APLIKASI PENGUJIAN WHITE-BOX IBII ONLINE JUDGE

Handy[1)] dan Joko Susilo[2)]
[1)] Alumni Program Studi Sistem Informasi
[2)] Staf Pengajar Program Studi Sistem Informasi
Institut Bisnis dan Informatika Kwik Kian Gie
Jl. Yos Sudarso Kav.87 Sunter Jakarta Utara 14350
http://www.kwikkiangie.ac.id
www.joko.susilo@kwikkiangie.ac.id

ABSTRACT

The development of information and communication technology assist people in various aspects of human life. One aspect of those is education. Utilization of information and communication technology in education is useful for users to understand the learning materials quickly and easily. Currently there has been online judge system which used for practice work on the problems of programming. IBII provides a means for students majoring in information systems and information technology for practicing programming capabilities through IBII Online Judge. In IBII Online Judge, there is a problem with the requirements of the use of algorithms and data structures. The purpose of this study is to design and create a white-box testing application that can assist test on IBII Online Judge. Application of white-box testing IBII Online Judge in this study developed using the Java programming language. The design of white-box testing utilized the theory of lexical analysis with knowledge of algorithms and data structures. This study was developed using the methods of systems development life cycle (SDLC) waterfall model. The design of the system used in this study described as an object-oriented by using Unified Modeling Language (UML) such as use case diagram, class diagrams, sequence diagrams and activity diagrams. The results of this study were white-box testing application that can assist test on IBII Online Judge. The conclusions of this research is the application of white-box testing IBII Online Judge useful to perform testing on the user's submitted source code to the requirements of IBII Online Judge problem in the use of algorithms and data structures.

Keyword: *white-box testing, online judge.*

1. PENDAHULUAN

Perkembangan teknologi dan informasi yang maju membantu kehidupan manusia di berbagai aspek, salah satunya adalah aspek pendidikan. Pemanfaatan teknologi pada pendidikan bermanfaat bagi penggunanya untuk memahami suatu materi pembelajaran dengan praktis dan mudah.

Saat ini telah terdapat sistem *online judge* yang dimanfaatkan untuk berlatih mengerjakan soal pemrograman. Pada awalnya *online judge* digunakan dalam lomba pemrograman dan persiapan latihan mengikuti perlombaan. Namun, seiring perkembangan teknologi dan informasi, *online judge* tidak hanya digunakan dalam batasan lomba pemrograman namun juga untuk terbuka umum. Setiap orang dapat mendaftarkan diri di situs *online judge* dan berlatih mengerjakan soal-soal pemrograman.

Algoritma dan struktur data adalah mata kuliah wajib yang diajarkan pada mahasiswa jurusan informatika yang mengajarkan tentang dasar pemrograman. Beberapa mahasiswa sulit untuk memahami algoritma yang telah dipelajari di kelas karena jarang menulis kode program dan mengulangnya kembali.

Kampus IBII menyediakan sarana bagi mahasiswa jurusan informatika untuk berlatih kemampuan pemrograman melalui IBII Online Judge. Pada IBII Online Judge, mahasiswa dapat

mengerjakan soal-soal algoritma yang tersedia dengan persyaratan dalam menyelesaikan soal. Untuk menjawab, mahasiswa dapat mengunggah jawabannya ke sistem, kemudian sistem penilaian akan memproses dan memberikan respon hasil penilaian. Soal yang diberikan mempunyai jawaban yang unik dan tidak membutuhkan pengujian spesifik untuk standar keluarannya. Namun pada soal dengan persyaratan penggunaan algoritma dan struktur data, penilaian tersebut dilakukan secara manual. Keterbatasan waktu dosen untuk memeriksa jawaban mahasiswa secara manual menyebabkan mahasiswa lama mendapatkan hasil penilaian.

White-box testing merupakan salah satu metode pengujian program yang bertujuan untuk memeriksa komponen program apakah berjalan semestinya dengan melihat internal kode program tersebut. Dengan menggunakan metode *white-box* maka alur program jawaban mahasiswa dapat diuji. Hal ini melatarbelakangi penulis untuk melakukan pengembangan aplikasi pengujian *white-box* IBII Online Judge.

2. IDENTIFIKASI MASALAH

a. Mahasiswa sulit untuk memahami algoritma yang telah dipelajari di kelas karena jarang menulis kode program dan mengulangnya kembali.
b. Pada IBII Online Judge terdapat soal dengan persyaratan penggunaan algoritma dan struktur data yang penilaiannya dilakukan secara manual.
c. Keterbatasan waktu dosen untuk memeriksa jawaban mahasiswa.

