Aplikasi Backtracking Dalam rm

Penerapan algoritma backtrack untuk flag -r pada rm dan cp

Tanur Rizaldi Rahardjo / 13519214
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): tanurrizaldi@gmail.com

***Abstract*—Aplikasi utilitas merupakan aplikasi yang sering sekali digunakan oleh pengguna komputer. Beberapa aplikasi utilitas yang sering digunakan adalah rm dan cp. Kedua utilitas sering sekali digunakan untuk melakukan operasi file pada sistem operasi turunan UNIX. Untuk melakukan operasi terhadap folder, kedua utilitas meminta flag -r yang digunakan untuk melakukan operasi penghapusan atau pengcopyan secara rekursif. Implementasi flag -r pada kedua utilitas dapat menggunakan algoritma backtracking.**

Keywords— Algoritma, Backtracking, Utilitas, rm, -r

I. PENDAHULUAN

Sistem operasi merupakan salah satu lapisan abstraksi yang sangat penting pada komputer modern. Sistem operasi memperbolehkan pengguna komputer untuk melakukan operasi terhadap file, melakukan komunikasi antar sistem, dan hal-hal lain tanpa perlu memperhatikan detail pada hardware. Selain menyediakan layanan *low level* yang dapat digunakan pengguna komputer, sistem operasi umumnya juga memiliki aplikasi-aplikasi utilitas yang memberikan lapisan abstraksi yang lebih tinggi kepada pengguna komputer.

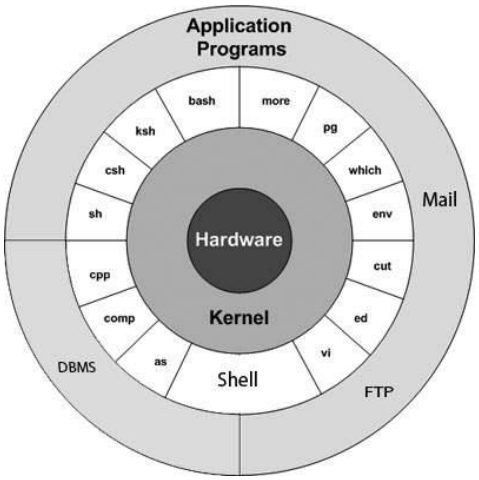
Tujuan utama dalam abstraksi adalah pengguna tidak perlu mengetahui detail pembuatan dan mekanisme alat yang akan digunakan dalam menggunakannya. Pengguna awam dapat menggunakan aplikasi utilitas seperti *explorer* pada sistem operasi *Windows* dan *rm* pada sistem operasi UNIX tanpa perlu mengetahui cara kerja *filesystem* dan *storage device* berkerja.

Dalam implementasi utilitas *rm* dan *cp* umumnya juga menambahkan *flag -r* yang digunakan untuk melakukan penghapusan atau peng-*copyan* secara rekursif pada suatu folder. Implementasi *flag -r* dapat menggunakan algoritma *backtracking* dan struktur data *stack.*

II. LANDASAN TEORI

A. Sistem operasi

Definisi sistem operasi umumnya suatu *software* yang berjalan diatas *hardware* secara langsung yang menyediakan operasi dan layanan *low level* dan aplikasi-aplikasi utilitas yang mengimplementasikan layanan *low level* yang sering digunakan ketika mengoperasikan komputer. Bagian pertama definisi sistem operasi umumnya disebut dengan *kernel.*



Gambar 2.1 Diagram *Ring* pada sistem operasi

(Sumber https://www.tutorialspoint.com/unix/unix-getting-started.htm)

Gambar diatas menggambarkan bagaimana aplikasi pengguna berjalan diatas suatu mesin komputer. Kernel bertanggung jawab penuh terhadap hardware yang menjalankannya. Manajemen memori, media penyimpanan, berkomunikasi dengan *device* eksternal, dan hal-hal lain yang krusial dalam pengoperasian mesin akan diatur oleh kernel.

