STUDI DAN IMPLEMENTASI *DYNAMIC PROGRAMMING FOR HEIGHT AND WEIGHT* TERHADAP DOKUMEN-DOKUMEN XML

M. Noversada – NIM : 13503029
Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13029@students.if.itb.ac.id

Abstrak

Makalah ini membahas studi dan implementasi *Dynamic Programming for Height and Weight* (DHW) untuk mempartisi dokumen-dokumen XML yang akan diproses. DHW merupakan sebuah algoritma *Optimal Tree Sibling Partitioning*. Algoritma ini adalah sebuah algoritma yang dikembangkan untuk mengganti algoritma konvensional yang terfokus pad a hubungan orang tua-anak saja.

Pada makalah ini juga dibahas beberapa algoritma yang lain sebagai pembanding untuk DHW dan juga beberapa adalah pendahulu dari DHW. Algoritma-algoritma tersebut antara lain FDW (*F*lat trees, *D*ynamic programming for tree *W*idth), GHDW (employing a *G*reedy strategy for the *H*eight of the tree, and *D*ynamic programming for the *W*idth), DFS (Depth First Search), BFS (Breadth First Search), Rightmost Siblings (RS), Kundu and Misra (KM), dan Enhanced Kundu and Misra (EKM).

Kemudian dilakukan uji coba terhadap algoritma-algoritma tersebut menggunakan file XML dengan berbagai variasi. Hasil uji menunjukkan bahwa DHW cukup memuaskan meskipun masih jauh dari sempurna. Selain itu juga membuktikan bahwa algoritma *Optimal Tree Sibling Partitioning* lebih baik dibandingkan dengan algoritma lama.

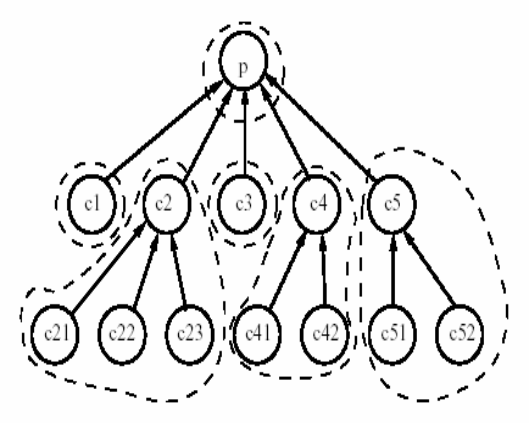
Kata kunci*: Dynamic Programming for Height and Weight, Optimal Tree Sibling Partitioning, Flat trees, Dynamic programming for tree Width, employing a Greedy strategy for the Height of the tree, and Dynamic programming for the Width, Depth First Search, Breadth First Search, Rightmost Siblings, Kundu and Misra, Enhanced Kundu and Misra* .

1. Pendahuluan

Secara umum, masalah yang ada yaitu pembagian pohon ditinjau dari sudut pandang *native XML Data Stores (XDS)*. Secara spesifik, masalah yang ada yaitu menyangkut kualitas representasi penyimpanan dokumen-dokumen XML pada sistem secara mendasar menyimpan representasi pohon dari dokumen XML tersebut yang telah diurutkan dan diberi label serta menggunakan petunjuk primitive untuk mengakses representasi tersebut pada saat pemrosesan *query*. Semua media penyimpanan dirancang untuk menyimpan pohon yang memerlukan lebih banyak tempat dibandingkan dengan sebuah media penyimpanan sekunder pasti memiliki sebuah algoritma pembagian pohon. Pembagian pohon merubah pohon dokumen lojik ke dalam partisi yang lebih kecil daripada limit yang ada, yang berhubungan langsung dengan kapasitas media penyimpanan. Algoritma pembagian pohon ini dapat berupa tambahan pada beberapa system yang secara sembarangan menempatkan simpul di mana saja selama ada tempat yang tersedia. Namun secara umum, merancang algoritma pembagian untuk XDS merupakan ide bagus karena:

1. jumlah dan struktur partisi adalah factor penting dalam pemrosesan *query* karena perpindahan antar media penyimpanan termasuk mahal.
2. performa dari algoritma pembagian itu menentukan performa system secara keseluruhan karena penambahan dokumen merupakan operasi yang cukup sering dilakukan.

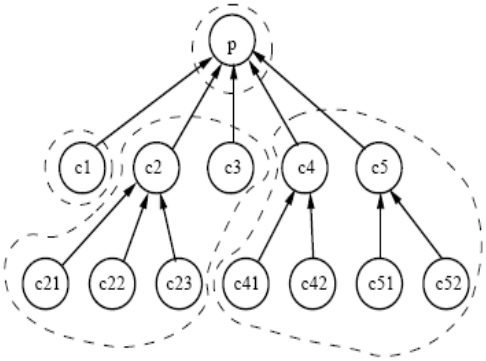
Fitur penting dari model data XML adalah order/urutan, dan hal ini harus benar-benar diperhatikan saat merancang algoritma pembagian. Media penyimpanan XDS tidak hanya harus menyimpan hubungan parent-child dari sebuah pohon, tetapi juga harus menyertakan hubungan kekerabatan antar simpul. Media penyimpanan untuk native XDS seperti IBM's System RX/DB2 Viper dan Natix system menyediakan penyimpanan pohon yang telah diberi order.



Gambar 1: Pembagian dengan memperhatikan hubungan parent-child saja

Kedua media tersebut bahkan telah menyediakan fasilitas penyimpanan yang telah dioptimalkan untuk kekerabatan yang berkelanjutan yang berbagi media penyimpanan., bahkan jika parent mereka terletak di media penyimpanan yang lain. Tanpa optimasi seperti ini, akses terhadap simpul-simpul dengan jumlah anak yang besar akan memiliki performa yang buruk. Sebagai contoh, lihat gambar 1. Asumsikan bahwa akar p tidak bisa disimpan pada sebuah media penyimpanan bersama dengan subpohon anak-anaknya. Jika format penyimpanan tidak memperbolehkan peletakan kerabat yang berkelanjutan pada sebuah media penyimpanan yang tidak mengandung orang tua mereka, hasil pembagian bisa dilihat seperti yang ditunjukkan oleh garis putus-putus pada gambar 1. Pada kasus ini, setiap anak akan disimpan terpisah dan setiap bagian berkorespondensi pada sebuah subpohon. Evaluasi *query* dengan bahasa XML *query* seperti XPath dan X*Query* sangat mahal. Dalam kasus sebuah in-order traversal dari semua anak atau keturunan dari *p*, evaluasi terhadap anak atau keturunan yang dimulai untuk konteks simpul p akan mengakses media yang berbeda untuk setiap anak *p*, kira-kira 5 media penyimpanan. Jika kerabat dapat berbagi media penyimpanan bahkan jika orang tuanya terletak pada media penyimpanan yang lain, maka akan didapat hasil seperti yang ditunjukkan oleh gambar 2.

Di sini, beberapa subpohon dapat berbagi partisi, selama akarnya memiliki hubungan kekerabatan. Tipe partisi seperti ini disebut juga pembagian kekerabatan. Hal ini menghasilkan lebih sedikit perlintasan antar media penyimpanan (pada makalah ini, sebagai contoh ada 3), yang pada gilirannya meningkatkan performa *query*.