3. TINJAUAN PUSTAKA

Black Box Testing

Menurut Myers, Glenford J., Corey Sandler, dan Tom Badgett (2011:8), *black box testing* merupakan salah satu strategi pengujian yang dikenal juga sebagai *data driven testing* atau *input/output testing*. Untuk menggunakan metode ini, program dipandang sebagai kotak hitam. Tujuannya adalah untuk mengabaikan *internal behaviour* dan struktur program. Sehingga penguji dapat berkonsentrasi menemukan keaTindaan saat program tidak berjalan sesuai spesifikasinya. Pada pendekatan ini, kasus data hanya dibuat berdasarkan spesifikasinya tanpa perlu mengetahui struktur internal program.

White-Box Testing

Menurut Pressman (2010:485), *white-box testing*, atau yang disebut *glass-box testing*, adalah metode perancangan *test case* yang menggunakan penjelasan struktur kontrol sebagai bagian dari *component-level design* untuk membuat *test cases*. Dengan menggunakan metode *white-box testing*, *software engineer* dapat membuat *test case* yang (1) menjamin semua jalur indepeden di dalam modul telah dieksekusi sekurangnya sekali, (2) menguji semua keputusan logikal pada nilai *true* dan *false*, (3) menjalankan semua perulangan pada batasannya dan dalam batas operasionalnya, dan (4) menguji struktur data internal untuk memastikan kebenarannya.

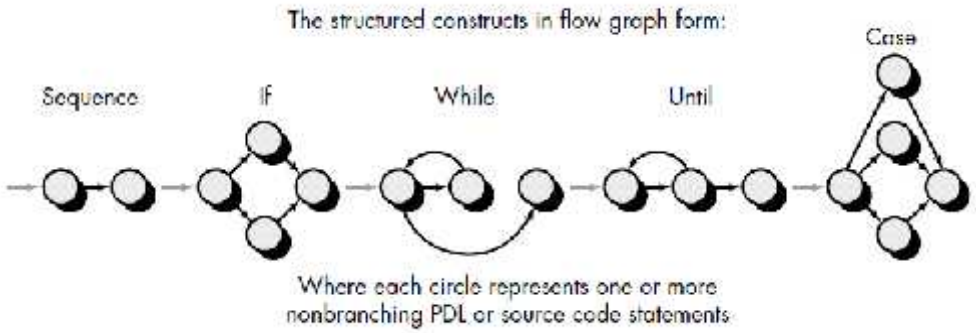
a. Basis Path Testing

Basis path testing adalah teknik *white-box testing* yang pertama dikemukakan oleh Tom McCabe (1976:308-320). Metode *basis path* memungkinkan perancang *test case* untuk membuat pengukuran kompleksitas logikal dari rancangan prosedural dan menggunakan pengukuran ini sebagai panduan untuk mendefinisikan himpunan basis dari jalur eksekusi. *Test case* yang dibuat untuk menguji himpunan basis dijamin akan mengeksekusi setiap *statement* di dalam program sekurangnya sekali pada saat pengujian.

b. Notasi *Flow Graph*

Sebelum mempelajari metode *basis path*, notasi sederhana untuk merepresentasi *control flow* yang disebut *flow graph* (atau *program graph*) harus dipahami terlebih dahulu. *Flow graph* menggambarkan alur kontrol logikal menggunakan notasi yang diilustrasikan oleh gambar 2.1. Setiap struktur yang dibangun mempunyai simbol *flow graph* yang berpasangan.

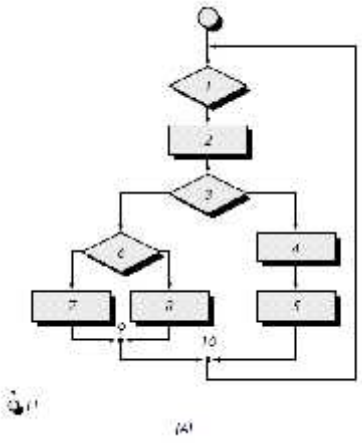
Gambar 2.1
Notasi *flow graph*



Untuk mengilustrasikan pemakaian *flow graph*, perhatikan contoh representasi rancangan prosedural pada gambar 2.2. Pada gambar tersebut *flowchart* digunakan untuk menggambarkan struktur kontrol program. Gambar 2.3 memetakan *flowchart* ke dalam *flow graph* yang terhubung (diasumsikan tidak ada kondisi majemuk pada *decision diamond* pada *flowchart*). Mengacu pada gambar 2.3, setiap lingkaran, disebut sebagai *flow graph node*, merepresentasikan satu atau beberapa *statement* prosedural. Urutan kotak proses dan *decision diamonds* dapat dipetakan menjadi satu *node*. Panah pada *flow graph*, disebut *edges* atau *links*, merepresentasikan alur kontrol, sama seperti panah pada *flowchart*. Sebuah *edge* harus berakhir pada *node*, walaupun *node* tersebut tidak merepresentasikan prosedur apapun. Area yang dibatasi oleh *edges* dan *nodes* disebut *regions*. Ketika menghitung jumlah *regions*, area luar *graph* dimasukkan sebagai *region*