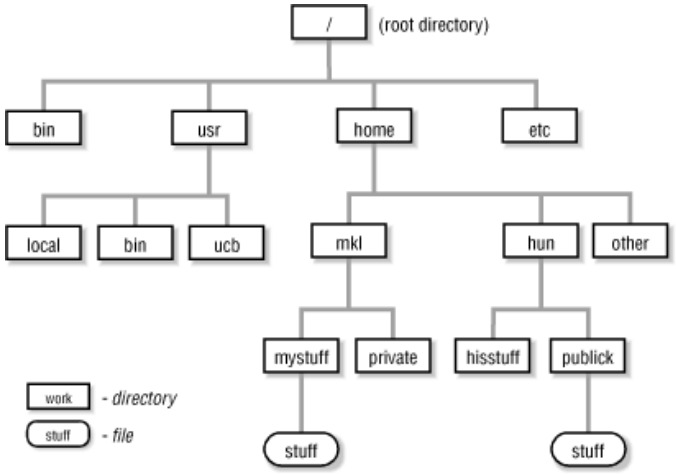
Pengguna juga dapat menggunakan fitur yang disediakan oleh kernel dengan menggunakan *system call* atau suatu *wrapper* yang disediakan pada *library* sistem operasi. Abstraksi hardware oleh kernel sistem operasi dapat mempermudah pengguna untuk melakukan aktivitas dan meningkatkan keamanan sistem.

Aplikasi utilitas yang disertakan pada sistem operasi merupakan alat-alat yang sering digunakan ketika mengoperasikan komputer, seperti operasi pemindahan file, penggantian nama file, melakukan koneksi ke internet, dan lain-lain. Pengguna komputer dapat langsung menggunakan utilitas yang disertakan pada sistem operasi tanpa mengimplementasikannya ulang.

Pengorganisasi media penyimpanan disebut dengan *filesystem*. Banyak aplikasi utilitas merupakan operasi terhadap file dan folder yang ada sehingga membutuhkan layanan kernel tentang manipulasi *filesystem.*

B. *Filesystem*

Filesystem menyimpan informasi detail tentang suatu file atau folder pada sistem operasi. Dalam implementasi, filesystem merupakan serangkaian pilihan *trade-off* oleh desainer sistem sehingga banyak kategori filesystem yang dapat digunakan. Salah satu filesystem yang sering digunakan adalah UNIX filesystem.

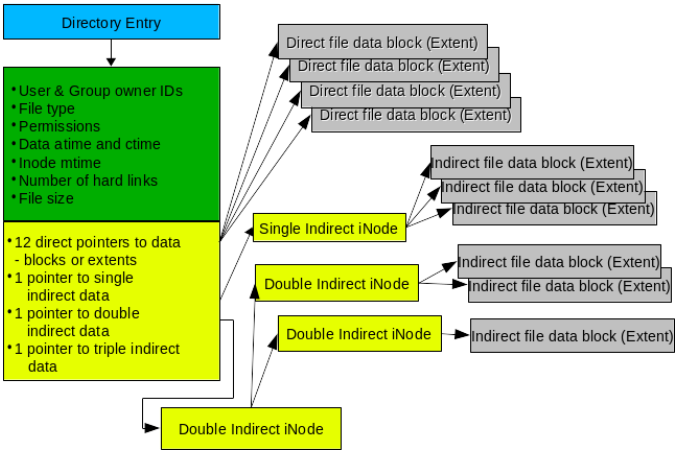


Gambar 2.2 Struktur Pohon UNIX filesystem

(Sumber https://docstore.mik.ua/orelly/unix3/upt/ch01_14.htm)

Sistem operasi yang menggunakan kernel Linux umumnya menerapkan prinsip UNIX filesystem dalam pengorganisasian media penyimpanan. Filesystem ini menggunakan satu direktori akar yang biasanya ditandai dengan / dan mewakili partisi media penyimpanan dalam bentuk suatu direktori.

