alternatif dari algoritma Pagerank pada penelitian Page dkk. (1999) yang selanjutnya akan disebut sebagai Pagerank Original yang dapat dijalankan dengan baik pada satu mesin komputer dengan memori utama terbatas, dengan membandingkannya dengan beberapa algoritma Pagerank pada penelitian-penelitian lainnya.

E. Manfaat Penelitian

Penelitian ini diharapkan akan memberikan manfaat bagi beberapa pihak yaitu:

1. Bagi Penulis

   Mengasah dan mengaplikasikan pengetahuan yang diperoleh selama berkuliah khususnya di bidang *search engine* dan Pagerank sekaligus untuk memperoleh gelar sarjana ilmu komputer.

2. Bagi Universitas Negeri Jakarta

   Penelitian ini dapat dijadikan acuan untuk penelitian selanjutnya terutama yang berkaitan dengan *search engine* dan Pagerank. Selain itu juga, memperkaya ragam tulisan akademik yang diterbitkan oleh Universitas Negeri Jakarta.

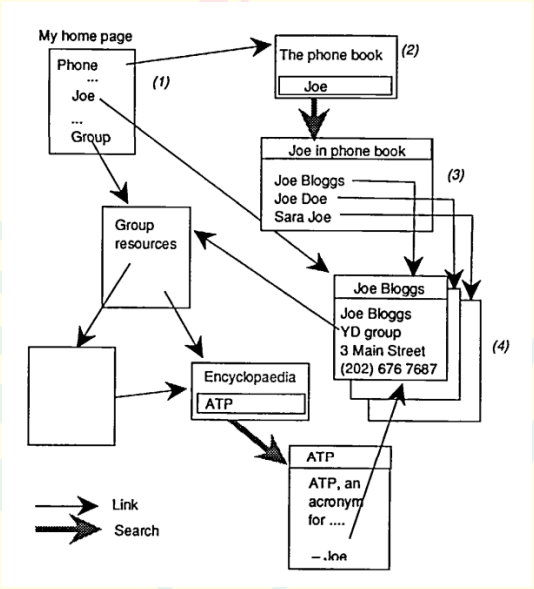
3. Bagi Masyarakat

   Menunjukan masalah implementasi algoritma Pagerank original pada satu mesin komputer yang memiliki memori utama terbatas, dan memberikan alternatif algoritma Pagerank lain dengan segala kelebihan dan kekurangannya.

BAB II

KAJIAN PUSTAKA

A. World Wide Web (WWW)



Gambar 2.1: Gambar sebuah web yang terdiri atas kumpulan *link* dan indeks (Berners-Lee dkk., 1992)

WWW merupakan proyek Tim Berner-lee bersama teman-temannya, yang ditunjukan pada publik pada tahun 1991. WWW didesain untuk membawa sebuah semesta informasi global menggunakan teknologi yang ada. Dengan adanya WWW manusia dapat mengakses seluruh informasi melalui sebuah *platform browsing* apapun. Pada masa itu, sudah ada teknologi serupa yang membuat WWW mungkin untuk dilakukan. Sistem *hypertext* yang sudah ada pada saat itu, terbatas pada sistem *file* lokal atau terdistribusi dan kadang hanya dikembangkan pada *platform* tertentu. Selain itu, juga ada sistem pengambilan dan akses informasi seperti Alex, Gopher, Prospero, dan WAIS yang sudah mencakup area yang luas, tetapi tanpa fungsionalitas *hypertext*. WWW menggabungkan teknik *hypertext*, *information*

retrieval, dan *wide area networking* (Berners-Lee dkk., 1992).

Model yang dipakai WWW menggunakan dua paradigma dari *hypertext link* dan pencarian teks yang saling melengkapi. Gambar 2.1 menunjukkan bagaimana sebuah web yang berisi informasi pribadi terbentuk pada paradigma ini. Pembaca mulai pada halaman *home* (1) lalu menggunakan *link* grup atau publik untuk mencari bahan informasi. Indeks seperti buku telepon (2) ditampilkan sebagai dokumen yang memungkinkan untuk melakukan input pencarian. Hasil dari pencarian berupa dokumen *hypertext* virtual (3) yang menunjuk pada dokumen yang ditemukan (4) (Berners-Lee dkk., 1992).

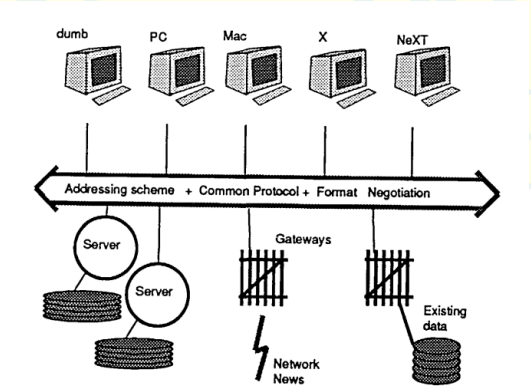
Terdapat fitur-fitur yang ditawarkan pada WWW yaitu:

- Informasi hanya direpresentasikan sekali, sebuah rujukan (*reference*) dibuat menggantikan salinan informasi.
- *Link* memungkinkan untuk topologi informasi terus berkembang, sehingga dapat memodelkan pengetahuan manusia setiap saat tanpa kendala.
- Web berisi hal yang beragam mulai dari catatan kecil pada sebuah ruang kerja sampai pada basis data raksasa di benua lain.
- Indeks berasal dari dokumen dan dapat dicari atau melalui penelusuran *link*. Sebuah indeks ditampilkan ke pengguna sebagai "halaman sampul" yang mendeskripsikan data yang diindeks dan properti dari *search engine*.
- Dokumen pada web tidak harus memiliki bentuk fisik layaknya *file*. Dokumen bisa berbentuk virtual yang dibentuk oleh server sebagai respon dari sebuah pencarian atau nama dokumen. Akibatnya dokumen dapat berbentuk *views* dari basis data, atau *snapshot* perubahan data.

Kebanyakan pengembangan tentang sistem *hypertext* pada saat itu

kebanyakan hanya berfokus pada *User Interface* (UI) dan penulisan pertanyaan alih-alih pengembangan sistem yang bersifat luas (*wide-area*) dan distribusi informasi dalam jangka panjang. Akibatnya, arsitektur pada sistem tersebut mengasumsikan bahwa pengguna menggunakan program aplikasi komputer yang sama pada sistem *file* yang sama. Berbeda dengan WWW yang bersifat global, sehingga dihadapi dengan komputer pengguna terdistribusi dan beragam dengan tipe perangkat yang berbeda-beda. Hal ini membuat arsitektur WWW mengadopsi model klien-server. Klien bertugas memproses sebuah alamat dokumen menjadi sebuah dokumen dengan protokol jaringan tertentu. Sedangkan server menyediakan data dalam bentuk *hypertext* sederhana atau dalam bentuk teks biasa, atau format data lain dengan bernegosiasi dengan klien (Berners-Lee dkk., 1992).

Tantangan dari arsitektur tersebut adalah harus mengembangkan sebuah *hypertext browser*, lebih sulit daripada mengembangkan tampilan *front-end* pada sistem informasi tertentu. Walaupun demikian, memisahkan program klien dan server dengan "*information bus*" akan terbayar ketika semakin banyak klien dan server bermunculan dan pembacaan universal tercapai. Menulis kode untuk sebuah server untuk data secara umum lebih sederhana karena tidak membutuhkan UI (Berners-Lee dkk., 1992).



Gambar 2.2: Gambar arsitektur WWW *link* dan indeks (Berners-Lee dkk., 1992)

Terdapat beberapa protokol yang dipakai WWW, *File Transfer Protocol* (FTP), *Network New Transfer Protocol* (NNTP), akses ke sistem *file* terpasang (*mounted*). Sebuah protokol baru yang bersifat cari dan dapatkan (*search and retrieve* yang disebut HTTP, juga dianggap penting dan digunakan WWW. Lebih cepat dari FTP untuk menarik dokumen, HTTP juga memungkinkan untuk pencarian indeks. HTTP mirip dengan implementasi protokol internet yang disebutkan sebelumnya dan mirip dengan fungsionalitas protokol WAIS.

Saat ini WWW terbukti sukses dalam mewujudkan cita-citanya dalam memudahkan akses informasi global ke pengguna. Terdapat lebih dari 1 miliar situs web yang terindeks WWW (Huss, 2022) dan angka tersebut akan terus bertambah. Selain itu juga, terdapat miliaran pengguna internet yang secara otomatis juga merasakan manfaat WWW ketika melakukan pencarian di *browser*. Belum lagi menyebutkan teknologi turunan yang muncul karena WWW, seperti *web development*, *search engine*, *web API*, dan masih banyak lagi.

B. *Search Engine*

Search engine adalah program perangkat lunak yang memungkinkan untuk mencari kumpulan situs web berdasarkan kata kunci yang dimasukkan oleh pengguna. *Search engine* lalu mencocokkan kata pencarian dengan basis data yang dimiliki. *Search engine* adalah contoh sistem pengambilan informasi berskala masif (Seymour dkk., 2011).

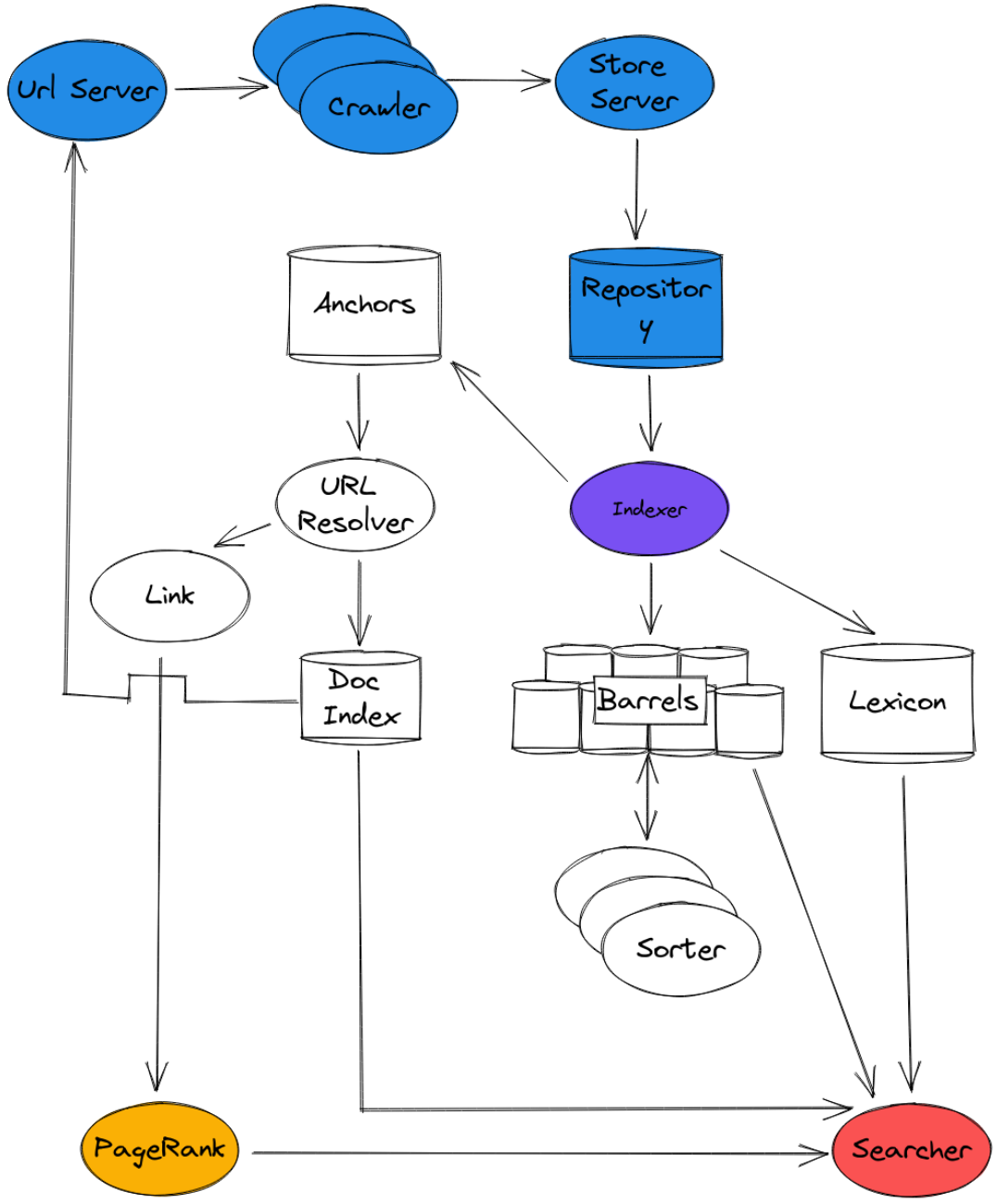
Sejarah dimulai pengembangan *search engine* dimulai pada tahun 1990. Alat pertama yang digunakan untuk mencari di internet adalah Archie. Archie dibuat pada 1990 oleh Alan Emtage, seorang mahasiswa di universitas McGill di Montreal. Basis data Archie terdiri atas direktori *file* dari ratusan sistem. Ketika pengguna mencari di basis data Archie dalam bentuk nama *file*, Archie dapat memberi tahu

lokasi dari salinan *file* tersebut. Archie tidak mengindeks konten dari *file* tersebut. Pada periode tertentu, Archie mengunjungi situs - situs FTP terbuka yang diketahui, membuat daftar *file* tersebut, dan membuat indeks yang dapat dicari. Perintah-perintah yang digunakan untuk Archie adalah perintah UNIX, sehingga untuk bisa menggunakan Archie, pengguna harus memiliki pengetahuan tentang UNIX (Seymour dkk., 2011).

Pada tahun 1994, seorang mahasiswa *Computer Science Engineer* Universitas Washington, membuat WebCrawler pada waktu senggangnya. Awalnya WebCrawler merupakan aplikasi Desktop, bukan layanan web seperti saat ini. WebCrawler hidup di atas web dengan basis data berisi halaman dari 4000 situs web berbeda. WebCrawler merupakan *search engine* web pertama yang menyediakan fitur pencarian teks secara penuh. WebCrawler berhasil dibuat pada 20 April 1994 oleh Brian Pinkerton dan dibeli America Online pada 1 Juni 1995, lalu dijual lagi ke Excite pada 1 April 1997. Yang membedakan WebCrawler dengan pendahulunya adalah penggunaan robot web pertama yang mampu mengindeks setiap kata pada halaman web, menyimpan URL, dan sebuah judul maksimal 100 kata (Seymour dkk., 2011).

AltaVista, dibuat pada tahun 1995, pernah menjadi *search engine* yang paling populer sebelum naiknya Google. *Crawler* yang dipakai AltaVista dibuat oleh Louis Moner, dan yang membuat pengindeks adalah Michael Burrows. AltaVista merupakan *search engine* tercepat pada masanya yang dapat menangani jutaan *hit* tiap harinya tanpa adanya penurunan performa. Satu hal yang sangat membedakan AltaVista dengan *search engine* lain pada masa itu adalah mampu memproses bahasa natural sebagai kata masukan untuk mencari web. Pengguna dapat menulis kalimat atau pertanyaan untuk mendapatkan respon pintar. Sebagai contoh, pengguna dapat menulis “Dimana London ?” tanpa mendapatkan jutaan hasil

pencarian yang tidak diinginkan karena mengandung “di” atau “mana” (Seymour dkk., 2011).



Gambar 2.3: *High Level Google Architecture* (Brin dan Page, 1998)

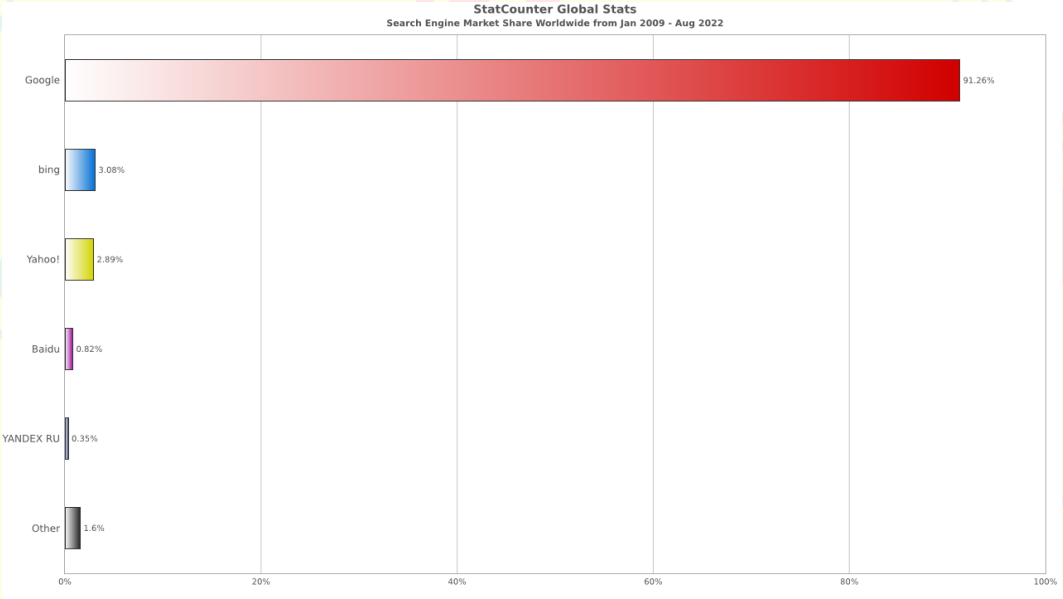
Google dibuat pada tahun 1998, oleh Lawrence Page dan Sergey Brin. Pada saat itu, metode utama untuk menelusuri WWW adalah dengan menggunakan

search engine atau melalui situs kumpulan indeks berkualitas tinggi yang disusun oleh manusia, yang pada saat itu, yang paling populer adalah Yahoo!. Kedua metode tersebut memiliki permasalahan. Yahoo! walaupun mampu menyajikan topik - topik populer tetapi bersifat sangat subjektif, memiliki biaya yang mahal karena membutuhkan tenaga manusia, dan tidak bisa mencakup topik-topik yang bersifat esoterik (Brin dan Page, 1998). Sedangkan untuk *search engine* otomatis yang berlandaskan pada pencocokkan kata kunci biasanya menampilkan halaman web berkualitas rendah. Untuk memperburuk keadaan, pengiklan berusaha menarik perhatian pengguna dengan memanfaatkan kelemahan *search engine*. Google dibuat dengan harapan untuk menjawab masalah - masalah tersebut.

Google diharapkan dapat menjadi *search engine* yang *scalable*, karena halaman web yang terus bertumbuh pesat. Dibutuhkan teknologi *crawling* yang cepat untuk bisa mengumpulkan dokumen web dan memperbaharuinya. Perangkat penyimpanan harus digunakan seefisien mungkin untuk menyimpan indeks, dan jika memungkinkan, menyimpan dokumen web itu sendiri. Sistem pengindeksan harus bisa memproses ratusan *Giga Byte* data secara efisien. Pemrosesan kueri harus ditangani secepat mungkin pada tingkat ratusan atau ribuan kueri per-detik. Hal ini juga didukung dengan semakin cepat, besar, dan murahnya perangkat keras komputer dari waktu ke waktu.

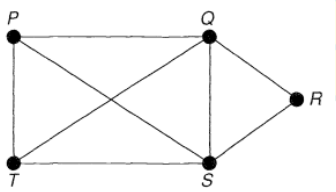
Selain *scalable*, Google juga diharapkan menghasilkan hasil pencarian yang lebih berkualitas. Pada saat Google dikembangkan, hasil pencarian *search engine* dipenuhi dengan hasil "sampah" karena hanya mengandalkan indeks tanpa menilai apakah halaman web yang ditampilkan memenuhi keinginan pengguna. Pengguna tidak dapat mengecek satu-per-satu dari ratusan sampai ribuan hasil pencarian yang ditampilkan. Algoritma Pagerank dikembangkan untuk Google digunakan untuk merangking seberapa penting halaman web berdasarkan struktur *link* graf WWW.

Sampai saat ini, *search engine* baru terus bermunculan. Pada 2004, Yahoo! yang sebelumnya terkenal sebagai direktori web, alih-alih sebagai *search engine* otomatis, meluncurkan *search engine* sendiri dengan menggabungkan fitur-fitur beberapa *search engine* yang mereka akuisisi. Dilanjutkan pada tahun 2005 MSN Search, dan pada tahun 2009 Bing oleh Microsoft (Seymour dkk., 2011). Walaupun demikian, Google tetap mendominasi pasar *search engine* hingga saat ini (lihat gambar 2.4).

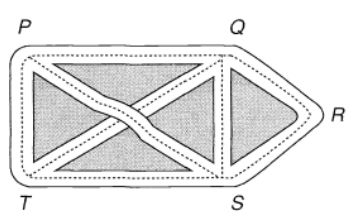


Gambar 2.4: Pangsa pasar *search engine* (GlobalStatCounter, 2022)

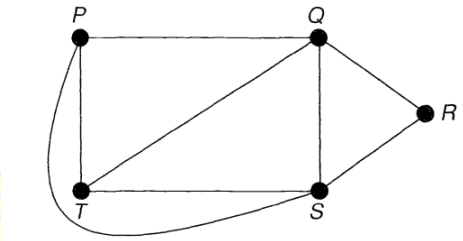
C. Teori Graf



Gambar 2.5: Contoh graf (Wilson, 1996)

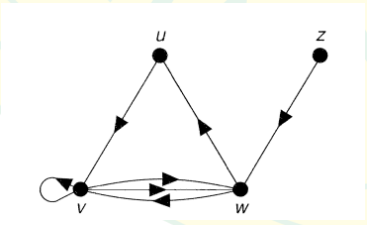


Gambar 2.6: Contoh peta jalan yang dapat diandaikan sebagai graf (Wilson, 1996)

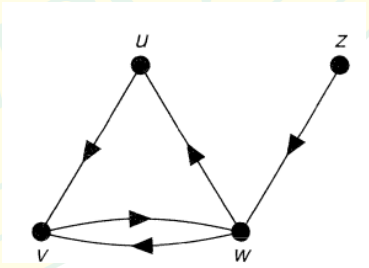


Gambar 2.7

Graf adalah sebuah representasi dari himpunan titik (*node / vertice*) dan bagaimana titik-titik tersebut saling terhubung tanpa memperdulikan sifat metriknya (Wilson, 1996). Pada gambar 2.5 merupakan contoh graf, dengan P, Q, R, S, T merupakan titik, dan masing-masing terhubung dengan garis (*edge*). Garis yang menghubungkan titik P dan S disebut dengan PS, sedangkan garis yang menghubungkan titik Q dan T disebut dengan QT. Persilangan antara PS dan QT tidak disebut sebagai titik, karena keduanya tidak saling bersilangan, melainkan saling melompati layaknya gambar 2.6. Selanjutnya, kedua graf dianggap sama, jika dan hanya jika titik yang berkorespondensi sama-sama terhubung dengan garis yang sama dengan garis pada graf lainnya (Wilson, 1996). Sebagai contoh, graf pada gambar 2.7 merupakan graf yang sama dengan graf pada gambar 2.5 (Wilson, 1996).



Gambar 2.8: Contoh digraf (Wilson, 1996)



Gambar 2.9: Contoh digraf sederhana (Wilson, 1996)

Garis pada graf dapat diberikan arah. Garis pada graf yang berarah disebut sebagai busur (*arc*). Graf yang memiliki arah pada garisnya disebut dengan graf

berarah (*directed graph / digraph* / digraf) yang terdiri atas himpunan titik dan busur (Wilson, 1996). Pada digraf di gambar 2.8 terdapat himpunan titik u, v, w, dan z, dengan busur uv, vv, $vw(2\times)$, wv, wu, dan zw. Sebuah digraf disebut sebagai digraf sederhana jika himpunan busurnya tidak ada yang sama (*distinct*) dan tidak memiliki *loop* (contoh: busur vv) (Wilson, 1996). Digraf pada gambar 2.9 adalah contoh digraf sederhana.

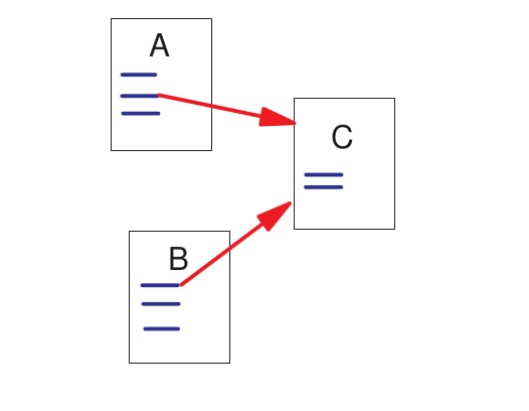
D. Pagerank

World Wide Web memberikan tantangan baru dalam hal memperoleh informasi karena besar dan beragam isi-nya. Terdapat miliaran halaman web saat ini. Halaman web tersebut sangat beragam mulai dari "Apa yang Joe makan hari ini ?" sampai ke jurnal tentang *information retrieval*. Belum lagi tantangan lain *search engine* harus menghadapi pengguna yang kurang berpengalaman dan banyaknya halaman web yang sudah dibuat sedemikian rupa untuk memanipulasi fungsi perangkingan yang dipakai *search engine*.

Walaupun demikian, tidak seperti dokumen biasa, halaman web berisi *hypertext* dan menyediakan banyak informasi tambahan pada setiap teks yang ada pada halaman web, seperti struktur *link* dan *link* teks. Pagerank memanfaatkan keuntungan struktur *link* pada web untuk menghasilkan sebuah *ranking* seberapa penting pada setiap halaman web secara global. *Ranking* ini membantu *search engine* dan pengguna untuk menelusuri luas dan beragamnya World Wide Web.

Jika World Wide Web diibaratkan pada sebuah graf berarah, halaman web adalah titik graf, *link* adalah garis. Lalu *link* yang menunjuk keluar dari titik disebut *forward link*, sedangkan *link* yang menunjuk kedalam titik disebut *backlink* (Lihat gambar 2.10). Sangat sulit untuk mengetahui semua *backlink* suatu halaman web, tetapi sangat mudah untuk mengetahui semua *forward link* suatu halaman web yaitu

dengan cara mengunduh halaman web tersebut (Page dkk., 1999). Setiap halaman web memiliki jumlah *backlink* yang beragam. Pada saat Pagerank diteliti, halaman *home* NetScape memiliki 62.804 *backlink* dibandingkan halaman web kebanyakan yang hanya memiliki beberapa *backlink*. Secara umum suatu halaman web jika memiliki banyak *backlink* dapat dianggap lebih penting daripada halaman web lain yang memiliki lebih sedikit *backlink*. Perhitungan jumlah sitasi sederhana pernah digunakan untuk memprediksi pemenang Nobel masa depan (Page dkk., 1999).



Gambar 2.10: A dan B adalah *backlink* dari C (Page dkk., 1999)

Yang membuat Pagerank menarik adalah sekedar menghitung jumlah sitasi atau *backlink* saja tidak cukup untuk membuat *ranking* halaman web sesuai dengan apa yang pengguna anggap sebagai penting. Sebagai contoh, jika ada suatu halaman web yang memiliki *backlink* dari situs terkenal, misal halaman *home* Yahoo.com, mungkin itu hanya satu *link* tetapi berasal dari halaman yang penting. Halaman tersebut seharusnya memiliki *ranking* di atas halaman web yang memiliki banyak *backlink* tetapi berasal dari tempat yang tidak jelas. Pagerank berusaha untuk mewujudkan perangkingan yang selaras dengan makna penting di mata pengguna hanya dengan menggunakan struktur *link* (Page dkk., 1999).

Pagerank dapat didefinisikan pada persamaan 1.1. Halaman web merupakan u. Himpunan F_u adalah kumpulan halaman u yang menunjuk halaman lain atau

disebut dengan *forward link* dan B_u adalah himpunan halaman yang menunjuk ke u atau disebut dengan *backlink*. $C_u = |F_u|$ adalah jumlah *link* dari u, sedangkan c adalah faktor yang digunakan untuk normalisasi (sehingga jumlah total *ranking* semua halaman web adalah konstan) dan $c < 1$. $E(u)$ adalah vektor yang berkorespondensi dengan *ranking* halaman web. $||\pi'||_1 = 1$. Iterasi perhitungan terus dilakukan sampai konvergen. Jika diubah kedalam persamaan matriks, maka persamaan 1.1 dapat diubah menjadi persamaan 1.2. Dimana X adalah matriks persegi yang baris dan kolomnya berkorespondensi dengan halaman web, dengan elemen $X_{u,v} = \frac{1}{C_v}$ jika terdapat *link* pada halaman v ke halaman u atau $X_{u,v} = 0$ jika tidak ada.

Berdasarkan persamaan 1.2, Algoritma Pagerank secara sederhana dapat didefinisikan pada algoritma 1. Vektor ranking awal-awal dapat didefinisikan sebagai vektor apapun yang berkorespondensi dengan halaman web (misal E).

Algoritma 1 Algoritma Pagerank (Page dkk., 1999)

1: $\pi_0 \leftarrow$ vektor ranking awal-awal (misal E)
2: **do**
3: $\quad \pi_{i+1} \leftarrow X\pi_i$
4: $\quad \delta \leftarrow ||\pi_i||_1 - ||\pi_{i+1}||_1$
5: $\quad \pi_{i+1} \leftarrow \pi_{i+1} + \delta E$
6: **while** $||\pi_{i+1} - \pi_i||_1 > \epsilon$

E. *Distributed Pagerank Computation* (DPC)

Telah dijelaskan sebelumnya, masalah dari algoritma Pagerank biasa adalah besarnya memori utama yang dibutuhkan untuk bisa menyimpan matriks X (lihat persamaan 1.2). Oleh karena itu, dirumuskan algoritma Pagerank terdistribusi. DPC, dirumuskan oleh Zhu dkk. (2005), memakai mekanisme interaksi sederhana antara

cluster dan lalu lintas komunikasi rendah. Ditinjau dari perspektif matematika, dibuktikan bahwa algoritma DPC setara dengan metode *Iterative Aggregation-Disaggregation* (IAD) dengan *Block Jacobi smoothing*. DPC juga memiliki keunggulan dibandingkan dengan algoritma Pagerank biasa yaitu, matriks-matriks dipecah menjadi matriks agregasi dan matriks lokal sehingga ukurannya cukup kecil untuk disimpan di memori utama, sehingga mempercepat komputasi karena tiap iterasi memerlukan sedikit operasi I/O pada *disk*. Selanjutnya, Vektor Pagerank lokal konvergen lebih cepat pada beberapa *cluster* tertentu, berbeda dengan Pagerank biasa karena terdapat komputasi tidak perlu pada *cluster* yang sudah konvergen (Zhu dkk., 2005).