Gambar 2: Pembagian dengan parent-child dan hubungan kekerabatan

Untuk menjaga agar jumlah perlintasan serendah mungkin, algoritma pembagian pohon untuk XDS harus menciptakan pembagian kekerabatan dan meminimalkan jumlah partisi total.

Tujuan utama mempelajari masalah pembagian pohon kekerabatan adalah adanya pengalaman dengan media penyimpanan pada native XML data store Natix. Natix menggunakan format penyimpanan dimana unit penyimpanan berupa physical record, dengan masing-masing mengandung bagian dari pohon dokumen yang simpul-simpulnya dihubungkan oleh garis orangtua-anak atau garis kekerabatan. Natix memiliki dua algoritma untuk menentukan simpul mana yang berbagi physical record. Algoritma node-at-a-time mempertahankan format penyimpanan penglompokan XML pada tambahan pembaruan. Penyisipan seluruh dokumen ditangani oleh komponen bulkload, yang rancangan dan implementasinya dideskripsikan pada. Algoritma pembagian standar untuk dokumen impor berupa heuristic yang sederhana.

Dalam pengujian, untuk beberapa kasus diperiksa keputusan pembagian yang aneh oleh algoritma yang sederhana ini yang berujung pada performa *query* yang tidak dapat diterima. Percobaan tambahan untuk memperbaiki heuristik ini tidak terlalu baik, yaitu selalu rentan terhadap kasus-kasus patologi baru (beberapa di antaranya akan diungkap pada makalah ini). Untuk dapat menilai kualitas dari berbagai algoritma dan untuk mendapatkan gambaran

bagaimana caranya untuk menghasilkan algoritma yang lebih baik, perlu diketahui batas optimal secara teori, yaitu membagi dengan jumlah paling minimal. Meskipun begitu, menentukan jumlah minimal pembagian untuk dokumen-dokumen tertentu bukanlah pekerjaan yang mudah, jumlah pembagian kerabat potensial berupa fungsi eksponensial terhadap jumlah simpul, jadi menggunakan algoritma *brute force* untuk menentukan jumlah optimal tidaklah layak.

Dalam beberapa dekade ini, jumlah algoritma untuk pembagian pohon telah berkembang. Beberapa diantaranya dirancang khusus untuk media penyimpanan saat ini. Algoritma pembagian pohon telah dipelajari dalam konteks *hierarchical DBMS*, *object-oriented DBMS*, dan baru-baru ini, XDS. Sayangnya tidak satupun dari algoritma-algoritma tersebut mempertimbangkan urutan kekerabatan atau mengizinkan subpohon yang berkerabat untuk berbagi partisi jika orang tua mereka berada di partisi yang lain.

2. Permasalahan

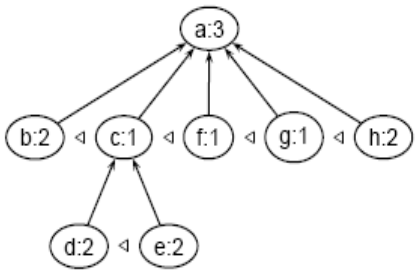
2.1. Istilah dan definisi

Asumsikan $T = (V, t, p, \blacktriangleleft, w)$ adalah sebuah pohon yang memiliki akar, telah diberi urutan**,** dan memiliki bobot dengan V adalah simpul-simpul pohon tersebut, t adalah akar, sebuah fungsi orang tua p, urutan kerabat yang menghantar ◄, dan sebuah fungsi bobot w. p memetakan setiap simpul bukan akar ke setiap orang tuanya dan akar pada NIL, dan w memetakan masing-masing simpul pada bobot yang nilainya integer positif. Berikutnya, istilah pohon selalu menunjukkan pada pohon berakar, memiliki urutan dan berbobot.

Gambar 3 menunjukkan contoh pohon $T = (\{a,b,c,d,e,f,g,h\}, a, p, \blacktriangleleft, w)$, yang akan digunakan untuk menunjukkan definisi di bawah. Pada gambar, simpul-simpul direpresentasikan sebagai oval dengan petunjuk, fungsi orang tua direpresentasikan dengan panah orang tua-anak, urutan kekerabatan direpresentasikan oleh simbol ◄ (dengan hubungan transitif seperti menghilangkan b◄g), dan bobot simpul w adalah angka pada oval.

Dengan $T = (V, t, p, \blacktriangleleft, w)$, menunjukkan subpohon yang *induced* oleh simpul $v \in V$ dengan T_v. Bobot subpohon $W_T(v)$ adalah jumlah dari seluruh bobot semua simpul pada T_v. Pada contoh, pohon T_c terdiri dari simpul c, d, dan e. Bobot subpohon c $W_T(c)$ adalah 5.

Interval kekerabatan $(l, r)_T$ dari T adalah set dari kerabat-kerabat yang berturutan yang ditentukan oleh kerabat pertama l dan kerabat terakhir r dengan l◄r, sehingga $(l, r)_T := \{x | x = r \vee x = l \vee l \blacktriangleleft x \blacktriangleleft r\}$. Pembagian pohon kekerabatan P dari T adalah set dari interval-interval kerabat yang tidak berturutan. Bobot subpohon dari interval kerabat ini adalah $W_T(l, r) := \Sigma_{x \in (l,r)_T} W_T(x)$. Bobot dari set S dari interval kerabat $W_T(S)$ adalah jumlah dari bobot dari interval-interval yang dikandungnya. Pada contoh, interval $(b,f)_T$ terdiri dari simpul-simpul b,c, dan f dan memiliki bobot



subpohon 8.

Gambar 3: contoh pohon

Diberikan pohon T dan pembagian pohon kekerabatan P seperti di atas, pembagian hutan F_{PT} dari T dengan memperhatikan P adalah set dari pohon-pohon yang dihasilkan dari T saat memotong garis orang tua dari simpul-simpul yang memiliki interval kekerabatan di P. Hal ini ekuivalen dengan memiliki fungsi orang tua p^P sehingga untuk semua $(l, r)T \in P$, $\forall v \in (l,r)T$ $pP(v) := NIL$. Sebab itu, dalam F_{PT} m setiap simpul yang terdapat dalam interval dalam P menjadi akar dari pohon. Pembagian didefinisikan sebagai interval (l, r) adalah set dari semua pohon dari F_{PT} dengan akar terletak pada $(l, r)_T$. Pada contoh, partisi didefinisikan sebagai (b,f)T is {Tb, Tc, Tf }.

Bobot partisi $W_{PT}(v)$ dari simpul v adalah bobot subpohon dalam F_{PT}. Analoginya, bobot partisi dari $W_{PT}(l, r)$ adalah jumlah dari semua bobot partisi dari simpul-simpulnya, dan bobot partisi dari set dari interval kekerabatan adalah jumlah dari bobot partisi dari setiap interval-intervalnya. Bobot akar dari sebuah pembagian adalah bobot partisi dari simpul akar, $W_{PT}(t)$. Pada contoh, anggap pembagian P := {(b,f)T }. Bobot akar dari P adalah 6, karena hanya simpul a, g, dan h tersisa pada akar a setelah garis orang tua dari b, c, dan f dihapus.

