Perancangan Sistem agar Tidak Terjadi Deadlock dengan Backtracking

Tifany Angelia / 13518067
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518067@std.stei.itb.ac.id

***Abstraksi*—Semua proses yang terjadi di komputer diatur oleh sistem operasi. Sistem operasi mengatur semua proses yang sedang berjalan agar tidak terjadi deadlock. Deadlock terjadi apabila pada himpunan proses, setiap proses menunggu resource sampai resource tersebut di terminate oleh proses lainnya. Pencegahan terjadinya deadlock sangat penting untuk keberjalanan sistem operasi. Algoritma backtracking dapat digunakan untuk merancang sistem operasi agar himpunan proses pada sistem tersebut tidak terjadi deadlock.**

Kata Kunci—deadlock; backtracking

I. PENDAHULUAN

Sistem operasi mengatur penjadwalan proses yang akan berjalan serta mengalokasikan *resource* yang dibutuhkan proses. Di antara proses tersebut, ada proses-proses yang akan memakai *resource* secara bersamaan, ada yang me-*request resource* yang sama. Jika penjadwalan tiap proses kurang baik dan ada *resource* dengan jumlah *instance* yang terbatas, maka ada suatu proses yang dalam keadaan *waiting*. Jika proses tersebut tidak akan mendapat giliran untuk menggunakan *resource* yang di *request*, maka kemungkinan besar akan terjadi deadlock. Sebelum deadlock terjadi, sebaiknya dilakukan pencegahan dengan merancang sistem dalam penjadwalan proses-proses serta memperhitungkan jumlah ketersediaan *resource* dan yang akan digunakan proses-proses tersebut. Namun, dalam kenyataannya kita tidak dapat mengatur penjadwalan proses suatu sistem. Dan saat terjadi deadlock pun, sebagian besar sistem operasi tidak menangani hal tersebut. Disini hanya mengimplementasikan bagaimana penjadwalan proses-proses suatu sistem yang baik dengan algoritma runut-balik.

II. TEORI DASAR

A. Graf Berarah

Graf G = (V, E), yang dalam hal ini:

V = himpunan tidak-kosong dari simpul-simpul (vertices)
= { v1 , v2 , ... , vn }

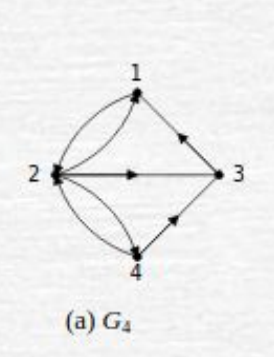
E = himpunan sisi (edges) yang menghubungkan sepasang simpul

= {e1 , e2 , ... , en }

Berdasarkan orientasi arah pada sisi, maka secara umum graf dibedakan atas 2 jenis:

1. Graf tak-berarah (undirected graph)Graf yang sisinya tidak mempunyai orientasi arah disebut graf tak-berarah. Tiga buah graf pada Gambar 2 adalah graf tak-berarah.

2. Graf berarah (directed graph atau di graph) Graf yang setiap sisinya diberikan orientasi arah disebut sebagai graf berarah. Dua buah graf pada Gambar 3 adalah graf berarah.

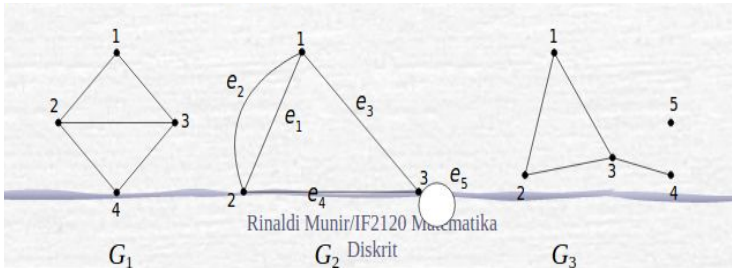


Gambar 1 graf berarah

Lintasan (Path)

Lintasan yang panjangnya n dari simpul awal v0 ke simpul tujuan vn di dalam graf G adalah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk v0, e1, v1, e2, v2,... , vn–1, en, vn sedemikian sehingga e1 = (v0, v1), e2 = (v1, v2), ... , en = (vn-1, vn) adalah sisi-sisi dari graf G.