1. Algoritma Pagerank versi DPC

Secara esensi algoritma Pagerank pada artikel asli Page dkk. (1999), dan algoritma Pagerank yang dipakai pada algoritma DPC pada artikel Zhu dkk. (2005) adalah sama. Hanya terdapat beberapa penyesuaian. Pertama vektor E, merupakan probabilitas *random walker* melompat ke halaman web lain secara acak, diganti dengan $(1-d)$. d disebut sebagai *damping factor* merupakan probabilitas *random walker* mengikuti *link* yang tersedia. Nilai d pada penelitian Zhu dkk. (2005) adalah 0.85. Yang kedua, jika vektor E pada algoritma Pagerank di artikel Page dkk. (1999) digunakan pada langkah tersendiri, sedangkan nilai d pada artikel Zhu dkk. (2005) dipakai langsung dalam menentukan nilai elemen pada matriks transisi.

$$P_{ij} = \begin{cases} \frac{d}{C_j} + \frac{(1-d)}{N} & j \to i \\ \frac{(1-d)}{N} & j \not\to i \text{ dan } C_j \neq 0 \\ \frac{1}{N} & C_j = 0 \end{cases} \tag{2.1}$$

Matriks transisi pada algoritma DPC didefinisikan sebagai matriks P. Matriks

P didefinisikan pada persamaan 2.1. C_j adalah jumlah *forward link* dari halaman j. $j \rightarrow i$ adalah halaman j memiliki *link* menuju halaman i.

Selanjutnya algoritma Pagerank yang dipakai DPC didefinisikan pada algoritma 2

Algoritma 2 Algoritma Pagerank yang dipakai DPC (Zhu dkk., 2005)

1: Definisikan π^0 awal-awal
2: $k \leftarrow 0$
3: $\pi^{\sim k+1} \leftarrow P\pi^k$
4: $\pi^{k+1} \leftarrow \frac{\pi^{\sim k+1}}{||\pi^{\sim k+1}||_1}$
5: Jika $||\pi^{k+1} - \pi^k|| < \epsilon$ berhenti dan kembalikan nilai π^{k+1}
6: $k \leftarrow k + 1$
7: Ulangi langkah 3

2. Algoritma IAD

Jika *link* pada kumpulan web dibuat kedalam graf, maka graf tersebut akan memiliki sebuah struktur menyerupai blok, karena mayoritas dari *link* tersebut bersifat *intra-host*, merujuk halaman yang masih di dalam *host* yang sama. Oleh karena itu, jika dilakukan perjalanan acak pada kumpulan web tersebut dapat dilihat sebagai rantai Markov *Nearly Completely Decomposable* (NCD) (Zhu dkk., 2005). Rantai Markov NCD adalah rantai Markov yang dapat dipartisi sehingga peluang dari keadaan awal ke keadaan selanjutnya lebih sering menunjuk keadaan yang berada di dalam partisinya dibandingkan di luar partisinya (Kontovasilis dan Mitrou, 1995). Sebelum dijelaskan tentang DPC, akan dijelaskan metode IAD terlebih dahulu.

Misal G adalah himpunan bilangan bulat $\{1, ..., N\}$. Misal $G_1, ...G_n, n \leq N$ adalah grup agregasi dari elemen-elemen di G. Himpunan-himpunan $G_i, i = 1, ..., n$,

adalah saling lepas dan $\cup_{i=1}^{n} G_i = G$. Misal N_i adalah ordo dari himpunan G_i, atau jumlah elemen-elemen di G_i (Zhu dkk., 2005).

Misal R adalah matriks agregasi $n \times N$, yang memenuhi persamaan 2.2 (Zhu dkk., 2005).

$$R_{ij} = \begin{cases} 1 & j \in G_i \\ 0 & \text{lainnya} \end{cases} \tag{2.2}$$

Dilakukan partisi pada vektor positif π menjadi $(\pi_1^T, \pi_2^T, ..., \pi_n^T)^T$ berdasarkan $\{G_i\}$. π_i adalah subvektor dengan dimensi N_i (Zhu dkk., 2005).

Maka dapat didefinisikan matriks disagregasi $S(\pi)$ $N \times n$ sebagai persamaan 2.3 (Zhu dkk., 2005).

$$S(\pi) = \begin{pmatrix} S(\pi)_1 & 0 & \dots & 0 \\ 0 & S(\pi)_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & S(\pi)_n \end{pmatrix} \tag{2.3}$$

Dimana $S(\pi)_i$ adalah sebuah kolom vektor yang mewakili *censored stationary distribution* dari halaman-halaman di *cluster* G_i. Ingat bahwa $RS(\pi) = I$ (Zhu dkk., 2005).

Algoritma 3 Algoritma IAD (Zhu dkk., 2005)

1: $||\pi^0|| \leftarrow 1$

2: $\pi^0 \leftarrow$ Pilih bilangan bulat positif

3: $k \leftarrow 0$

4: **do**

5: Buat matriks agregasi $RPS(\pi^k)$ dan selesaikan persamaan linear 2.4.

Dimana $||z|| = 1$

$$RPS(\pi^k)z^k = z^k \tag{2.4}$$

6: $\pi^{\sim k+1} \leftarrow TS(\pi^k)z^k$

7: $\pi^{k+1} \leftarrow \frac{\pi^{\sim k+1}}{||\pi^{\sim k+1}||_1}$

8: $k \leftarrow k + 1$

9: **while** $||\pi^{k+1} - \pi^k|| < \epsilon$

Misal $T = M^{-1}N$ adalah sebuah matriks berasal dari operasi pemisahan matriks dari $I - P = M - N$. Dimana M adalah matriks non-singular (bisa dilakukan invers) dan operasi pemisahan matriksnya adalah *weak regular*, yang berarti $M^{-1} \geq 0$ dan $M^{-1}N \geq 0$ (Mishra, 2016). Untuk menyelesaikan persamaan linear $(I - P)\pi = 0$, maka dapat dirumuskan algoritma Iteration Aggregation Disaggregation (IAD) yang dapat dilihat pada algoritma 3 (Zhu dkk., 2005).

3. Algoritma DPC

Sebelum langsung membahas algoritma DPC, akan didefinisikan beberapa notasi terlebih dahulu. $e = (1, ..., 1)^T$. Matriks transisi P dipartisi menjadi beberapa blok berdasarkan $\{G_i\}$ menjadi seperti persamaan 2.5 (Zhu dkk., 2005).

$$P = \begin{pmatrix} P_{11} & P_{12} & \dots & P_{1n} \\ P_{21} & P_{22} & \dots & P_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & \dots & P_{nn} \end{pmatrix} \tag{2.5}$$

Dilambangkan blok baris ke-i sebagai persamaan 2.6

$$P_{i*} \triangleq (P_{i1}, \dots, P_{in}) \tag{2.6}$$

dan dilambangkan blok kolom ke-i sebagai persamaan 2.7

$$P_{*i} \triangleq \begin{pmatrix} P_{1i} \\ \vdots \\ P_{ni} \end{pmatrix} \tag{2.7}$$

Setiap blok diagonal P_{ii} adalah matriks persegi dan merupakan matriks *intra-cluster link* dari *cluster* G_i, sementara blok-blok di luar diagonal merupakan struktur *link* antar-*cluster*. Selanjutnya, matriks agregat $A = RPS(\pi)$ adalah matriks transisi antar *cluster*. Maka dapat dibuat algoritma DPC pada algoritma 4 (Zhu dkk., 2005).

Algoritma 4 Algoritma DPC (Zhu dkk., 2005)

1: Buat matriks transisi lokal untuk tiap *cluster* G_i berdasarkan P

$$Q_i = LocalTransitionMatrix(G_i) \forall G_i \in G \tag{2.8}$$

2:

$$\pi_i^0 = Pagerank(Q_i, \frac{e}{N_i}, \epsilon) \forall G_i \in G \tag{2.9}$$

Ket: $e = [1, 1, ..., 1]^T$; $N_i \rightarrow$ jumlah anggota G_i

3: Inisialisasi $k = 0$

4:

$$A^k = RPS(\pi^k) \tag{2.10}$$

Ket: $R \rightarrow$ persamaan 2.2; $P \rightarrow$ persamaan 2.1; $S(\pi) \rightarrow$ persamaan 2.3

5:

$$z^k = Pagerank(A^k, \frac{e}{n}, \epsilon) \tag{2.11}$$

Ket: $n \rightarrow$ banyaknya anggota G

6: $\forall G_i \in G$ buat sebuah *extended local transition matrix* $(N_i + 1) \times (N_i + 1)$.

Dimana skalar α^k membuat jumlah nilai kolom dari B_i^k adalah satu

$$B_i^k = \begin{pmatrix} P_{ii} & \frac{(P_{i*}S(\pi^k)z^k - P_{ii}\pi_i^k z_i)}{(1-z_i^k)} \\ e^T P_{*i} & \alpha^k \end{pmatrix} \tag{2.12}$$

7: Hitung vektor *extended local Pagerank*. Dimana β_i^{k+1} adalah skalar

$$\begin{pmatrix} \omega_i^{k+1} \\ \beta_i^{k+1} \end{pmatrix} = Pagerank(B_i^k, \frac{e}{(N_i+1)}, \epsilon) \tag{2.13}$$

8:

$$\pi_i^{\sim k+1} = \frac{1-z_i^k}{\beta_i^{k+1}} \omega_i^{k+1} \tag{2.14}$$

9:

$$\pi^{k+1} = \frac{\pi^{\sim k+1}}{||\pi^{\sim k+1}||_1} \tag{2.15}$$

10: $k = k+1$

11: Jika persamaan 2.16 terpenuhi, berhenti dan berikan π^k sebagai hasil akhir. Jika sebaliknya, kembali ke langkah 2.10

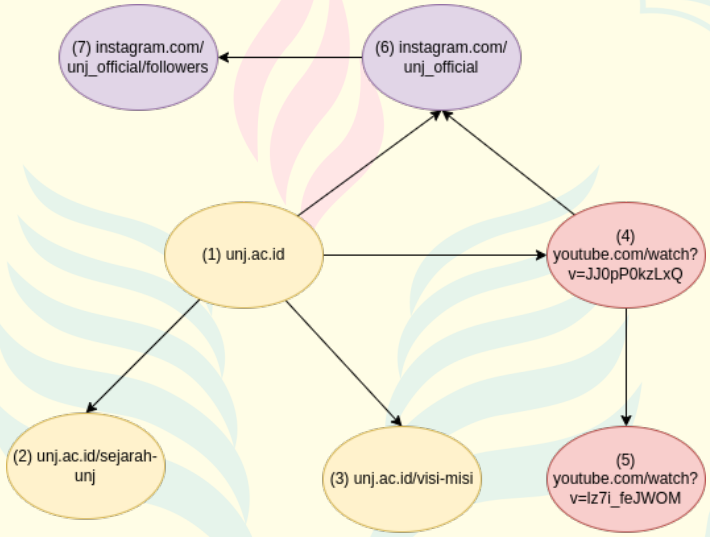
$$||\pi^{k+1} - \pi^k|| < \epsilon \tag{2.16}$$

---

4. Contoh Nilai dari tiap simbol algoritma DPC

Akan diberikan contoh nilai dari tiap simbol algoritma DPC yang sudah dijelaskan sebelumnya. Misal diberikan graf berarah halaman web pada gambar 2.11. Jika dibuat matriks P, berdasarkan persamaan 2.1, dan jika $d = 0,85$, maka matriks P yang terbentuk akan menjadi seperti matriks 2.17. Nilai dari elemen $P_{1,1}$ diperoleh dari perhitungan $\frac{1-d}{N} = \frac{1-0,85}{7} = 0,02143$, karena antara halaman 1

("unj.ac.id") tidak memiliki *forward link* ke dirinya sendiri. Nilai dari elemen $P_{2,1}$ diperoleh dari perhitungan $\frac{d}{C_j} + \frac{1-d}{N} = \frac{0{,}85}{5} + \frac{1-0{,}85}{7} = 0,23393$ karena antara halaman 1 memiliki *forward link* ke halaman 2 ("unj.ac.id/sejarah-unj"). Nilai elemen $P_{1,2}$ sampai elemen $P_{7,2}$ diperoleh dari perhitungan $\frac{1}{N} = \frac{1}{7} = 0,14286$, karena halaman 2 tidak memiliki *forward link* ke halaman manapun.



Gambar 2.11

$$P = \begin{pmatrix} 0,02143 & 0,14286 & 0,14286 & 0,02143 & 0,14286 & 0,02143 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 & 0,02143 & 0,14286 & 0,02143 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 & 0,02143 & 0,14286 & 0,02143 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 & 0,02143 & 0,14286 & 0,02143 & 0,14286 \\ 0,02143 & 0,14286 & 0,14286 & 0,44643 & 0,14286 & 0,02143 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 & 0,44642 & 0,14286 & 0,02143 & 0,14286 \\ 0,02143 & 0,14286 & 0,14286 & 0,02143 & 0,14286 & 0,87143 & 0,14286 \end{pmatrix} \quad (2.17)$$

Dari matriks P sebelumnya, maka dapat dipecah layaknya pada persamaan 2.5, menjadi matriks 2.19.

$$P = \begin{pmatrix} P_{1,1} & P_{1,2} & P_{1,3} \\ P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,1} & P_{3,2} & P_{3,3} \end{pmatrix} \tag{2.18}$$

$$P = \begin{pmatrix} \begin{pmatrix} 0,02143 & 0,14286 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 \end{pmatrix} & \begin{pmatrix} 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \end{pmatrix} & \begin{pmatrix} 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \end{pmatrix} \\ \begin{pmatrix} 0,23393 & 0,14286 & 0,14286 \\ 0,02143 & 0,14286 & 0,14286 \end{pmatrix} & \begin{pmatrix} 0,02143 & 0,14286 \\ 0,44643 & 0,14286 \end{pmatrix} & \begin{pmatrix} 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \end{pmatrix} \\ \begin{pmatrix} 0,23393 & 0,14286 & 0,14286 \\ 0,02143 & 0,14286 & 0,14286 \end{pmatrix} & \begin{pmatrix} 0,44642 & 0,14286 \\ 0,02143 & 0,14286 \end{pmatrix} & \begin{pmatrix} 0,02143 & 0,14286 \\ 0,87143 & 0,14286 \end{pmatrix} \end{pmatrix} \tag{2.19}$$

Dari matriks $P_{1,1}$, $P_{2,2}$, dan $P_{3,3}$ dapat diperoleh matriks transisi lokal Q_1, Q_2, dan Q_3 yang tiap kolomnya dinormalisasi yang dapat dilihat pada matriks 2.20.

$$Q_1 = \begin{pmatrix} 0,0438 & 0,33333 & 0,33333 \\ 0,4781 & 0,33333 & 0,33333 \\ 0,4781 & 0,33333 & 0,33333 \end{pmatrix} Q_2 = \begin{pmatrix} 0,0458 & 0,5 \\ 0,9542 & 0,5 \end{pmatrix} \quad Q_3 = \begin{pmatrix} 0,024 & 0,5 \\ 0,976 & 0,5 \end{pmatrix} \tag{2.20}$$

Selanjutnya dari graf di gambar 2.11 dapat ditemukan 3 domain, yaitu domain 1 adalah "unj.ac.id" dengan anggota halaman "unj.ac.id", "unj.ac.id/sejarah-unj", dan "unj.ac.id/visi-misi" yang dapat dinotasikan sebagai $G_1 = \{1, 2, 3\}$. Domain 2 adalah "youtube.com" dengan anggota halaman "youtube.com/watch?v=JJ0pP0kzLxQ" dan "youtube.com/watch?v=lz7i_feJWOM"

yang dapat dinotasikan sebagai $G_2 = \{4, 5\}$. Yang terakhir domain 3 "instagram.com" dengan anggota halaman "instagram.com/unj_official" dan "instagram.com/unj_official/followers" $G_3 = \{6, 7\}$. Jika dibuat matriks R maka dapat dilihat pada matriks 2.21. Elemen $R_{1,1}$ sampai elemen $R_{1,3}$ mendapat nilai 1 karena halaman 1 sampai 3 merupakan anggota G_1, sedangkan elemen $R_{1,4}$ sampai elemen $R_{1,7}$ mendapat nilai 0 karena halaman 4 sampai 7 bukan anggota G_1.

$$R = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \tag{2.21}$$

Pada setiap halaman web tersebut, pada tiap anggota domain-nya dapat dijalankan algoritma Pagerank. Setelah konvergen, keluaran dari algoritma Pagerank pada tiap domainnya adalah beberapa vektor $S(\pi)_1 = \{0,25974; 0,37013; 0,37013\}$, $S(\pi)_2 = \{0,35088; 0,64912\}$, dan $S(\pi)_3 = \{0,35088; 0,64912\}$. Jika dibuat matriks $S(\pi)$ maka akan terbentuk matriks 2.22.

$$S(\pi) = \begin{pmatrix} 0,25974 & 0 & 0 \\ 0,37013 & 0 & 0 \\ 0,37013 & 0 & 0 \\ 0 & 0,35088 & 0 \\ 0 & 0,64912 & 0 \\ 0 & 0 & 0,35088 \\ 0 & 0 & 0,64912 \end{pmatrix} \tag{2.22}$$

Setelah diperoleh matriks R, P, $S(\pi)$ maka dapat diperoleh matriks A, dengan mengalikan ketiga matriks tersebut, sehingga memperoleh matriks 2.23.

$$A = \begin{pmatrix} 0,44434882 & 0,30075792 & 0,30075792 \\ 0,27783429 & 0,34962928 & 0,20050528 \\ 0,27783429 & 0,34962577 & 0,49875328 \end{pmatrix} \tag{2.23}$$

Selanjutnya diperoleh vektor z dengan melakukan perhitungan Pagerank sampai konvergen dengan memasukan matriks A sebagai matriks transisi, dan vektor $\frac{e}{n} = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]^T$ sebagai vektor *ranking* awal-awal. Hasil dari vektor z dapat dilihat pada vektor 2.24.

$$z = \begin{pmatrix} 0,35118 \\ 0,26756 \\ 0,38127 \end{pmatrix} \tag{2.24}$$

Dari vektor dan matriks sebelumnya dapat dibentuk matriks B_i. Karena terdapat 3 domain, maka matriks B_i yang terbetuk adalah matriks B_1, B_2, dan B_3. Proses pembentukan matriks B_1 dapat dilihat pada persamaan 2.25.

$$\begin{aligned} B_1 &= \begin{pmatrix} P_{1,1} & \frac{P_{1,*} \times S(\pi) \times z - P_{1,1} \times \pi_1 \times z_1}{1 - z_1} \\ e^T \times P_{*,1} & \alpha \end{pmatrix} \\ &= \begin{pmatrix} \begin{pmatrix} 0,02143 & 0,14286 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 \end{pmatrix} & \begin{pmatrix} 0,10025 \\ 0,10025 \\ 0,10025 \end{pmatrix} \\ \begin{pmatrix} 1,00001 & 1,00002 & 1,00002 \end{pmatrix} & \begin{pmatrix} 0,69924 \end{pmatrix} \end{pmatrix} \end{aligned} \tag{2.25}$$

$$= \begin{pmatrix} 0,02143 & 0,14286 & 0,14286 & 0,10025 \\ 0,23393 & 0,14286 & 0,14286 & 0,10025 \\ 0,23393 & 0,14286 & 0,14286 & 0,10025 \\ 1,00001 & 1,00002 & 1,00002 & 0,69924 \end{pmatrix}$$

Dari *extended local matrix* B_1 dapat diperoleh *local pagerank vector* untuk domain 1. Vektor yang diperoleh dapat dilihat pada vektor 2.26. Di mana elemen baris 1 sampai baris 3 merupakan nilai vektor ω_1, sedangkan baris 4 merupakan nilai β_1.

$$\begin{pmatrix} \omega_1 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} 0,09005 \\ 0,10689 \\ 0,10689 \\ 0,69617 \end{pmatrix} \tag{2.26}$$

5. Analisis Konvergen

Dibuktikan konvergensi metode *Block Jacobi* pada skenario Pagerank. Pertama-tama diberikan sebuah lema yang diambil dari penelitian Courtois dan Semal (1986) (Zhu dkk., 2005):

Lema 1. Iterasi

$$\pi^{k+1} = \frac{T\pi^k}{||T\pi^k||_1} \tag{2.27}$$

akan konvergen ketika kondisi-kondisi berikut terpenuhi:

- $\rho(T) = 1$
- T *irreducible*

- T asiklik

$\rho(T)$ adalah *spectral radius* dari matriks T atau merupakan nilai mutlak terbesar dari himpunan *eigenvalue* matriks T. Nilai *eigenvalue* dari matriks T dapat diperoleh dengan cara menghitung determinan dari $T - \lambda I$. Dimana λ adalah vektor atau himpunan *eigenvalue*.

Neumann dan Plemmons (1978) membuktikan bahwa iterasi matriks yang diturunkan dari *weak regular splitting* matriks $I - P$ akan memenuhi syarat pertama dan kedua dari Lema 1 jika matriks P stokastik dan *irreducible*.

Misal D adalah blok diagonal dari $I - P$. Misal L adalah blok bagian segitiga bawah dari P, dan U adalah blok bagian segitiga atas dari P. Berdasarkan dari operasi pemisahan matriks $I - P = D - (L + U)$, iterasi matriks dari metode *Block Jacobi* adalah persamaan 2.28

$$T = D^{-1}(L + U) \tag{2.28}$$

Karena $(I - P_{ii})^{-1} \geq 0$ dan $(L + U) \geq 0$, maka operasi pemisahan di atas adalah *weak regular*. Karena P adalah stokastik dan *irreducible*, maka T memenuhi syarat pertama dan kedua Lema 1.

Namun, sifat asiklik dari P tidak cukup untuk menjadi sifat asiklik dari T. Untungnya skenario Pagerank juga terdapat lema lain:

Lema 2. Jika $P > 0$ adalah matriks transisi dari sebuah rantai markov dan dipartisi berdasarkan persamaan 2.5. Misal T adalah matriks iterasi yang didefinisikan pada persamaan 2.28, atau disebut dengan matriks *Block Jacobi*. T adalah asiklik jika dan hanya jika $n > 2$.

Semua syarat pada Lema 1 terpenuhi ketika $n > 2$. Akibatnya dapat dirumuskan teorema:

Teorema 1. Jika $P > 0$ adalah matriks transisi dari rantai Markov dan

dipartisi berdasarkan persamaan 2.5. Misal T adalah matriks iterasi yang didefinisikan pada persamaan 2.28, atau disebut dengan matriks *Block Jacobi*. Jika $n > 2$, maka persamaan 2.27 akan selalu konvergen tepat pada titik $\hat{x}$ dari $P\hat{x} = \hat{x}$.

Penjelasan lebih lengkap dari analisis konvergen dapat dibaca pada penelitian Zhu dkk. (2005).

6. *Communication Overhead*

Karena DPC bersifat distributif, maka dibutuhkan komunikasi bagi setiap *cluster* untuk menyatukan *ranking* halaman web. Dianalisa *communication overhead* atau ongkos memori saat komunikasi dari algoritma DPC. Pesan yang dikirim berupa vektor dan tidak ada matriks yang dikirim. Vektor v dikirim kedalam bentuk kumpulan data berbentuk *pair* yang berisi index i dan nilai $v_i > 0$. Index adalah kombinasi dari ID *cluster* dan sebuah nilai *hash* dari string URL. $Pos(.)$ melambangkan jumlah elemen positif pada sebuah vektor atau matriks. Sehingga ukuran dari pesan sebanding dengan nilai $Pos(v)$. Matriks *sparse* (matriks yang mengandung banyak nilai 0) $\bar{P}$ dipakai dibandingkan matriks P. Misal $\bar{L}$ dan $\bar{U}$ adalah blok yang terletak di segitiga bawah dan atas pada matriks $\bar{P}$ secara terpisah. Matriks $\bar{P}$ dapat didefinisikan sebagai berikut:

$$\bar{P}_{ij} = \begin{cases} \frac{d}{C_j} & j \rightarrow i \\ 0 & \text{lainnya} \end{cases} \tag{2.29}$$

Baris ke-2.8 dan ke-2.9 dari algoritma DPC membutuhkan komunikasi *trivial*. Pada baris ke-2.10 dan baris ke-2.11 algoritma DPC, *cluster* G_i mengirim koordinator dalam bentuk sebuah vektor $\bar{P}_{*i}\pi_i$, yang sama dengan kolom ke-i dari $\bar{P}S(\pi)$. Perlu dicatat bahwa subvektor ke-i dari $\bar{P}_{*i}\pi_i$ dikirim sebagai skalar $e^T\bar{P}_{ii}\pi_i$. Nilai dari lalu lintas komunikasi sebanding dengan $Pos((\bar{L}+\bar{U})S(\pi))$ yang berarti lebih kecil dari

$Pos(\bar{L} + \bar{U})$. Tabel 2.1 menunjukkan perbandingan pada graf web nyata (Zhu dkk., 2005).

Tabel 2.1: Perbandingan jumlah elemen positif pada graf web yang diuji (Zhu dkk., 2005)

Graf web	$Pos(\bar{L} + \bar{U})$ (juta)	$Pos((\bar{L} + \bar{U})S(\pi))$ (juta)
ST01	40	8
ST03	484	165
CN04	150	35

Pada baris ke-6 sampai baris ke-8, koordinator mengirim subvektor ke-i dari $\bar{P}S(\pi)z$ ke *cluster* G_i. Jadi biaya komunikasi adalah $Pos((\bar{L}+\bar{U})S(\pi)z)$, lebih kecil dari N.

Pada baris ke-9, *cluster-cluster* lokal mengirim vektor $\widetilde{\pi}_i^{k+1}$ ke koordinator yang melakukan normalisasi. Biaya komunikasinya adalah $O(N)$.

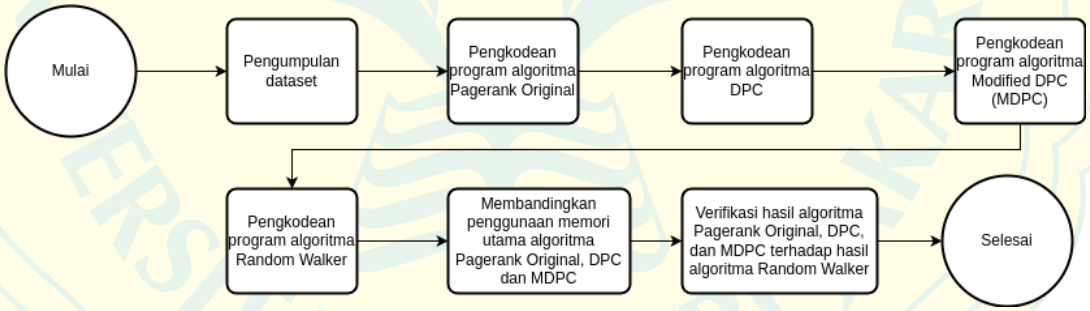
Secara keseluruhan, seluruh *communication overhead* besarnya adalah $O(Pos((\bar{L} + \bar{U})S(\pi))) + O(N)$.

BAB III

METODOLOGI PENELITIAN

A. Tahapan Penelitian

Terdapat tahapan-tahapan yang harus dilalui untuk melaksanakan penelitian ini. Tahapan penelitian dapat dilihat pada diagram 3.1. Terdapat beberapa algoritma yang belum dijelaskan sebelumnya seperti Modified DPC (MDPC) dan Random Walker. Algoritma MDPC dirumuskan karena kekurangan dari algoritma DPC yang akan dijelaskan pada bagian selanjutnya. Sedangkan algoritma Random Walker merupakan program yang mensimulasikan pergerakan kunjungan halaman web dan merupakan basis dari algoritma Pagerank (Page dkk., 1999), sehingga sangat cocok untuk dijadikan sebagai acuan untuk melakukan verifikasi hasil.



Gambar 3.1: Diagram tahapan penelitian

B. Pengumpulan Dataset

Penelitian ini menggunakan data yang berasal dari basis data penelitian Khatulistiwa (2022) ditambah dengan mengumpulkan data tambahan yang sama-sama diperoleh dengan menjalankan program *crawling*. Data yang diperoleh disimpan ke dalam basis data MySQL dan terdapat 13 tabel atau *entity* yang strukturnya dapat dilihat pada tabel-tabel berikut:

Tabel 3.1: Struktur tabel *crawling*

crawling		
No.	**Atribut**	**Tipe Data**
1.	id_crawling	int
2.	start_urls	text
3.	keyword	text
4.	total_page	int
5.	duration_crawl	time
6.	created_at	timestamp

Tabel 3.2: Struktur tabel *page_information*

page_information		
No.	**Atribut**	**Tipe Data**
1.	id_page	int
2.	crawl_id	int
3.	url	text
4.	html5	tinyint
5.	title	text
6.	description	text
7.	keywords	text
8.	content_text	text
9.	hot_url	tinyint
10.	size_bytes	bigint
11.	model_crawl	text
12.	duration_crawl	time
13.	created_at	timestamp

Tabel 3.3: Struktur tabel *tfidf_word*

tfidf_word		
No.	**Atribut**	**Tipe Data**
1.	id_word	int
2.	word	text
3.	page_id	int
4.	tfidf_score	double

Dari 13 tabel tersebut, nantinya tabel yang akan dipakai adalah tabel *page_linking*, *page_information*, dan tabel *pagerank*. Tabel *page_linking* memuat informasi dari halaman mana *link* berasal melalui atribut *page_id* dan ke mana *link* tersebut menunjuk melalui atribut *url*. Untuk tabel *page_information* atribut yang

Tabel 3.4: Struktur tabel *tfidf*

tfidf		
No.	**Atribut**	**Tipe Data**
1.	id_tfidf	int
2.	keyword	text
3.	page_id	int
4.	tfidf_total	double

Tabel 3.5: Struktur tabel *pagerank*

pagerank		
No.	**Atribut**	**Tipe Data**
1.	id_pagerank	int
2.	page_id	int
3.	pagerank_score	double

Tabel 3.6: Struktur tabel *page_linking*

page_linking		
No.	**Atribut**	**Tipe Data**
1.	id_linking	int
2.	page_id	int
3.	outgoing_link	text

Tabel 3.7: Struktur tabel *page_tables*

page_tables		
No.	**Atribut**	**Tipe Data**
1.	id_table	int
2.	page_id	int
3.	table_str	text

Tabel 3.8: Struktur tabel *page_forms*

page_forms		
No.	**Atribut**	**Tipe Data**
1.	id_form	int
2.	page_id	int
3.	form	text

dipakai hanya *id_page* dan *url*. Lalu setelah perhitungan selesai, maka hasilnya akan disimpan kedalam tabel *pagerank*.

Tabel 3.9: Struktur tabel *page_images*

page_images		
No.	**Atribut**	**Tipe Data**
1.	id_image	int
2.	page_id	int
3.	image	text

Tabel 3.10: Struktur tabel *page_scripts*

page_scripts		
No.	**Atribut**	**Tipe Data**
1.	id_script	int
2.	page_id	int
3.	script	text

Tabel 3.11: Struktur tabel *page_list*

page_list		
No.	**Atribut**	**Tipe Data**
1.	id_list	int
2.	page_id	int
3.	list	text

Tabel 3.12: Struktur tabel *page_styles*

page_styles		
No.	**Atribut**	**Tipe Data**
1.	id_style	int
2.	page_id	int
3.	style	text

Pada penelitian ini digunakan dua *dataset*. *Dataset* pertama yang nantinya disebut sebagai Dataset 1 merupakan *dataset* gabungan dari *dataset* yang diperoleh dari penelitian Khatulistiwa (2022) dan data lanjutan yang diperoleh dengan cara *crawling*. Sebelumnya Dataset 1 hanya memiliki tidak lebih dari 11.000 baris halaman web *page_information* menjadi 20.493 baris dan 2.915.842 baris *page_linking*. Data *page_information* pada Dataset 1 dapat dikelompokan ke dalam 560 *cluster* berdasarkan *domain*-nya yang dapat dilihat pada tabel 3.13.