Kernel Linux secara default menggunakan ext4 filesystem. Filesystem tersebut menyimpan metadata tentang file dalam inode.



Gambar 2.3 Struktur data inode

(Sumber https://opensource.com/article/17/5/introduction-ext4-filesystem)

Setiap file pada filesystem ext4 akan memiliki inode yang menyimpan metadata terkait file itu sendiri. Inode menyimpan metadata dimana lokasi block file disimpan, ukuran file, tipe file, dan lain-lain. Aplikasi utilitas pada Linux memanipulasi metadata pada inode untuk memenuhi operasinya.

C. *Aplikasi Utiltias*

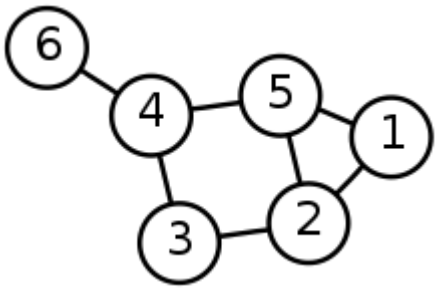
Aplikasi utilitas umumnya merujuk kepada aplikasi yang melakukan operasi sehari-hari seperti pemindahan file, pembuatan suatu folder baru, dan lain-lain yang bertujuan untuk mempermudah pengguna komputer.

Kernel telah menyediakan layanan *low level* seperti mengambil dan memanipulasi inode, melakukan operasi pada *device* dan lain-lain. Pengguna komputer dapat memakai layanan tersebut dalam pembuatan aplikasi yang membutuhkan layanan-layanan khusus dengan hardware. Namun tidak seluruh pengguna komputer ingin dan memiliki waktu untuk mengimplementasikan aplikasi sehingga sistem operasi umumnya juga menyediakan utilitas-utilitas umum yang dapat digunakan secara langsung oleh *end-user* tanpa melakukan implementasi ulang.

Aplikasi utilitas berkaitan dengan manipulasi file dan folder yang biasanya disediakan oleh sistem operasi yang menggunakan kernel Linux adalah *rm, cp, touch, ln, mv, mkdir, ls, cat,* dan lain-lain.

D. *Graf*

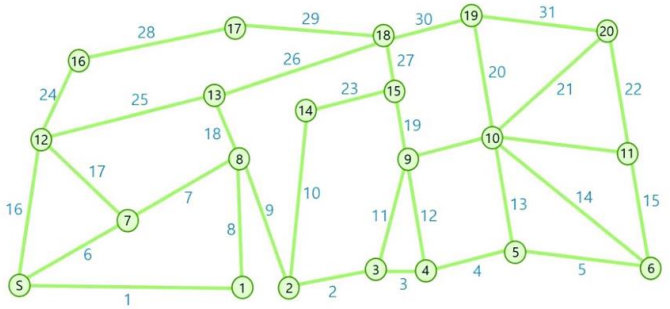
Graf adalah salah satu struktur diskrit pada matematika. Definisi graf merupakan suatu koleksi himpunan yang memiliki himpunan simpul atau *node* dan himpunan sisi atau *edge*. Sederhananya graf adalah suatu kumpulan titik-titik dan sisi yang menghubungkan antara dua titik. Graf dapat diklasifikasikan berdasarkan adanya arah atau nilai pada setiap sisi, adanya kalang atau *loop*, dan pohon.



Gambar 2.4 Graf sederhana

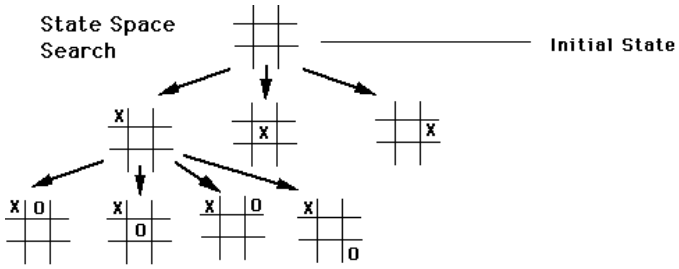
(Sumber https://en.wikipedia.org/wiki/Graph_theory)

Kesederhanaan definisi graf menyebabkan mudahnya pengaplikasian struktur graf pada suatu permasalahan. Graf dapat digunakan untuk memodelkan suatu hal yang memiliki *notion* keterhubungan. Beberapa contoh dari hal yang dapat dimodelkan dengan graf adalah rute jalan, ruang status atau *state space* dan lain-lain.