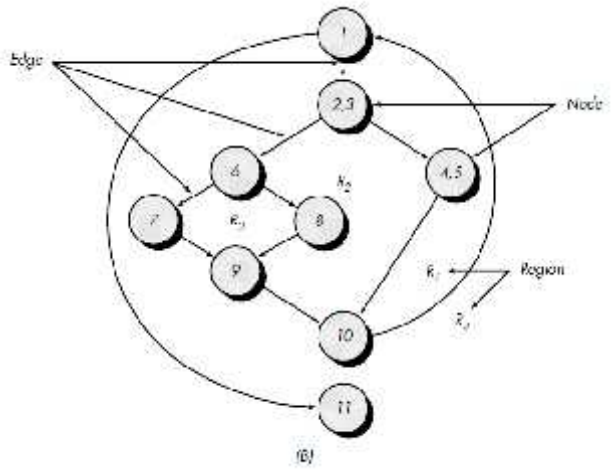
.

Gambar 2.2
Flowchart



Ketika kondisi majemuk ditemui dalam rancangan prosedural, pembentukan *flow graph* menjadi lebih rumit. Kondisi majemuk terjadi ketika satu atau beberapa operator boolean (OR, AND, NAND, NOR) disebutkan dalam *stament* kondisional.

Gambar 2.3
Flow Graph



Gambar 2.4
PDL dengan *Node* yang Teridentifikasi

```
PROCEDURE average;
   * This procedure computes the average of 100 or fewer
     numbers that lie between bounding values; it also computes the
     sum and the total number valid.

   INTERFACE RETURNS average, total.input, total.valid;
   INTERFACE ACCEPTS value, minimum, maximum;

   TYPE value[1:100] IS SCALAR ARRAY;
   TYPE average, total.input, total.valid;
      minimum, maximum, sum IS SCALAR;
   TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;                          2
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100         3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum     6
5        THEN increment total.valid by 1;
7              sum = sum + value[i]
         ELSE skip
8     ENDIF
      increment i by 1;
9  ENDDO
   IF total.valid > 0                                      10
11    THEN average = sum / total.valid;
12       ELSE average = -999;
13 ENDIF
END average
```

Berdasarkan gambar 2.5, segmen PDL (*Program Design Language* atau *pseudocode*) diterjemahkan ke dalam *flow graph* yang ditampilkan. Sebagai catatan, *node* yang terpisah dibuat untuk setiap kondisi a dan b pada *statement* IF a OR b. Setiap *node* yang mempunyai kondisi/persyaratan disebut *predicate node* dan dikarakteristikkan dengan dua atau beberapa *edges* yang keluar dari *node* tersebut.

c. Jalur Independen Program

Jalur independen adalah semua jalur yang melewati program yang menuju sekurangnya satu himpunan dari *statement* proses atau kondisi baru. Ketika diartikan dalam *flow graph*, sebuah jalur independen harus berpindah sekurangnya satu *edge* yang belum pernah dikunjungi sebelum jalur didefinisikan. Sebagai contoh, himpunan dari jalur independen pada *flow graph* pada gambar 2.3 adalah:

Path 1 : 1-11

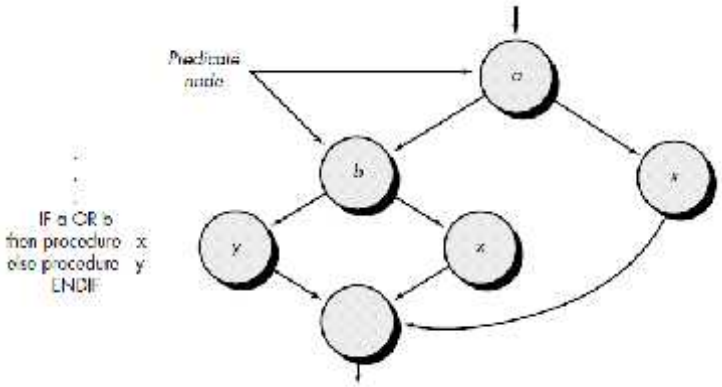
Path 2 : 1-2-3-4-5-10-1-11

Path 3 : 1-2-3-6-8-9-10-1-11

Path 4 : 1-2-3-6-7-9-10-1-11

Sebagai catatan, setiap jalur baru melewati *edge* baru.