Tinjau graf G1: lintasan 1, 2, 4, 3 adalah lintasan dengan barisan sisi (1,2), (2,4), (4,3). Panjang lintasan adalah jumlah sisi dalam lintasan tersebut. Lintasan 1, 2, 4, 3 pada G1 memiliki panjang 3.



Gambar 2 Lintasan Graf

Siklus (Cycle) atau Sirkuit (Circuit)

Lintasan yang berawal dan berakhir pada simpul yang sama disebut sirkuit atau siklus. Tinjau graf G1: 1, 2, 3, 1 adalah sebuah sirkuit. Panjang sirkuit adalah jumlah sisi dalam sirkuit tersebut. Sirkuit 1, 2, 3, 1 pada G1 memiliki panjang 3.

B. Algoritma Runut Balik (*Backtracking*)

Backtracking dapat dipandang sebagai salah satu dari dua hal berikut:

1. Sebagai sebuah fase di dalam algoritma traversal DFS
2. Sebagai sebuah metode pemecahan masalah yang mangkus, terstruktur, dan sistematis.

Algoritma runut-balik merupakan perbaikan dari *exhaustive search*. Pada *exhaustive search*, semua kemungkinan solusi dieksplorasi satu per satu. Pada *backtracking*, hanya pilihan yang mengarah ke solusi yang dieksplorasi, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan lagi, dengan prinsip *pruning* yaitu pemotongan pada simpul yang tidak mengarah solusi.

Properti Umum Metode Runut-balik

Solusi Persoalan

Solusi dinyatakan sebagai vektor dengan n-tuple: X= (x1, x2, ..., xn), xi Si. Mungkin saja S1= S2= ... = Sn.

Fungsi Pembangkit nilai xk

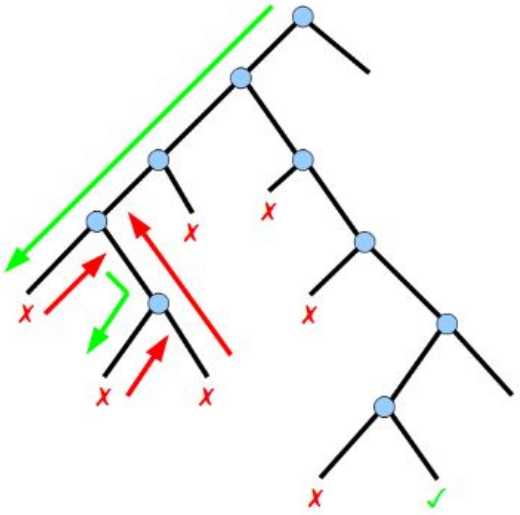
Dinyatakan sebagai predikat:T(k). T(k) membangkitkan nilai untuk xk, yang merupakan komponen vektor solusi.

Fungsi Pembatas

Dinyatakan sebagai predikat B(x1, x2, ..., xk). B bernilai true jika (x1, x2, ..., xk) mengarah ke solusi. Jika true, maka pembangkitan nilai untuk xk+1 dilanjutkan, tetapi jika false, maka (x1, x2, ..., xk) dibuang.

Prinsip Pencarian Solusi dengan Metode Runut-balik

- Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan depth-first order (DFS).
- Simpul-simpul yang sudah dilahirkan dinamakan simpul hidup(live node).
- Simpul hidup yang sedang diperluas dinamakan simpul-E(Expand-node).
- Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang.
- Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" sehingga menjadi simpul mati(dead node).
- Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas (bounding function).
- Simpul yang sudah mati tidak akan pernah diperluas lagi.
- Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian backtrack ke simpul aras diatasnya
- Lalu, teruskan dengan membangkitkan simpul anak yang lainnya.
- Selanjutnya simpul ini menjadi simpul-E yang baru.
- Pencarian dihentikan bila kita telah sampai pada goal node.



Gambar 3 Pohon Solusi dengan Backtracking

C. Deadlock

Pada himpunan proses sistem operasi dikatakan *deadlock*, jika masing-masing proses dalam keadaan menunggu resource yang akan dipakai proses tersebut di release oleh proses lain. Karena seluruh proses sedang menunggu, tidak ada proses yang akan *terminated* dan membangkitkan proses lainnya, yang mengakibatkan seluruh proses dalam keadaan *wait* selamanya.