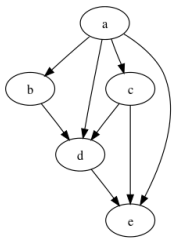
Gambar 2.5 Pemodelan Rute jalan dengan *weighted undirected graph* *(Sumber https://www.researchgate.net/figure/Example-of-a-network-in-the-traveling-salesman-problem-TSP_fig1_338071918)*

Rute jalan merupakan salah satu penerapan graf yang paling sering dibahas dalam bidang *computer science*. Graf tidak harus mewakilkan suatu objek fisik nyata pada dunia sehingga "rute jalan" bisa saja mewakilkan suatu koneksi antar komputer dan komputer lain.



Gambar 2.6 *State space* permainan Tic-Tac-Toe
(Sumber https://www.computing.dcu.ie/~humphrys/Notes/AI/statespace.html)

Selain memodelkan rute, graf juga dapat memodelkan suatu ruang abstrak yang memiliki keterhubungan satu sama lain dengan operasi-operasi tertentu. Ruang status merupakan salah satu ruang yang sering digunakan sebagai daerah pengaplikasian graf pada bidang *computer science*.



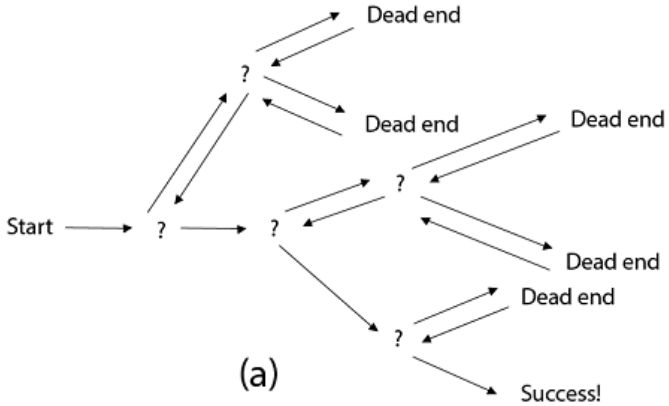
Gambar 2.7 *Directed acyclic graph*
(Sumber https://en.wikipedia.org/wiki/Directed_acyclic_graph)

Selain kedua pemodelan diatas banyak sekali hal-hal yang dapat dimodelkan dengan graf seperti *linked list* yang biasanya *directed acyclic graph* dengan batasan sisinya hanya satu untuk setiap simpul. Banyak hal lain yang dapat ditranslasikan menjadi permasalahan graf, sehingga algoritma untuk menyelesaikan masalah graf dapat diaplikasikan pada banyak hal.

Algoritma seperti *depth first search*, *breath first search*, *backtracking*, *branch and bound* merupakan algoritma yang ditujukan untuk menemukan suatu simpul dengan titik awal dan aturan-aturan tertentu.