Tabel 3.13: Data *cluster* pada Dataset 1

No.	Domain	Jumlah Halaman
1	detik.com	2.215
2	unj.ac.id	2.208
3	sport.detik.com	1.279
4	finance.detik.com	1.098
5	repository.unj.ac.id	1.089
6	news.detik.com	802
7	oto.detik.com	779
8	inet.detik.com	671
9	support.google.com	630
10	food.detik.com	626
⋮	⋮	⋮
558	codingcompetitions.withgoogle.com	1
559	googledevelopers.blogspot.com	1
560	skillshop.exceedlms.com	1

Dapat dilihat pada tabel 3.13, halaman web didominasi oleh domain "detik.com" dan "unj.ac.id" beserta sub domainnya, hal ini karena saat dilakukan *crawling* titik halaman web awal-awal yang di-*crawl* adalah "detik.com" dan "unj.ac.id". Hal ini juga membuktikan kecendrungan dari halaman web memiliki *link* yang bersifat *intra-link* yaitu *link* yang menunjuk halaman lain yang masih di dalam satu domain-nya, yang merupakan salah satu basis dari algoritma DPC (Zhu dkk., 2005).

Yang kedua, Dataset 2 merupakan *dataset* kecil dan *domain* kecil yang sengaja dikumpulkan untuk melihat perbedaan performa algoritma antara *dataset* yang berisi

banyak *domain* besar dengan *dataset* yang berisi *domain* kecil. Batasan pada tiap *domain* yang dipakai ketika mengumpulkan data adalah 20 halaman web per *domain*. Pada Dataset 2 terdapat 100 baris *page_information*, 5.944 baris *page_linking*, serta 5 *cluster* atau *domain*. Data *cluster* pada Dataset 2 dapat dilihat pada tabel 3.14.

Tabel 3.14: Data *cluster* pada Dataset 2

No.	Domain	Jumlah Halaman
1	unj.ac.id	20
2	ppid.unj.ac.id	20
3	fip.unj.ac.id	20
4	fbs.unj.ac.id	20
5	fmipa.unj.ac.id	20

C. Kelemahan Algoritma Pagerank Original

Algoritma Pagerank Original bekerja dengan cara melakukan iterasi perkalian antara vektor *ranking* halaman web terhadap matriks transisi graf situs-situs web. Permasalahan muncul karena matriks transisi tersebut membutuhkan memori utama yang cukup besar, yaitu dengan kompleksitas $O(N^2)$. Misal jika tipe data yang dipakai adalah *floating point* 64 bit dan jika pada dataset terdapat 10 ribu halaman web, maka matriks transisi yang terbentuk adalah matriks persegi 10 ribu $\times$ 10 ribu dan secara memori utama diperlukan $10.000 \times 10.000 \times 64$ bit $= 6.400.000.000$ bit $= 800$ Mega Byte. Walaupun pada contoh sebelumnya cukup kecil, pada kenyataannya internet memiliki miliaran halaman web yang, jika menggunakan algoritma Pagerank Original tanpa penanganan khusus, akan memakan *Peta Byte* atau bahkan *Exa Byte* memori. Akibatnya jika dilakukan pada satu mesin komputer pribadi yang hanya

menggunakan memori 4 *Giga Byte* sampai 32 *Giga Byte*, program akan *crash*.

D. Penjelasan Lanjut Algoritma DPC

Algoritma dimulai dengan memasukan input matriks transisi P dan himpunan *cluster* halaman web G. Keduanya didapatkan dari basis data penelitian Khatulistiwa (2022), tersusun atas dua *entity* yang akan dijelaskan pada nanti pada bagian berikutnya. *damping factor* d yang nilainya mengikuti penelitian Zhu dkk. (2005) yakni $d = 0.85$, dan nilai $0 < \epsilon < 1$ untuk toleransi *error*.

Selanjutnya untuk setiap *cluster* G_i dibuat matriks lokal transisi ukuran $N_i \times N_i$ diambil dari nilai matriks transisi P. Setelah itu hitung nilai Pagerank lokal π_i^0 dengan memasukan Q_i, nilai toleransi *error* ϵ, dan nilai Pagerank awal-awal adalah $\frac{e}{N_i}$. Ingat $\epsilon = [1, 1, ..., 1]^T$ atau vektor kolom seragam yang semua nilainya satu dan N_i adalah jumlah anggota G_i maka $\frac{e}{N_i}$ = $[\frac{1}{N_i}, \frac{1}{N_i}, ..., \frac{1}{N_i}]$. Hasil dari Pagerank lokal awal tersebut adalah vektor dengan dimensi $N_i \times 1$.

Langkah selanjutnya sudah memasuki *looping*. k adalah jumlah iterasi. Dibuat matriks agregat $A^k = RPS(\pi^k)$. R adalah matriks $n \times N$, dimana n adalah panjang G_i dan N adalah banyaknya halaman web secara keseluruhan. R didefinisikan pada persamaan 2.2. P adalah matriks transisi secara keseluruhan dengan dimensi $N \times N$. $S(\pi^k)$ matriks disagregasi $N \times n$ yang didefinisikan pada persamaan 2.3. Matriks A^k dipakai sebagai input matriks transisi untuk perhitungan Pagerank pada level kasar $z^k = Pagerank(A^k, \frac{e}{n}, \epsilon)$. Perlu diingat, karena dimensi A^k adalah $n \times n$, maka dimensi vektor $\frac{e}{n}$ adalah $n \times 1$. Akibatnya dimensi vektor z^k adalah $n \times 1$. Langkah ini disebut dengan solusi kasar *coarse level* (Zhu dkk., 2005).

Setelah itu, setiap *cluster* G_i buat sebuah matriks $(N_i + 1) \times (N_i + 1)$ lokal transisi yang diperbesar B_i^k seperti persamaan 2.12. Pada bagian kiri atas matriks B_i^k, P_{ii} adalah matriks transisi lokal *cluster* G_i. Pada bagian kiri bawah terdapat

perkalian vektor baris e^T dengan dimensi $1 \times N$ dan P_{*i} yang merupakan matriks transisi dari halaman anggota G_i ke halaman anggota G dengan dimensi $N \times N_i$. Hasil akhir dari perkalian tersebut adalah sebuah vektor baris $1 \times N_i$ yang akan menempati baris terbawah dari matriks B_i^k bersama dengan skalar α^k, yang menjadi elemen paling bawah dan kanan dari matriks B_i^k. Nilai dari α^k tergantung dari jumlah total kolom paling kanan matriks B_i^k yaitu sama dengan 1. Pada bagian kanan atas terdapat matriks P_{i*}, berdimensi $N_i \times N$ sekaligus merupakan matriks transisi dari halaman anggota G ke halaman anggota G_i, dikalikan dengan matriks disagregasi $S(\pi^k)$ dan vektor z^k. Selanjutnya hasil perkalian dari ketiga matriks dan vektor tersebut dikurangi dengan perkalian matriks P_{ii}, vektor π_i^k, dan skalar z_i. Dari hasil pengurangan tersebut dibagi dengan pengurangan 1 dikurangi z_i^k. Hasil akhir dari operasi bagian kanan atas matriks B_i^k adalah sebuah vektor kolom $N_i \times 1$ sekaligus, bersama α^k merupakan kolom paling kanan dari matriks B_i^k.

Selanjutnya hitung algoritma Pagerank dengan masukan matriks B_i^k sebagai matriks transisi, vektor $\frac{e}{(N_i+1)}$ sebagai vektor *ranking* awal-awal, dan nilai ϵ. Hasil akhir dari Pagerank adalah vektor kolom $N_i \times 1$. Baris pertama sampai baris ke-N_i adalah vektor ω_i^{k+1} dan baris ke-(N_i+1) adalah nilai skalar β_i^{k+1}. Langkah ini disebut sebagai penghalusan (*smoothing*) pada tingkat lebih halus (Zhu dkk., 2005).

Setelah itu, nilai *ranking* halaman pada tiap *cluster* yang dilambangkan dengan $\pi_i^{\sim k+1}$ diperbaharui dengan melakukan perhitungan sesuai persamaan 8. Selanjutnya *ranking* halaman tersebut dinormalisasi sesuai persamaan 9. Hitung selisih *ranking* halaman web iterasi saat ini dengan terasi sebelumnya, jika sudah kurang dari toleransi *error* ϵ maka algoritma selesai mengembalikan nilai π^{k+1}. Jika tidak, maka ulangi langkah *looping*.

E. Kelemahan Algoritma DPC

Pada algoritma DPC, langkah untuk mendapatkan matriks A memerlukan perkalian matriks R, matriks P, dan matriks $R(\pi)$. Melakukan perkalian matriks P secara utuh akan menimbulkan masalah dan memakan memori utama yang cukup besar, permasalahan yang sama yang dihadapi pada Pagerank Original. Untuk mengatasi hal tersebut, diperlukan penanganan khusus saat melakukan perkalian agar bisa muat pada memori utama, dengan konsekuensi algoritma dijalankan lebih lambat karena terdapat proses memecah matriks P lebih kecil, dalam penelitian ini memecah matriks P menjadi beberapa vektor kolomnya ketika akan mengalikan matriks R dengan P.

Esensi pada langkah empat sampai langkah tujuh algoritma DPC adalah mencari *ranking* domain dan menyesuaikan *ranking* domain tersebut dengan *ranking* halaman web yang masih terisolasi pada domain-nya masing-masing. Matriks A sejatinya adalah matriks transisi domain, sedangkan vektor z adalah vektor *ranking* dari domain. Langkah-langkah tersebut dapat disimplifikasi, pada penelitian ini dirumuskan algoritma modifikasi dari algoritma DPC yang dinamakan algoritma Modified DPC (MDPC).

F. Algoritma Modified DPC

Sebelumnya telah dibahas dua algoritma yaitu Pagerank Original pada penelitian Page dkk. (1999), dan Distributed Pagerank Computation pada penelitian Zhu dkk. (2005). Diusulkan algoritma modifikasi dari algoritma DPC atau Modified DPC (MDPC). Secara garis besar MDPC hanya memangkas langkah-langkah pada algoritma DPC, berdasarkan gagasan utama DPC yaitu melakukan perhitungan terpisah *ranking* halaman web berdasarkan *domain*-nya, dan melakukan perhitungan

penggabungan dengan menghitung *ranking domain* itu sendiri. Masalah utama dari algoritma DPC adalah langkah memperoleh matriks A pada langkah 2.10 yang melakukan perkalian matriks P secara utuh.

Definisi matriks A versi MDPC yang selanjutnya akan berganti notasi menjadi A_{mdpc} dapat dilihat pada persamaan 3.1. Makna notasi P_{**} merupakan partisi matriks transisi P yang sudah didefinisikan pada persamaan 2.5, sedangkan makna dari *sum* adalah nilai total dari elemen matriks.

$$A_{mdpc} = \begin{pmatrix} \frac{\text{sum}(P_{11})}{\text{sum}(P_{*1})} & \cdots & \frac{\text{sum}(P_{1i})}{\text{sum}(P_{*i})} \\ \vdots & \ddots & \vdots \\ \frac{\text{sum}(P_{i1})}{\text{sum}(P_{*1})} & \cdots & \frac{\text{sum}(P_{ii})}{\text{sum}(P_{*i})} \end{pmatrix} \tag{3.1}$$

Dari web graf di gambar 2.11, dapat dibuat contoh matriks A_{mdpc}. Proses perhitungan matriks A_{mdpc} dapat dilihat pada persamaan 3.2.

$$\text{sum}(P_{*,1}) = \text{sum}(\begin{pmatrix} 0,02143 & 0,14286 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 \\ 0,02143 & 0,14286 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 \\ 0,02143 & 0,14286 & 0,14286 \end{pmatrix}) = 3,00005$$

$$\text{sum}(P_{*,2}) = \text{sum}(\begin{pmatrix} 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,44643 & 0,14286 \\ 0,44642 & 0,14286 \\ 0,02143 & 0,14286 \end{pmatrix}) = 2,00002 \quad \text{sum}(P_{*,3}) = \text{sum}(\begin{pmatrix} 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,87143 & 0,14286 \end{pmatrix}) = 2,00003 \tag{3.2}$$

$$A_{mdpc} = \begin{pmatrix} \frac{\text{sum}(\begin{pmatrix} 0,02143 & 0,14286 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 \\ 0,23393 & 0,14286 & 0,14286 \end{pmatrix})}{3,00005} & \frac{\text{sum}(\begin{pmatrix} 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \end{pmatrix})}{2,00002} & \frac{\text{sum}(\begin{pmatrix} 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \end{pmatrix})}{2,00003} \\ \frac{\text{sum}(\begin{pmatrix} 0,23393 & 0,14286 & 0,14286 \\ 0,02143 & 0,14286 & 0,14286 \end{pmatrix})}{3,00005} & \frac{\text{sum}(\begin{pmatrix} 0,02143 & 0,14286 \\ 0,44643 & 0,14286 \end{pmatrix})}{2,00002} & \frac{\text{sum}(\begin{pmatrix} 0,02143 & 0,14286 \\ 0,02143 & 0,14286 \end{pmatrix})}{2,00003} \\ \frac{\text{sum}(\begin{pmatrix} 0,23393 & 0,14286 & 0,14286 \\ 0,02143 & 0,14286 & 0,14286 \end{pmatrix})}{3,00005} & \frac{\text{sum}(\begin{pmatrix} 0,44642 & 0,14286 \\ 0,02143 & 0,14286 \end{pmatrix})}{2,00002} & \frac{\text{sum}(\begin{pmatrix} 0,02143 & 0,14286 \\ 0,87143 & 0,14286 \end{pmatrix})}{2,00003} \end{pmatrix}$$

$$A_{mdpc} = \begin{pmatrix} \frac{1,34645}{3,00005} & \frac{0,49287}{2,00002} & \frac{0,49287}{2,00003} \\ \frac{0,8268}{3,00005} & \frac{0,75358}{2,00002} & \frac{0,32858}{2,00003} \\ \frac{0,8268}{3,00005} & \frac{0,75357}{2,00002} & \frac{1,17858}{2,00003} \end{pmatrix}$$

$$A_{mdpc} = \begin{pmatrix} 0,44881 & 0,24643 & 0,24643 \\ 0,2756 & 0,37679 & 0,16429 \\ 0,2756 & 0,37678 & 0,58928 \end{pmatrix}$$

Langkah-langkah algoritma MDPC dapat dilihat pada algoritma 5.

Algoritma 5 Algoritma MDPC

1: Buat matriks transisi lokal $N_i \times N_i$ untuk tiap *cluster* G_i berdasarkan P

$$Q_i = LocalTransitionMatrix(G_i) \forall G_i \in G \tag{3.3}$$

Ket: $N_i \rightarrow$ jumlah anggota G_i

2:

$$\pi_i = Pagerank(Q_i, \frac{e}{N_i}, \epsilon) \forall G_i \in G \tag{3.4}$$

Ket: $e = [1, 1, ..., 1]^T$; Dimensi π_i adalah $N_i \times 1$

3:

$$z = Pagerank(A_{mdpc}, \frac{e}{n}, \epsilon) \tag{3.5}$$

Ket: $n \rightarrow$ banyaknya anggota G; Dimensi z adalah $n \times 1$

4: Perbaharui nilai π_i dengan dikalikan nilai skalar z_i

$$\pi_i \leftarrow \pi_i \times z_i \forall i \in 1 \ldots n \tag{3.6}$$

5: kembalikan π sebagai hasil akhir. π adalah vektor gabungan $N \times 1$ dari semua π_i.

G. Algoritma Random Walker

Basis intiutif dari algoritma Pagerank adalah *random walk* pada graf. Versi penyederhanaan dalam bentuk distribusi probabilitas *random walk* pada sebuah graf web (Page dkk., 1999). Akan dibuat sebuah program yang mensimulasikan dari proses *random walk* ini yang disebut Algoritma Random Walker. Algoritma Random Walker dipakai untuk membandingkan hasil perhitungan algoritma DPC,

MDPC, dan Pagerank asli. Langkah-langkah pada Algoritma Random Walker dapat dilihat pada algoritma 6.

Pada langkah pertama jumlah iterasi diperlukan karena pada dasarnya, berbeda dengan algoritma Pagerank yang bisa konvergen, algoritma Random Walker tidak bisa berhenti selain membatasi jumlah iterasinya. Langkah kedua mensimulasikan bahwa pengguna internet dapat berasal dari halaman web mana pun. Langkah ketiga untuk menentukan besarnya peluang *walker* berpindah dari halaman web awal ke halaman lain. Matriks P ini juga berisi peluang *walker* melompat ke halaman web lain yang tidak memiliki *link* satu sama lain atau tidak terhubung langsung.

Algoritma 6 Algoritma Random Walker

1: Tentukan jumlah iterasi dan jumlah $walker$ awal-awal pada setiap halaman web
2: Instansiasi $walker$ sama banyak pada setiap halaman web
3: Bentuk matriks P sesuai persamaan 2.1
4: Untuk setiap $walker$, pindahkan $walker$ ke halaman web selanjutnya berdasarkan peluang P_{*i}, untuk i adalah indeks halaman web $walker$ saat ini.
5: Ulangi langkah 4 sebanyak jumlah iterasi. *Ranking* halaman web ditentukan pada banyaknya jumlah $walker$ yang tinggal.

Pada penelitian ini, algoritma Random Walker tidak dipakai untuk membandingkan performa, melainkan sebatas pembanding hasil dari algoritma Pagerank, DPC, dan MDPC.

H. Alat dan Bahan Penelitian

Untuk melaksanakan penelitian ini digunakan perangkat keras dan lunak untuk membuat program. Perangkat keras dan perangkat lunak yang digunakan adalah sebagai berikut:

1. Komputer *Desktop* dengan spesifikasi CPU AMD Ryzen 5 3600, GPU RTX 2060, dan RAM 16GB
2. Monitor Resolusi 1920x1080
3. Koneksi internet
4. Sistem Operasi Ubuntu 22.04
5. Editor kode Visual Studio Code
6. DBeaver Community Edition untuk akses basis data MySQL
7. Bahasa pemrograman Python 3

I. Tahapan Pengembangan

Akan dilakukan pengkajian efisiensi penggunaan memori utama pada algoritma Pagerank, DPC, dan MDPC. Penelitian ini merupakan sub-penelitian dari penelitian utama tentang pengembangan *search engine*. Telah dilakukan penelitian sebelumnya oleh Qoriiba (2021) yang membuat modul *crawler* dan modul pendukung lain. Selanjutnya Khatulistiwa (2022) menggabungkan modul *crawler* Qoriiba (2021), Google Pagerank, dan *searcher* berbasis TF IDF menjadi *search engine* berbasis konsol. Di saat yang berdekatan, Pratama (2022) membuat modul *indexer* menggunakan *Induced Generalized Suffix Tree* dan Zalghornain (2022) menggunakan *Continuous-Bag-of-Words* dan Model *Continuous Skip-Gram*.

Karena penelitian ini merupakan sub-penelitian, maka *stack* teknologi yang digunakan dalam implementasi penelitian ini akan disamakan dengan penelitian induk yaitu menggunakan bahasa pemrograman Python 3 dan basis data MySQL. Implementasi algoritma akan mengikuti algoritma-algoritma yang sudah dijelaskan sebelumnya. Perkiraan pengerjaan adalah satu bulan, dan tidak menutup

kemungkinan akan memakan waktu lebih lama atau lebih cepat tergantung kesulitan dan kompleksitas yang dihadapi.

Untuk menguji penggunaan memori, waktu eksekusi, dan hasil algoritma *ranking* halaman web maka dilakukan langkah-langkah berikut:

1. Dilakukan pengkodean empat program algoritma yaitu Pagerank Original, DPC, MDPC, dan Random Walker
2. Ditentukan nilai masukan *damping factor*, toleransi *error*, serta dataset yang sama
3. Jalankan program Pagerank Original, DPC, MDPC, dan Random Walker terhadap dataset 1 (20.493 Halaman)
4. Bandingkan lamanya waktu, dan penggunaan memori utama algoritma Pagerank Original, DPC, dan MDPC
5. Verifikasi kemiripan hasil antara algoritma Pagerank Original, DPC, dan MDPC terhadap algoritma Random Walker
6. Jalankan program Pagerank Original, DPC, MDPC, dan Random Walker terhadap dataset 2 (100 Halaman)
7. Bandingkan lamanya waktu, dan penggunaan memori utama algoritma Pagerank Original, DPC, dan MDPC
8. Verifikasi kemiripan hasil antara algoritma Pagerank Original, DPC, dan MDPC terhadap algoritma Random Walker

J. Metode Perbandingan

Terdapat tiga hal yang dibandingkan pada penelitian ini, yang pertama adalah perbandingan penggunaan memori utama, lama waktunya algoritma dijalankan, dan kemiripan hasil yang dikeluarkan oleh ketiga algoritma. Tidak ada metode khusus untuk membandingkan penggunaan memori utama, dan lama waktu eksekusi. Ketiga algoritma hanya dijalankan lalu dapat dilihat dan dibandingkan penggunaan memori utama dan lamanya waktu eksekusi tiap algoritma. Sedangkan untuk kemiripan hasil, penelitian ini mengikuti metode yang dipakai oleh Zhu dkk. (2005), digunakan perhitungan jarak *Kendall's* τ atau dapat disebut sebagai $KDist$.

Rumus dari jarak $KDist$ antara vektor *ranking* halaman web π terhadap $\hat{\pi}$ secara berurutan dapat dilihat pada persamaan 3.8, di mana $0 \leq KDist(\pi, \hat{\pi}) \leq 1$. Semakin kecil nilai jaraknya maka dianggap lebih mirip.

$$K_{ij}(\pi, \hat{\pi}) = \begin{cases} 1 & \pi_i \geq \pi_j \text{ and } \hat{\pi}_i < \hat{\pi}_j \\ 1 & \pi_i < \pi_j \text{ and } \hat{\pi}_i \geq \hat{\pi}_j \\ 0 & \text{lainnya} \end{cases} \tag{3.7}$$

$$KDist(\pi, \hat{\pi}) = \frac{\Sigma_{1 \leq i < j \leq N} K_{ij}(\pi, \hat{\pi})}{\frac{1}{2}N(N-1)} \tag{3.8}$$

BAB IV

HASIL DAN PEMBAHASAN

Pada bab ini akan dibagi ke dalam dua bagian yaitu pengkodean dan hasil. Pada bagian pengkodean akan dibahas implementasi algoritma ke dalam kode pemrograman. Selanjutnya pada bagian hasil akan dijelaskan hasil pengujian program terhadap data yang dipakai sehingga dapat dibuat sebuah kesimpulan pada bab selanjutnya.

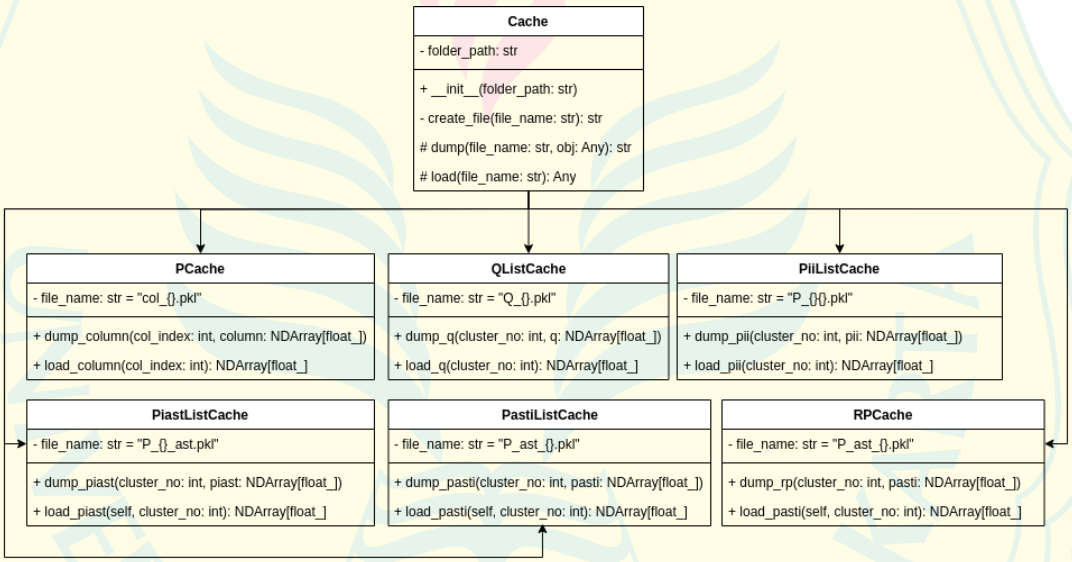
A. Pengkodean

Akan dibahas implementasi algoritma ke dalam kode pemrograman Python. Program dapat dipisah menjadi empat bagian, yaitu tiga bagian berbeda berdasarkan algoritma yang dipakai dan bagian kode program yang dipakai bersama (*shared code*). Pada bagian kode program bersama, terdapat *Helper Class* yang bertugas untuk *caching* dan mengakses basis data atau *Helper Function* yang melakukan tugas umum seperti perhitungan matematika, mengambil alamat *domain* dari *string url*.

1. *Shared Code*

Untuk melakukan *caching*, *library* Pickle dipakai. Pickle mengubah objek pada program menjadi *byte stream* yang disimpan ke dalam *file* berformat ".pkl". Tujuan dilakukan *caching* ini untuk meringankan penggunaan memori utama dengan menyimpan objek ke *disk*, dan hanya dimuat ke memori utama ketika hanya akan digunakan. Objek-objek yang disimpan ke dalam *cache* adalah matriks P yang dipecah berdasarkan kolomnya, matriks Q, matriks P_{ii}, matriks P_{i*}, matriks P_{*i}, dan matriks RP.

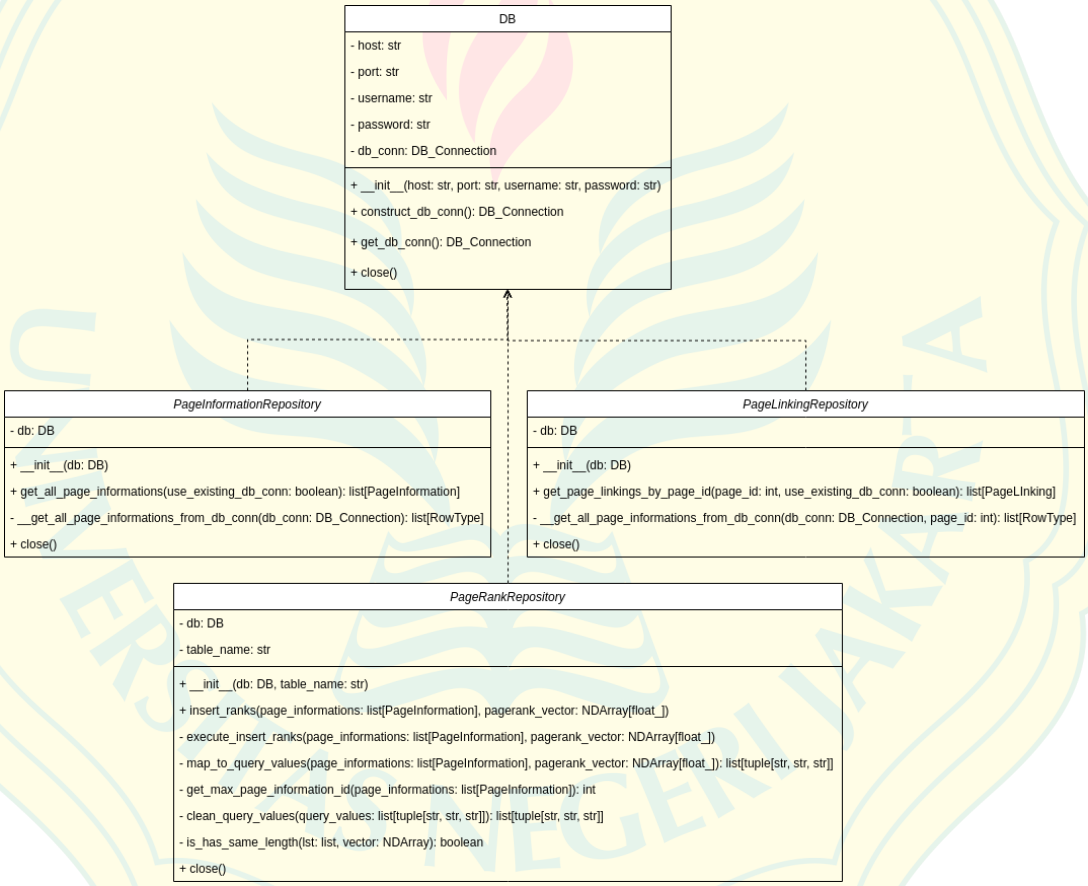
Potongan kode 3 berisi *class* untuk *caching Class* Cache merupakan *Parent Class* dari PCache, yang sesuai namanya, bertugas melakukan *Caching* untuk matriks P. Selain PCache juga ada *Cache Class* lain seperti, QListCache untuk matriks Q, PiiListCache untuk matriks P_{ii}, PiastListCache untuk matriks P_{i*}, PastiListCache untuk matriks P_{*i}, dan RPCache untuk matriks RP. Secara garis besar *class* tersebut memiliki *method* dan *property* yang mirip, seperti nama *file*, dan *method* untuk menyimpan dan memuat *cache*.



Gambar 4.1: *Class Diagram* dari *class* Cache dan turunannya

Selanjutnya terdapat *Class* yang bertugas menghubungkan basis data ke program. *Class* DB merupakan *Class* dasar untuk mengurus koneksi basis data ke program. *Class* DB dibungkus ke dalam *Repository Class* yang dibatasi oleh tabel tertentu dengan maksud agar tidak ada *Class* yang terlalu besar karena bertanggung jawab pada banyak tugas. Sebagai contoh pada *Class* PageInformationRepository bertugas mengurus tabel *page_information*. PageInformationRepository memiliki *method get_all_page_informations*, sesuai namanya, mengambil semua baris dari tabel *page_information* dan dikembalikan dalam bentuk List yang berisi *Class*

PageInformation (lihat kode 1 dan kode 2). Selain PageInformationRepository, juga ada *Repository Class* lain seperti PageLinkingRepository yang tugas utamanya mengambil data tabel *page_linking*, dan PagerankRepository yang tugas utamanya memasukan data hasil *ranking* halaman ke tabel *page_rank_original_pagerank_dpc_paper_version*, *page_rank_dpc*, dan *page_rank_modified_dpc_v2*.