Pada sistem terdapat beberapa sumber daya (resource) yang digunakan untuk proses-proses untuk menyelesaikan task. Sumber daya yang pada sistem terdiri dari tipe resource CPU cycle, ruang memori, perangkat I/O yang disebut dengan tipe sumber daya R1, R2, . . ., Rm. Setiap tipe sumber daya Ri mempunyai beberapa anggota Wi. Setiap proses yang menggunakan sumber daya menjalankan urutan operasi sebagai berikut :

meminta (request) : meminta sumber daya

memakai (use) : memakai sumber daya

melepaskan (release) : melepaskan sumber daya

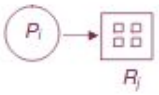
Garis berarah dari proses Pi ke tipe sumber daya Rj dinotasikan dengan Pi -> Rj artinya proses Pi meminta satu anggota dari tipe sumber daya Rj dan sedang menunggu sumber daya tersebut. Garis berarah dari tipe sumber daya Rj ke proses Pi dinotasikan dengan Rj -> Pi artinya satu anggota tipe sumber daya Rj dialokasikan ke proses Pi. Garis berarah Pi -> Rj disebut request edge dan garis berarah Rj -> Pi disebut assignment edge.

Notasi-notasi yang digunakan pada resource allocation graph adalah :

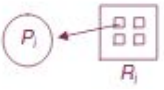
Proses

Tipe *resource* dengan 4 *instances*

Pi *request instance* dari Rj



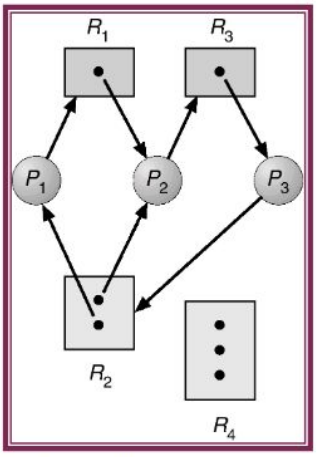
Pi *use one instance resource* R



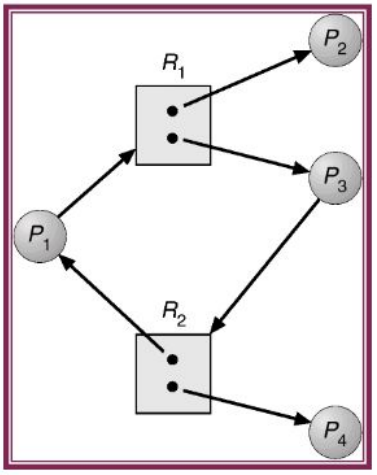
Karakteristik Deadlock

Deadlock terjadi bila terdapat empat kondisi berikut ini secara simultan.

a. Mutual Exclusion : hanya satu proses pada satu waktu yang dapat menggunakan sumber daya.
b. Genggam dan Tunggu (Hold and Wait) : suatu proses membawa sedikitnya satu sumber daya menunggu mendapatkan tambahan sumber daya baru yang dibawa oleh proses
c. Non-Preemption : sebuah sumber daya dapat dibebaskan dengan sukarela oleh proses yang memegangnya setelah proses menyelesaikan task.
d. Menunggu Secara Sirkuler (Circular Wait) : Terdapat sekumpulan proses {P0, P1, ..., P0} yang menunggu sumber daya dimana P0 menunggu sumber daya yang dibawa P1, P1 menunggu sumber daya yang dibawa P2, dan seterusnya, Pn–1 menunggu sumber daya yang dibawa oleh Pn, dan Pn menunggu sumber daya yang dibawa P0.



Gambar 4 Resource allocation graph yang terjadi deadlock



Gambar 5 Resource allocation graph yang tidak terjadi deadlock

Untuk mencegah deadlock dilakukan dengan meniadakan salah satu dari syarat perlu sebagai berikut :

1. Mencegah Mutual Exclusion

   Mutual exclusion benar-benar tak dapat dihindari. Hal ini dikarenakan tidak ada sumber daya yang dapat digunakan bersama-sama, jadi sistem harus membawa sumber daya yang tidak dapat digunakan bersama-sama.