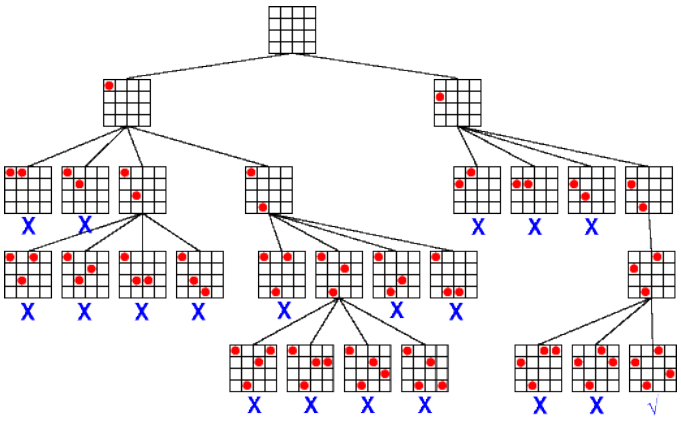
E. Algoritma Backtracking

Backtracking merupakan salah satu cara dalam pencarian jalan dalam suatu graf. Ide dasar ini sering diaplikasikan oleh manusia ketika mencari alamat jalan suatu tempat. Jika terdapat perempatan jalan, pencari alamat dapat memilih salah satu jalan dan mengecek apakah didalam jalan tersebut ada alamat yang dicari atau tidak. Jika tidak kembali ke jalan sebelumnya dan lakukan pada jalan yang lain.



Gambar 2.8 *Backtracking* ketika mencari jalan
(Sumber https://www.javatpoint.com/backtracking-introduction)

Karena graf juga dapat memodelkan ruang status suatu permasalahan yang ingin diselesaikan, algoritma *backtracking* juga dapat diaplikasikan untuk mencari *state* yang diinginkan. Algoritma backtrack juga dapat ditambah dengan heuristik yang diinginkan untuk mempercepat proses pencarian.



Gambar 2.9 Ruang status *N-Queen Problem*

(Sumber https://ktiml.mff.cuni.cz/~bartak/constraints/propagation.html)

Heuristik dapat mempercepat waktu dan mengurangi ruang memori yang digunakan ketika melakukan pencarian jalur. Dengan heuristik yang tepat dengan aturan-aturan yang ada pada graf, backtracking akan jauh lebih cepat. Penggunaan heuristik untuk menghilangkan *state* yang tidak valid merupakan teknik pada algoritma *branch and bound*.

Pengkategorian algoritma seperti algoritma *backtracking* dan *branch and bound* hanya mempermudah orang lain untuk memahami konsep dasar apa yang digunakan dalam suatu algoritma. Secara realita, algoritma tidak harus ekslusif backtracking atau branch and bound. Algoritma pencarian jalan pada graf dapat menggunakan backtracking sebagai konsep utama dan menerapkan konsep heuristik branch and bound untuk mempercepat waktu pencarian.

III. Implementasi

A. *Implementasi backtracking*

Penulis menerapkan konsep algoritma backtracking pada implementasi *flag -r* pada mata kuliah sistem operasi. Aplikasi utilitas *rm* dan *cp* berdasarkan standar POSIX hanya menghapus atau meng-*copy* file dan menolak input folder. *Flag -r* digunakan ketika pengguna ingin melakukan penghapusan atau peng-*copy*-an folder secara rekursif.

Arti rekursif pada kalimat sebelumnya adalah seluruh file dan folder yang terdapat pada folder target akan dihapus atau dicopy juga file didalamnya.

Sistem operasi yang digunakan merupakan sistem operasi 16-bit yang berjalan diatas emulator. Filesystem yang digunakan adalah filesystem sederhana yang setiap inode-nya menyimpan direktori lokasi file disimpan, *single indirect pointer* kepada tabel yang telah ditentukan, dan nama file sendiri.

Untuk mempermudah *backtracking* pada *flag -r*, diperlukan suatu prosedur khusus yang digunakan untuk menghapus atau meng-*copy* suatu file. Berikut adalah contoh *function signature* untuk operasi terhadap file

```
void file_delete(char *filename, byte dir_idx);

void file_copy(char *source_name,
byte source_dir_idx, char *target_name,
byte target_dir_idx);
```

Tipe data *byte* merupakan tipe data *unsigned char* yang digunakan sebagai *byte*. Kedua *function signature* memiliki nama file yang dijadikan sumber dan lokasi direktori atau folder disimpannya file terkait. Prosedur diatas akan tidak melakukan apa-apa jika sumber yang diberikan adalah folder.