Diberikan T dan sebuah integer positif K, sebuah pembagian kekerabatan P dari T dianggap layak jika $(t, t)T \in P$ and $\forall_{(l,r)T \in P} W_{PT}(l, r) \leq K$.

Pembagian yang layak dari contoh pohon di atas dan K = 5 adalah P := {(a,a)$_T$, (b,b)$_T$, (c,c)$_T$, (f,g)$_T$ }. Di sini, h terletak di partisi yang sama dengan akar, dan bobot akar adalah 5.

Sebuah pembagian pohon kekerabatan dikatakan minimal jika *feasible* dan memiliki kemungkinan kardinalitas terkecil dari semua partisi yang layak. Sebuah pembagian pohon kekerabatan P dikatakan *lean* jika bobot akarnya adalah minimal di antara semua peartisi dengan kardinalitas yang sama. Sebuah pembagian pohon kekerabatan dikatakn optimal jika memenuhi persyaratan minimal dan lean. Pada contoh, R := {(a,a)$_T$, (c,c)$_T$, (f,h)$_T$ } adalah partisi minimal (K = 5) dengan kardinalitas 3. b terdapat pada partisi yang sama dengan akar, sehingga R memiliki bobot akar 5. Meskipun begitu, R tidak *lean*. Terdapat partisi dengan kardinalitas yang sama dan bobot akar yang lebih kecil: pada P := {(a,a)$_T$, (c,h)$_T$, (d,e)$_T$ }, bobot akar adalah 3. P optimal. Pada makalah ini, pembagian pohon kekerabatan optimal akan disimbolkan dengan huruf P atau D.

2.2. Masalah pembagian pohon kekerabatan

Diberikan istilah-istilah berikut, masalah yang ingin diselesaikan secara formal dinyatakan sbb:

Pembagian pohon kekerabatan: Diberikan pohon T dan batas bobot K, tentukan pembagian pohon kekerabatan minimal.

Untuk menyelesaikan masalah di atas, dikembangkan sebuah algoritma yang menemukan pembagian dengan properti yang lebih kuat, yang disebut optimalitas. Sesuai definisi, hal ini menunjukkan bahwa partisi tersebut harus memiliki bobot akar minimal di antara partisi minimal yang lain. Di bawah akan ditunjukkan bahwa alasan untuk hal ini terletak pada pendekatan rekursif, bottom-up: Saat minimalitas adalah hal yang kita butuhkan untuk solusi secara keseluruhan, submasalah yang kita selesaikan juga harus lean untuk menjamin minimalitas pada level yang lebih tinggi.

3. Pembagian pohon kekerabatan yang optimal

Jumlah dari pembagian pohon kekerabatan yang layak untuk pohon yang telah diberikan dengan simpul-simpul sebanyak n sangat banyak, bahkan jika batas bobot K telah ditentukan. Untuk semua simpul orang tua, harus ditentukan subset dari anak-anak yang harus diletakkan pada partisi yang sama dengan simpul orang tua. Untuk sisa anak-anak, harus ditentukan bagaimana cara mengkombinasikan kekerabatan dalam partisi. Namun tidak semudah itu untuk menemukan pembagian minimal dengan waktu yang proporsional terhadap n, walaupun batas bobot partisi K telah diberikan. Bahkan, dapat dilihat bahwa pada versi yang telah disederhanakan dari suatu masalah tidak selalu bisa diselesaikan dalam waktu yang linear.

Di sini dicari strategi tambahan. Pendekatan pembagian pohon kekerabatan diuji secara formal, yang menyediakan rangkaian properti yang memungkinkan untuk mengembangkan secara progresif algoritma yang lebih baik.

Di sini diawali dengan menunjukkan bahwa pendekatan bottom-up dapat berjalan karena dapat mengkombinasikan pembagian untuk subpohon untuk mendapatkan solusi global. Langkah kedua, dihadirkan algoritma pemrograman dinamik yang dapat membagi pohon yang flat (yaitu pohon yang semua simpul kecuali akarnya adalah daun) dalam waktu O(nK2).

Sayangnya, akan dilihat bahwa pada aplikasi bottom-up dari algoritma ini untuk pohon yang dalam/tinggi tidak selalu menghasilkan hasil yang optimal: Kadang-kadang harus dipilih solusi suboptimal pada level yang lebih rendah dari pohon untuk menghindari partisi ekstra pada level yang lebih tinggi nantinya. Namun, dapat ditunjukkan bahwa pada tiap langkah, hanya diperlukan pemilihan antara solusi yang optimal dan yang mendekati optimal, untuk menunjukkan definisi mudah dari "mendekati optimal". Juga ditunjukkan bagaimana menggabungkan pilihan ini ke dalam algoritma pemrograman dinamik yang telah dimiliki sebelumnya, yang berujung pada algoritma O(nK$_3$) untuk pembagian pohon kekerabatan yang optimal.

3.1. Pembagian pohon secara bottom-up

Algoritma yang ada berlandaskan pada asumsi bahwa untuk menentukan pembagian optimal secara global, dapat dipilih simpul v dari pohon dan menentukan pembagian untuk subpohon yang di-*induce* oleh simpul tersebut. Kemudian, secara rekursif dapat ditentukan pembagian global untuk sisa dari pohon tersebut dan mengkombinasikan dua solusi untuk

mendapatkan solusi global. Selanjutnya asumsi ini akan dicoba untuk diformalkan.
Ingat bahwa sebuah solusi dianggap optimal jika solusi tersebut minimal dan *lean*. Alasan untuk hal ini akan dijelaskan di bawah.
Pada *lemma* berikut, tidak diasumsikan bahwa pembagian subpohon lokal untuk subpohon T_v, yang disebut S, adalah solusi optimal lokal. Di sini hanya diasumsikan bahwa untuk suatu alasan tertentu bahwa S adalah bagian dari solusi global dan menunjukkan bagaimana cara mendapatkan solusi global berdasarkan S. Hal ini dilakukan dengan cara menghancurkan T_v dari pohon yang sebenarnya menjadi sebuah daun v dengan bobot yang merepresentasikan subpohon yang telah dihancurkan. Solusi optimal eP ditentukan secara rekursif untuk pohon baru eT, dan menggabungkan hasilnya dengan S untuk mendapatkan solusi optimal P`.

LEMMA 1. Anggap T = (V, t, p, ◄,w) adalah pohon. Anggap v ∈ V adalah simpul dari T. Anggap V_v adalah simpul dari T_v. Anggap S adalah pembagian pohon kekerabatan yang layak dari T_v, sehingga terdapat beberapa pembagian pohon kekerabatan yang optimal P dari T yang mengandung S dan tidak memiliki interval-interval lain di antara keturunan-keturunan v, yaitu dengan $S - \{(v, v)_T\} =$
$\{(l, r)_T \in P | (l, r)_T \subseteq T_v\}$
Lebih jauh, anggap eT = ($V - V_v \cup \{v\}$, t, ep, e◄, ew) adalah pohon T dengan keturunan-keturunan dari v disingkirkan, sehingga ep, e◄, ew adalah fungsi-fungsi dari T yang terbatas pada eT, dengan pengecualian pada bobot baru $v{:}ew(v) := W_{ST}(v)$. Anggap eP adalah pembagian pohon kekerabatan dari T.
Sehingga $P` := eP \cup (S - \{(v, v)_T\})$ adalah pembagian pohon kekerabatan yang optimal dari T.