2. Mencegah Hold and Wait

   Untuk mencegah hold and wait, sistem harus menjamin bila suatu proses meminta sumber daya, maka proses tersebut tidak sedang memegang sumber daya yang lain. Proses harus meminta dan dialokasikan semua sumber daya yang diperlukan sebelum proses memulai eksekusi atau mengijinkan proses meminta sumber daya hanya jika proses tidak membawa sumber daya lain. Model ini

mempunyai utilitas sumber daya yang rendah dan kemungkinan terjadi starvation jika proses membutuhkan sumber daya yang popular sehingga terjadi keadaan menunggu yang tidak terbatas karena setidaknya satu dari sumber daya yang dibutuhkannya dialokasikan untuk proses yang lain.

3. Mencegah Non Preemption Peniadaan non preemption mencegah proses-proses lain harus menunggu. Seluruh proses menjadi preemption, sehingga tidak ada tunggu menunggu. Cara mencegah kondisi non preemption :

   o Jika suatu proses yang membawa beberapa sumber daya meminta sumber daya lain yang tidak dapat segera dipenuhi untuk dialokasikan pada proses tersebut, maka semua sumber daya yang sedang dibawa proses tersebut harus dibebaskan.

   o Proses yang sedang dalam keadaan menunggu, sumber daya yang dibawanya ditunda dan ditambahkan pada daftar sumber daya.

   o Proses akan di restart hanya jika dapat memperoleh sumber daya yang lama dan sumber daya baru yang diminta.

4. Mencegah Kondisi Menunggu Sirkular

   Sistem mempunyai total permintaan global untuk semua tipe sumber daya. Proses dapat meminta proses kapanpun menginginkan, tapi permintaan harus dibuat terurut secara numerik. Setiap proses yang membutuhkan sumber daya dan memintanya maka nomor urut akan dinaikkan. Cara ini tidak akan menimbulkan siklus. Masalah yang timbul adalah tidak ada cara pengurutan nomor sumber daya yang memuaskan semua pihak.

Penanganan Deadlock

Terdapat beberapa cara dalam menangani *deadlock*, yang secara umumnya ada 4 cara untuk menanganinya, yaitu:

1. Mengabaikan masalah.
2. Mendeteksi dan memperbaiki.
3. Penghindaran.
4. Pencegahan.

Mengabaikan Masalah Deadlock

Untuk memastikan sistem tidak memasuki *deadlock*, sistem dapat menggunakan pencegahan *deadlock* atau penghindaran *deadlock*. Penghindaran *deadlock* membutuhkan informasi tentang sumber daya yang mana yang akan suatu proses meminta dan berapa lama akan digunakan. Dengan informasi tersebut dapat diputuskan apakah suatu proses harus menunggu atau tidak. Hal ini disebabkan oleh keberadaan sumber daya, apakah ia sedang digunakan oleh proses lain atau tidak.

Metode ini lebih dikenal dengan Algoritma Ostrich. Dalam algoritma ini dikatakan bahwa untuk menghadapi *Deadlock* ialah dengan berpura-pura bahwa tidak ada masalah apa pun. Hal ini seakan-akan melakukan suatu hal yang fatal, tetapi sistem operasi Unix menanggulangi *Deadlock* dengan cara ini dengan tidak mendeteksi *Deadlock* dan membiarkannya secara otomatis mematikan program sehingga seakan-akan tidak terjadi apa pun. Jadi jika terjadi *Deadlock*, maka tabel akan penuh, sehingga proses yang menjalankan proses melalui operator harus menunggu pada waktu tertentu dan mencoba lagi.

Mendeteksi dan memperbaiki.

Hal-hal yang terjadi dalam mendeteksi adanya *Deadlock* adalah:

1. Permintaan sumber daya dikabulkan selama memungkin-kan.
2. Sistem operasi memeriksa adakah kondisi *circular wait* secara periodik.
3. Pemeriksaan adanya *Deadlock* dapat dilakukan setiap ada sumber daya yang hendak digunakan oleh sebuah proses.
4. Memeriksa dengan algoritma tertentu.

Penghindaran Deadlock

Kondisi aman yang dimaksudkan selanjutnya disebut sebagai safe-state, sedangkan keadaan yang tidak memungkinkan untuk diberikan sumber daya yang diminta disebut unsafe-state.

- a. Kondisi Aman (Safe State)
  Suatu keadaan dapat dinyatakan sebagai safe state jika tidak terjadi deadlock dan terdapat cara untuk memenuhi semua permintaan sumber daya yang ditunda tanpa menghasilkan deadlock. Dengan cara mengikuti urutan tertentu.
- b. Kondisi Tidak Aman (Unsafe State)
  Suatu state dinyatakan sebagai state tak selamat (unsafe state) jika tidak terdapat cara untuk memenuhi semua permintaan yang saat ini ditunda dengan menjalankan proses-proses dengan suatu urutan.