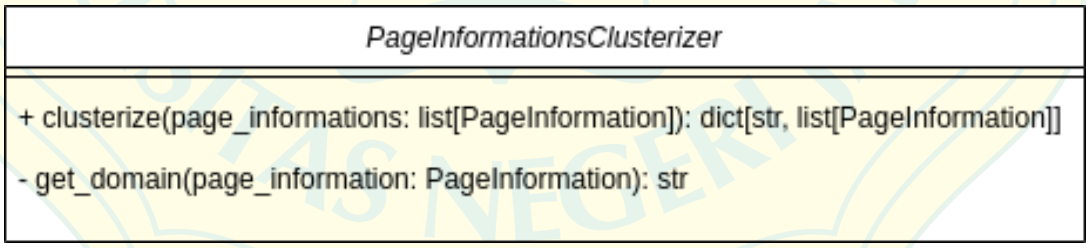


Gambar 4.2: *Class Diagram* dari *class* DB dan *class* lain yang memiliki dependensi terhadapnya

Sebelumnya sempat disinggung *Class* PageInformation tanpa penjelasan lebih lanjut. *Class* PageInformation merupakan salah satu dari dua *Model Class* yang ada di program penelitian ini. *Model Class* merupakan *Class* sederhana yang memodelkan data pada program. Pada kode 4 terdapat PageInformation dan

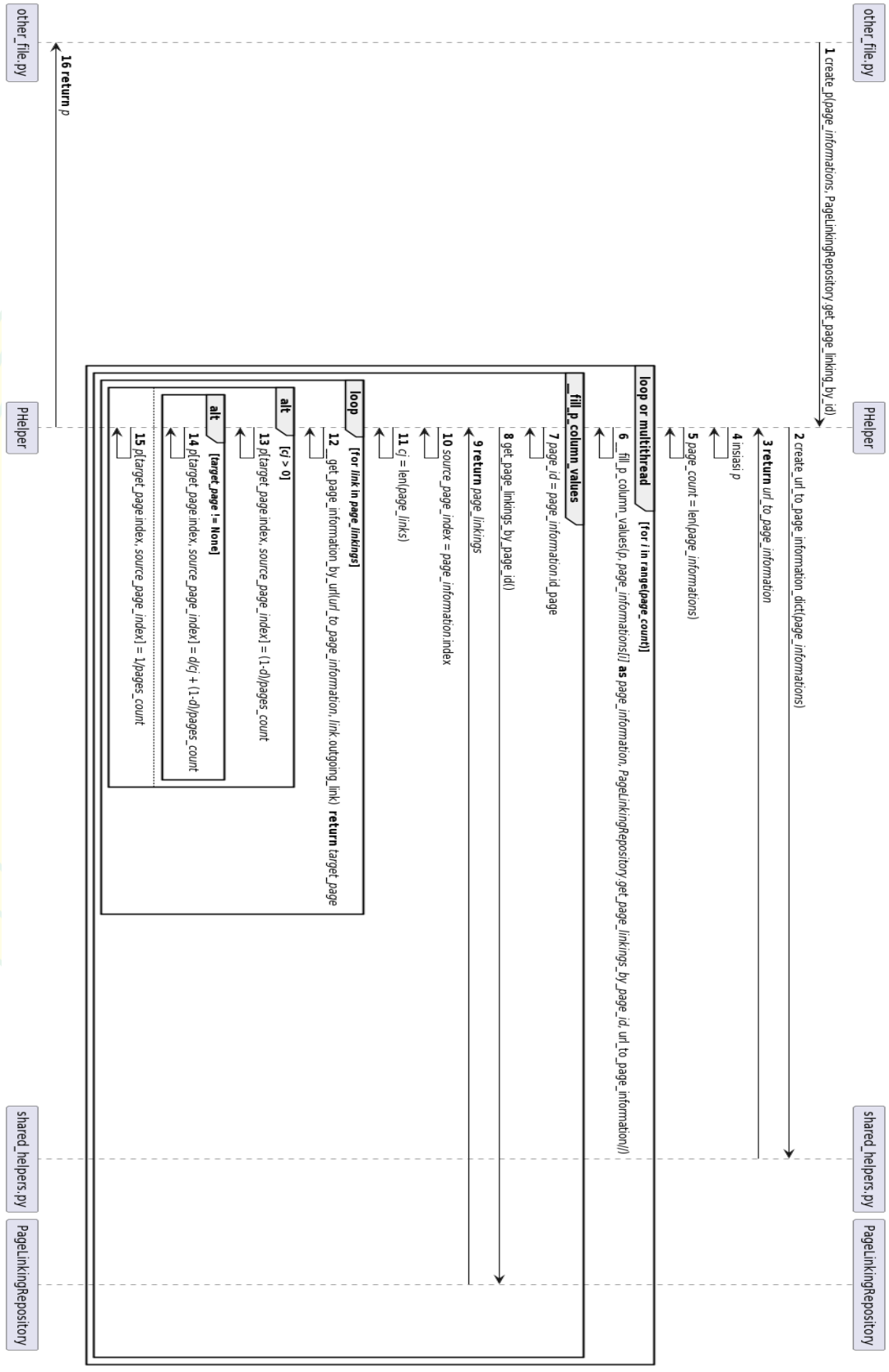
PageLinking. PageInformation menampung data baris pada tabel *page_information*. PageInformation memiliki tiga atribut yaitu *index* urutan PageInformation dalam matriks, *id_page* yang diambil dari basis data, dan url yang juga diambil dari basis data. Yang kedua ada PageLinking menampung data baris pada tabel *page_linking* yang memiliki atribut *outgoing_link* yang diambil dari basis data.

Pada kode 5 terlihat *helper functions* yang memiliki kegunaan tertentu yang bersifat umum. Sebagai contoh fungsi *l1_norm* mengembalikan nilai *l1 norm* dari suatu List. Selanjutnya ada *create_url_to_page_information_dict* yang mengubah List yang berisi PageInformation menjadi sebuah *dictionary* dengan *url* sebagai kunci, dan PageInformation merupakan isinya. Fungsi *sort_page_information_by_domain* mengurutkan List PageInformation berdasarkan domain-nya. Selain *helper functions*, di *file* yang sama, juga ada *class* PageInformationClusterizer yang tugasnya adalah mengelompokan List yang berisi PageInformation berdasarkan domain-nya, hasil akhirnya adalah sebuah *dictionary* yang *key*-nya merupakan domain, dan *value*-nya merupakan List PageInformation yang sudah dikelompokan sebelumnya.

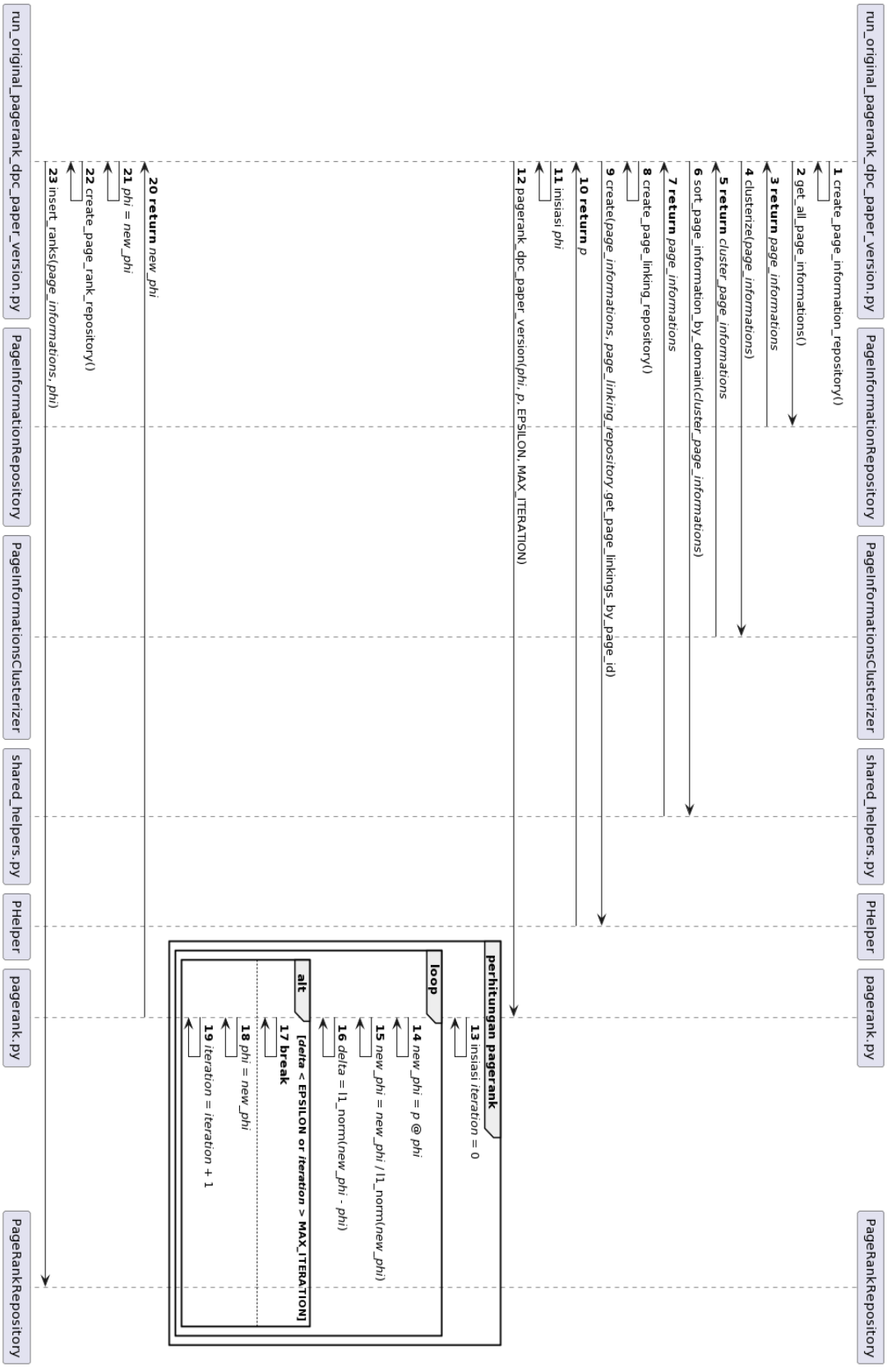


Gambar 4.3: *Class Diagram* dari *class* PageInformationClusterizer

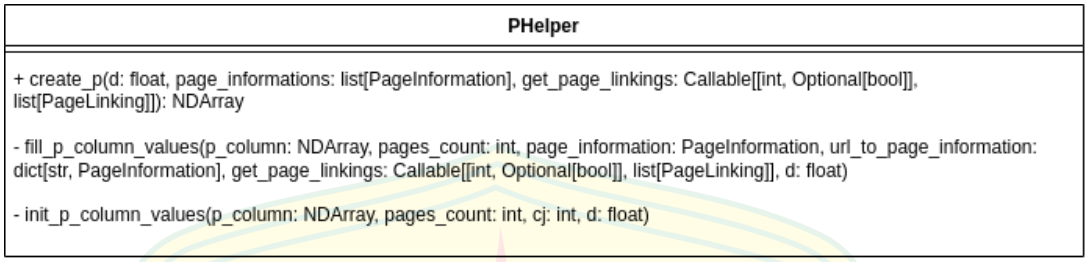
Class PHelper adalah sebuah *class* yang bertugas untuk membuat matriks transisi P. *Class* PHelper memiliki dependensi terhadap *class* PageLinkingRepository karena untuk menentukan nilai elemennya didasari pada *link* pada halaman web. Kode dan *Class Diagram* PHelper secara berurutan dapat dilihat pada kode 6 dan gambar 4.6.



Gambar 4.4: Diagram alir ketika membentuk matriks transisi P di *class* PHelper



Gambar 4.5: Diagram alir dari program Pagerank



Gambar 4.6: *Class Diagram* PHelper

Alur kerja dari PHelper dapat dilihat pada diagram alir 4.4. Kotak abu-abu pada kiri dan kanan diagram menunjukan nama *file* atau *class* terjadinya proses, tanda panah menunjukan perpindahan alur eksekusi dari *file*/*class* yang satu ke yang lainnya, nama di atas panah merupakan nama *method* atau *function* yang dipanggil, kata "*return*" berarti hasil keluaran dari *method*/*function*, kata "*loop*" berarti pengulangan, kata "*alt*" berarti percabangan alur program. Pada langkah satu *method create_p* dipanggil dengan argumen *page_informations* dan sebuah *method PageLinkingRepository.get_page_linking_by_id*. Pada langkah dua dibuat sebuah *dictionary* yang berisi PageInformation dan *url*-nya sebagai kunci. Selanjutnya pada langkah enam, dilakukan *looping* berdasarkan jumlah banyaknya halaman web yang ada pada *dataset*, proses ini juga dapat dilakukan secara pararel dengan *multi threading*. Pada langkah enam dilakukan pengisian matriks P pada setiap kolomnya. Alasan dilakukan pada tiap kolom karena terdapat nilai yang dipakai secara bersama yaitu banyaknya jumlah *PageLinking* suatu halaman web. Langkah 12 merupakan langkah penting, yaitu perhitungan nilai elemen matriks P. Dasar dari perhitungan dari persamaan 2.1.

2. Program Pagerank

Alur algoritma Pagerank dalam program dapat dilihat pada gambar 4.5. Langkah satu dilakukan instansiasi PageInformationRepository untuk mengambil

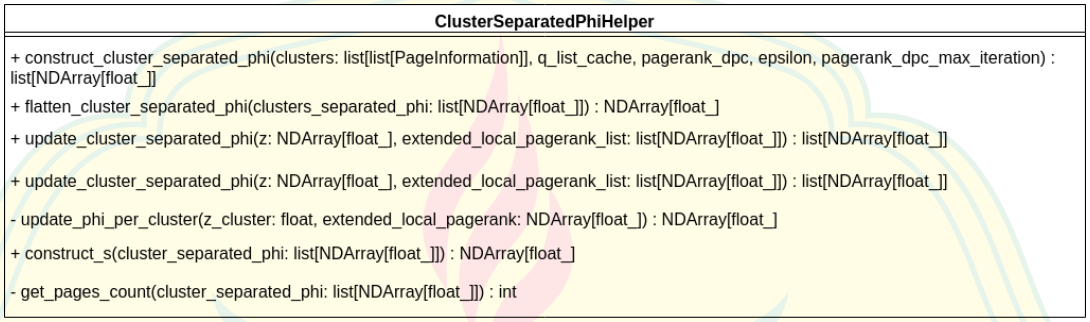
semua data dari tabel *page_information* dari basis data. Selanjutnya pada langkah empat dilakukan klasterisasi dari data *page_information* untuk dilakukan pengurutan berdasarkan *cluster*-nya. Hal ini dilakukan untuk mempermudah proses pemetaan antara vektor *ranking* halaman web π dengan *list page_information*. Pada langkah delapan dilakukan instansiasi PageLinkingRepository untuk membuat matriks transisi P yang memerlukan koneksi basis data untuk memperoleh data dari tabel *page_linking*. Setelah membuat matriks P, pada langkah sepuluh dilakukan inisiasi nilai π dan e, dan selanjutnya dilakukan perhitungan *pagerank*. Setelah melakukan perhitungan *pagerank*, pada langkah 22 dilakukan instansiasi PagerankRepository, untuk menyimpam hasil perhitungan *pagerank* ke basis data.

3. Program DPC

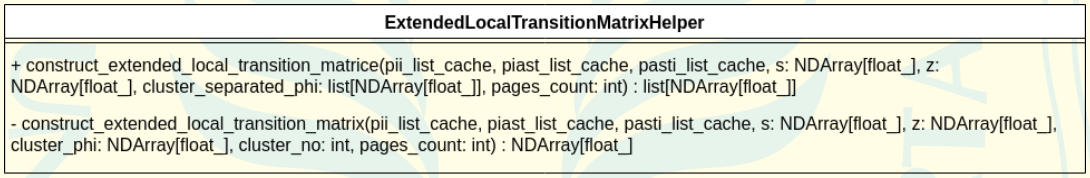
Selanjutnya akan dibahas kode program DPC. Berbeda dengan program Pagerank yang cenderung lebih sederhana, perhitungan pada program DPC lebih kompleks. Akibatnya dibutuhkan *class* pendukung yang lebih banyak. Terdapat delapan *class* pendukung yang dipakai oleh program DPC, yaitu ClusterSeparatedPhiHelper, ExtendedLocalTransitionMatrixHelper, PWithCacheHelper, PartitionedPHelper, PastiHelper, PiastHelper, RPHelper, dan DPCExecutor.

ClusterSeparatedPhiHelper merupakan salah satu *helper class* yang berisi *method-method* yang berkaitan dengan vektor π yang dipecah sesuai dengan *cluster*-nya masing-masing. *Method construct_cluster_separated_phi* melakukan perhitungan pada langkah 2.9 pada algoritma DPC, menghitung nilai π pada tiap *cluster*. Selanjutnya *method flatten_cluster_separated_phi* menyatukan vektor-vektor π yang terpisah menjadi ke dalam satu vektor. *Method update_cluster_separated_phi* melakukan perhitungan untuk memperbaharui nilai

vektor π sesuai dengan langkah 8 pada algoritma DPC. Yang terakhir ada *method construct_s* yang membuat sebuah matriks S berdasarkan vektor-vektor π yang terpisah berdasarkan *cluster*-nya.



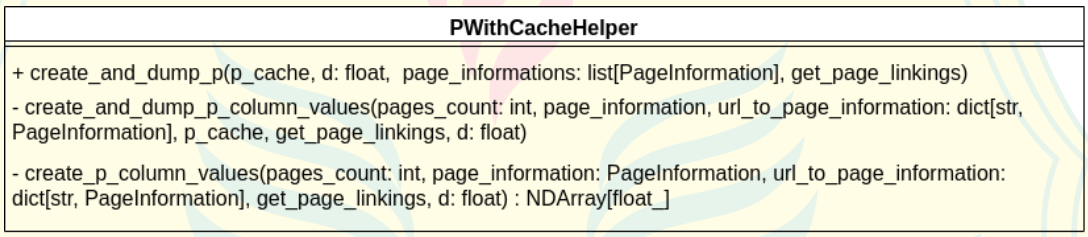
Gambar 4.7: *Class Diagram* ClusterSeparatedPhiHelper



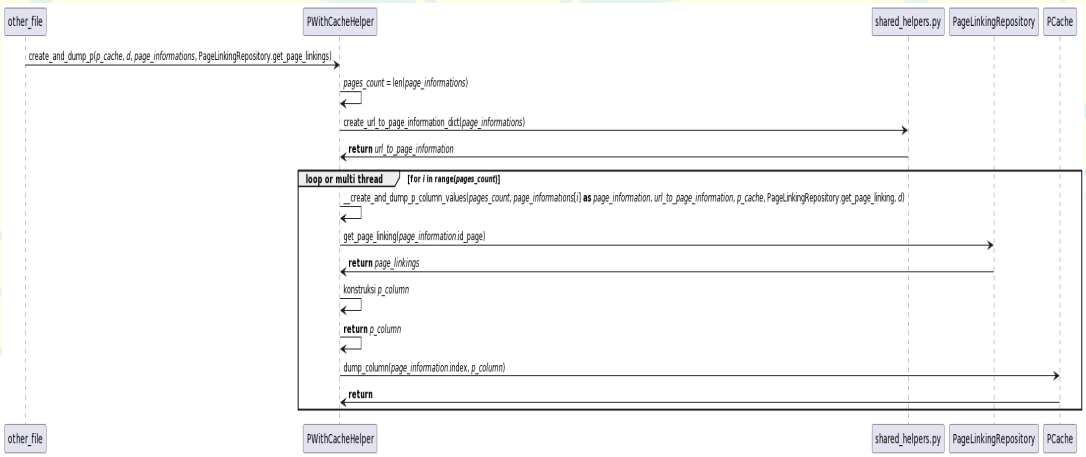
Gambar 4.8: *Class Diagram* ExtendedLocalTransitionMatrixHelper

ExtendedLocalTransitionMatrixHelper, sesuai namanya, merupakan *helper class* yang berkaitan dengan matriks *extended local transition matrix* yang dalam algoritma DPC berada pada langkah 2.12. Pada *method construct_extended_local_transition_matrice* membentuk himpunan matriks B pada setiap *cluster*. Sesuai dengan langkah 2.12 pada algoritma DPC, *method* ini memuat matriks P_{ii} yang sudah di-*caching* untuk dimasukkan ke beberapa kolom dan baris pertama matriks B. Selanjutnya, pada baris paling bawah diisi hasil perkalian vektor e dan matriks P_{*i} yang dimuat dari *class* PastiListCache. Selanjutnya pada kolom paling kanan dilakukan perhitungan yang cukup panjang sesuai pada langkah 2.12 algoritma DPC. Sel kanan bawah matriks B dapat bernilai apa saja, selama nilai total dari kolom paling kanan matriks B bernilai satu.

Selanjutnya, PWithCacheHelper, sama dengan PHelper, menangani proses pembentukan matriks P tetapi juga menyimpannya ke dalam *disk* atau *caching*. *Class* PWithCacheHelper hanya memiliki satu *public method* yaitu *create_and_dump_p*. Proses pembentukan matriks P pada *class* ini adalah dengan melakukan perhitungan layaknya matriks P pada *class* PHelper tetapi yang membedakan adalah, untuk mengakomodasi matriks yang lebih besar, dilakukan *caching* pada setiap kolom matriksnya.



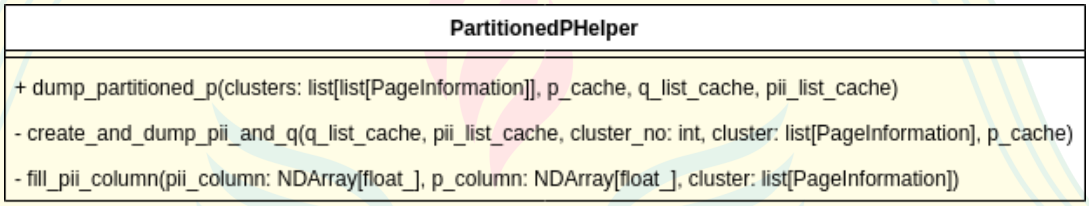
Gambar 4.9: *Class Diagram* PWithCacheHelper



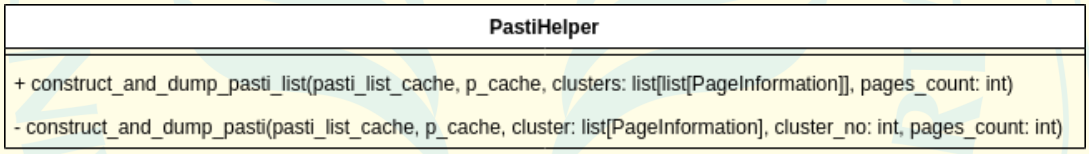
Gambar 4.10: Diagram alir PWithCacheHelper

Setelah matriks P terbentuk menggunakan *class* PWithCacheHelper, selanjutnya matriks P dapat dipecah menjadi matriks P_{ii} dan matriks Q_i. Matriks P_{ii} mirip dengan matriks Q_i yang merupakan matriks transisi lokal antara halaman web *cluster-*i terhadap *cluster-*i, tetapi yang membedakan adalah nilai kolom dari

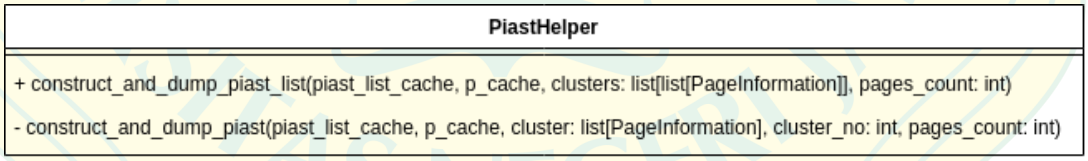
matriks Q_i sudah dinormalisasi. *Class* PartitionedPHelper melalui *method dump_partitioned_p* memuat kolom-kolom pada matriks P di PCache dipecah menjadi matriks-matriks P_{ii} dimasukkan ke PiiListCache, lalu tiap kolom P_{ii} dinormalisasi menjadi matriks Q_i dan dimasukkan ke dalam QListCache. PiastHelper dan PastiHelper memiliki cara kerja mirip dengan PartitionedPHelper, memuat matriks P dari PCache untuk dibentuk matriks P_{i*} dan matriks P_{*i}.



Gambar 4.11: *Class Diagram* PartitionedPHelper



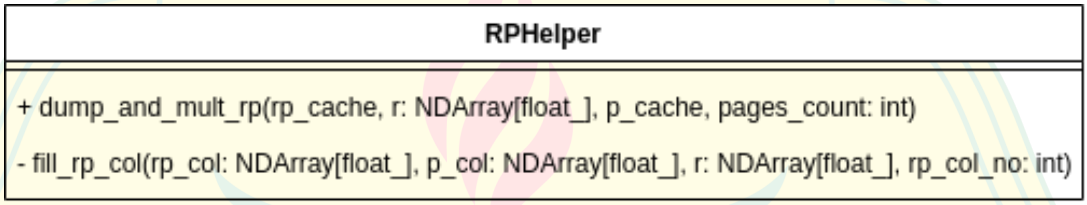
Gambar 4.12: *Class Diagram* PastiHelper



Gambar 4.13: *Class Diagram* PiastHelper

RPHelper bertugas menangani perkalian matriks R dan matriks P pada langkah 2.10 algoritma DPC. Alasan kenapa dibutuhkan *Helper Class* untuk perkalian kedua matriks tersebut, alih-alih menggunakan operator perkalian matriks biasa, karena matriks P memiliki dimensi yang besar dan harus dipecah ke dalam bentuk yang kecil, dalam penelitian ini, matriks P dipecah berdasarkan kolom. Akibatnya, dalam sudut pandang program, perkalian matriks R dan P membutuhkan

langkah-langkah kecil. Agar kode lebih rapih langkah-langkah kecil tersebut dimasukkan ke dalam *class* tersendiri. Pada *method dump_and_mult_rp* dilakukan *looping* terhadap kolom matriks RP yang nilainya masih kosong, tiap elemen dari kolom R didapatkan dari perkalian antara baris matriks R dan kolom matriks P. Hasil perkalian yaitu matriks RP disimpan ke dalam *class* RPCache.

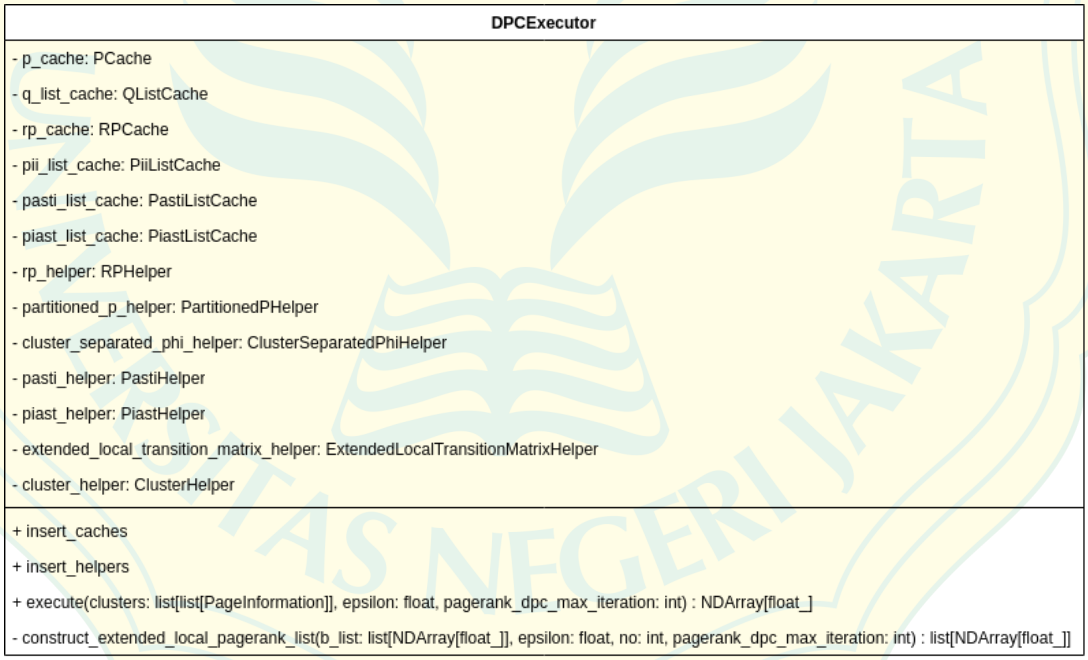


Gambar 4.14: *Class Diagram* RPHelper

DPCExecutor merupakan kelas utama yang mengorkestrasi *class* dan fungsi pendukung dalam mengeksekusi program DPC. Terdapat tiga *public method* yang dimiliki DPCExecutor. *insert_caches* menerima *class* Cache yang sudah diinstansiasi untuk disimpan ke dalam properti DPCExecutor. Selanjutnya, *insert_helpers* menerima *helper class* yang sudah diinstansiasi untuk disimpan ke dalam properti DPCExecutor. Yang terakhir ada *method execute* yang mengeksekusi program DPC dibantu dengan *cache class* dan *helper class* yang sudah dimasukkan terlebih dahulu.

Sama seperti program Pagerank yang sudah dibahas sebelumnya, program DPC dijalankan dimulai dari sebuah *file* Python, *run_dpc.py*. Kode dari *run_dpc.py* dapat dilihat pada kode 18. Langkah-langkah dari program DPC di *run_dpc.py* dapat dilihat di gambar 4.16. Langkah pertama dilakukan instansiasi PageInformationRepository untuk mengambil semua data PageInformation di langkah dua. PageInformation yang diambil dari PageInformationRepository, dilakukan klasterisasi pada langkah empat, hasil dari klasterisasi tersebut adalah sebuah klaster dalam bentuk *hashmap*, *key*-nya adalah domain, dan isinya adalah *list*

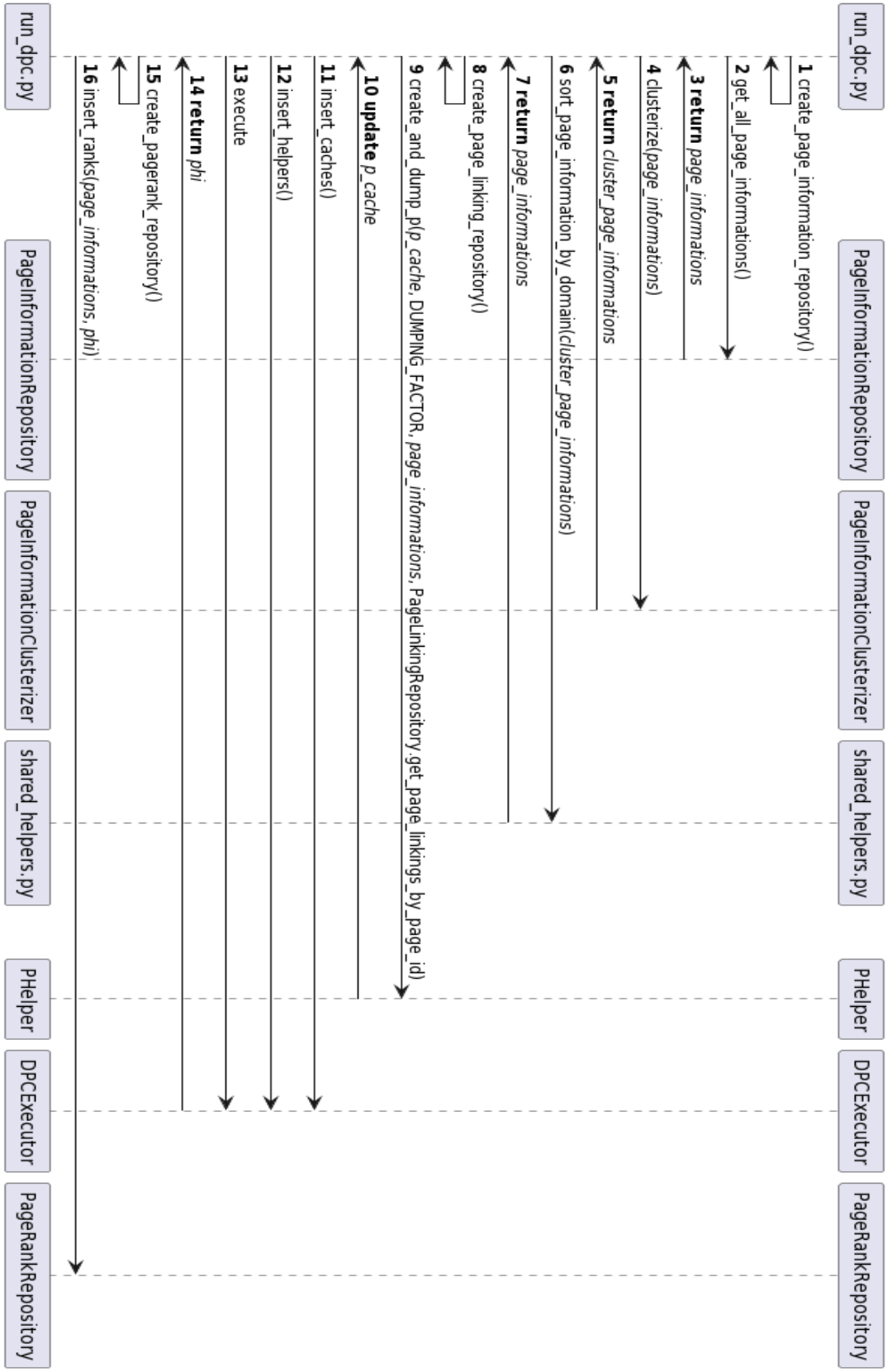
yang berisi PageInformation. Dari data klaster tersebut dapat dilakukan pengurutan *list* PageInformation berdasarkan domain-nya di langkah enam. Selanjutnya pada langkah delapan dilakukan instansiasi PageLinkingRepository untuk mengambil data PageLinking untuk menghitung nilai dari sel matriks transisi P yang dibuat pada langkah sembilan, lalu disimpan ke dalam PCache. Selanjutnya, akan dijalankan algoritma DPC di program melalui DPCExecutor. Setelah dilakukan instansiasi, pada langkah sebelas dan dua belas dimasukkan *cache class* dan *helper class* ke DPCExecutor. Algoritma DPC dijalankan pada langkah 13, yang secara detil dapat dilihat pada gambar 4.17. Setelah selesai, dikembalikan nilai π lalu disimpan ke basis data melalui PagerankRepository.