Implementasi file_delete() menggunakan system call deleteFile() dan file_copy() menggunakan readFile() dan writeFile() yang telah disediakan oleh kernel. Kegagalan penghapusan atau pengcopyan akan di *handle* oleh file_delete() dan file_copy() untuk memudahkan *error handling* pada utilitas *rm* dan *cp*.

Utilitas *rm* dan *cp* akan memanggil prosedur file_delete() atau file_copy() jika tidak diberikan *flag -r* dalam pemanggilan kedua utilitas. Pada flag -r digunakan algoritma backtracking untuk menghapus atau meng-copy secara rekursif seluruh file dan folder yang ada pada folder sumber.

Untuk algoritma backtracking utiltias rm dan cp, dapat digunakan sebuah struktur data *stack* untuk menyimpan jalur yang telah dilalui, dan struktur data *list* untuk menyimpan informasi folder mana saja yang telah dilalui algoritma backtracking.

```
push(&stack_route_rm, source_dir_idx);
```

Pada utilitas rm, *stack* akan diinisiasi dengan folder sumber dan melakukan penghapusan pada filesystem untuk semua file yang terletak pada folder tersebut dengan prosedur file_delete(). Jika semua isi file pada folder telah dihapus, catat pada struktur data list bahwa folder tersebut telah dihapus. Pada tahap penghapusan isi folder, jika ditemukan sebuah folder, tambahkan folder tersebut pada stack rute.

```
push(&stack_route_rm, iter_idx);
```

Variabel iter_idx mewakili folder yang berada didalam folder. Setelah penghapusan pada isi file pada folder telah selesai, lakukan operasi pop dan pengecekan, apakah masih ada folder yang isinya masih belum dihapus atau tidak, jika masih ada, lakukan operasi penghapusan kembali pada folder tersebut.

Lakukan langkah diatas hingga seluruh stack rute kosong. Ketika selesai, ambil list yang digunakan sebagai catatan folder dan hapus menggunakan system call deleteFolder() yang disediakan pada kernel.

Untuk implementasi utilitas cp, diperlukan juga list yang digunakan untuk menyimpan nama folder target dan lokasi direktori folder target. Operasi penghapusan menggunakan file_delete() akan diganti dengan file_copy() dalam peng-copy-an file.

B. *Pemodelan graf pada implementasi*

Kedua implementasi algoritma diatas merupakan implementasi algoritma backtracking pada suatu graf. Lebih spesifiknya tabel direktori yang menunjuk lokasi dan tidak memperbolehkan adanya penunjukan secara siklik merupakan *directed acyclic graph.* Jika penunjukan satu arah pada tabel direktori dianggap *undirected edge*, maka tabel direktori akan membentuk suatu pohon atau *tree.*

Meskipun tabel direktori yang terdapat pada filesystem bersifat datar pada media penyimpanan, adanya penunjukan lokasi direktori menyebabkan setiap file memiliki keterhubungan dengan folder. Keterhubungan antara file dan folder dapat dimodelkan menggunakan struktur matematis graf.

Gambar 3.1 Utilitas *tree*

(Sumber Dokumen pribadi penulis)

Stack yang digunakan untuk menyimpan rute merupakan struktur data yang digunakan sebagai alat backtracking. Setiap folder yang belum diproses akan dimasukkan kedalam stack dan dicatat apakah telah dihapus atau belum. Jika tidak ditemukan folder pada daun graf yang sedang ditempati, lakukan backtrack dengan melakukan pop pada stack rute. Lakukan backtrack hingga ditemukan folder yang belum diproses atau sudah mencapai akar sumber folder.

C. *Algoritma backtracking dibandingkan alternatif*

Sistem operasi berjalan pada sistem x86, 16-bit yang memiliki memori yang relatif jauh lebih kecil dibandingkan sistem 32-bit dan 64-bit pada umumnya. Alternatif algoritma untuk melakukan *flag -r* adalah melakukan secara operasi secara rekursif.