Lemma 1 menyatakan bahwa algoritma melewati pohon secara *bottomup*. Untuk setiap simpul non-daun v, tentukan pembagian S untuk T_v sehingga merupakan bagian dari solusi global. Kemudian hapus interval kekerabatan dari S (kecuali $(v, v)_T$), dan menggantikan keseluruhan subpohon T_v sebuah simpul yang bobotnya setara dengantotal bobot dari semua simpul pada T_v yang bukan merupakan bagian dari interval dalam S. Selanjutnya dilanjutkan ke simpul berikutnya.
Pendekatan ini mengurangi masalah sebenarnya dalam menemukan pembagian untuk pohon sembarang menjadi masalah yang lebih mudah dalam menemukan pembagian hanya untuk pohon *flat*. Pendekatan *bottom-up* menjamin bahwa, setelah mencapai simpul bagian dalam, semua subpohon yang terletak di bawah simpul tersebut telah dipangkas dan hanya pohon *flat* yang tersisa.
Bottom-up traversal juga merupakan alasan mengapa diperlukan solusi lokal yang *lean* dan juga minimal: Dengan memotong sebanyak mungkin mungkin bobot suatu pohon, akan tercipta submasalah yang lebih sederhana pada level yang lebih tinggi nantinya dari pohon tersebut. Hal ini dilakukan hanya jika tambahan interval kekerabatan tidak dimasukkan, karena tujuan utamanya adalah untuk menemukan pembagian minimal.

3.2. Pembagian pohon flat

Sebelum melihat cara paling optimal untuk menentukan pembagian pohon kekerabatan paling optimal untuk pohon flat, maka perlu ditegaskan bahwa algoritma brute-force sederhana tidaklah mencukupi. Lihat jumlah pembagian pohon kekerabatan yang layak pada pohon flat. Asumsikan bahwa pohon flat dengan n simpul daun yang semua simpulnya berbobot 1. dalam kasus ini, dapat diletakkan sebanyak K-1 daun dari n daun pada partisi yang sama dengan akar. Terdapat K-1 kemungkinan untuk melakukan hal ini. Sebab itu, ikatan yang lebih rendah untuk jumlah partisi akar adalah $\Omega(n^{K-1})$. Perkiraan ini belum mencakup perbedaan kemungkinan untuk menggabungkan simpul-simpul kerabat yang tersisa ke dalam interval. Karena itu, cukup beralasan untuk menyimpulkan bahwa algoritma brute-force tidak layak untuk nilai-nilai tertentu dari K.
Pendekatan terhadap masalah ini dengan menunjukkan bahwa solusi optimal untuk pohon dengan n simpul daun mengandung solusi optimal untuk pohon dengan jumlah daun lebih sedikit daripada n. Kemudian, pengetahuan ini digunakan untuk mengembangkan algoritma pemrograman dinamik yang menemukan solusi optimal dalam waktu $O(nK^2)$. Akhirnya potensi optimisasi dari algoritma ini akan dibahas kemudian.

3.2.1 Substruktur optimal untuk pohon flat

Anggap bahwa terdapat beberapa pilihan terhadap sebuah daun dalam solusi: Daun tersebut bisa diletakkan dalam parisi yang sama dengan akar atau termasuk ke dalam interval dari hasil pembagian. Bersama dengan fakta bahwa terdapat jumlah yang terbatas dari interval yang

layak untuk daun dimasukkan ke dalamnya, hal ini membentuk substruktur masalah yang berguna untuk pemrograman dinamik.
Lemma berikut menytakan bahwa dapat ditemukan pembagian pohon kekerabatan yang optimal untuk pohon flat dengan n simpul daun dengan memilih kandidat solusi terbaik dari salah satu dari dua submasalah berikut: (1) solusi sama dengan pembagian pohon kekerabatan optimal untuk pohon dengan n-1 simpul, yang merepresentasikan pohon aslinya dimana anak terakhir diletakkan ke dalam partisi akar, atau (2) solusi dapat dibentuk dengan menambahkan sebuah interval pada pemabgian optimal untuk pohon yang lebih kecil.

LEMMA 2. Anggap $T = (\{t, c1, \ldots, cn\}, t,p,\blacktriangleleft, w)$ adalah pohon dengan semua simpul kecuali akar adalah daun, yaitu $p(ci) = t$. $\blacktriangleleft$ mengurutkan ci sesuai nilai indeks i. Pohon T_{sj}

1. $D^{s'}_{j-1} \subseteq D^s_j$ *with* $s' := s + w(c_j)$.
2. *For some* m *with* $0 \leq m < j \wedge m < K$: $\forall_{M \in D^s_{j-m-1}} (M \cup \{(c_{j-m}, c_j)_T\}) \in D^s_j$.

$=(\{t, c_1, \ldots, c_j\}, t, p, \blacktriangleleft, w_{sj})$ adalah pohon dengan semua anak $\{c_i | i > j\}$ dihapus dan bobot s yang berbeda untuk akar t, dengan p dan $\blacktriangleleft$ dianggap sebagai batasan untuk $\{c_1, \ldots, c_j\}$, dan $w_{sj}(t) := s$ dan untuk $v \in \{c_1, \ldots, c_j\}$, $w_{sj}(c_i) := w(c_i)$. Anggap D_{sj} adalah set dari pembagian pohon kekerabatan optimal untuk T_{sj}. Kemudian, untuk j=0 dan untuk setiap $s \leq K$ menentukan $D_{s0} = \{\{(t, t)_T\}\}$. Untuk setiap j dengan $1 \leq j \leq n$, sedikitnya satu dari pernyataan-pernyataan berikut benar:

Dengan notasi yang diberikan di atas, masalah yang harus diselesaikan dengan algoritma yang telah ada dapat dinyatakan sbb: Temukan sembarang elemen dari $D_{w(t)n}$. karena sembarang elemen dari $D_{w(t)n}$ dapat dihitung dari pembagian kekerabatan optimal D_{sj} untuk pohon yang lebih kecil (j<n), masalah mudah diseleesaikan dengan pendekatan pemrograman dinamik, seperti ditunjukkan di bawah.