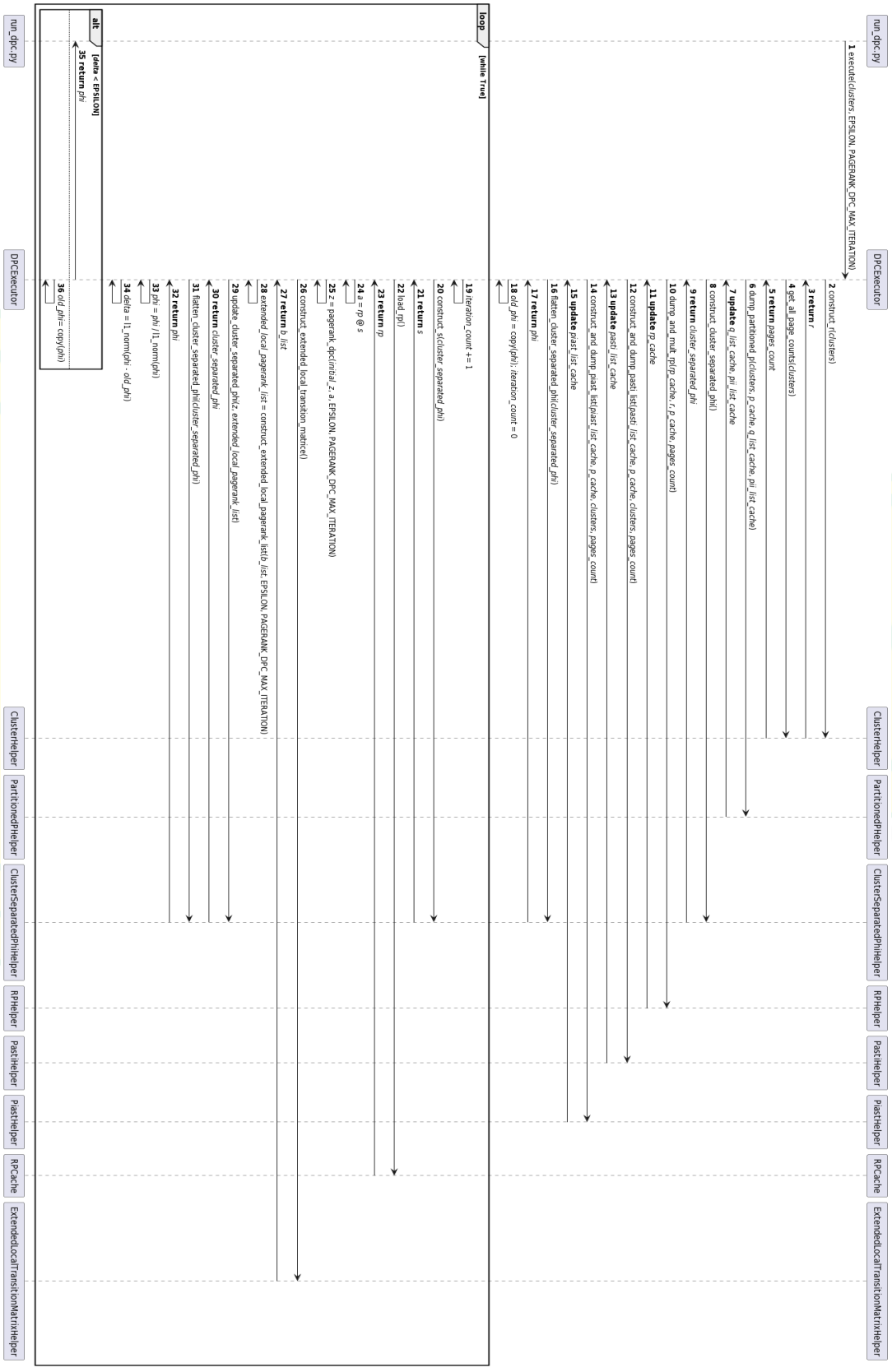
Gambar 4.15: *Class Diagram* DPCExecutor

DPCExecutor dipanggil melalui *method execute* dengan memasukan data klaster, ϵ, dan max iterasi. Pertama-tama pada langkah dua sampai lima dengan data klaster dibentuk matriks R dan diperoleh jumlah total halaman. Selanjutnya pada langkah enam dan tujuh, dipartisi nilai matriks P yang diambil dari PCache menjadi

himpunan matriks Q dan P_{ii}. Lalu pada langkah delapan dan sembilan, dilakukan perhitungan Pagerank pada setiap halaman web yang masih terpisah pada tiap domainnya. Pada langkah sepuluh dan sebelas dilakukan perkalian matriks R dan P. Alasan perkalian matriks R dan P dilakukan secara terpisah alih-alih langsung dikalikan dengan matriks S, karena dimensi matriks P sangat besar dan nilai matriks R dan P tidak akan berubah, sehingga lebih baik dilakukan secara terpisah terlebih dahulu, dan disimpan ke dalam *cache*. Selanjutnya pada langkah 12 sampai 15 dibentuk matriks P_{*i} dan P_{i*}, dengan alasan yang sama dengan matriks R dan P yang nilainya tidak akan berubah. Setelah itu, pada langkah 16 nilai vektor π yang masih terpisah pada tiap domainnya, disatukan tanpa ada perubahan nilai. Memasuki *loop* dibentuk matriks S berdasarkan nilai vektor π yang masih terpisah terhadap domainnya. Setelah itu, dilakukan perkalian matriks RP dengan matriks S untuk menghasilkan matriks A. Matriks A dapat disebut sebagai matriks transisi antara domain/klaster. Pada langkah 25, diperoleh nilai vektor z dengan melakukan perhitungan Pagerank dengan masukan matriks A. Vektor Z dapat disebut sebagai *ranking* pada tiap domain. Pada langkah 26 sampai 28 dilakukan *smoothing* dengan membentuk matriks *extended local transition matrix* B, dan vektor *extended local pagerank* pada setiap domain-nya. Selanjutnya pada langkah 29 nilai vektor π diperbaharui seperti pada langkah 8 algoritma DPC. Setelah itu, nilai π dinormalisasi pada langkah 33 dan dihitung nilai delta antara π saat ini, dengan π iterasi sebelumnya. Jika nilai deltanya kurang dari ϵ, maka perhitungan selesai dan dikembalikan nilai π sebagai nilai akhir. Jika tidak, kembali pada langkah awal ketika memasuki *loop*, diawali dengan menambahkan variabel $iteration\_count$ dengan satu untuk menghitung jumlah iterasi yang sudah dilalui. Jumlah iterasi maksimal dapat ditentukan, apabila iterasi tidak kunjung konvergen. Jadi selain nilai delta, jumlah iterasi dapat ditentukan batasnya untuk keluar dari *loop*.



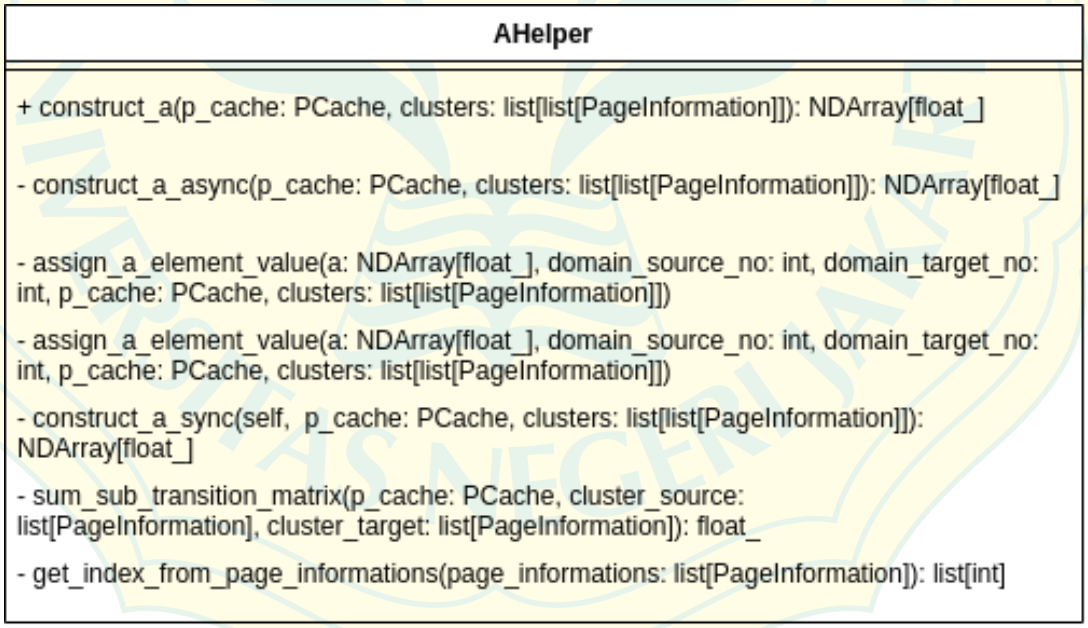
Gambar 4.16: Diagram alir *run_dpc.py*



Gambar 4.17: Diagram alir DPCExecutor

4. Program MDPC

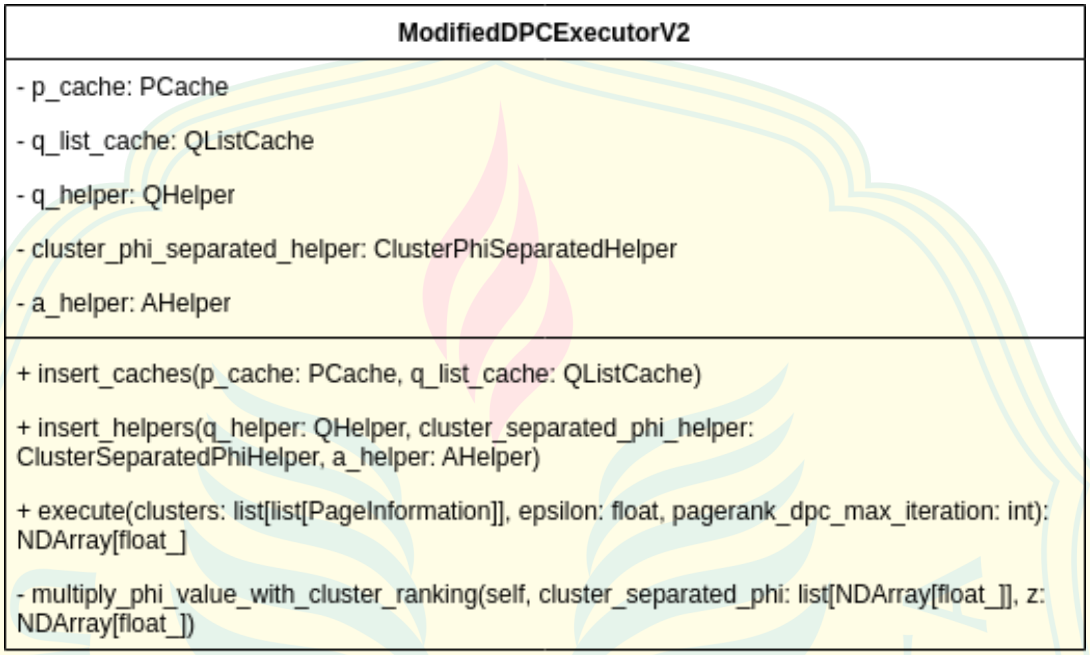
MDPC merupakan versi lebih sederhana dari algoritma DPC, sehingga kode yang dibuat sangat lebih sedikit dibanding program DPC. Selain itu juga, ada beberapa *class* yang ada pada DPC juga dipakai pada program MDPC. Namun, karena matriks transisi antar *cluster* A_{mdpc} tidak ada pada DPC, maka dibuat *class* AHelper untuk melakukan perhitungan dan membentuk matriks tersebut. Kode program dari *class* AHelper dapat dilihat pada kode 21. Secara garis besar, proses pembentukan matriks A_{mdpc} di *class* ini dimulai dari *public method construct_a*, dilakukan perhitungan nilai pada tiap sel matriks, lalu dilakukan normalisasi pada setiap kolom-nya. Adapun untuk dasar perhitungannya sesuai dengan persamaan 3.1 dan dilakukan pada *method __sum_sub_transition_matrix*.



Gambar 4.18: *Class Diagram* AHelper

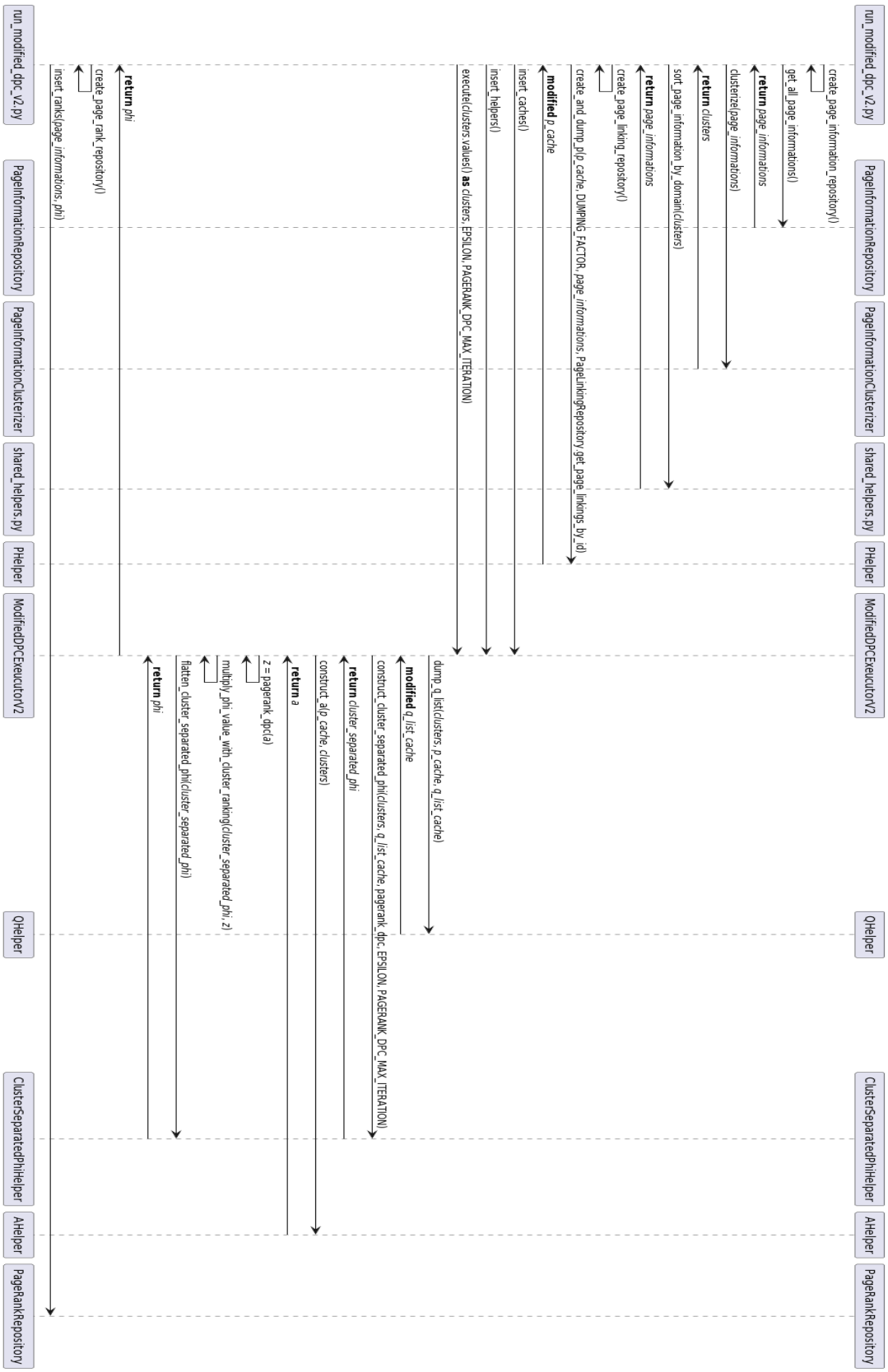
Selain AHelper, program MDPC juga memiliki *class* ModifiedDPCV2Executor yang berperan sebagai eksekutor dari program MDPC. Secara struktur ModifiedDPCV2Executor mirip dengan *class* DPCExecutor, terdapat

public method insert_caches, insert_helpers dan masing-masing dimulai dari *public method execute*.



Gambar 4.19: *Class Diagram* ModifiedDPCExecutorV2

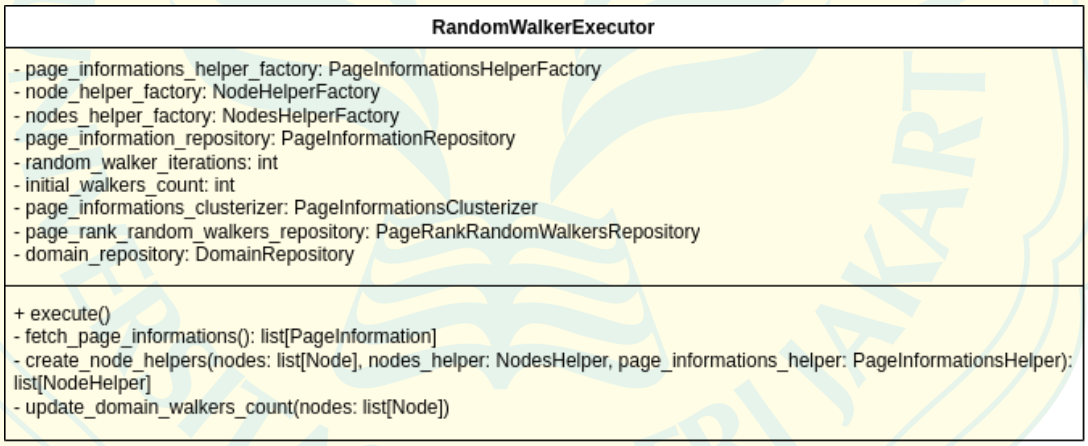
Ketika *method execute* pada *class* ModifiedDPCV2Executor dipanggil, hal pertama yang dilakukan adalah mengkonstruksi matriks Q dan disimpan ke dalam *cache*. Selanjutnya dilakukan perhitungan *ranking* halaman web atau Pagerank yang masih terisolasi pada setiap domain-nya, di dalam program disimpan ke dalam variabel *cluster_separated_phi*, sesuai dengan langkah 3.4 algoritma MDPC. Setelah itu dibuat matriks transisi antar *cluster* A_{mdpc} menggunakan *class* AHelper. Selanjutnya dilakukan perhitungan *ranking* tiap *cluster* dengan menggunakan A_{mdpc} sebagai matriks transisi. Setelah diperoleh *cluster ranking*, dilakukan perkalian *ranking* halaman web yang masih terisolasi terhadap *cluster* dengan *cluster ranking*. Nilai Pagerank pada *cluster_separated_phi* dianggap tidak terisolasi terhadap *cluster* dan dilakukan penggabungan menjadi sebuah vektor Pagerank tunggal dan dikembalikan sebagai hasil akhir.



Gambar 4.20: Diagram alir program MDPC

Serupa dengan program DPC, program MDPC dimulai dari sebuah *Python file run_modified_dpc_v2.py*. Alur dari program MDPC secara keseluruhan dapat dilihat pada diagram alir gambar 4.20. Sama dengan program-program sebelumnya, langkah awal yaitu instansiasi repositori dan diambil semua data informasi halaman web PageInformation, diklasterisasi, diurutkan, dan lalu dapat dibentuk matriks P yang disimpan ke dalam PCache. Selanjutnya dilakukan instansiasi *cache class* dan *helper class* lain lalu dimasukkan ke dalam ekskutor ModifiedDPCExecutorV2. Selanjutnya dipanggil *public method execute*. Setelah diperoleh nilai π, lalu disimpan ke basis data melalui PagerankRepository.

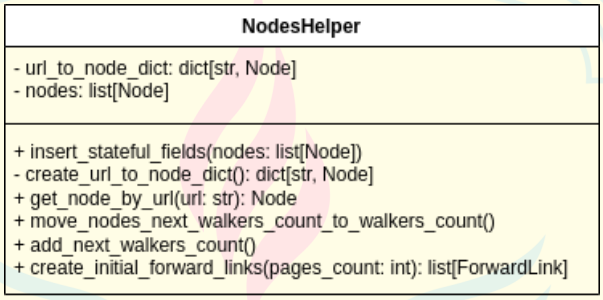
5. Program Random Walker



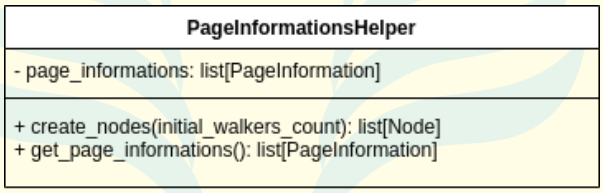
Gambar 4.21: *Class Diagram* RandomWalkerExecutor

Program Random Walker merupakan sebuah simulasi dari Pagerank. Secara program, Random Walker memiliki sebuah *executor class* dan beberapa *factory helper* dan *helper class*. Program dimulai dengan menjalankan berkas Python *run_random_walker.py*. Berkas Python tersebut memanggil *method execute* pada *class* RandomWalkerExecutor. Pertama, diambil data semua PageInformation di basis data melalui PageInformationRepository. Selanjutnya, tiap PageInformation

disimpan ke dalam *class* Node. Node memiliki properti *page_information*, *walkers_count* untuk menampung jumlah *walker* yang berada di Node saat ini, dan *next_walkers_count* untuk menampung jumlah *walker* yang berada di Node pada iterasi selanjutnya.

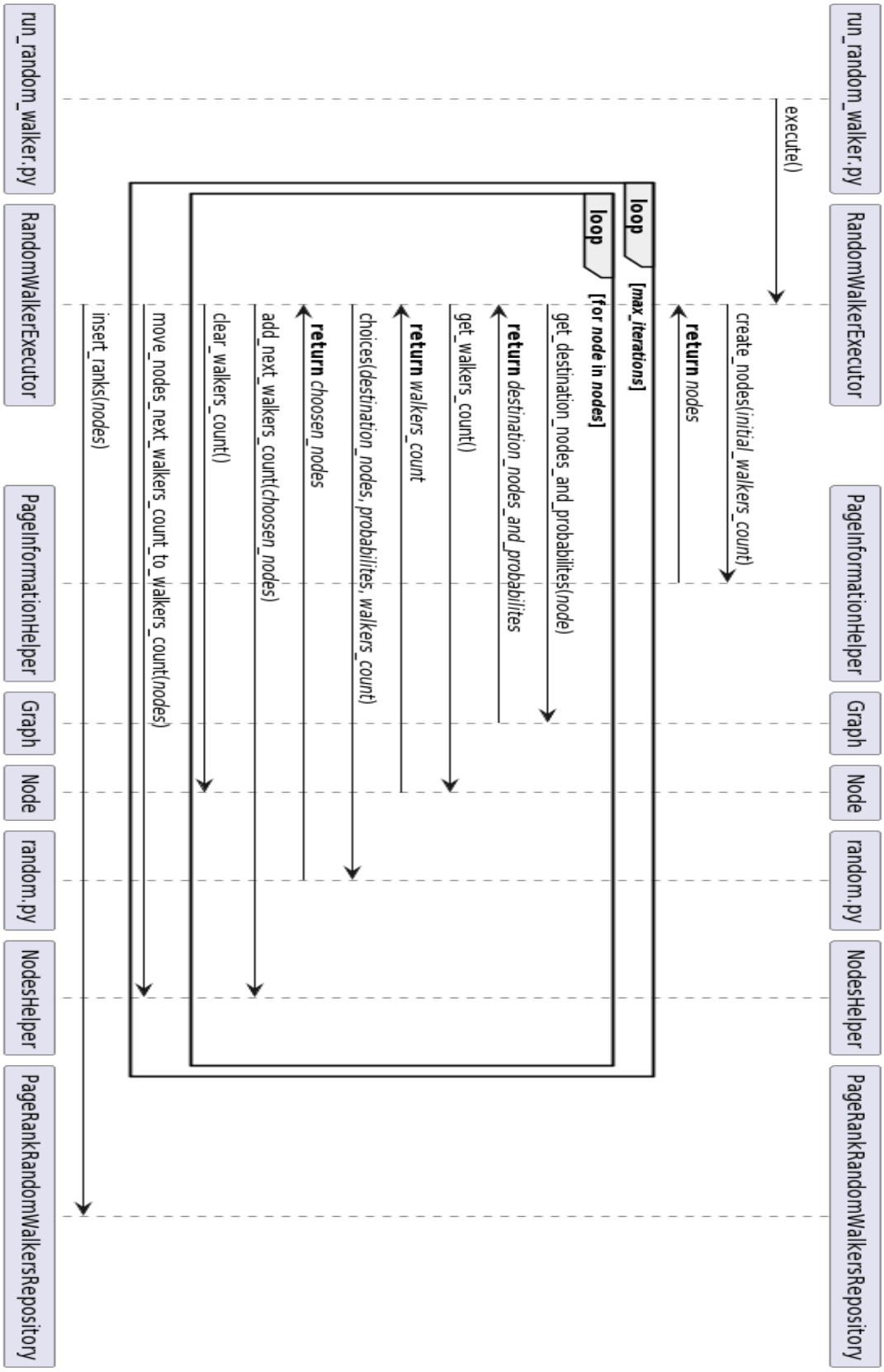


Gambar 4.22: *Class Diagram* NodesHelper



Gambar 4.23: *Class Diagram* PageInformationsHelper

Sebelum memasuki langkah iterasi, akan dilakukan iterasi sebanyak yang ditentukan, pada penelitian ini jumlah iterasinya adalah 20 iterasi. Memasuki iterasi, untuk setiap Node, diambil Node tujuan, dan probabilitas setiap Node tujuan. Melalui fungsi *random.choices* dimasukkan jumlah *walker*, *list* Node tujuan, dan *list* probabilitas Node tujuan. Keluaran dari fungsi *random.choices* adalah *list* Node dari Node tujuan sebanyak jumlah *walker*, yang dianggap sebagai Node tujuan terpilih pada setiap *walker*. Pada setiap Node terpilih dilakukan penambahan nilai *next_walker_count* sebanyak satu. Setelah satu iterasi selesai, nilai *walkers_count* di-perbaharui dengan memindahkan nilai *next_walkers_count* ke *walkers_count*. Langkah pada iterasi diulangi, sampai iterasi selesai. Penjelasan program dapat dilihat dalam bentuk diagram alir pada gambar 4.24.



Gambar 4.24: Diagram alir program Random Walker

B. Hasil

Setelah program dijalankan terhadap semua *dataset*, maka diperoleh hasil berikut. Pada Dataset 2 puncak penggunaan memori pada ketiga algoritma relatif sama, hal ini dikarenakan data yang cukup kecil dibandingkan *overhead* memory pada program. Selanjutnya pada Dataset 1 puncak penggunaan memori terbesar ada pada algoritma pagerank asli yaitu sebesar 3,4 GB dimulai saat matriks 20.493×20.493 P dibentuk. Dengan perhitungan setiap sel matriks merupakan angka desimal *floating point* 64bit maka besarnya matriks P pada memori adalah $\pm$3,4 GB.

Selanjutnya, pada algoritma DPC penggunaan memori terbesar adalah 842 MB yang terjadi pada pembentukan dan penyimpanan matriks P_{*i} ke *cache*. Mengingat domain terbesar di Dataset 1 yaitu "detik.com" dengan jumlah 2.215 halaman, maka dimensi matriks P_{*i} terbesar adalah 20.493×2.215 yang akan memakan memori sebesar ± 363 MB, dan ketika akan disimpan ke dalam *cache* menggunakan *pickle library* akan dilakukan penyalinan objek sebelum ditulis ke dalam *cache*, sehingga penggunaan memori $2 \times \pm 363MB = \pm 726MB$.

Pada algoritma MDPC puncak memori sebesar ± 86 MB, penggunaan memori puncak terjadi pada langkah pembentukan dan penyimpanan matriks Q ke *cache*. Karena matriks Q adalah matriks transisi lokal suatu domain, maka matriks Q terbesar adalah matriks Q domain "detik.com" dengan 2.215 halaman. Ukuran matriks Q yang terbentuk adalah 2.215×2.215 yang secara ukuran di memori adalah ± 39 MB, dan di tengah proses penyimpanan ke dalam *cache* dilakukan penyalinan objek sementara, sehingga ukuran memorinya menjadi dua kali lipat yaitu ± 86 MB. Perbandingan puncak memori dan lamanya waktu eksekusi algoritma Pagerank Original, algoritma Distributed Pagerank Computation (DPC), dan algoritma Modified DPC (MDPC), dapat dilihat pada tabel 4.1.

Tabel 4.1: Puncak penggunaan memori dan waktu eksekusi

Algoritma	**Puncak Penggunaan Memori(*Mega Byte*)**	**Waktu(Detik)**
Dataset 1 (20.493 halaman)		
Pagerank	3.417,63239	328,803
DPC	842,48153	1.151,628
MDPC	86,58194	816,375
Dataset 2 (100 halaman)		
Pagerank	2,04511	0,568
DPC	2,0563	19,372
MDPC	2,05595	0,652

Selanjutnya akan dihitung nilai kemiripan antara nilai Pagerank yang dihasilkan program Pagerank asli, DPC, MDPC, dan Random Walker. Nilai Pagerank berbentuk vektor dan diurutkan berdasarkan *id* halaman. Pada Dataset 1, nilai $KDist$ tiap vektor *ranking* halaman web yang dihasilkan oleh algoritma Pagerank, DPC, MDPC, dan Random Walker dapat dilihat pada tabel 4.2.