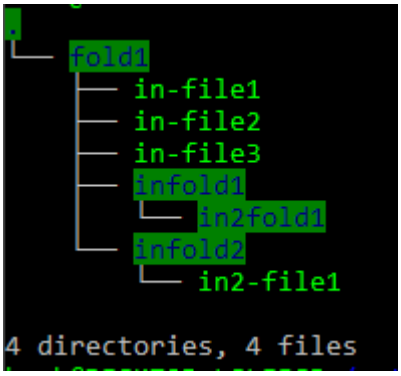
Pada sistem operasi yang digunakan, operasi peng-copy-an membutuhkan buffer untuk menyimpan file pada setiap operasi writeFile() dan readFile(). Sehingga jika melakukan secara naif, rekursif akan dengan cepat menghabiskan memori yang ada pada segment yang digunakan.

Backtracking digunakan daripada rekursif karena operasi rekursif yang sangat mahal pada bahasa C. Implementasi backtracking juga cukup sederhana dibandingkan melakukan optimisasi memori rekursif yang membutuhkan waktu tambah dalam debugging dan testing.

IV. PENGUJIAN DAN PEMBAHASAN

A. *Skenario Pengujian*

Pengujian dilakukan dalam sistem operasi yang dibuat dan dijalankan pada emulator bochs. Host operating system adalah Windows 10 dan menggunakan WSL untuk menjalankan bochs. Dibuat sebuah unit test bernama recursion_test yang akan membuat direktori file seperti berikut

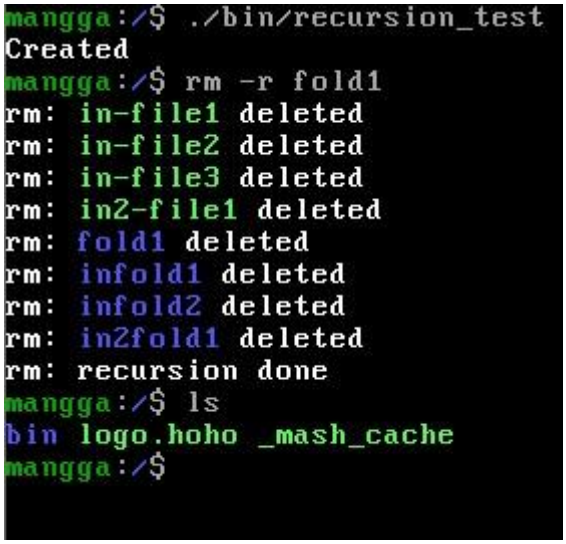


Gambar 4.1 Unit testing untuk flag -r

(Sumber Dokumen pribadi penulis)

B. *Pengujian*

1. Pengujian *flag -r* pada rm



2. Pengujian *flag -r* pada cp

C. *Pembahasan*

Pengujian pertama membuat folder dengan unit testing recursion_test dan menghapus isi folder tersebut menggunakan *rm -r*

Pengujian kedua membuat folder dengan recursion_test dan meng-copy seluruhnya pada folder target bernama rec_test. Utilitas ls digunakan untuk memperlihatkan hasil copy flag -r utilitas cp.

V. KESIMPULAN

Backtracking merupakan golongan algoritma yang cukup mudah dipahami dan diimplementasikan. Penggunaan backtracking pada rm dan cp juga mengurangi permasalahan memori yang dimiliki pada algoritma naif rekursif.

VI. REFERENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/

Sumber kode penulis terdapat pada repository

https://github.com/Lock1/OS-IF2230

(Catatan : Release yang digunakan pada makalah ini adalah release Original Milestone 3.)

UCAPAN TERIMA KASIH

Penulis berterima kasih kepada Bapak Dr. Ir. Rinaldi Munir, MT selaku dosen pengajar program studi informatika yang telah mengajar penulis selama ini dan memberikan ilmu terkait ke-informatika-an sehingga penulis dapat menyelesaikan makalah ini dengan baik.

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, 11 Mei 2021

Tanur Rizaldi Rahardjo