3.2.2. Algoritma FDW untuk pohon flat

Algoritma FDW *(Flat trees, Dynamic programming for tree Width)* menggunakan pemrograman dinamik dengan memulai dari pohon yang hanya mengandung akar. Daun kemudian ditambahkan dengan urutan dari kiri ke kanan dan beriterasi sepanjang semua bobot potensial akar. Untuk setiap pohon kelas menengah, dilakukan penghitungan pembagian optimal. Untuk setiap simpul, harus ditentukan apakah akan dimasukkan menjadi interval kekerabatan dalam solusi atau akan dimasukkan bersama partisi akar. Keputusan ini berdasarkan solusi optimal untuk pohon kelas menengah yang telah diproses.
Secara formal, untuk setiap j<n dan untuk setiap $s \in \{w(t), \ldots, K\}$, ditentukan sebuah elemen tunggal $P \in D_{sj}$, dan disimpan pada tabel yang berindeks j dan s.
Untuk j=0, terdapat $D_{s0} = \{\{(t, t)_T\}\}$ untuk semua s. karena itu $P = \{(t, t)_T\}$. Untuk j>0, Lemma 2 menyatakan bahwa hanya perlu dipertimbangkan sejumlah kandidat yang terbatas: apakah partisi P adalah sembarang elemen dari D_{s_j-1}, atau untuk setiap m dengan $0 \leq m \leq K$, adalah sembarang elemen dari D_{sj-m-1} bersama dengan $(c_{j-m}, c_j)_T$.
Karena itu, terdapat paling banyak K+1 kandidat untuk setiap langkah. Langkah-langkah diproses sesuai dengan peningkatan urutan dari j. Hal ini akan memastikan bahwa partisi dari setiap D_{si} with i < j telah diketahui. Untuk menentukan P pada setiap langkah, hanya perlu dilakukan pengecekan terhadap semua kandidat untuk kelayakan dan menyimpan kandidat dengan kardinalitas terkecil yang juga berbobot paling kecil
Algoritma pada gambar 4 mengimplemntasikan strategi ini. Algoritma ini menggunakan tabel D(s, j) untuk menyimpan pembagian dari D_{sj}. Diasumsikan bahwa akses yang melewati batas pada D (dimana s>K) selalu mengembalikan *dummy* interval dengan card = ∞. Hal ini memudahkan penjelasan algoritma ini.
Lemma 2 menjelaskan bahwa setiap partisi D(s, j) adalah sama denganpartisi yang telah ada atau menambah partisi yang ada paling banyak satu interval. Karena itu, cukup menyimpan sebagai catatan pada D sebagai tiruan dari partisi lain atau hanya menambahkan interval dan sebagai pointer ke rantai interval yang tersisa. Pointer ini diimplementasikan sebagai sepasang *indice* dari partisi lain dalam tabel. Interval tiruan atau baru direpresentasikan dengan dua ikatan simpul (*begin* dan *end*). Sebagai tambahan, setiap catatan pada D (gambar 4) memilliki dua *fields* yang menyimpan kardinalitas dan bobot dari partisi akar untuk menghindari penghitungan ulang selama proses perbandingan.

```
input T = ({t, c_1, ..., c_n}, t, p, ◁, w) flat tree
output D dynamic programming table with final result in D(w(t), n)

for s := w(t) to k
   D(s,0).begin:=t
   D(s,0).end:=t
   D(s,0).card:=1
   D(s,0).rootweight:=w(t)
   D(s,0).next:=(0,0)
for j := 1 to n
   for s := w(t) to K
      s' := s + w(c_j)
      P := D(s', j-1)
      w := 0
      m := 0
      while m < j ∧ m < K ∧ w < K
            w := w + w(c_{j-m})
            if w ≤ K
               crd:= D(s, j-m-1).card+1
               rw:= D(s, j-m-1).rootweight
               if crd< P.card∨(crd= P.card∧rw< P.rootweight)
                  P.begin:=c_{j-m}
                  P.end:=c_j
                  P.card:=crd
                  P.rootweight:=rw
                  P.next:=(s, j-m-1)
            m := m + 1
      D(s,j) := P
```

Entries in the dynamic programming table $D(i,j)$

begin	first node of interval
end	last node of interval
card	cardinality of best parititioning so far (length of next chain)
rootweight	rootweight of best parititioning so far
next	index of next interval

Gambar 4: Algoritma FDW untuk pembagian pohon flat

Setelah menjalankan algoritma di atas, pembagian pohon kekerabatan yang optimal dapat diperoleh dengan memulai pada interval dalam D(w(t), n) dan melewati daftar dari pointer-pointer berikutnya sampai mencapai pointer yang memiliki nilai (0,0).

Cukup mudah untuk menspesifikkan masa hidup algoritma ini: terdapat tiga loop bertingkat, loop terluar diproses paling banyak n kali, dan dua loop yang lebih dalam diproses paling banyak K kali. Jadi algoritma ini memiliki kasus terburuk $O(nK^2)$.

3.2.3 Optimalisasi

Untuk setiap nilai s, tidak perlu ditentukan D(s, j). untuk setiap j, hanya perlu dipertimbangkan nilai s yang dibutuhkan oleh nilai yang lebih tinggi. Hal ini selalu merupakan jumlah dari bobot-bobot simpul. Karena itu, hanya diperlukan nilai s yang merupakan jumlah dari bobot akar dan bobot simpul di kanan c_j.

Cara yang mudah untuk mencapainya adalah menggunakan teknik memorisasi. Semua catatan pada D tidak perlu dihitung sebagaimana ditunjukkan pada *pseudocode* di atas. Sebaliknya, hanya perlu melakukan komputasi berdasarkan permintaan dan mengingatnya dalam D. Permintaan *subsequent* untuk catatan yang sama dapat dipenuhi dalam waktu O(1). Hal ini tidak hanya mempengaruhi kompleksitas asimptotik, tapi juga secara signifikan mengurangi *runtime* dalam dunia nyata: hanya sebagian dari seluruh tabel D yang benar-benar dibutuhkan untuk pohon sebenarnya. (contoh lihat 3.3.6)

3.3 Pembagian pohon *deep* secara optimal

Pada bagian ini algoritma FDW dikembangkan untuk menyelesaikan perdoalan pohon *deep*. Algoritma hasil pengembangan FDW ini disebut GHDW *(for Greedy–Height/Dynamic–Width)*. GHDW menggunakan pembagian optimal lokal untuk menyusun pemabgian global. Namun, terdapat beberapa kasus dimana GHDW menghasilkan hasil yang suboptimal, dan juga akan ditunjukkan bahwa solusi optimal global untuk masalah pembagian ini kadang-kadang membutuhkan solusi optimal lokal. Hal ini dapat menjadi karakteristik pembagian suboptimal lokal yang dibutuhkan dan bagaimana cara menghasilkannya. Hal ini kemudian dimasukkan dalam algoritma pemrograman dinamik. Akhirnya, dihasilkan algoritma waktu linear untuk pembagian pohon kekerabatan lokal optimal.