Tabel 4.2: Nilai jarak Kendall vektor Pagerank pada dataset 1 (20.493 halaman)

	Pagerank	**DPC**	**MDPC**	**Random Walker**
Pagerank	0,0	0,24956	0,25985	0,02716
DPC	0,24956	0,0	0,27681	0,25208
MDPC	0,25985	0,27681	0,0	0,26387
Random Walker	0,02716	0,25208	0,26387	0,0

Vektor yang paling mirip atau memiliki nilai $KDist$ terkecil adalah vektor Pagerank dan Random Walker yaitu sebesar $0,02716$ atau $2,7$ persen perbedaan urutan. Sedangkan $KDist$ untuk DPC terhadap Pagerank dan Random Walker

secara berurutan adalah $0,24956$ dan $0,25208$ atau 25 persen dan $25,2$ persen perbedaan urutan. Selanjutnya untuk $KDist$ MDPC terhadap DPC, Pagerank, dan Random Walker secara berurutan adalah $0,27681$, $0,25985$, dan $0,26387$ atau $27,7$ persen, 26 persen, dan $26,4$ persen perbedaan urutan.

Pada Dataset 2, nilai $KDist$ tiap vektor *ranking* halaman web yang dihasilkan oleh algoritma Pagerank, DPC, MDPC, dan Random Walker dapat dilihat pada tabel 4.3. Vektor yang paling mirip atau memiliki nilai $KDist$ terkecil adalah vektor Pagerank dan Random Walker yaitu sebesar $0,03293$ atau $3,3$ persen perbedaan urutan. Sedangkan $KDist$ untuk DPC terhadap Pagerank dan Random Walker secara berurutan adalah $0,17131$ dan $0,19131$ atau $17,1$ persen dan $19,1$ persen perbedaan urutan. Selanjutnya untuk $KDist$ MDPC terhadap DPC, Pagerank, dan Random Walker secara berurutan adalah $0,11091$, $0,12586$, dan $0,14949$ atau $11,1$ persen, $12,6$ persen, dan $14,9$ persen perbedaan urutan.

Tabel 4.3: Nilai jarak Kendall vektor Pagerank pada dataset 2 (100 halaman)

	Pagerank	**DPC**	**MDPC**	**Random Walker**
Pagerank	0,0	0,17131	0,12586	0,03293
DPC	0,17131	0,0	0,11091	0,19131
MDPC	0,12586	0,11091	0,0	0,14949
Random Walker	0,03293	0,19131	0,14949	0,0

Secara lebih detil, perbedaan peringkat halaman satu per satu berdasarkan tiap algoritmanya dapat dilihat pada tabel 4.5 untuk Dataset 1, dan tabel 4.7 untuk Dataset 2. Kolom *Rank* menunjukan urutan peringkat halaman web, nilai pada kolom *value* merupakan nilai dari peringkat halaman web yang merupakan nilai peluang ($0 \leq$ *value* ≤ 1), dan pada nilai pada kolom *page_id* merupakan nomor pembeda atau *Primary Key* yang disematkan pada basis data relasional.

Tabel 4.4: Perbandingan peringkat halaman web Dataset 1 (20.493 halaman) bag. 1

Dataset 1 (20.493 halaman)				
	Pagerank		DPC	
Rank	***page_id***	***value***	***page_id***	***value***
1	13.343	0,00975	9.262	0,00489
2	2.858	0,0093	9.276	0,00489
3	11.365	0,00635	9.285	0,00489
4	18.179	0,0048	16.214	0,00489
5	7	0,00465	16.217	0,00489
⋮	⋮	⋮	⋮	⋮
20.489	13.012	0,00001	1.654	0,0
20.490	13.015	0,00001	1.656	0,0
20.491	13.004	0,00001	1.655	0,0
20.492	13.005	0,00001	1.653	0,0
20.493	10.723	0,00001	1.658	0,0

Tabel 4.5: Perbandingan peringkat halaman web Dataset 1 (20.493 halaman) bag. 2

Dataset 1 (20.493 halaman)				
	MDPC		Random Walker	
Rank	***page_id***	***value***	***page_id***	***value***
1	2.087	0,01244	13.343	0,00966
2	13.343	0,00985	2.858	0,00933
3	2.858	0,0093	11.365	0,00621
4	2.859	0,00718	18.179	0,00483
5	699	0,00654	7	0,00466

Dataset 1 (20.493 halaman)				
	MDPC		Random Walker	
Rank	***page_id***	***value***	***page_id***	***value***
⋮	⋮	⋮	⋮	⋮
20.489	16.464	0,0	9.192	0,00001
20.490	16.504	0,0	9.167	0,00001
20.491	15.644	0,0	16.005	0,00001
20.492	13.898	0,0	9.179	0,00001
20.493	16.347	0,0	15.866	0,00001

Tabel 4.6: Perbandingan peringkat halaman web Dataset 2 (100 halaman) bag. 1

Dataset 2 (100 halaman)				
	Pagerank		DPC	
Rank	***page_id***	***value***	***page_id***	***value***
1	72	0,08431	72	0,1306
2	62	0,01856	95	0,02583
3	54	0,01856	91	0,02583
4	55	0,01856	87	0,02583
5	59	0,01856	97	0,02583
⋮	⋮	⋮	⋮	⋮
96	63	0,00199	1	0,00043
97	64	0,00199	25	0,00037
98	80	0,00199	63	0,00037
99	28	0,00184	64	0,00037
100	1	0,0015	80	0,00037

Tabel 4.7: Perbandingan peringkat halaman web Dataset 2 (100 halaman) bag. 2

Dataset 2 (100 halaman)				
	MDPC		Random Walker	
Rank	***page_id***	***value***	***page_id***	***value***
1	72	0,16809	72	0,08381
2	54	0,03158	54	0,01856
3	55	0,03158	62	0,01847
4	59	0,03158	55	0,01847
5	62	0,03158	59	0,01842
⋮	⋮	⋮	⋮	⋮
96	25	0,00048	80	0,002
97	63	0,00048	63	0,00198
98	64	0,00048	64	0,00197
99	80	0,00048	28	0,0018
100	1	0,00022	1	0,00151

Selain perbedaan nilai peringkat, perbedaan selisih nilai atau hasil pengurangan antara pada setiap halaman web antara algoritma satu dengan yang lainnya dapat dilihat pada tabel 4.8 dan tabel 4.9 untuk Dataset 1, dan tabel 4.10 dan tabel 4.11 untuk Dataset 2. Mengetahui perbedaan selisih nilai penting untuk mengetahui seberapa jauh deviasi atau perubahan hasil nilai *ranking* antara keempat algoritma (Pagerank Original, DPC, MDPC, dan Random Walker). Semakin besar nilai selisihnya, semakin besar pula perbedaannya. Hal ini juga merupakan keniscayaan karena perbedaan algoritma yang dipakai. Namun, umumnya, selisih yang dibawah 10^{-5} dapat ditoleransi dan dianggap sama. Oleh karena itu nilai yang ditampilkan di tabel hanya lima angka desimal, atau lima angka di belakang koma.

Tabel 4.8: Selisih nilai peringkat halaman web Dataset 1 (20.493 halaman) Bagian 1

Dataset 1 (20.493 halaman)			
page_id	**Pagerank - DPC**	**Pagerank - MDPC**	**DPC - MDPC**
1	0,00002	0,00001	0,0
2	0,00245	0,00008	0,00237
3	0,00245	0,00008	0,00237
4	0,00251	0,00217	0,00034
5	0,00244	0,0019	0,00054
⋮	⋮	⋮	⋮
20.490	0,00006	0,00001	0,00005
20.491	0,00003	0,00003	0,0
20.492	0,00003	0,00003	0,0
20.493	0,00006	0,00001	0,00005
20.494	0,00003	0,00003	0,0

Tabel 4.9: Selisih nilai peringkat halaman web Dataset 1 (20.493 halaman) bagian 2. Ket: Random Walker (RW)

Dataset 1 (20.493 halaman)			
page_id	**RW - Pagerank**	**RW - DPC**	**RW - MDPC**
1	0,0	0,00002	0,00001
2	0,00001	0,00246	0,00009
3	0,0	0,00246	0,00008
4	0,00001	0,00252	0,00218
5	0,0	0,00244	0,0019
⋮	⋮	⋮	⋮

Dataset 1 (20.493 halaman)			
page_id	**RW - Pagerank**	**RW - DPC**	**RW - MDPC**
20.490	0,0	0,00005	0,0
20.491	0,0	0,00003	0,00003
20.492	0,0	0,00003	0,00003
20.493	0,0	0,00005	0,0
20.494	0,0	0,00003	0,00003

Selanjutnya akan ditampilkan perbedaan selisih keempat algoritma pada Dataset 2 yang berisi 100 halaman web. Sama dengan tabel untuk Dataset 1, angka yang ditampilkan di bawah hanya lima angka di belakang koma.

Tabel 4.10: Selisih nilai peringkat halaman web Dataset 2 (100 halaman) bagian 1

Dataset 2 (100 halaman)			
page_id	**Pagerank - DPC**	**Pagerank - MDPC**	**DPC - MDPC**
1	0,00107	0,00128	0,00021
2	0,00088	0,00112	0,00024
3	0,00156	0,00314	0,0047
4	0,00571	0,00079	0,00492
5	0,00571	0,00079	0,00492
⋮	⋮	⋮	⋮
96	0,00269	0,00257	0,00012
97	0,00784	0,00038	0,00746
98	0,00328	0,00482	0,0081
99	0,00328	0,00348	0,0002
100	0,00277	0,00244	0,00033

Tabel 4.11: Selisih nilai peringkat halaman web Dataset 2 (100 halaman) bagian 2. Ket: Random Walker (RW)

Dataset 2 (100 halaman)			
page_id	**RW - Pagerank**	**RW - DPC**	**RW - MDPC**
1	0,00001	0,00108	0,00128
2	0,00001	0,00089	0,00113
3	0,00001	0,00157	0,00313
4	0,00004	0,00576	0,00083
5	0,00008	0,00563	0,00071
⋮	⋮	⋮	⋮
96	0,00004	0,00266	0,00253
97	0,00004	0,0078	0,00033
98	0,00001	0,00327	0,00483
99	0,00008	0,00336	0,00356
100	0,0	0,00278	0,00244

Perbedaan hasil cukup signifikan antara Distributed Pagerank Computation (DPC) dan Modified DPC (MDPC) terhadap Pagerank Original dan Random Walker disebabkan karena dasar dari perhitungannya yang berbeda. DPC dan MDPC memisahkan halaman web berdasarkan domain-nya, sedangkan Pagerank dan Random Walker melakukan perhitungan secara utuh.

Dapat disimpulkan, algoritma Pagerank Original paling cepat dan paling mirip hasilnya dengan algoritma Random Walker dalam melakukan pe-*ranking*-an halaman web, namun memerlukan memori utama yang lebih besar. Sedangkan algoritma DPC dan MDPC memerlukan memori utama lebih kecil, sehingga cocok untuk komputer satu mesin dengan memori utama terbatas, tetapi dengan

konsekuensi perbedaan hasil yang cukup besar terhadap algoritma Pagerank Original dan algoritma Random Walker dan waktu eksekusi yang lebih lambat.

BAB V

KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan hasil implementasi dan pengujian program Algoritma Pagerank, DPC, MDPC, dan Random Walker, maka diperoleh kesimpulan sebagai berikut:

1. Algoritma Pagerank merupakan algoritma untuk menghitung *ranking* halaman web yang berbasis pada Random Walk di graf halaman web (Page dkk., 1999). Algoritma Pagerank memiliki masalah pada penggunaan memori yang besar. Algoritma DPC yang memakai metode *divide and conquer* (Zhu dkk., 2005) dipakai untuk menjawab permasalahan pada algoritma Pagerank. Algoritma MDPC merupakan modifikasi dari algoritma DPC yang dirumuskan pada penelitian ini karena terdapat langkah-langkah yang bisa disederhanakan pada algoritma DPC. Program simulasi Random Walker dibuat untuk menjadi pembanding dari hasil algoritma Pagerank, DPC, dan MDPC.

2. Dari hasil pengujian algoritma pe-*ranking*-an halaman web tercepat dalam waktu eksekusi dipegang oleh algoritma Pagerank, sedangkan dari segi penggunaan memori puncak, MDPC dan DPC jauh lebih kecil dibandingkan algoritma Pagerank. Walaupun demikian, setelah dilakukan uji hasil *ranking* dengan menghitung nilai $KDist$ antara masing-masing algoritma, hasil dari algoritma Pagerank sangat mirip dengan hasil dari algoritma Random Walker dibandingan dengan algoritma DPC, dan MDPC terhadap Random Walker. Sehingga dapat disimpulkan algoritma DPC dan MDPC cocok untuk komputer satu mesin dengan memori utama terbatas, tetapi dengan mengorbankan kemiripan hasil dan waktu eksekusi lebih lambat.

B. Saran

Adapun saran untuk penelitian selanjutnya adalah:

1. Menguji keempat algoritma lebih lanjut dengan dataset yang lebih besar dan beragam
2. Mencari algoritma alternatif lain dari Pagerank yang secara performa memori dan waktu lebih baik, tetapi perbedaan hasil *ranking* di bawah 10 persen

DAFTAR PUSTAKA

Allah, K. K., Ismail, N. A., dan Almgerbi, M. (2021). Designing web search ui for the elderly community: a systematic literature review. *Journal of Ambient Intelligence and Humanized Computing*.

Amstrong, M. (2021). How many websites are there? https://www.statista.com/chart/19058/number-of-websites-online/. Diakses pada 30-07-2022.

Berners-Lee, T., Cailliau, R., Groff, J.-F., dan Pollermann, B. (1992). World-wide web: The information universe. *Internet Research*, 2(1):52–58.

Brin, S. dan Page, L. (1998). The anatomy of a large-scale hypertextual web search engine.

Chen, Y.-Y., Suel, T., dan Markowetz, A. (2006). Efficient query processing in geographic web search engines. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 277–288.

Courtois, P. dan Semal, P. (1986). Block iterative algorithms for stochastic matrices. *Linear Algebra and its Applications*, 76:59–70.

GeeksForGeeks (2022). Introduction of b-tree. https://www.geeksforgeeks.org/introduction-of-b-tree-2/. Diakses pada 28-09-2022.

GlobalStatCounter (2022). Search engine market share worldwide - july 2022. https://gs.statcounter.com/search-engine-market-share#monthly-200901-202208-bar. Diakses pada 28-09-2022.

Huss, N. (2022). How many websites are there in the world? (2022). https://siteefy.com/how-many-websites-are-there/. Diakses pada 28-09-2022.

Khatulistiwa, L. (2022). Perancangan arsitektur search engine dengan mengintegrasikan web crawler, algoritma page ranking, dan document ranking.

Kontovasilis, K. P. dan Mitrou, N. M. (1995). Markov-modulated traffic with nearly complete decomposability characteristics and associated fluid queueing models. *Advances in Applied Probability*, 27(4):1144–1185.

Mishra, D. (2016). Proper weak regular splitting and its application to convergence of alternating iterations.

MySQLDoc (2022). How mysql uses indexes. https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html. Diakses pada 30-07-2022.

Neumann, M. dan Plemmons, R. J. (1978). Convergent nonnegative matrices and iterative methods for consistent linear systems. *Numerische Mathematik*, 31(3):265–279.

Page, L., Brin, S., Motwani, R., dan Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab.

Pratama, Z. (2022). Perancangan modul pengindeks pada search engine berupa induced generalized suffix tree untuk keperluan perangkingan dokumen.

Qoriiba, M. F. (2021). Perancangan crawler sebagai pendukung pada search engine.

Sample, I. (2018). What is the internet? 13 key questions answered. https://www.theguardian.com/technology/2018/oct/22/what-is-the-internet-13-key-questions-answered. Diakses pada 30-07-2022.

Seymour, T., Frantsvog, D., dan Kumar, S. (2011). History of search engines. *International Journal of Management & Information Systems (IJMIS)*, 15(4):47.

Techopedia (2020). Website. https://www.techopedia.com/definition/5411/website. Diakses pada 30-07-2022.

Wilson, R. J. (1996). *Introduction to Graph Theory*. Longman Group Ltd.

Zalghornain, M. (2022). Rancang bangun sistem pencarian teks dengan menggunakan model continuous-bag-of-words dan model continuous skip-gram pada koleksi dokumen.

Zhu, Y., Ye, S., dan Li, X. (2005). Distributed pagerank computation based on iterative aggregation-disaggregation methods. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 578–585.

LAMPIRAN KODE PROGRAM

Kode 1: *Class* DB

```
1  from mysql.connector import connect, CMySQLConnection
2  from mysql.connector.pooling import PooledMySQLConnection
3  from mysql.connector.connection import MySQLConnection
4  from typing import Union
5
6  DB_Connection = Union[PooledMySQLConnection, MySQLConnection,
       CMySQLConnection]
7
8  class DB:
9      """wrapper class for mysql db connection"""
10
11     __host: str
12     __port: str
13     __username: str
14     __password: str
15     __db_name: str
16     __db_conn: DB_Connection
17
18     def __init__(self, host: str, port: str, username: str, password:
           str, db_name: str) -> None:
19         self.__host = host
20         self.__port = port
21         self.__username = username
22         self.__password = password
23         self.__db_name = db_name
24         self.__db_conn = self.construct_db_conn()
25
26     def construct_db_conn(self) -> DB_Connection:
27         """
28         create and get new db connection
29         """
30
31         return connect(host=self.__host, port=self.__port, user=self.
               __username, password=self.__password, database=self.
               __db_name)
32
33     def get_db_conn(self) -> DB_Connection:
34         """
35         get existing db connection
36         """
37
38         return self.__db_conn
39
40     def close(self):
41         """
42         close db connection
```

```
43      """
44
45      self.__db_conn.close()
```

Kode 2: *Class* PageInformationRepository

```
1   from typing import cast
2   from data.db import DB, DB_Connection
3   from model import PageInformation
4   from mysql.connector.types import RowType
5
6
7   class PageInformationRepository:
8     """
9     Intermediary for CRUD page_information table in database
10    """
11
12    __db: DB
13    __table_name = "page_information"
14
15    def __init__(self, db: DB) -> None:
16      self.__db = db
17
18    def get_all_page_informations(self, use_existing_db_conn = True)
        -> list[PageInformation]:
19      """
20      get all page_information entries
21      """
22
23      db_conn = self.__db.get_db_conn() if(use_existing_db_conn) else
          self.__db.construct_db_conn()
24      page_informations = self.
          __get_all_page_informations_from_db_conn(db_conn)
25      output = []
26
27      if(not use_existing_db_conn):
28        db_conn.close()
29
30      for page_information in page_informations:
31        output.append(PageInformation(0, cast(int, page_information
            [0]), cast(str, page_information[1])))
32
33      return output
34
35    def __get_all_page_informations_from_db_conn(
36      self,
37      db_conn: DB_Connection
38    ) -> list[RowType]:
39      cursor = db_conn.cursor()
40      cursor.execute(f"SELECT id_page, url FROM {self.__table_name}")
41      output = cursor.fetchall()
42      cursor.close()
```

```
43      return output
44
45      def get_page_information_by_id(self, page_information_id: int) ->
            PageInformation:
46          """
47          get a page_information entry with matched id, raise exception
                if not exist
48          """
49
50          db_conn = self.__db.get_db_conn()
51          cursor = db_conn.cursor()
52          cursor.execute(f"SELECT id_page, url FROM {self.__table_name}
                WHERE id_page={page_information_id}")
53          query_result = cursor.fetchone()
54
55          if(query_result is None):
56              raise Exception("page_information_id not exist")
57
58          return PageInformation(0, cast(int, query_result[0]), cast(str,
                query_result[1]))
59
60      def update_backlink_count_by_page_ids(self, page_ids: list[int],
            backlink_counts: list[int]):
61          """
62          update backlink_count value of page_information entries with
                matching page_ids argument respectively
63          """
64
65          if(len(page_ids) != len(backlink_counts)):
66              raise Exception("page_ids and backlink_counts must have same
                    length")
67
68          db_conn = self.__db.get_db_conn()
69          cursor = db_conn.cursor()
70          query = f"UPDATE {self.__table_name} SET backlink_count=%s
                WHERE id_page=%s"
71          query_values: list[tuple[int, int]] = []
72
73          for i in range(len(page_ids)):
74              query_values.append((backlink_counts[i], page_ids[i]))
75
76          cursor.executemany(query, query_values)
77          db_conn.commit()
78          cursor.close()
79
80      def update_domain_id_by_page_ids(self, page_ids: list[int],
            domain_ids: list[int]):
81          """
82          update domain_id value of page_information entries with
                matching page_ids argument respectively
83          """
84
```

```
85      if(len(page_ids) != len(domain_ids)):
86        raise Exception()
87
88      db_conn = self.__db.get_db_conn()
89      cursor = db_conn.cursor()
90      query = f"UPDATE {self.__table_name} SET domain_id=%s WHERE
            id_page=%s"
91      query_values: list[tuple[int, int]] = []
92
93      for i in range(len(page_ids)):
94        query_values.append((domain_ids[i], page_ids[i]))
95
96      cursor.executemany(query, query_values)
97      db_conn.commit()
98      cursor.close()
99
100   def update_backlink_count_by_page_id(self, page_id: int,
          backlink_count: int):
101     """
102     update backlink_count value of page_information entries with
            matching page_ids argument respectively
103     """
104
105     db_conn = self.__db.get_db_conn()
106     cursor = db_conn.cursor()
107
108     query = f"UPDATE {self.__table_name} SET backlink_count={
            backlink_count} WHERE id_page={page_id}"
109     cursor.execute(query)
110     db_conn.commit()
111
112     cursor.close()
```

Kode 3: Kode untuk *caching*

```
1   """
2   this package contain Cache classes that act as intermediary between
        the program and pickle library
3   """
4
5   from pickle import dump, load
6   from numpy.typing import NDArray
7   from numpy import float_
8   from typing import Any
9
10  class Cache:
11    """
12    Properties:
13      - folder_path: .pkl file folder that store .pkl files
14
15    Methods:
16      - private create_file(self, file_name: str) -> str
```

```
17          - create .pkl file
18
19      - protected dump(self, file_name: str, obj: Any)
20          - dump object into .pkl file
21
22      - protected load(self, file_name: str) -> Any
23          - load .pkl file into python object
24    """
25
26    __folder_path: str
27
28    def __init__(self, folder_path: str) -> None:
29      self.__folder_path = folder_path
30
31    def __create_file(self, file_name: str) -> str:
32      file_location = f"{self.__folder_path}/{file_name}"
33
34      try:
35        open(file_location, "x").close()
36      except FileExistsError:
37        pass
38      finally:
39        return file_location
40
41    def _dump(self, file_name: str, obj: Any):
42      file_location = self.__create_file(file_name)
43      with open(file_location, "wb") as file_write:
44        dump(obj, file_write)
45
46    def _load(self, file_name: str) -> Any:
47      file_location = f"{self.__folder_path}/{file_name}"
48
49      try:
50        with open(file_location, "rb") as file_open:
51          return load(file_open)
52      except FileNotFoundError:
53        return None
54
55  class PCache(Cache):
56    """
57    caching for P matrix
58
59    Properties:
60      - file_name: .pkl file name format
61    """
62
63    __file_name = "col_{}.pkl"
64
65    def dump_column(self, col_index: int, column: NDArray[float_]):
66      """
67      dump matrix P by its column into .pkl file / cache
68
```

```
69      Args:
70        - col_index: column index of matrix P
71        - column: N x 1 vector of matrix P column
72      """
73
74      self._dump(self.__file_name.format(str(col_index)), column)
75
76
77    def load_column(self, col_index: int) -> NDArray[float_]:
78      """
79      load P matrix column from .pkl file / cache
80
81      Args:
82        - col_index: column index of matrix P
83
84      Returns:
85        - N x 1 P column vector
86      """
87
88      return self._load(self.__file_name.format(str(col_index)))
89
90  class QListCache(Cache):
91    """
92    caching for Q matrix
93
94    Properties:
95      - file_name: .pkl file name format
96    """
97
98    __file_name = "Q_{}.pkl"
99
100   def dump_q(self, cluster_no: int, q: NDArray[float_]):
101     """
102     dump Ni x Ni matrix Qi into .pkl file / cache
103
104     Args:
105       - cluster_no: "i" number in Qi
106       - q: Qi matrix
107     """
108
109     self._dump(self.__file_name.format(cluster_no), q)
110
111   def load_q(self, cluster_no: int) -> NDArray[float_]:
112     """
113     load Ni x Ni matrix Qi from .pkl file / cache
114
115     Args:
116       - cluster_no: "i" number in Qi
117
118     Returns:
119       Ni x Ni Qi matrix
120     """
```

```
121
122         return self._load(self.__file_name.format(cluster_no))
123
124 class PiiListCache(Cache):
125     """
126     caching for Pii matrix
127
128     Properties:
129       - file_name: .pkl file name format
130     """
131
132     __file_name = "P_{}{}.pkl"
133
134     def dump_pii(self, cluster_no: int, pii: NDArray[float_]):
135         """
136         dump Ni x Ni matrix Pii into cache
137
138         Args:
139           - cluster_no: "i" number in Pii
140           - pii: Pii matrix
141         """
142
143         self._dump(self.__file_name.format(cluster_no, cluster_no), pii
                )
144
145     def load_pii(self, cluster_no: int) -> NDArray[float_]:
146         """
147         load Ni x Ni matrix Pii from cache
148
149         Args:
150           - cluster_no: "i" number in Pii
151
152         Returns:
153           Ni x Ni matrix Pii
154         """
155
156         return self._load(self.__file_name.format(cluster_no,
                cluster_no))
157
158 class PiastListCache(Cache):
159     """
160     caching for Pi* (Pi-ast) matrix
161
162     Properties:
163       - file_name: .pkl file name format
164     """
165
166     __file_name = "P_{}_ast.pkl"
167
168     def dump_piast(self, cluster_no: int, piast: NDArray[float_]):
169         """
170         dump Ni x N matrix Pi* into cache
```

```
171
172         Args:
173           - cluster_no: "i" number in Pi*
174           - piast: Pi* matrix
175         """
176
177         self._dump(self.__file_name.format(cluster_no), piast)
178
179       def load_piast(self, cluster_no: int) -> NDArray[float_]:
180         """
181         load Ni x N matrix Pi*  from cache
182
183         Args:
184           - cluster_no: "i" number in Pi*
185
186         Returns:
187           Ni x N matrix Pi*
188         """
189
190         return self._load(self.__file_name.format(cluster_no))
191
192     class PastiListCache(Cache):
193       """
194       caching for P*i (Past-i) matrix
195
196       Properties:
197         - file_name: .pkl file name format
198       """
199
200       __file_name = "P_ast_{}.pkl"
201
202       def dump_pasti(self, cluster_no: int, pasti: NDArray[float_]):
203         """
204         dump N x Ni matrix P*i into cache
205
206         Args:
207           - cluster_no: "i" number in P*i
208           - pasti: P*i matrix
209         """
210
211         self._dump(self.__file_name.format(cluster_no), pasti)
212
213       def load_pasti(self, cluster_no: int) -> NDArray[float_]:
214         """
215         load N x Ni matrix P*i from cache
216
217         Args:
218           - cluster_no: "i" number in P*i
219
220         Returns:
221           N x Ni matrix P*i
222         """
```

```
223
224         return self._load(self.__file_name.format(cluster_no))
225
226 class RPCache(Cache):
227     """
228     caching for RP matrix
229
230     Properties:
231         - file_name: .pkl file name
232     """
233
234     __file_name = "RP.pkl"
235
236     def dump_rp(self, rp: NDArray[float_]):
237         """
238         dump RP matrix into cache
239
240         Args:
241             - rp: RP matrix
242         """
243
244         self._dump(self.__file_name, rp)
245
246     def load_rp(self) -> NDArray[float_]:
247         """
248         load RP matrix from cache
249
250         Returns:
251             RP matrix
252         """
253
254         return self._load(self.__file_name)
```

Kode 4: *Class* Model

```
1  """
2  package that contain plain data class without complex logic
3  """
4
5  class PageInformation():
6      """
7      representation of page_information table entry
8      """
9
10     index: int
11     id_page: int
12     url: str
13
14     def __init__(self, index: int, id_page: int, url: str) -> None:
15         self.index = index
16         self.id_page = id_page
17         self.url = url
```

```
18
19
20  class PageLinking():
21    """
22    representation of page_linking table entry
23    """
24
25    outgoing_link: str
26
27    def __init__(self, outgoing_link: str) -> None:
28      self.outgoing_link = outgoing_link
29
30  class Domain():
31    """
32    representation of domain_information table entry
33    """
34
35    domain_id: int
36    url: str
37    mdpcv2_rank_value: float
38    dpc_rank_value: float
39    random_walkers_count: int
40    random_walkers_count_normalized: float
41
42    def __init__(self, domain_id: int, url: str, mdpcv2_rank_value:
        float, dpc_rank_value: float, random_walkers_count: int,
        random_walkers_normalized: float) -> None:
43      self.domain_id = domain_id
44      self.url = url
45      self.mdpcv2_rank_value = mdpcv2_rank_value
46      self.dpc_rank_value = dpc_rank_value
47      self.random_walkers_count = random_walkers_count
48      self.random_walkers_count_normalized =
          random_walkers_normalized
49
50  class Node:
51    """
52    Node representation for random_walker algorithm
53    """
54
55    __page_information: PageInformation
56    __walkers_count: int
57    __next_walkers_count: int
58
59    def __init__(self, page_information: PageInformation,
        walkers_count: int) -> None:
60      self.__page_information = page_information
61      self.__walkers_count = walkers_count
62      self.__next_walkers_count = 0
63
64    def clear_walkers_count(self):
65      self.__walkers_count = 0
```

```
66
67    def add_next_walkers_count(self):
68      self.__next_walkers_count += 1
69
70    def move_next_walkers_count_to_walkers_count(self):
71      self.__walkers_count = self.__next_walkers_count
72      self.__next_walkers_count = 0
73
74    def get_walkers_count(self):
75      return self.__walkers_count
76
77    def get_next_walkers_count(self):
78      return self.__next_walkers_count
79
80    def get_page_information(self):
81      """get copy of PageInformation which this node has"""
82
83      return PageInformation(self.__page_information.index, self.
            __page_information.id_page, self.__page_information.url)
84
85    def get_page_information_index(self) -> int:
86      """get index of page_information that has been sorted by its
            cluster"""
87
88      return self.__page_information.index
```