Pada sistem kebanyakan permintaan terhadap sumber daya dilakukan sebanyak sekali saja. Sistem sudah harus dapat mengenali bahwa sumber daya itu aman atau tidak (tidak terkena deadlock), setelah itu baru dialokasikan.

Pada sistem deadlock avoidance (penghindaran) dilakukan dengan cara memastikan bahwa program memiliki maksimum permintaan. Dengan kata lain cara sistem ini memastikan terlebih dahulu bahwa sistem akan selalu dalam kondisi aman. Baik mengadakan permintaan awal ataupun saat meminta permintaan sumber daya tambahan, sistem harus selalu berada dalam kondisi aman.

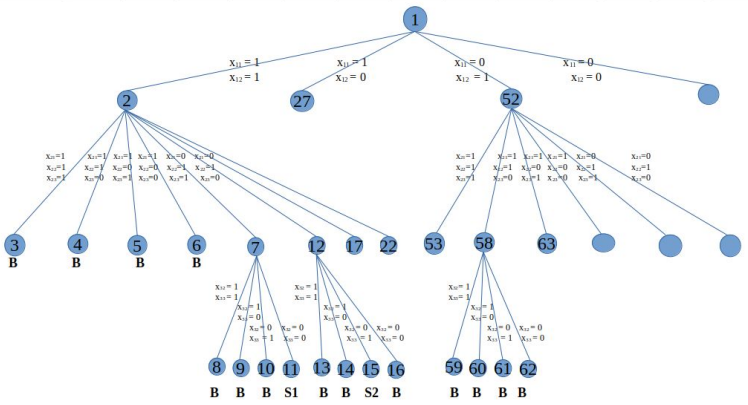
III. PEMBAHASAN

A. Strategi Perancangan

1. Inisialisasi jumlah *instance* setiap *resource* yang dipakai sistem. Buatlah matriks dengan baris merepresentasikan proses dan kolom untuk *resource*. Lalu inisialisasi matrik tersebut dengan nilai -1.
2. Pada simpul pertama, lakukan inisialisasi x pada matriks, jika proses memegang sebuah resource maka x bernilai 1, jika proses menunggu resource maka x bernilai 0.
3. Lakukan pengecekan apakah jumlah *instance* dari setiap *resource* kurang dari jumlah proses yang memegang resource tersebut, dengan mengurangkan jumlah *instance resource* dengan satu, jika simpul sebelumnya menginisialisasi x=1. Jika *resource* tersebut bernilai negatif, lanjutkan ke langkah kelima.
4. Lakukan pembangkitan simpul berikutnya dan lakukan inisialisasi x, ulangi langkah ketiga sampai semua simpul dibangkitkan. Jika semua simpul sudah dibakitkan, lakukan pengecekan apakah jumlah *instance resource* tersebut bernilai 0, jika iya lanjutkan ke langkah keenam. Jika tidak bernilai 0, maka lakukan backtrack.
5. Nilai x pada matriks yang merepresentasikan proses dengan resource tersebut di*assign* dengan nilai -1 dan lakukan backtracking. *Expand* simpul berikutnya ke langkah ketiga.
6. Lakukan pengecekan apakah dapat terjadi deadlock,
7. Untuk setiap proses, lakukan pengecekan terhadap matriks tadi, buatlah sebuah list untuk merepresentasikan rute sistem proses dengan resource.
8. Lakukan pengecekan nilai matriks[i][j], jika bernilai 1 lakukan langkah 9, jika bernilai 0 lakukan langkah ke 10.
9. Lakukan bactrack, dan cek nilai matriks[i][j+1] dan ulangi langkah kedua.
10. Masukkan proses i dan resource j ke list, telusuri apakah terjadi siklus, dengan mencari proses lainnya yang memegang resource j, cek apakah proses tersebut terdapat dalam list, jika proses tersebut terdapat dalam list, maka terdapat siklus dan hentikan proses. Pada simpul tersebut dapat terjadi deadlock, lakukan backtrack pada simpul tersebut dan lakukan langkah ketiga. Jika proses tersebut bukan anggota list, cek nilai matriks[i][1] dan lakukan langkah kedelapan.
11. Jika semua resource pada proses tersebut sudah dicek, lakukan backtrack sampai menemukan solusi lain, jika tidak ada maka kosongkan list dan pindah ke proses selanjutnya, cek matriks[i][j] dan lakukan langkah kedelapan. Jika sudah tidak ada proses, maka sistem tidak ada siklus, program dihentikan.