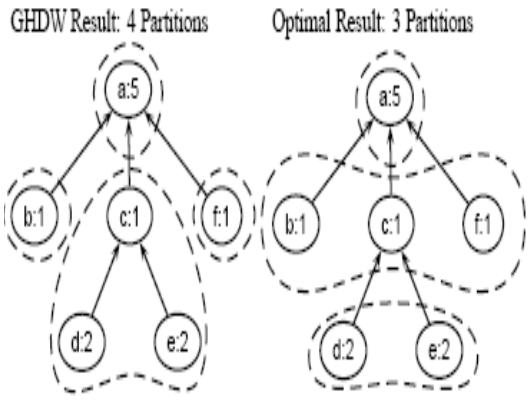
```
input T tree
output D dynamic programming table

for all nodes v of T in postorder
   for s := w_T(v) to K
      D(v,s,0).begin:=NIL
      D(v,s,0).end:=NIL
      D(v,s,0).card:=0
      D(v,s,0).rootweight:=s
      D(v,s,0).next:=(0,0)
   for j := 1 to childcount(v)
      for s := w_T(v) to K
         s' := s + D(v).rootweight
         P := D(v, s', j-1)
         w := 0
         m := 0
         while m < j ∧ m < K ∧ w < K
               w := w + D(c_{j-m}(v)).rootweight
               if w ≤ K
                  crd:= D(v,s,j-m-1).card+1
                  rw:= D(v,s,j-m-1).rootweight
                  if crd< P.card
                        ∨(crd= P.card∧rw< P.rootweight)
                     P.begin:=c_{j-m}(v)
                     P.end:=c_j(v)
                     P.card:=crd
                     P.rootweight:=rw
                     P.next:=(s, j-m-1)
               m := m + 1
         D(v,s,j) := P
```

Gambar 5: Algoritma GHDW untuk pembagian pohon *deep*

3.3.1 Algoritma GHDW

Pseudocode untuk algoritma ditunjukkan oleh gambar di atas. Kode tersebut mirip dengan FDW dengan tambahan loop yang lebih luar untuk setiap simpul. Namun, sekarang pohon yang dihadapi adalah pohon *deep* dan menggunakan primitif yang sesuai untuk mengakses struktur pohon tersebut: digunakan $c_j(v)$ untuk menspesifikasi anak ke j dari v dan childcount(v) untuk menunjukkan jumlah anak dari v.

Pada GHDW, D memiliki dimensi ekstra dibandingkan FDW karena memiliki satu hasil "flat" untuk setiap simpul P. Hasil D(v,w(c_j (v)), childcount(v)) dapat disingkat menjadi D(v), yang merupakan partisi terbaik untuk T_v yang dapat ditemukan oleh GHDW. Perlu diingat bahwa card field dalam D hanya menghitung partisi ekstra yang merupakan hasil dari partisi anak-anak v, bukan dari level yang lebih rendah. Jumlah sebenarnya dari partisi tidak diperlukan, namun hanya perlu untuk memilih salah satu yang memiliki nilai minimal lokal. Menghancurkan pembagian subpohon optimal menjadi satu simpul pada level yang lebih tinggi berikutnya diwujudkan dalam cara yang sederhana: jika FDW menggunakan bobot dari simpul, GHDW menggunakan field bobot akar dari pembagian subpohon optimal untuk simpul-simpul pada level yang lebih rendah.



Gambar 6: kegagalan strategi greedy (K = 5)

Ingat bahwa loop awal s untuk j =0 berbeda dengan FDW; yang disimpan bukanlah interval $(v, v)_T$ untuk akar subpohon v, melainkan interval dummy kosong, karena interval sebenarnya dari v ditentukan pada level yang lebih tinggi. Solusi global diambil dari tabel D dengan mengembalikan rantai partisi selanjutnya untuk setiap simpul (tanpa interval kosong), dan akhirnya menambahkan tambahan interval akar $(t, t)_T$.

Kompleksitas GHDW sama dengan FDW: $O(nK^2)$. Walaupun terdapat ekstra loop yang lebih luar pada GHDW, namun total jumlah iterasinya tidak berubah sehingga kompleksitasnya sama dengan FDW.

Pada Bab 5 nanti akan ditunjukkan bahwa algoritma GHDW menghasilkan nilai yang bagus. Namun hasil yang didapat tidak selalu optimal.

Contoh hasil suboptimal GHDW dapat dilihat pada gambar 6. Batas bobot K=5, dan angka melambangkan bobot simpul.

3.3.2. Definisi

Anggap T = (V, t, p,◀,w) adalah pohon dan Q

$$\Delta W(v) := \begin{cases} 0 & \text{if } Q \text{ does not exist} \\ W^{\mathcal{P}}_{T_v}(v) - W^{Q}_{T_v}(v) & \text{otherwise} \end{cases}$$

adalah pembagian pohon kekerabatan untuk T. Q dikatakan mendekati minimal jika mengandung tepat lebih satu interval daripada partisi minimal. Q dikatakan mendekati optimal jika Q mendekati minimal dan lean.

ΔW(v) mendeskripsikan sembarang $v \in V$ dimana perbedaan bobot akar dari pembagian optimal dan mendekati optimal pada subpohon T_v di-induce oleh v. Anggap p adalah partisi optimal dan Q adalah partisi mendekati optimal untuk T_v. Maka dapat didefinisikan:

LEMMA 3. Anggap T = (V, t, p,◀,w) adalah pohon, dan anggap $v \in V$ adalah simpul pada T. Kemudian, terdapat pembagian pohon kekerabatan yang optimal P dari T dan pembagian pohon kekerabatan S dari T_v sehingga s optimal atau mendekati optimal, dan $S - \{(v, v)_T\} = \{(l, r)_T \in P | (l, r)_T \subseteq T_v\} - \{(v, v)_T\}$, yaitu set interval pada P yang terdapat pada subpohon di bawah P adalah tepat S. Lebih jauh lagi, jika S mendekati optimal, maka $\Delta W(v) > 0$.

Lemma di atas menunjukkan bahwa jika pembagian subpohon optimal untuk sebuah pohon bukan merupakan solusi optimal global, maka solusi optimal globalnya adalah yang mendekati optimal.

LEMMA 4. Anggap T = (V, t, p,◀,w) adalah pohon, dan P adalah pembagian pohon kekerabatan optimal untuk T. Anggap T` = (V, t,

p,◀,w`) adalah pohon T dengan perubahan bobot untuk akar $w'(t) := w(t) + K - W_{PT}(t) + 1$, dan untuk $v \in V - \{t\}$, $w'(v) := w(v)$. Maka, setiap pembagian pohon kekerabatan optimal untuk T` adalah pembagian pohon kekerabatan mendekati optimal untuk T. lebih jauh, jika tidak terdapat pembagian pohon kekerabatan optimal untuk T`, maka $\Delta W(t) = 0$.

Meskipun Lemma 4 menyediakan cara yang mudah untuk mendapatkan pembagian yang mendekati optimal, masih harus diperhatikan bahwa tidak mungkin memilih solusi optimal atau mendekati optimal sampai mencapai tingkat yang lebih tinggi. Cara untuk menyelesaikan persoalan ini secara efisien akan dijelaskan pada bahasan selanjutnya.

3.3.5 Algoritma DHW untuk pohon *deep*

DHW *(Dynamic programming for Height and Width)* merupakan hasil penggabungan pemrograman dinamik dan GHDW. Pemrograman dinamik digunakan untuk memutuskan apakah solusi optimal atau mendekati optimal untuk setiap simpul anak pada subpohon, dan kemudian menggabungkan pilihan ini ke dalam algoritma GHDW.
Lemma berikut menunjukkan bahwa penggunaan solusi mendekati optimal lebih merupakan pengecualian daripada sebuah aturan. Umumnya, hanya solusi optimal yang dianggap sebagai bagian solusi optimal global. Secara khusus, berdasarkan lemma tersebut: (1) pembagian optimal dianggap mencukupi bila v berbagi partisi dengan orang tuanya, dan (2) pembagian mendekati optimal untuk subpohon dari simpul v hanya perlu dipertimbangkan bila semua simpul lain u berada dalam interval yang sama dengan ΔW telah menggunakan pembagian mendekati optimal untuk subpohonnya.