Kode 5: *helper functions*

```
1  """
2  this package contain helper classes and functions that are used for
       all page ranking methods
3  """
4
5  import tracemalloc
6  from numpy import float_, full, float64
7  from numpy.typing import NDArray
8  from model import PageInformation
9  from urllib.parse import urlparse
10 from typing import cast
11 from time import time
12 from datetime import datetime
13
14 def l1_norm(list: NDArray[float_]) -> float_:
15   """
16   calculate of level 1 normalization of N x 1 list or vector
17   """
18
19   total = cast(float_, 0)
20   for i in list:
21     total += abs(i)
22
23   return total
```

```
24
25  def create_url_to_page_information_dict(page_informations: list[
        PageInformation]) -> dict[str, PageInformation]:
26      """
27      create a python dictionary with PageInformation.url as key, and
            the PageInformation itself as the content
28      """
29
30      url_to_page_information = {}
31
32      for page_information in page_informations:
33          url_to_page_information[page_information.url] =
                page_information
34
35      return url_to_page_information
36
37  def sort_page_information_by_domain(clusters: dict[str, list[
        PageInformation]]) -> list[PageInformation]:
38      """
39      convert clusters dictionary into PageInformation-s list sorted
            with following logic:
40
41      {"abc.com": [page_information0, page_information1], "aaa.com": [
            page_information2]} -> [page_information0, page_information1,
            page_information2]
42      """
43
44      page_informations = []
45      for cluster in clusters.values():
46          for index, page_information in enumerate(cluster):
47              page_information.index = len(page_informations) + index
48          page_informations += cluster
49
50      return page_informations
51
52  class PageInformationsClusterizer:
53      """
54      helper class related to clusterizing PageInformation-s list into
            a cluster dictionary
55      """
56
57      def clusterize(self, page_informations: list[PageInformation]) ->
            dict[str, list[PageInformation]]:
58          """
59          convert PageInformation-s list into a cluster dictionary
60          """
61
62          clusters: dict[str, list[PageInformation]] = {}
63
64          for page_information in page_informations:
65
66              try:
```

```
67          domain = self.get_domain(page_information)
68          if(domain in clusters.keys()):
69            clusters[domain].append(page_information)
70          else:
71            clusters[domain] = [page_information]
72        except TypeError as e:
73          print(e)
74
75      return clusters
76
77    def get_domain(self, page_information: PageInformation) -> str:
78      """
79      extract domain from PageInformation.url
80      """
81
82      parse_result = urlparse(page_information.url)
83      domain = parse_result.hostname
84
85      if(domain is None):
86        raise TypeError(f"cannot get hostname from {page_information.
            url}")
87
88      if(domain[0:4] == "www."):
89          domain = domain[4:]
90
91      return domain
92
93  def full_matrix(shape, value: float) -> NDArray[float64]:
94    """
95    wrapper function to create matrix or factor using numpy.full
96    """
97
98    return full(shape, value, dtype=float64)
99
100 def show_and_reset_smallest_and_highest_memory_usage(step_name: str
      , start_time: float | None = None):
101   """
102   display peak and lowest memory usage, and reset the tracemalloc
        after that
103   """
104
105   traced_memory = tracemalloc.get_traced_memory()
106   print("\n==========")
107   print(f"{step_name};{get_time_string(start_time)}")
108   print(f"smallest memory usage: {traced_memory[0]} Bytes")
109   print(f"highest memory usage: {traced_memory[1]} Bytes")
110   print("==========\n")
111   tracemalloc.reset_peak()
112
113 def get_time_string(start_time: float | None) -> str:
114   if(start_time is None):
115     return ""
```

```
116
117     return f" seconds since start: {round(time() - start_time, 3)};
            time: {get_datetime_string_now()}"
118
119 def get_datetime_string_now() -> str:
120     """
121     get current datetime string with format YYYY/MM/DD HH:mm:ss
122     """
123
124     now = datetime.now()
125     return f"{now.year}/{now.month}/{now.day} {now.hour}:{now.minute
            }:{round(now.second)}"
```

Kode 6: PHelper

```
1  from abstracts import Multithreadable
2  from typing import Callable, Optional
3  from numpy.typing import NDArray
4  from numpy import float_
5  from model import PageInformation, PageLinking
6  from concurrent.futures import ThreadPoolExecutor
7  from shared_helpers import create_url_to_page_information_dict,
       full_matrix
8  import threading
9
10
11 class PHelper(Multithreadable):
12     """
13     Helper class for creating full P matrix without division or
           caching base on DPC paper
14     """
15
16     def create_p(
17         self,
18         d: float,
19         page_informations: list[PageInformation],
20         get_page_linkings: Callable[[int, Optional[bool]], list[
               PageLinking]]
21     ) -> NDArray[float_]:
22         """
23         Create full P matrix base on DPC paper without caching or
               division
24
25         Args:
26             - d: damping factor, usually 0.85
27             - page_informations: N-length list of PageInformation-s
28             - get_page_linkings: method or function to get all page
                   linkings of page information id
29
30         Returns:
31             N x N "P" matrix or transition matrix
32         """
```

```
33
34      print(f"Create P matrix start")
35
36      pages_count = len(page_informations) # N in paper
37      url_to_page_information = create_url_to_page_information_dict(
            page_informations)
38      p = full_matrix((pages_count, pages_count), 0)
39
40      if(self._is_multithread):
41        with ThreadPoolExecutor(self._max_workers) as executor:
42          executor.map(
43            self.__fill_p_column_values,
44            [p[:, col_no] for col_no in range(pages_count)],
45            [pages_count for _ in range(pages_count)],
46            [page_information for page_information in
                page_informations],
47            [url_to_page_information for _ in range(pages_count)],
48            [get_page_linkings for _ in range(pages_count)],
49            [d for _ in range(pages_count)],
50          )
51      else:
52        for i in range(pages_count):
53          self.__fill_p_column_values(
54            p[:, i],
55            pages_count,
56            page_informations[i],
57            url_to_page_information,
58            get_page_linkings,
59            d
60          )
61
62      print(f"Create P matrix done")
63      return p
64
65    def __fill_p_column_values(
66      self,
67      p_column: NDArray,
68      pages_count: int,
69      page_information: PageInformation,
70      url_to_page_information: dict[str, PageInformation],
71      get_page_linkings: Callable[[int, Optional[bool]], list[
            PageLinking]],
72      d: float
73    ):
74      try: # need to wrap it with try block because, in Threadpool,
            error will not be displayed
75        page_linkings = get_page_linkings(page_information.id_page,
              False)
76        cj = len(page_linkings)
77
78        self.__init_p_column_values(p_column, pages_count, cj, d)
79
```

```
80          for page_linking in page_linkings:
81            target_page_information: PageInformation
82
83            try:
84              target_page_information = url_to_page_information[
                    page_linking.outgoing_link]
85            except KeyError as e:
86              print(page_linking.outgoing_link)
87
88              raise e
89
90            target_index = target_page_information.index
91            p_column[target_index] = d/cj + (1-d)/pages_count
92
93        except Exception as e:
94          print(f"create_p_column_values: {e}")
95
96        print(f"fill_p_column_values for page_index = {page_information
              .index} done; thread_id={threading.get_native_id()}")
97
98      def __init_p_column_values(self, p_column: NDArray, pages_count:
            int, cj: int, d: float):
99        initial_p_column_values = full_matrix(pages_count, (1-d)/
              pages_count if(cj > 0) else 1/pages_count)
100
101       for i in range(pages_count):
102         p_column[i] = initial_p_column_values[i]
```

Kode 7: Program Pagerank

```
1  from factories import close_db, create_p_helper,
       create_page_information_repository,
       create_page_informations_clusterizer,
       create_page_linking_repository, create_page_rank_repository
2  from methods.original_pagerank_dpc_paper_version.pagerank import
       pagerank_dpc_paper_version
3  from consts import DAMPING_FACTOR, MAX_ITERATION, EPSILON
4  from shared_helpers import
       show_and_reset_smallest_and_highest_memory_usage,
       sort_page_information_by_domain, full_matrix
5  from time import time
6  import tracemalloc
7
8  def main():
9    start_time = time()
10   tracemalloc.start()
11   show_and_reset_smallest_and_highest_memory_usage("start",
         start_time)
12
13   page_information_repository = create_page_information_repository
         ()
```

```
14    page_informations = page_information_repository.
          get_all_page_informations()
15    clusterizer = create_page_informations_clusterizer()
16    page_informations = sort_page_information_by_domain(clusterizer.
          clusterize(page_informations))
17    page_linking_repository = create_page_linking_repository()
18    p_helper = create_p_helper()
19    show_and_reset_smallest_and_highest_memory_usage("instantiate
          step", start_time)
20
21
22    p = p_helper.create_p(
23      DAMPING_FACTOR,
24      page_informations,
25      page_linking_repository.get_page_linkings_by_page_id
26    )
27    show_and_reset_smallest_and_highest_memory_usage("create p matrix
           step", start_time)
28
29    phi = full_matrix(len(p), 1/len(p))
30    phi = pagerank_dpc_paper_version(phi, p, EPSILON, MAX_ITERATION)
31    show_and_reset_smallest_and_highest_memory_usage("computation
          step", start_time)
32
33    page_rank_repository = create_page_rank_repository("
          page_rank_original_pagerank_dpc_paper_version")
34    page_rank_repository.insert_ranks(page_informations, phi)
35
36    show_and_reset_smallest_and_highest_memory_usage("original
          pagerank end", start_time)
37    close_db()
38    tracemalloc.stop()
39
40  if(__name__ == "__main__"):
41    main()
```

Kode 8: Fungsi Pagerank pada Program Pagerank

```
1  from typing import cast
2  from numpy.typing import NDArray
3  from numpy import float_
4  from shared_helpers import l1_norm
5
6  def pagerank_dpc_paper_version(
7    phi: NDArray[float_],
8    p: NDArray[float_],
9    epsilon: float,
10   max_iteration: int
11 ) -> NDArray[float_]:
12   """
13   page rank computation from DPC paper
14
```

```
15    Args:
16      epsilon: maximum l1 norm difference/delta between pagerank
            vector current iteration
17      p: transition N x N matrix
18      phi: initial pagerank N x 1 vector
19      max_iteration: iteration maximum
20
21    Returns:
22      phi: final pagerank N x 1 vector
23    """
24
25    iteration = 0
26
27    while True:
28      new_phi = cast(NDArray[float_], p @ phi)
29      new_phi = cast(NDArray[float_], new_phi / l1_norm(new_phi))
30
31      delta = l1_norm(phi - new_phi)
32      print(f"iteration={iteration}; delta={delta}")
33
34      if(delta < epsilon or iteration > max_iteration):
35        return new_phi
36
37      iteration += 1
38      phi = new_phi
```

Kode 9: ClusterSeparatedPhiHelper

```
1  from abstracts import Multithreadable
2  from concurrent.futures import ThreadPoolExecutor
3  from model import PageInformation
4  from cache import QListCache
5  from numpy import float_, concatenate, array
6  from numpy.typing import NDArray
7  from typing import Callable
8  from shared_helpers import full_matrix
9
10 class ClusterSeparatedPhiHelper(Multithreadable):
11   def construct_cluster_separated_phi(
12     self,
13     clusters: list[list[PageInformation]],
14     q_list_cache: QListCache,
15     pagerank_dpc: Callable[[NDArray[float_], NDArray[float_], float
           , int, str], NDArray[float_]],
16     epsilon: float,
17     pagerank_dpc_max_iteration: int
18   ) -> list[NDArray[float_]]:
19     """
20     contruct Ni x 1 pagerank vector for each cluster/domain
21
22     Args:
```

```
23        - clusters: page informations that are grouped by their
              domain forming n x Ni nested list. First level contain
              list of page informations that have common domain, while
              second level contain the page information itself
24        - q_list_cache: object to access matrix Q in cache
25        - pagerank_dpc: function or method to computing pagerank
              computation
26        - epsilon: maximum l1 norm difference/delta between pagerank
              vector current iteration
27        - pagerank_dpc_max_iteration: max iteration for pagerank
              computation
28
29      Returns:
30        n-length list containing Nix1 pagerank vektor
31      """
32
33      return [
34        pagerank_dpc(
35          full_matrix(len(clusters[cluster_no]),1/len(clusters[
                cluster_no])),
36          q_list_cache.load_q(cluster_no),
37          epsilon,
38          pagerank_dpc_max_iteration,
39          f"cluster_separated_phi-{cluster_no}"
40        ) for cluster_no in range(len(clusters))
41      ]
42
43    def flatten_cluster_separated_phi(self, clusters_separated_phi:
          list[NDArray[float_]]) -> NDArray[float_]:
44      """
45      flatten n-length list of Nix1 pagerank vector that still
            separated by its cluster into N x 1 pagerank vector
46
47      ex: [[0.1, 0.2], [0.3]] -> [0.1, 0.2, 0.3]
48
49      Args:
50        - clusters_separated_phi: n-length list containing Nix1
              pagerank vektor
51
52      Returns:
53        N x 1 pagerank vector
54      """
55
56      output = array([])
57      for phi in clusters_separated_phi:
58        output = concatenate((output, phi), axis = 0)
59
60      return output
61
62    def get_new_cluster_separated_phi(
63      self, z: NDArray[float_],
64      extended_local_pagerank_list: list[NDArray[float_]]
```

```
65    ) -> list[NDArray[float_]]:
66      """
67      get new cluster separated phi base on "z" vector and extended
            local pagerank vectors see DPC algorithm in the paper for
            detail
68
69      Args:
70        - z: n x 1 cluster rank vector
71        - extended_local_pagerank_list: n-length list containing (Ni
              +1) x 1 extended local pagerank/extended cluster-separated
              phi
72
73      Returns:
74        n-length list containing new Ni x 1 local pagerank/extended
              cluster-separated phi vector
75      """
76
77      clusters_count = len(extended_local_pagerank_list)
78
79      return [
80        self.__update_phi_per_cluster(
81          z[cluster_no], extended_local_pagerank_list[cluster_no]
82        ) for cluster_no in range(clusters_count)
83      ]
84
85    def __update_phi_per_cluster(
86      self, z_cluster: float,
87      extended_local_pagerank: NDArray[float_]
88    ) -> NDArray[float_]:
89      beta = extended_local_pagerank[-1]
90      omega = extended_local_pagerank[0:-1]
91
92      return (1 - z_cluster)/beta * omega
93
94    def construct_s(self, cluster_separated_phi: list[NDArray[float_
        ]]) -> NDArray[float_]:
95      """
96      construct N x n matrix S
97
98      Args:
99        - cluster_separated_phi: n-length list containing Ni x 1
              local pagerank vector
100
101     Returns:
102       N x n matrix S
103     """
104
105     pages_count = self.__get_pages_count(cluster_separated_phi)
106     s = full_matrix((pages_count, len(cluster_separated_phi)), 0)
107     row_index = 0
108     for col_index, phi in enumerate(cluster_separated_phi):
109       s[row_index:row_index+len(phi), col_index] = phi
```

```
110         row_index += len(phi)
111       return s
112
113     def __get_pages_count(self, cluster_separated_phi: list[NDArray[
            float_]]) -> int:
114       pages_count = 0
115       for phi in cluster_separated_phi:
116         pages_count += phi.shape[0]
117
118       return pages_count
```

Kode 10: ExtendedLocalTransitionMatrixHelper

```
1  from cache import PiiListCache, PiastListCache, PastiListCache
2  from numpy.typing import NDArray
3  from numpy import float_
4  from shared_helpers import full_matrix
5
6  class ExtendedLocalTransitionMatrixHelper:
7    def construct_extended_local_transition_matrice(
8      self, pii_list_cache: PiiListCache, piast_list_cache:
           PiastListCache,
9      pasti_list_cache: PastiListCache, s: NDArray[float_], z:
           NDArray[float_],
10     cluster_separated_phi: list[NDArray[float_]], pages_count: int
11   ) -> list[NDArray[float_]]:
12     """
13     construct n-length list containing (Ni+1) x (Ni+1) extended
           local transition matrix, refer to the DPC paper for more
           detail
14
15     Args:
16       - pii_list_cache: object to access Ni x Ni "pii" matrix in
             cache
17       - piast_list_cache: object to access Ni x N "pi*" matrix in
             cache
18       - pasti_list_cache: object to access N x Ni "p*i" matrix in
             cache
19       - s: N x n "s" matrix
20       - z: n x 1 cluster rank vector
21       - cluster_separated_phi: n-length list containing Ni x 1
             local pagerank
22       - pages_count: all page informations count in database or
             known as N
23
24     Returns:
25       n-length list containing (Ni+1) x (Ni+1) extended local
             transition matrix
26     """
27
28     clusters_count = len(cluster_separated_phi)
29
```

```
30      return [self.__construct_extended_local_transition_matrix(
31        pii_list_cache,
32        piast_list_cache,
33        pasti_list_cache,
34        s, z, cluster_separated_phi[cluster_no],
35        cluster_no, pages_count
36      ) for cluster_no in range(clusters_count)]
37
38    def __construct_extended_local_transition_matrix(
39      self, pii_list_cache: PiiListCache, piast_list_cache:
          PiastListCache, pasti_list_cache: PastiListCache,
40      s: NDArray[float_], z: NDArray[float_], cluster_phi: NDArray[
          float_], cluster_no: int, pages_count: int
41    ) -> NDArray[float_]:
42      ni = cluster_phi.shape[0]
43      b = full_matrix((ni + 1, ni + 1), 0)
44      b[0:ni, 0:ni] = pii_list_cache.load_pii(cluster_no)
45      b[ni, 0:ni] = full_matrix(pages_count, 1) @ pasti_list_cache.
          load_pasti(cluster_no)
46      right_most_col = full_matrix(ni+1, 0)
47      piast_s_z = piast_list_cache.load_piast(cluster_no) @ s @ z
48      right_most_col[0:ni] = (piast_s_z - (pii_list_cache.load_pii(
          cluster_no) @ cluster_phi * z[cluster_no]))/(1-z[cluster_no
          ])
49      right_most_col[ni] = 1 - sum(right_most_col)
50
51      return b
```

Kode 11: PWithCacheHelper

```
1  from cache import PCache
2  from model import PageInformation, PageLinking
3  from typing import Callable, Optional
4  from numpy.typing import NDArray
5  from numpy import float_
6  from concurrent.futures import ThreadPoolExecutor
7  from shared_helpers import create_url_to_page_information_dict,
     full_matrix
8  from abstracts import Multithreadable
9  import threading
10
11 class PWithCacheHelper(Multithreadable):
12   def create_and_dump_p(
13     self,
14     p_cache: PCache,
15     d: float,
16     page_informations: list[PageInformation],
17     get_page_linkings: Callable[[int, Optional[bool]], list[
         PageLinking]]
18   ):
19     """
20     create matrix P and dump it into cache by modifying p_cache
```

```
21
22      Args:
23        - p_cache: object to write P matrix into cache
24        - d: damping_factor, usually 0.85
25        - page_informations: N-length list containing all page
              informations in database
26        - get_page_linkings: function or method to get page linkings
              of page information id
27      """
28
29      print(f"Create and dump P matrix start")
30
31      pages_count = len(page_informations) # N in paper
32      url_to_page_information = create_url_to_page_information_dict(
            page_informations)
33
34      if(self._is_multithread):
35        with ThreadPoolExecutor(self._max_workers) as executor:
36          executor.map(
37            self.__create_and_dump_p_column_values,
38            [pages_count for _ in range(pages_count)],
39            [page_information for page_information in
                page_informations],
40            [url_to_page_information for _ in range(pages_count)],
41            [p_cache for _ in range(pages_count)],
42            [get_page_linkings for _ in range(pages_count)],
43            [d for _ in range(pages_count)],
44          )
45      else:
46        for i in range(pages_count):
47          self.__create_and_dump_p_column_values(
48            pages_count, page_informations[i],
49            url_to_page_information, p_cache,
50            get_page_linkings, d)
51
52      print(f"Create and dump P matrix done")
53
54    def __create_and_dump_p_column_values(
55      self,
56      pages_count: int,
57      page_information: PageInformation,
58      url_to_page_information: dict[str, PageInformation],
59      p_cache: PCache,
60      get_page_linkings: Callable[[int, Optional[bool]], list[
            PageLinking]],
61      d: float
62    ):
63      try:
64        p_column = self.__create_p_column_values(pages_count,
              page_information, url_to_page_information,
              get_page_linkings, d)
65        p_cache.dump_column(page_information.index, p_column)
```

```
66          print(f"create_and_dump_p_column_values for page_index: {
                page_information.index} done; thread_id={threading.
                get_native_id()}")
67        except Exception as e:
68          print(f"create_and_dump_p_column_values: {e}")
69
70      def __create_p_column_values(
71        self,
72        pages_count: int,
73        page_information: PageInformation,
74        url_to_page_information: dict[str, PageInformation],
75        get_page_linkings: Callable[[int, Optional[bool]], list[
              PageLinking]],
76        d: float
77      ) -> NDArray[float_]:
78
79        try: # need to wrap it with try block because, in Threadpool,
              error will not be displayed
80          page_linkings = get_page_linkings(page_information.id_page,
                False)
81          cj = len(page_linkings)
82          output = full_matrix(pages_count, (1-d)/pages_count if(cj >
                0) else 1/pages_count)
83
84          for page_linking in page_linkings:
85            target_page_information: PageInformation
86
87            try:
88              target_page_information = url_to_page_information[
                    page_linking.outgoing_link]
89            except KeyError as e:
90              print(page_linking.outgoing_link)
91
92              raise e
93
94            target_index = target_page_information.index
95            output[target_index] = d/cj + (1-d)/pages_count
96
97          return output
98
99        except Exception as e:
100         print(f"create_p_column_values: {e}")
101         raise e
```

Kode 12: PartitionedPHelper

```
1  from model import PageInformation
2  from cache import PCache, QListCache, PiiListCache
3  from numpy import float_
4  from numpy.typing import NDArray
5  from shared_helpers import full_matrix
6  import threading
```

```
7
8  class PartitionedPHelper:
9    def dump_partitioned_p(
10     self, clusters: list[list[PageInformation]], p_cache: PCache,
11     q_list_cache: QListCache, pii_list_cache: PiiListCache
12   ):
13     """
14     partitioning N x N P matrix into Ni x Ni Pii matrix and
           normalized Pii matrix as Ni x Ni Qi matrix. Dump it into
           cache using q_list_cache and pii_list_cache
15
16     Args:
17       - clusters: page informations that are grouped by their
             domain forming n x Ni nested list. First level contain
             list of page informations that have common domain, while
             second level contain the page information itself
18       - p_cache: object to access matrix P in cache
19       - q_list_cache: object to write Qi matrix into cache
20       - pii_list_cache: object to write Pii matrix into cache
21     """
22
23     print(f"dump_partitioned_p start")
24
25     for cluster_no in range(len(clusters)):
26       self.__create_and_dump_pii_and_q(q_list_cache, pii_list_cache
             , cluster_no, clusters[cluster_no], p_cache)
27
28     print(f"dump_partitioned_p done")
29
30   def __create_and_dump_pii_and_q(
31     self, q_list_cache: QListCache, pii_list_cache: PiiListCache,
32     cluster_no: int, cluster: list[PageInformation], p_cache:
         PCache
33   ):
34     try:
35       pii = full_matrix((len(cluster), len(cluster)), 0)
36
37       for page_no in range(len(cluster)):
38         page_information = cluster[page_no]
39
40         self.__fill_pii_column(pii[:, page_no], p_cache.load_column
             (page_information.index), cluster)
41
42       pii_list_cache.dump_pii(cluster_no, pii)
43
44       # convert pii to q by normalize its columns
45       for i in range(pii.shape[1]):
46         pii[:, i] = pii[:, i]/sum(pii[:, i])
47
48       q_list_cache.dump_q(cluster_no, pii)
49       print(f"dump_partitioned_p cluster-{cluster_no} done;
           thread_id={threading.get_native_id()}")
```

```
50      except Exception as e:
51        print(f"dump_partitioned_p: {e}")
52
53    def __fill_pii_column(self, pii_column: NDArray[float_], p_column
          : NDArray[float_], cluster: list[PageInformation]):
54      try:
55        for q_index, page_information in enumerate(cluster):
56          pii_column[q_index] = p_column[page_information.index]
57
58      except Exception as e:
59        print(f"fill_pii_column: {e}")
```

Kode 13: PastiHelper

```
1  from cache import PastiListCache, PCache
2  from model import PageInformation
3  from shared_helpers import full_matrix
4
5  class PastiHelper:
6    def construct_and_dump_pasti_list(self, pasti_list_cache:
         PastiListCache, p_cache: PCache, clusters: list[list[
         PageInformation]], pages_count: int):
7      """
8      construct N x Ni P*i matrix and dump it into cache using
           pasti_list_cache
9
10     Args:
11       - pasti_list_cache: object to write P*i matrix into cache
12       - p_cache: object to load matrix P from cache
13       - clusters: page informations that are grouped by their
             domain forming n x Ni nested list. First level contain
             list of page informations that have common domain, while
             second level contain the page information itself
14       - pages_count: all page informations count in the database /
             N
15     """
16
17     print("construct_and_dump_pasti_list start")
18
19     for cluster_no in range(len(clusters)):
20       self.__construct_and_dump_pasti(pasti_list_cache, p_cache,
             clusters[cluster_no], cluster_no, pages_count)
21
22     print("construct_and_dump_pasti_list end")
23
24   def __construct_and_dump_pasti(self, pasti_list_cache:
         PastiListCache, p_cache: PCache, cluster: list[PageInformation
         ], cluster_no: int, pages_count: int):
25     try:
26       pasti = full_matrix((pages_count, len(cluster)), 0)
27       for index, page in enumerate(cluster):
28         pasti[:, index] = p_cache.load_column(page.index)
```

```
29
30          pasti_list_cache.dump_pasti(cluster_no, pasti)
31          print(f"construct_and_dump_pasti done for cluster_no = {
                cluster_no}")
32        except Exception as e:
33          print(f"construct_and_dump_pasti {e}")
```

Kode 14: PiastHelper

```
1  from cache import PiastListCache, PCache
2  from model import PageInformation
3  from shared_helpers import full_matrix
4
5  class PiastHelper:
6    def construct_and_dump_piast_list(self, piast_list_cache:
         PiastListCache, p_cache: PCache, clusters: list[list[
         PageInformation]], pages_count: int):
7      """
8      construct Ni x N Pi* matrix and dump it into cache using
           piast_list_cache
9
10     Args:
11       - piast_list_cache: object to write Pi* matrix into cache
12       - p_cache: object to load matrix P from cache
13       - clusters: page informations that are grouped by their
             domain forming n x Ni nested list. First level contain
             list of page informations that have common domain, while
             second level contain the page information itself
14       - pages_count: all page informations count in the database /
             N
15     """
16
17     print("construct_and_dump_piast_list start")
18
19     for cluster_no in range(len(clusters)):
20       self.__construct_and_dump_piast(piast_list_cache, p_cache,
             clusters[cluster_no], cluster_no, pages_count)
21
22     print("construct_and_dump_piast_list end")
23
24   def __construct_and_dump_piast(self, piast_list_cache:
         PiastListCache, p_cache: PCache, cluster: list[PageInformation
         ], cluster_no: int, pages_count: int):
25     try:
26       piast = full_matrix((len(cluster), pages_count), 0)
27       min_index = cluster[0].index
28       max_index = cluster[-1].index + 1
29
30       for i in range(pages_count):
31         piast[:, i] = p_cache.load_column(i)[min_index:max_index]
32
33       piast_list_cache.dump_piast(cluster_no, piast)
```

```
34        print(f"construct_and_dump_piast done for cluster_no = {
              cluster_no}")
35
36      except Exception as e:
37        print(f"construct_and_dump_piast {e}")
```

Kode 15: PiastHelper

```
1  from cache import RPCache, PCache
2  from numpy import float_
3  from numpy.typing import NDArray
4  from shared_helpers import full_matrix
5
6  class RPHelper:
7    def dump_and_mult_rp(
8      self, rp_cache: RPCache, r: NDArray[float_],
9      p_cache: PCache, pages_count: int
10   ):
11     """
12     multiply matrix R and matrix P and store the product matrix RP
           into rp_cache
13
14     Args:
15       - rp_cache: object to write RP matrix
16       - r: Ni x N matrix R
17       - p_cache: object to load matrix P from cache
18       - pages_count: all page informations count in the database /
             N
19     """
20
21     rp = full_matrix((r.shape[0], pages_count), 0)
22
23     for col_n in range(pages_count) :
24       self.__fill_rp_col(rp[:, col_n], p_cache.load_column(col_n),
             r, col_n)
25
26     rp_cache.dump_rp(rp)
27
28   def __fill_rp_col(
29     self, rp_col: NDArray[float_], p_col: NDArray[float_],
30     r: NDArray[float_], rp_col_no: int
31   ):
32     try:
33       for i in range(len(rp_col)):
34         rp_col[i] = r[i] @ p_col
35
36       print(f"dump_and_mult_rp.fill_rp_col col-{rp_col_no} done")
37
38     except Exception as e:
39       print(f"fill_rp_col: {e}")
```

Kode 16: DPCExecutor

```
from cache import PCache, QListCache, RPCache, PiiListCache,
    PastiListCache, PiastListCache
from data.domain_repository import DomainRepository
from model import PageInformation
from shared_helpers import l1_norm, full_matrix,
    show_and_reset_smallest_and_highest_memory_usage
from numpy.typing import NDArray
from numpy import float_, copy
from methods.dpc.rp_helper import RPHelper
from methods.dpc.partitioned_p_helper import PartitionedPHelper
from methods.dpc.cluster_separated_phi_helper import
    ClusterSeparatedPhiHelper
from methods.dpc.pasti_helper import PastiHelper
from methods.dpc.piast_helper import PiastHelper
from methods.dpc.extended_local_transition_matrix_helper import
    ExtendedLocalTransitionMatrixHelper
from typing import cast
from shared_helpers.cluster_helper import ClusterHelper

class DPCExecutor:
  """
  main class for DPC algorithm program
  """

  __p_cache: PCache
  __q_list_cache: QListCache
  __rp_cache: RPCache
  __pii_list_cache: PiiListCache
  __pasti_list_cache: PastiListCache
  __piast_list_cache: PiastListCache

  __rp_helper: RPHelper
  __partitioned_p_helper: PartitionedPHelper
  __cluster_separated_phi_helper: ClusterSeparatedPhiHelper
  __pasti_helper: PastiHelper
  __piast_helper: PiastHelper
  __extended_local_transition_matrix_helper:
      ExtendedLocalTransitionMatrixHelper
  __cluster_helper: ClusterHelper

  __domain_repository: DomainRepository

  def insert_caches(
    self,
    p_cache: PCache,
    q_list_cache: QListCache,
    rp_cache: RPCache,
    pii_list_cache: PiiListCache,
    pasti_list_cache: PastiListCache,
    piast_list_cache: PiastListCache
```

```
46    ):
47        """
48        insert cache classes that will be used later
49        """
50
51        self.__p_cache = p_cache
52        self.__q_list_cache = q_list_cache
53        self.__rp_cache = rp_cache
54        self.__pii_list_cache = pii_list_cache
55        self.__pasti_list_cache = pasti_list_cache
56        self.__piast_list_cache = piast_list_cache
57
58    def insert_helpers(
59        self,
60        rp_helper: RPHelper,
61        partitioned_p_helper: PartitionedPHelper,
62        cluster_separated_phi_helper: ClusterSeparatedPhiHelper,
63        pasti_helper: PastiHelper,
64        piast_helper: PiastHelper,
65        extended_local_transition_matrix_helper:
              ExtendedLocalTransitionMatrixHelper,
66        cluster_helper: ClusterHelper,
67    ):
68        """
69        insert helper classes that will be used later
70        """
71
72        self.__rp_helper = rp_helper
73        self.__partitioned_p_helper = partitioned_p_helper
74        self.__cluster_separated_phi_helper =
              cluster_separated_phi_helper
75        self.__pasti_helper = pasti_helper
76        self.__piast_helper = piast_helper
77        self.__extended_local_transition_matrix_helper =
              extended_local_transition_matrix_helper
78        self.__cluster_helper = cluster_helper
79
80
81    def insert_repositories(self, domain_repository: DomainRepository
          ):
82        """
83        insert repository classes that will be used later
84        """
85
86        self.__domain_repository = domain_repository
87
88    def execute(
89        self,
90        clusters: dict[str, list[PageInformation]],
91        epsilon: float,
92        pagerank_dpc_max_iteration: int,
93        start_time: float
```

```
94    )-> NDArray[float_]:
95      """
96      entry point for DPC algorithm
97
98      Args:
99        - clusters: page informations that are grouped by their
              domain forming python dictionary.
100         - key: domain url
101         - value: list of page information that belong to the domain
102       - epsilon: maximum l1 norm difference/delta between pagerank
              vector current iteration
103       - pagerank_dpc_max_iteration: max iteration for pagerank
              iterations
104       - start_time: start time when this method is called in
              seconds
105
106     Returns:
107       - N x 1 pagerank vector
108     """
109
110     page_information_in_clusters = list(clusters.values())
111     r = self.__cluster_helper.construct_r(
            page_information_in_clusters)
112     pages_count = self.__cluster_helper.get_all_page_counts(
            page_information_in_clusters)
113
114     self.__partitioned_p_helper.dump_partitioned_p(
115       page_information_in_clusters,
116       self.__p_cache,
117       self.__q_list_cache,
118       self.__pii_list_cache
119     )
120
121     show_and_reset_smallest_and_highest_memory_usage("
            dump_partitioned_p step", start_time)
122
123     cluster_separated_phi = self.__cluster_separated_phi_helper.
            construct_cluster_separated_phi(
124       page_information_in_clusters,
125       self.__q_list_cache,
126       pagerank_dpc, epsilon,
127       pagerank_dpc_max_iteration
128     )
129
130     show_and_reset_smallest_and_highest_memory_usage("
            construct_cluster_separated_phi step", start_time)
131
132     self.__rp_helper.dump_and_mult_rp(
133       self.__rp_cache, r,
134       self.__p_cache, pages_count
135     )
136
```

```
137     show_and_reset_smallest_and_highest_memory_usage("
            dump_and_mult_rp step", start_time)
138
139     self.__pasti_helper.construct_and_dump_pasti_list(
140       self.__pasti_list_cache, self.__p_cache,
141       page_information_in_clusters, pages_count
142     )
143
144     show_and_reset_smallest_and_highest_memory_usage("
            construct_and_dump_pasti_list step", start_time)
145
146     self.__piast_helper.construct_and_dump_piast_list(
147       self.__piast_list_cache,
148       self.__p_cache, page_information_in_clusters,
149       pages_count
150     )
151
152     show_and_reset_smallest_and_highest_memory_usage("
            construct_and_dump_piast_list step", start_time)
```