B. Implementasi dengan Backtracking

Pada kasus ini, sistem memiliki 3 proses dan 3 *resource*, untuk Proses 1 membutuhkan *resource* 1 dan *resource* 2, untuk Proses 2 membutuhkan semua *resource*, dan Proses 3 membutuhkan *resource* 2 dan *resource* 3. *Resource* 1 memiliki 1 *instance*, *resource* 2 memiliki 2 *instance*, dan *resource* 3 memiliki 1 *instance*.

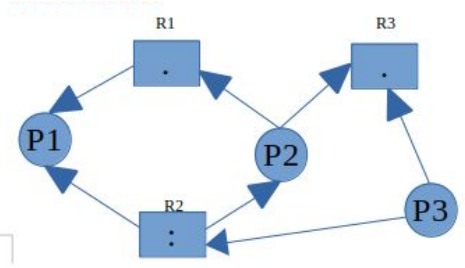


Gambar 6 Pohon ruang solusi

Pada simpul 3, simpul di bounded karena proses 1 dan proses 2 sedang dalam keadaaan *hold*, memegang *resource* 1, dimana *resource* 1 hanya memiliki 1 *instance*. Pada kondisi ini, tidak mungkin terjadi, maka simpul dikatakan tidak mengarah ke solusi dan simpul tersebut di bounded. Begitu juga dengan simpul 4, simpul 5 dan simpul 6.

Solusi pertama ditemukan pada simpul 11, sistem memungkinkan untuk berjalan, lalu solusi ini dicek apakah dapat terjadi deadlock atau tidak.

Untuk simpul ke 11



Gambar 7 Solusi dengan tidak terjadi deadlock

Pada Solusi ini, deadlock tidak terjadi.

Matriks yang terbentuk:

	P1	P2	P3
R1	1	0	-1
R2	1	1	0
R3	-1	0	0

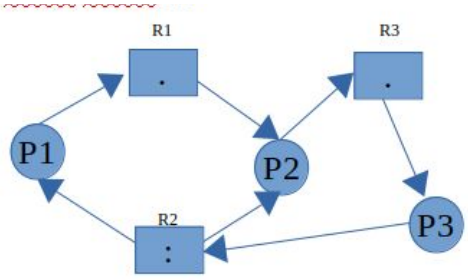
Isi list saat pengecekan deadlock:

- P2, R1
- P2, R3,
- P3, R2, P2, R1
- P3, R2, P2, R3

Solusi pertama ini tidak terjadi deadlock, maka pencarian dapat dihentikan.

Ada suatu kasus, dimana sistem dikatakan dapat berjalan. Tetapi solusi tersebut terjadi deadlock. Solusi ini didapatkan pada simpul 61.

Untuk simpul ke 61



Gambar 8 Solusi dengan terjadi deadlock

Matriks yang terbentuk:

	P1	P2	P3
R1	0	1	-1
R2	1	1	0
R3	-1	0	1

Isi list saat pengecekan:

- P1, R1, P2, R3, P3, R2, P1

Terjadi deadlock.

Karena solusi ini terjadi deadlock, maka simpul tersebut di bounded.

KESIMPULAN

Algoritma runut-balik (*Backtracking*) dapat digunakan untuk melakukan perancangan sistem. Algoritma ini masih perlu ditingkatkan agar dapat diimplementasikan pada sistem operasi.

LINK VIDEO DI YOUTUBE

https://youtu.be/1SGSOKgF3TM

REFERENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Algoritma-Runut-balik-(2020).pdf diakses pada tanggal 1 Mei 2020

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2015-2016/Graf%20(2015).pdf diakses pada tanggal 1 Mei 2020

[3] http://arna.lecturer.pens.ac.id/Diktat_SO/6.Deadlock.pdf diakses pada tanggal 1 Mei 2020

[4] https://www.gurupendidikan.co.id/deadlock-dan-starvation/ diakses pada tanggal 1 Mei 2020

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 4 Meil 2020

Tifany Angelia / 13518067