LEMMA 5. untuk setiap pohon $T = (V, t, p, \blacktriangleleft, w)$ terdapat pembagian pohon kekerabatan yang optimal P dari T sehingga untuk setiap $v \in V$, $P_v - \{(v, v)_T\} \subseteq P$ jika salah satu dari pernyataan berikut benar:
1. Tidak ada interval $(l, r)_T \in P$ sehingga $v \in (l, r)_T$.
2. Tidak ada interval $(l, r)_T \in P$ sehingga $v \in (l, r)_T$, dan terdapat *a* $u \in (l, r)_T$ untuk setiap Q_u dan $Q_u - \{(u, u)_T\} \subseteq P \wedge \Delta W(u) < \Delta W(v)$.

Hasil dari algoritma DHW dapat dilihat pada gambar 7. seperti telah dijelaskan sebelumnya, algoritma ini merupakan pengembangan dari GHDW.

```
input T tree
output D dynamic programming table

for all nodes v of T in postorder
  for s := w_T(v) to K
    D(v,s,0).begin:=t
    D(v,s,0).end:=t
    D(v,s,0).nearlyopt:=∅
    D(v,s,0).card:=1
    D(v,s,0).rootweight:=s
    D(v,s,0).next:=(0,0)
  for j := 1 to childcount(v)
    for s := w_T(v) to K
      s' := s + D(c_j(v), w_T(c_j(v)), childcount(c_j(v))).rootweigl
      P := D(v, s', j-1)
      w := 0
      dw := 0
      m := 0
      while m < j ∧ m < K ∧ w - dw < K
          w := w + D(c_{j-m}).rootweight
          dw = dw + ΔW(c_{j-m}(v))
          if w - dw ≤ K
            crd:= D(v, s, j-m-1).card+1
            rw:= D(v, s, j-m-1).rootweight
            C :=nodes (c_{j-m}(v), c_j(v))_T
                 ordered by descending ΔW(c_i(v))
            w' := w
            N := ∅
            while w' > K
              u := head(C)
              w' := w' - ΔW(u)
              N := N ∪ {u}
              C := C - {u}
              crd:=crd+1
            if crd< P.card∨(crd= P.card∧rw< P.rootweight)
              P.begin:=c_{j-m}(v)
              P.end:=c_j(v)
              P.nearlyopt:=N
              P.card:=crd
              P.rootweight:=rw
              P.next:=(s, j-m-1)
          m := m + 1
      D(v,s,j) := P
```

Gambar 7: Algoritma DHW untuk pembagian pohon *deep*

Pemrograman dinamik tabel D memiliki sebuah field baru nearlyopt yang mengandung subset dari simpul dalam interval yang menggunakan pembagian mendekati optimal.
Komputasi dari pembagian mendekati optimal tidak merubah kompleksitas asimptotik karena catatan yang diperlukan tersedia pada tabel D. sebagai tambahan terhadap loop yang dimiliki oleh GHDW, DHW memiliki tambahan loop dengan O(K) langkah untuk menentukan subset Q untuk setiap kandidat interval. DHW juga perlu menciptakan daftar urutan C, yang membutuhkan waktu O(K logK) pada kasus terburuk. Karena itu, kompleksitasnya bertambah untuk setiap langkah pada bagian dalam sebesar

$O(K \log K)$. Kompleksitas GHDW adalah $O(nK_2)$, jadi kompleksitas DHW adalah $O(nK_3 \log K)$. Hal ini berarti DHW adalah algoritma waktu linear untuk pembagian pohon kekerabatan optimal.

3.3.6. Optimalisasi

Algoritma DHW menyediakan beberapa peluang untuk optimalisasi. Pendekatan memeorisasi yang telah dibahas di awal, dapat digunakan. Sebagai contoh, dokumen contoh berukuran 20 MB dan K=256 menunjukkan bahwa rata-rata kurang dari 4 dari potensial 256 nilai untuk s yang benar-benar muncul pada simpul yang lebih dalam.
Lebih jauh lagi, dapat ditunjukkan bahwa untuk subpohon yang di-induce oleh simpul pertama dan terakhir sebagai interval, pembagian optimal selalu mencukupi untuk menghasilkan solusi optimal global.

4. Implementasi dan Algoritma perkiraan

DHW berjalan pada waktu yang linear dan membantu menentukan pembagian kekerabatan yang optimal untuk dokumen nyata dalam waktu yang masih dapat diterima. Sayangnya, DHW juga memiliki beberapa kelemahan yang membuatnya menjadi pilihan suboptimal untuk penyisipan dokumen, seperti yang akan ditunjukkan di bawah.
Bab ini akan menjelaskan alasan-alasan yang mempengaruhi pemilihan algoritma pembagian untuk XDS. Selain itu juga akan ditampilkan beberapa algoritma pembagian yang hasilnya tidak optimal namun cocok untuk dipakai di dunia nyata. Beberapa merupakan algoritma asli, beberapa adalah modifikasi kecil dari algoritma yang sudah ada, dan satu merupakan variasi baru dari algoritma yang telah ada

4.1. Implementasi

Algoritma pembagian untuk menambahkan dokumen dalam native XDS harus memenuhi beberapa persyaratan, di antaranya: (1)harus dapat menyelesaikan tugas secepat mungkin, (2) harus dapat mempartisi dokumen yang sangat besar, dan (3) kualitas hasil tidak boleh bervariasi terlalu besar untuk tipe-tipe dokumen yang berbeda.
Sayangnya, algoritma DHW memiliki persyaratan resource yang dapat menjadi masalah pada penggunaannya: (1)tabel pemrograman dinamik membutuhkan jumlah memori yang besar, (2) meskipun algoritmanya linear terhadap jumlah simpul, faktor K^3 tetap berpengaruh besar, dan (3) pembagian final baru dilakukan setelah seluruh pohon selesai diproses. Hal ini tidak bisa dilakukan terhadap dokumen yang sangat besar ukurannya.
Maka di makalah ini juga akan dibahas beberapa algoritma perkiraan yang berguna sebagai algoritma pembagi XDS.

4.2. Algoritma perkiraan *top-down*

4.2.1. DFS

Algoritma DFS memproses graf menggunakan *depth-first search*, memasukkan simpul secara *greedy* ke dalam kelompok yang ada. Kelompok baru dibentuk apabila kelompok yang ada sudah tidak mampu menampung simpul lagi.
Algoritma yang asli tidak peduli tentang keterhubungan antar partisi, seperti yang diminta oleh pembagian orang tua-anak. Namun algoritma ini dapat dimodifikasi sehingga dapat memulai partisi tidak hanya saat kelompok yang ada sekarang sudah penuh, juga saat simpul yang diproses tidak memiliki keterhubungan dengan simpul di partisi yang sudah ada.
Varian DFS ini bersifat *main-memory friendly* karena bisa ditentukan setiap simpul yang diproses menuju partisi yang seharusnya. Algoritma ini cocok untuk pemrosesan XML karena XML *parsers* mengirim dokumen masukan sebagai aliran dari *even parsing* dalam *depth-first preorder* pada pohon dokumen.