Kode 17: Fungsi pagerank pada program DPC

```
1     old_phi = copy(phi)
2
3     print("start iterating...")
4     iteration_count = 0
5     while True:
6       iteration_count += 1
7       s = self.__cluster_separated_phi_helper.construct_s(
            cluster_separated_phi)
8       a = self.__rp_cache.load_rp() @ s # A = RPS(phi^k)
9       z = pagerank_dpc(full_matrix(a.shape[0], 1/len(clusters)), a,
             epsilon, pagerank_dpc_max_iteration, f"pagerank_z-{
            iteration_count}")
10
11        b_list = self.__extended_local_transition_matrix_helper.
              construct_extended_local_transition_matrice(
12          self.__pii_list_cache,
13          self.__piast_list_cache,
14          self.__pasti_list_cache,
15          s, z, cluster_separated_phi,
16          pages_count
17        )
18
19        extended_local_pagerank_list = self.
              __construct_extended_local_pagerank_list(b_list, epsilon,
              iteration_count, pagerank_dpc_max_iteration)
20        cluster_separated_phi = self.__cluster_separated_phi_helper.
              get_new_cluster_separated_phi(z,
              extended_local_pagerank_list)
21        phi = self.__cluster_separated_phi_helper.
              flatten_cluster_separated_phi(cluster_separated_phi)
```

```
22          phi = cast(NDArray[float_], phi/l1_norm(phi))
23
24          delta = l1_norm(phi - old_phi)
25          print(f"delta = {delta}")
26          if(delta < epsilon or iteration_count >=
                pagerank_dpc_max_iteration):
27            self.__update_domain_ranks(z, clusters)
28            break
29
30          old_phi = copy(phi)
31
32      return phi
33
34    def __update_domain_ranks(self, z: NDArray[float_], clusters:
          dict[str, list[PageInformation]]):
35      self.__domain_repository.update_dpc_rank(list(clusters.keys()),
            [float(domain_rank) for domain_rank in z])
36
37    def __construct_extended_local_pagerank_list(
38      self, b_list: list[NDArray[float_]], epsilon: float, no: int,
            pagerank_dpc_max_iteration: int
39    ) -> list[NDArray[float_]]:
40      clusters_count = len(b_list)
41
42      return [pagerank_dpc(
43        self.__create_initial_phi(b_list[cluster_no].shape[0]),
44        b_list[cluster_no], epsilon, pagerank_dpc_max_iteration,
45        f"construct_extended_local_pagerank_list-{no}"
46      ) for cluster_no in range(clusters_count)]
47
48    def __create_initial_phi(self, ni_plus_1) -> NDArray[float_]:
49      return full_matrix(ni_plus_1, 1/ni_plus_1)
50
51
52  def pagerank_dpc(
53    phi: NDArray[float_],
54    p: NDArray[float_],
55    epsilon: float,
56    max_iteration: int,
57    log_name: str
58  ) -> NDArray[float_]:
59    """
60    pagerank computation for DPC algorithm
61
62    Args:
63      epsilon: maximum l1 norm difference/delta between pagerank
            vector current iteration
64      p: transition N x N matrix
65      phi: initial pagerank N x 1 vector
66      max_iteration: iteration maximum
67      log_name: for logging purpose name that will displayed in log
            for each iteration
```

```
68
69      Returns:
70        phi: final pagerank N x 1 vector
71      """
72
73      new_phi = full_matrix(phi.shape, 0)
74
75      iteration_count = 0
76      while True:
77        iteration_count += 1
78        new_phi: NDArray[float_] = p @ phi
79        new_phi = cast(NDArray[float_], new_phi / l1_norm(new_phi))
80
81        if(l1_norm(new_phi - phi) < epsilon or iteration_count >=
              max_iteration):
82          print(f"{log_name} {iteration_count} iteration count")
83          return new_phi
84
85        phi = new_phi
```

Kode 18: *run_dpc.py*

```
1  from factories import close_db, create_cluster_separated_phi_helper
       , create_domain_repository, create_page_information_repository,
       create_page_informations_clusterizer,
       create_page_linking_repository, create_page_rank_repository
2  from shared_helpers import
       show_and_reset_smallest_and_highest_memory_usage,
       sort_page_information_by_domain
3  from cache import PCache, QListCache, RPCache, PiiListCache,
       PastiListCache, PiastListCache
4  from consts import DPC_P_MATRIX_PATH, DAMPING_FACTOR,
       DB_MAX_WORKERS, EPSILON, DPC_Q_MATRIX_PATH, DPC_RP_MATRIX_PATH,
       DPC_PII_MATRIX_PATH, DPC_PAST_I_MATRIX_PATH,
       DPC_PI_AST_MATRIX_PATH, IS_MULTITHREAD,
       PAGERANK_DPC_MAX_ITERATION
5  from time import time
6  from methods.dpc.p_with_cache_helper import PWithCacheHelper
7  from methods.dpc import DPCExecutor
8  from methods.dpc.rp_helper import RPHelper
9  from methods.dpc.partitioned_p_helper import PartitionedPHelper
10 from methods.dpc.pasti_helper import PastiHelper
11 from methods.dpc.piast_helper import PiastHelper
12 from methods.dpc.extended_local_transition_matrix_helper import
       ExtendedLocalTransitionMatrixHelper
13 import tracemalloc
14
15 from shared_helpers.cluster_helper import ClusterHelper
16
17 def main():
18   start_time = time()
19   tracemalloc.start()
```

```
20
21    show_and_reset_smallest_and_highest_memory_usage("start",
          start_time)
22
23    page_information_repository = create_page_information_repository
          ()
24    page_informations = page_information_repository.
          get_all_page_informations()
25    clusterizer = create_page_informations_clusterizer()
26    clusters = clusterizer.clusterize(page_informations)
27    page_informations = sort_page_information_by_domain(clusters)
28
29    p_cache = PCache(DPC_P_MATRIX_PATH)
30    q_list_cache = QListCache(DPC_Q_MATRIX_PATH)
31    rp_cache = RPCache(DPC_RP_MATRIX_PATH)
32    pii_list_cache = PiiListCache(DPC_PII_MATRIX_PATH)
33    pasti_list_cache = PastiListCache(DPC_PAST_I_MATRIX_PATH)
34    piast_list_cache = PiastListCache(DPC_PI_AST_MATRIX_PATH)
35
36    show_and_reset_smallest_and_highest_memory_usage("initialize step
          ", start_time)
37
38    p_helper = PWithCacheHelper(IS_MULTITHREAD, DB_MAX_WORKERS)
39    page_linking_repository = create_page_linking_repository()
40    p_helper.create_and_dump_p(p_cache, DAMPING_FACTOR,
          page_informations, page_linking_repository.
          get_page_linkings_by_page_id)
41
42    show_and_reset_smallest_and_highest_memory_usage("
          create_and_dump_p step", start_time)
43
44    rp_helper = RPHelper()
45    partitioned_p_helper = PartitionedPHelper()
46    cluster_separated_phi_helper =
          create_cluster_separated_phi_helper()
47    pasti_helper = PastiHelper()
48    piast_helper = PiastHelper()
49    extended_local_transition_matrix_helper =
          ExtendedLocalTransitionMatrixHelper()
50    cluster_helper = ClusterHelper()
51
52    domain_repository = create_domain_repository()
53
54    dpc_executor = DPCExecutor()
55    dpc_executor.insert_caches(
56      p_cache, q_list_cache, rp_cache, pii_list_cache,
57      pasti_list_cache, piast_list_cache
58    )
59    dpc_executor.insert_helpers(
60      rp_helper, partitioned_p_helper, cluster_separated_phi_helper,
61      pasti_helper, piast_helper,
            extended_local_transition_matrix_helper,
```

```
62      cluster_helper
63    )
64    dpc_executor.insert_repositories(domain_repository)
65
66    print("DPC Start")
67
68    phi = dpc_executor.execute(clusters, EPSILON,
          PAGERANK_DPC_MAX_ITERATION, start_time)
69
70    show_and_reset_smallest_and_highest_memory_usage("dpc execute
          step", start_time)
71
72    page_rank_repository = create_page_rank_repository("page_rank_dpc
          ")
73    page_rank_repository.insert_ranks(page_informations, phi)
74
75    show_and_reset_smallest_and_highest_memory_usage("dpc end",
          start_time)
76    tracemalloc.stop()
77
78    close_db()
79
80  if(__name__ == "__main__"):
81    main()
```

Kode 19: AHelper

```
1   from cache import PCache
2   from model import PageInformation
3   from shared_helpers import full_matrix
4   from numpy.typing import NDArray
5   from numpy import float_
6   from typing import cast
7
8   class AHelper:
9     """
10    Helper class to create A matrix for modified dpc v2 algorithm see
          the skripsi for the definition
11    """
12
13    def construct_a(self, p_cache: PCache, clusters: list[list[
          PageInformation]]) -> NDArray[float_]:
14      """
15      create A matrix by merging P matrix values in PCache base on
            clusters
16
17      Args:
18        - p_cache: Object that can load P matrix in cache
19        - clusters: page informations that are grouped by their
              domain forming n x Ni nested list. First level contain
              list of page informations that have common domain, while
              second level contain the page information itself
```

```
20
21      Returns:
22        n x n "A" matrix for modified DPC algorithm
23      """
24
25      domain_count = len(clusters)
26      a = full_matrix((domain_count, domain_count), 0)
27
28      for domain_source_no in range(domain_count):
29        for domain_target_no in range(domain_count):
30          a[domain_target_no, domain_source_no] = self.
                __sum_sub_transition_matrix(
31            p_cache, clusters[domain_source_no], clusters[
                  domain_target_no]
32          )
33
34        print(f"construct_a for domain {domain_source_no} done")
35
36      return self.__normalize_a_column(a)
37
38    def __sum_sub_transition_matrix(
39      self,
40      p_cache: PCache,
41      cluster_source: list[PageInformation],
42      cluster_target: list[PageInformation]
43    ) -> float_:
44      total = 0
45      source_indices = self.__get_index_from_page_informations(
          cluster_source)
46      target_indices = self.__get_index_from_page_informations(
          cluster_target)
47
48      for source_index in source_indices:
49        p_column = p_cache.load_column(source_index)
50
51        for target_index in target_indices:
52          total += p_column[target_index]
53
54      return cast(float_, total)
55
56    def __get_index_from_page_informations(self, page_informations:
        list[PageInformation]) -> list[int]:
57      return [page_information.index for page_information in
          page_informations]
58
59    def __normalize_a_column(self, a: NDArray[float_]) -> NDArray[
        float_]:
60      column_count = a.shape[1]
61
62      for column_no in range(column_count):
63        a[:, column_no] = a[:, column_no] / sum(a[:, column_no])
64
```

```
65      return a
```

Kode 20: ModifiedDPCV2Executor

```
1   """
2   package for modified dpc v2 algorithm program. Origin of "v2" of
        the algorithm name due to existence of first modified dpc
        algorithm, but scrapped due to certain reasons.
3   """
4
5   from cache import PCache, QListCache
6   from data.domain_repository import DomainRepository
7   from methods.dpc.cluster_separated_phi_helper import
        ClusterSeparatedPhiHelper
8   from methods.modified_dpc_v2.q_helper import QHelper
9   from methods.modified_dpc_v2.a_helper import AHelper
10  from model import PageInformation
11  from shared_helpers import full_matrix,
        show_and_reset_smallest_and_highest_memory_usage
12  from numpy.typing import NDArray
13  from numpy import float_
14  from methods.dpc import pagerank_dpc
15
16  class ModifiedDPCV2Executor:
17    """
18    Class that contain core logic of modified DPC v2 algorithm (or
          refered as modified dpc in this research report/"skripsi")
19    """
20
21    __p_cache: PCache
22    __q_list_cache: QListCache
23
24    __q_helper: QHelper
25    __cluster_separated_phi_helper: ClusterSeparatedPhiHelper
26    __a_helper: AHelper
27
28    __domain_repository: DomainRepository
29
30    def insert_caches(
31      self,
32      p_cache: PCache,
33      q_list_cache: QListCache
34    ):
35      """
36      Insert cache classes that will be used later
37      """
38
39      self.__p_cache = p_cache
40      self.__q_list_cache = q_list_cache
41
42    def insert_helpers(
43      self,
```

```
44      q_helper: QHelper,
45      cluster_separated_phi_helper: ClusterSeparatedPhiHelper,
46      a_helper: AHelper,
47    ):
48      """
49      Insert helper classes that will be used later
50      """
51
52      self.__q_helper = q_helper
53      self.__cluster_separated_phi_helper =
            cluster_separated_phi_helper
54      self.__a_helper = a_helper
55
56    def insert_repositories(self, domain_repository: DomainRepository
          ):
57      """
58      Insert repository classes that will be used later
59      """
60
61      self.__domain_repository = domain_repository
62
63    def execute(
64      self,
65      clusters: dict[str, list[PageInformation]],
66      epsilon: float,
67      pagerank_dpc_max_iteration: int,
68      start_time: float
69    )-> NDArray[float_]:
70      """
71      Entry point to start mdofied dpc v2 program
72
73      Args:
74        - clusters: page informations that are grouped by their
              domain forming python dictionary.
75          - key: domain url
76          - value: list of page information that belong to the domain
77        - epsilon: maximum l1 norm difference/delta between pagerank
              vector current iteration
78        - pagerank_dpc_max_iteration: max iteration for pagerank
              iterations
79        - start_time: start time when this method is called in
              seconds
80
81      Returns:
82        - N x 1 pagerank vector
83      """
84
85      cluster_page_informations = list(clusters.values())
86
87      self.__q_helper.dump_q_list(
88        cluster_page_informations,
89        self.__p_cache,
```

```
90          self.__q_list_cache,
91      )
92
93      show_and_reset_smallest_and_highest_memory_usage("create and
            dump matrix Qs step", start_time)
94
95      cluster_separated_phi = self.__cluster_separated_phi_helper.
            construct_cluster_separated_phi(
96          cluster_page_informations,
97          self.__q_list_cache,
98          pagerank_dpc,
99          epsilon,
100         pagerank_dpc_max_iteration
101     )
102
103     show_and_reset_smallest_and_highest_memory_usage("
            cluster_separated_phi step", start_time)
104
105     a = self.__a_helper.construct_a(self.__p_cache,
            cluster_page_informations)
106
107     show_and_reset_smallest_and_highest_memory_usage("matrix a step
            ", start_time)
108
109     z = pagerank_dpc(full_matrix(a.shape[0], 1/len(clusters)), a,
            epsilon, pagerank_dpc_max_iteration, "pagerank_z")
110
111     show_and_reset_smallest_and_highest_memory_usage("vector z step
            ", start_time)
112
113     self.__insert_domain_ranks(clusters, z)
114     self.__multiply_phi_value_with_cluster_ranking(
            cluster_separated_phi, z)
115
116     return self.__cluster_separated_phi_helper.
            flatten_cluster_separated_phi(cluster_separated_phi)
117
118 def __multiply_phi_value_with_cluster_ranking(self,
        cluster_separated_phi: list[NDArray[float_]], z: NDArray[
        float_]):
119     for cluster_no, phi_per_cluster in enumerate(
            cluster_separated_phi):
120         for local_index in range(phi_per_cluster.shape[0]):
121             phi_per_cluster[local_index] *= z[cluster_no]
122
123 def __insert_domain_ranks(self, clusters: dict[str, list[
        PageInformation]], z: NDArray[float_]):
124     domain_ranks: list[float] = []
125     cluster_urls = list(clusters.keys())
126
127     for i in range(len(cluster_urls)):
128         domain_ranks.append(float(z[i]))
```

```
129
130         self.__domain_repository.update_mdpcv2_rank(cluster_urls,
                domain_ranks)
```

Kode 21: run_modified_dpc_v2.py

```
1   from time import time
2   import tracemalloc
3   from cache import PCache, QListCache
4   from consts import DB_MAX_WORKERS, DAMPING_FACTOR, EPSILON,
        IS_MULTITHREAD, MODIFIED_DPC_V2_P_MATRIX_PATH,
        MODIFIED_DPC_V2_Q_MATRIX_PATH, PAGERANK_DPC_MAX_ITERATION
5   from factories import close_db, create_cluster_separated_phi_helper
        , create_domain_repository, create_page_information_repository,
        create_page_linking_repository, create_page_rank_repository
6   from methods.dpc.p_with_cache_helper import PWithCacheHelper
7   from methods.modified_dpc_v2.q_helper import QHelper
8   from methods.modified_dpc_v2 import ModifiedDPCV2Executor
9   from methods.modified_dpc_v2.a_helper import AHelper
10  from shared_helpers import PageInformationsClusterizer,
        show_and_reset_smallest_and_highest_memory_usage,
        sort_page_information_by_domain
11
12  def main():
13      start_time = time()
14      tracemalloc.start()
15
16      show_and_reset_smallest_and_highest_memory_usage("start",
            start_time)
17
18      page_information_repository = create_page_information_repository
            ()
19      page_informations = page_information_repository.
            get_all_page_informations()
20      clusterizer = PageInformationsClusterizer()
21      clusters = clusterizer.clusterize(page_informations)
22      page_informations = sort_page_information_by_domain(clusters)
23
24      p_cache = PCache(MODIFIED_DPC_V2_P_MATRIX_PATH)
25      q_list_cache = QListCache(MODIFIED_DPC_V2_Q_MATRIX_PATH)
26      show_and_reset_smallest_and_highest_memory_usage("instantiate
            step", start_time)
27
28      p_helper = PWithCacheHelper(IS_MULTITHREAD, DB_MAX_WORKERS)
29      page_linking_repository = create_page_linking_repository()
30      p_helper.create_and_dump_p(p_cache, DAMPING_FACTOR,
            page_informations, page_linking_repository.
            get_page_linkings_by_page_id)
31      show_and_reset_smallest_and_highest_memory_usage("create and dump
             matrix P step", start_time)
32
33      q_helper = QHelper()
```

```
34    a_helper = AHelper()
35    cluster_separated_phi_helper =
          create_cluster_separated_phi_helper()
36
37    domain_repository = create_domain_repository()
38
39    print("Modified DPC V2 Start")
40
41    modified_dpc_executor = ModifiedDPCV2Executor()
42    modified_dpc_executor.insert_caches(p_cache, q_list_cache)
43    modified_dpc_executor.insert_helpers(q_helper,
          cluster_separated_phi_helper, a_helper)
44    modified_dpc_executor.insert_repositories(domain_repository)
45
46    show_and_reset_smallest_and_highest_memory_usage("prepare to
          execution step", start_time)
47
48    phi = modified_dpc_executor.execute(clusters, EPSILON,
          PAGERANK_DPC_MAX_ITERATION, start_time)
49
50    show_and_reset_smallest_and_highest_memory_usage("after execution
           step", start_time)
51
52    page_rank_repository = create_page_rank_repository("
          page_rank_modified_dpc_v2")
53    page_rank_repository.insert_ranks(page_informations, phi)
54
55    show_and_reset_smallest_and_highest_memory_usage("modified dpc v2
          end", start_time)
56    tracemalloc.stop()
57
58    close_db()
59
60  if(__name__ == "__main__"):
61    main()
```

Kode 22: *run_random_walker.py*

```
1
2   from time import time
3   import tracemalloc
4   from factories import close_db, create_random_walker_executor
5
6
7   def main():
8     tracemalloc.start()
9     start_time = time()
10
11    random_walker_executor = create_random_walker_executor()
12    random_walker_executor.execute()
13
14    traced_memory = tracemalloc.get_traced_memory()
```

```
15
16    print("random walker")
17    print(f"smallest memory usage: {traced_memory[0]} Bytes")
18    print(f"highest memory usage: {traced_memory[1]} Bytes")
19    print(f"{time() - start_time} seconds")
20
21    tracemalloc.stop()
22    close_db()
23
24  if(__name__ == "__main__"):
25    main()
```

Kode 23: RandomWalkerExecutor

```
1  from data.page_rank_random_walkers_repository import
       PagerankRandomWalkersRepository
2  from data.domain_repository import DomainRepository
3  from methods.random_walker.helper_factories.graph_factory import
       GraphFactory
4  from methods.random_walker.helper_factories.nodes_helper_factory
       import NodesHelperFactory
5  from methods.random_walker.helpers.page_informations_helper import
       PageInformationsHelper
6  from model import Node
7  from shared_helpers import PageInformationsClusterizer,
       show_and_reset_smallest_and_highest_memory_usage
8  from shared_helpers.p_helper import PHelper
9  import random
10
11 class RandomWalkerExecutor:
12   """
13   executor for random walker algorithm
14
15   Args:
16     - page_informations_helper: helper class to manage list of page
           informations
17     - nodes_helper_factory: factory class to construct nodes helper
18     - graph_factory: factory class to construct graph helper class
19     - random_walker_iterations: maximum iterations for random
           walker program
20     - initial_walkers_count: initial walkers count for each node
21     - page_informations_clusterizer: helper class to clusterize
           page informations
22     - page_rank_walkers_repository: class to store random walkers
           output into database
23     - domain_repository: class to store random walkers output for
           each page information domain
24   """
25
26   __page_informations_helper: PageInformationsHelper
27   __nodes_helper_factory: NodesHelperFactory
28   __graph_factory: GraphFactory
```

```
29    __p_helper: PHelper
30    __random_walker_iterations: int
31    __initial_walkers_count: int
32    __page_informations_clusterizer: PageInformationsClusterizer
33    __page_rank_random_walkers_repository:
          PagerankRandomWalkersRepository | None
34    __domain_repository: DomainRepository | None
35
36    def __init__(
37      self,
38      page_informations_helper: PageInformationsHelper,
39      nodes_helper_factory: NodesHelperFactory,
40      graph_factory: GraphFactory,
41      p_helper: PHelper,
42      random_walker_iterations: int,
43      initial_walkers_count: int,
44      page_informations_clusterizer: PageInformationsClusterizer,
45      page_rank_random_walkers_repository:
            PagerankRandomWalkersRepository | None = None,
46      domain_repository: DomainRepository | None = None
47    ) -> None:
48      self.__page_informations_helper = page_informations_helper
49      self.__nodes_helper_factory = nodes_helper_factory
50      self.__random_walker_iterations = random_walker_iterations
51      self.__initial_walkers_count = initial_walkers_count
52      self.__page_informations_clusterizer =
            page_informations_clusterizer
53      self.__p_helper = p_helper
54      self.__graph_factory = graph_factory
55      self.__page_rank_random_walkers_repository =
            page_rank_random_walkers_repository
56      self.__domain_repository = domain_repository
57
58    def execute(self):
59      """
60      main entry point for random walker algorithm
61      """
62      nodes = self.__page_informations_helper.create_nodes(self.
            __initial_walkers_count)
63      nodes_helper = self.__nodes_helper_factory.create_helper(nodes)
64      graph = self.__graph_factory.create_helper(self.__p_helper,
            nodes_helper, self.__page_informations_helper)
65
66      for i in range(self.__random_walker_iterations):
67        for current_node in nodes:
68          destination_nodes_and_probabilites = graph.
                get_destination_nodes_and_probabilites(current_node)
69          walkers_count = current_node.get_walkers_count()
70          choosen_nodes = random.choices(
                destination_nodes_and_probabilites[0], weights=
                destination_nodes_and_probabilites[1], k=walkers_count)
71          choosen_nodes_helper = self.__nodes_helper_factory.
```

```
            create_helper(choosen_nodes)
72          choosen_nodes_helper.add_next_walkers_count()
73          current_node.clear_walkers_count()
74          show_and_reset_smallest_and_highest_memory_usage(f"
                iteration-{i} for page index {current_node.
                get_page_information_index()} done")
75
76        nodes_helper.move_nodes_next_walkers_count_to_walkers_count()
77
78      self.__update_domain_walkers_count(nodes)
79      self.__insert_ranks(nodes)
80
81    def __update_domain_walkers_count(self, nodes: list[Node]):
82      if(self.__domain_repository is None):
83        return
84
85      domain_dict: dict[str, int] = {}
86
87      for node in nodes:
88        domain_url = self.__page_informations_clusterizer.get_domain(
              node.get_page_information())
89        if(domain_url in domain_dict.keys()):
90          domain_dict[domain_url] += node.get_walkers_count()
91        else:
92          domain_dict[domain_url] = node.get_walkers_count()
93
94      domain_urls = list(domain_dict.keys())
95      walkers_counts = [domain_dict[domain_url] for domain_url in
            domain_urls]
96      self.__domain_repository.update_domain_walkers_count(
            domain_urls, walkers_counts)
97
98    def __insert_ranks(self, nodes: list[Node]):
99      if(self.__page_rank_random_walkers_repository is None):
100       return
101
102     self.__page_rank_random_walkers_repository.insert_ranks(nodes)
```

Kode 24: NodesHelper

```
1  from model import Node
2
3  class NodesHelper:
4    """
5    helper class managing list of nodes
6
7    Properties
8      - sorted: true if nodes already sorted base on its
           page_information.index
9      - nodes: N-length list of nodes
10   """
11
```

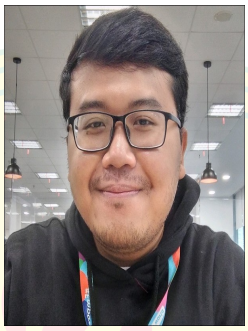
```
12    __sorted = False
13    __nodes: list[Node]
14
15    def insert_stateful_fields(self, nodes: list[Node]):
16      """
17      method to insert parameters that are stateful. Stateful means
            for every instances will have different mutable object
            params
18
19      Args:
20        - nodes: N-length list of nodes
21      """
22
23      self.__nodes = nodes
24
25    def get_nodes(self, sorted: bool = False) -> list[Node]:
26      """
27      get all nodes
28
29      Args:
30        - sorted: default False, set True will sort nodes by its
              page_information.index before return the nodes
31      """
32
33      if(sorted and not self.__sorted): # only sort nodes once
34        self.__sort_nodes_by_page_information_index()
35
36      return self.__nodes
37
38    def __sort_nodes_by_page_information_index(self):
39      self.__nodes.sort(key = lambda node: node.
            get_page_information_index())
40      self.__sorted = True
41
42    def move_nodes_next_walkers_count_to_walkers_count(self):
43      """
44      for each node set node.walkers_count = node.next_walkers_count
            then set node.next_walkers_count = 0
45      """
46
47      for node in self.__nodes:
48        node.move_next_walkers_count_to_walkers_count()
49
50    def add_next_walkers_count(self):
51      """
52      for each node increment node.walkers_count
53      """
54
55      for node in self.__nodes:
56        node.add_next_walkers_count()
```

Kode 25: PageInformationsHelper

```
1  from model import PageInformation, Node
2  from data.page_information_repository import
       PageInformationRepository
3  from shared_helpers import PageInformationsClusterizer,
       sort_page_information_by_domain
4
5
6  class PageInformationsHelper:
7    """
8    helper class managing all page informations in this program from
         page_information table in database
9
10   sort the list base on their domains, and set page_information.
         index base on the index in the list
11
12   Properties
13     - page_informations: N-length list of page informations
14   """
15
16   __page_informations: list[PageInformation]
17
18   def __init__(self, page_information_repository:
         PageInformationRepository, page_information_clusterizer:
         PageInformationsClusterizer) -> None:
19     page_informations = page_information_repository.
           get_all_page_informations()
20     clusters = page_information_clusterizer.clusterize(
           page_informations)
21     self.__page_informations = sort_page_information_by_domain(
           clusters)
22
23   def create_nodes(self, initial_walkers_count: int) -> list[Node]:
24     """
25     for each page information create a node with node.walkers_count
           = initial_walkers_count
26
27     Args:
28       - initial_walkers_count
29
30     Returns:
31       N-length list of nodes
32     """
33
34     return [Node(page_information, initial_walkers_count) for
           page_information in self.__page_informations]
35
36   def get_page_informations(self) -> list[PageInformation]:
37     """
38     get all page informations
39     """
```

```
40
41          return self.__page_informations
```

DAFTAR RIWAYAT HIDUP



Nama	: Farhan Herdian Pradana
Tempat, Tanggal Lahir	: Jakarta, 28 Januari 2000
Orang tua	: Dadang Budhi Herdiyana & Sri Rejeki
Email	: farhan.herdia123@gmail.com

Pendidikan

1. SD Negeri Utan Kayu Utara 01 Pagi (2006-2012)
2. SMP Negeri 7 Jakarta (2012-2015)
3. SMA Negeri 22 Jakarta, Matematika dan Ilmu Alam (2015-2018)
4. Universitas Negeri Jakarta, Ilmu Komputer (2018-)

Pengalaman Kerja

1. Front End Developer Intern, **Lingotalk** (Juli - November, 2021)
2. Android Engineer Intern, **Traveloka** (Februari - Juli 2022)
3. Associate Software Engineer, **Blibli** (September 2022 - Sekarang)