4.2.2 BFS

Strategi untuk DFS dapat pula diterapkan pada BFS. BFS tidak bersifat *main-memory friendly* karena semua simpul harus dikunjungi terlebih dahulu untuk melakukan pencarian BFS yang benar.

4.3. Algoritma perkiraan *bottom-up*

4.3.1. GHDW

Telah dijelaskan di bab sebelumnya. GHDW adalah pendahulu DHW, menghasilkan banyak hasil bagus saat pengujian, dan juga memory-friendly, karena algoritam ini menentukan subpohon yang jelas untuk setiap subpohon yang lebih berat daripada K.

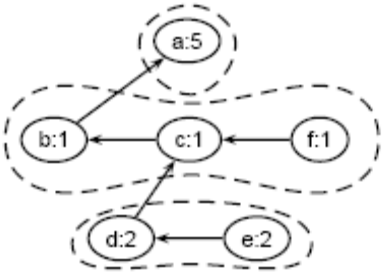
4.3.2. Rightmost Siblings (RS)

Algoritma penyisipan documen Natix menerapkan heuristic yang sangat sederhana. Saat memproses sebuah simpul yang subpohonnya lebih besar dari K, algoritma akan beriterasi sepanjang simpul anak-anak subpohon tersebut dari kanan ke kiri dan menambahkan kerabat sehingga bobot partisi mencapai K. algoritma ini akan terus bekerja untuk menciptakan partisi sampai bobot partisinya kurang dari K. pendekatan ini bersifat *main-memory friendly* dan mudah diimplementasikan.
This approach is main-memory friendly and very simple to implement.

4.3.3. Kundu and Misra (KM)

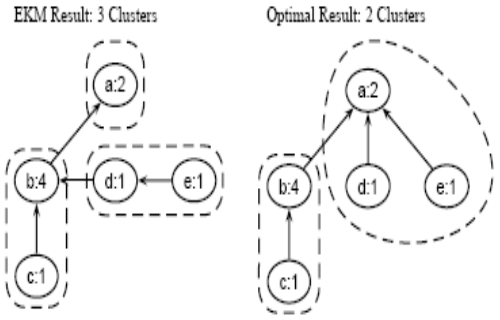
Algoritma Kundu and Misra meminimalkan jumlah total partisi untuk setiap pohon yang diberikan, dan memaksa keterhubungan dari partis-partisi, sekalipun hanya berupa garis orang tua-anak. Sepintas algoritma ini terlihat boros dan juga membutuhkan banyak tempat penyimpanan yang tidak perlu. Namun algoritma ini cukup cepat, berjalan pada waktu yang linear, memproses setiap simpul tepat sekali, dan *memory-friendly*, serta kompleksitasnya tidak tergantung pada batas bobot K.

4.3.4. Enhanced Kundu and Misra (EKM)



Gambar 8: Pohon biner untuk Enhanced KM (K = 5)

Ide dasar dari algoritma yaitu mencoba untuk membuat algoritma KM bekerja pada pohon biner dan bukan pada pohon n-ary. Hasilnya tampak seperti gambar 8. Hasil dari algoritma ini kadang-kadang sama dengan hasil DHW. Namun algoritma ini masih menyisakan masalah. Seperti bisa dilihat pada gambar 9.



Gambar 9: Kegagalan Enhanced KM (K = 5)

5. Pengujian

Setelah melihat tujuh algoritma yang disebut di atas, maka akan dilakukan pengujian terhadap ketujuh algoritma tersebut, terutama untuk membandingkan performa DHW dan DHGW dengan algoritma lainnya.
Dokumen XML yang akan digunakan untuk menguji terdiri dari 6 buah dokumen yang ukurannya bervariasi

Document	Size	Nodes	Weight/*K*
SigmodRecord.xml	477KB	42054	352
mondial-3.0.xml	1785KB	152218	1236
partsupp.xml	2242KB	96005	1026
uwm.xml	2338KB	189542	1446
orders.xml	5379KB	300005	2247
xmark0p1.xml	11670KB	549213	7532

Tabel 1: dokumen-dokumen pengujian

Algorithm						
DHW (Optimal)	GHDW	EKM	RS	DFS	KM	BFS
382	384	402	405	1153	1294	2987
1358	1376	1407	1433	3268	11625	17312
1083	1083	1091	1091	2282	15876	8192
1727	1790	1746	1817	4345	5449	11039
2476	2476	2482	2482	5832	29876	15474
8603	8838	8975	9631	25046	20519	42155

Tabel 2: hasil partisi

	Algorithm						
Document	DHW	GHDW	EKM	RS	DFS	KM	BFS
SigmodRecord.xml	24.83	0.28	<0.01	<0.01	<0.01	0.05	<0.01
mondial-3.0.xml	184.17	6.02	<0.01	<0.01	<0.01	0.11	0.02
partsupp.xml	474.13	5.55	<0.01	<0.01	<0.01	0.16	0.02
uwm.xml	401.38	1.18	<0.01	<0.01	<0.01	0.21	0.04
orders.xml	565.01	9.73	<0.01	<0.01	<0.01	0.35	0.07
xmark0p1.xml	2041.18	6.24	0.02	0.03	<0.01	0.63	0.11

Tabel 3: CPU time (dalam detik)

Hasil pengujian ini menunjukkan keunggulan dan kelemahan algoritma DHW dan DGHW. Keuntungannya adalah, untuk hasil yang optimal, DHW dan DGHW telah mampu mencapainya. Sementara kerugiannya yaitu, untuk hasil yang optimal maka waktu yang diperlukan juga akan berlipat-lipat.

6. Kesimpulan

Kesimpulan yang dapat diambil berdasarkan pengujian terhadap algoritma pembagian pohon adalah:

1. algoritma pembagian pohon kekerabatan berhasil memangkas hasil partisi menjadi 90% dibandingkan dengan algoritma orang tua-anak biasa.
2. DHW berhasil memenuhi tujuan untuk menghasilkan algoritma yang optimal untuk membagi dokumen XML menjadi partisi yang aling minimal.
3. Waktu yang dibutuhkan oleh DHL untuk menyelesaikan partisi dokumen jauh lebih lama dibandingkan algoritma yang lain.
4. Masalah kompleksitas DHW yaitu sebesar $O(nK^3)$ perlu dipecahkan agar waktu pemrosesan bisa dikurangi.

7. Daftar Pustaka

[1] Kanne, Carl-Christian, Guido Moerkotte. (2006). A Linear Time Algorithm for Optimal Tree Sibling Partitioning and Approximation Algorithms in Natix. *www.vldb.org/conf/2006/p91-kanne.pdf*. Tanggal akses: 3 Januari 2007 pukul 15.00

[2] Joseph A. Lukes. (1974). Efficient algorithm for the partitioning of trees. *IBM Journal of Research and Development*,18(3):217–224.

[3] Kanne, Carl-Christian and Guido Moerkotte, (2006). A linear time algorithm for optimal tree sibling partitioning and its application to XML data stores. Technical Report TR-2006-006, University of Mannheim, Department for Mathematics and Computer Science.