Gambar 2.5
Compound Logic



Jalur 1, 2, 3, dan 4 merupakan himpunan basis untuk *flow graph* pada gambar 2.3. Jika pengujian dapat dirancang untuk mendorong eksekusi dari jalur ini (himpunan basis), maka setiap *statement* di dalam program dijamin akan dijalankan sekurangnya sekali, dan setiap kondisi akan dijalankan pada kondisi *true* atau *false*. Perlu diketahui bahwa himpunan basis tidaklah unik. Faktanya, sejumlah himpunan basis yang berbeda dapat diturunkan dari sebuah rancangan prosedural yang diberikan.

Perhitungan *cyclomatic complexity* bermanfaat untuk mengetahui jumlah jalur yang perlu dicari. *Cyclomatic complexity* adalah *metric* piranti lunak yang menyediakan ukuran kuantitatif dari kompleksitas logikal program. Ketika digunakan dalam konteks metode pengujian *basis path*, nilai yang dihitung untuk *cyclomatic complexity* menyatakan jumlah jalur independen pada himpunan basis program dan menyatakan batas atas untuk jumlah kasus pengujian yang harus dilakukan sehingga menjamin semua *statement* dijalankan sekurangnya sekali.

Cyclomatic complexity mempunyai fondasi dalam teori *graph* dan dapat dihitung dengan satu dari tiga cara:

d. Jumlah *region* sama dengan *cyclomatic complexity*.

e. *Cyclomatic complexity*, V(G), untuk sebuah *flow graph*, G, didefnisikan sebagai:

$V(G) = E - N + 2$

E adalah jumlah *edge* pada *flow graph*, dan N adalah jumlah *node* pada *flow graph*.

f. *Cyclomatic complexity*, V(G),

untuk *flow graph*, G, juga didefinisikan sebagai:

V(G) = P + 1

P adalah jumlah *predicate nodes* yang terdapat pada *flow graph* G.

Berdasarkan gambar 2.3, *cyclomatic complexity* dapat dihitung dengan menggunakan setiap algoritma yang diberikan:

g. *Flow graph* mempunyai empat *regions*.

h. V(G) = 11 *edges* – 9 *nodes* + 2 = 4.

i. V(G) = 3 *predicate nodes* + 1 = 4.

j. Membuat *Test Case*

Metode pengujian *basis path* dapat diterapkan pada rancangan prosedural atau *source code*. Langkah-langkah berikut digunakan untuk membuat himpunan basis:

k. Dengan menggunakan rancangan atau *code* sebagai fondasi, gambarkan *flow graph* yang bersesuaian.

l. Menentukan *cyclomatic complexity* dari hasil *flow graph*.

m. Menentukan himpunan basis dari jalur independen yang linier.

n. Mempersiapkan *test case* yang akan mengeksekusi setiap jalur pada himpunan basis.

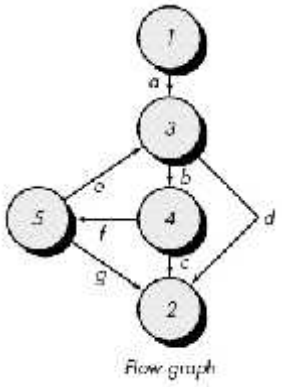
Penting untuk diketahui bahwa beberapa jalur independen tidak dapat diuji secara tersendiri. Penggabungan data dibutuhkan untuk menjelajahi jalur yang tidak dapat dicapai dengan alur biasa dari program. Pada beberapa kasus, jalur ini diuji sebagai bagian dari pengujian jalur lain.

o. Graph Matrix

Prosedur untuk membuat *flow graph* dan menentukan himpunan *basis path* dapat diterima berdasarkan mekanisme. Untuk mengembangkan alat piranti lunak yang membantu pengujian *basis path*, sebuah struktur data yang disebut *graph matrix*, dapat sangat bermanfaat.

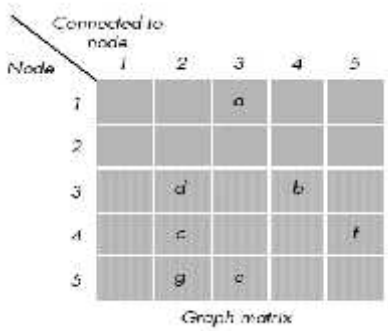
Graph matrix adalah matriks kotak yang ukurannya (jumlah baris dan kolom) sama untuk jumlah *node* pada *flow graph*. Setiap baris dan kolom berhubungan dengan *node* yang teridentifikasi, dan data matriks berhubungan dengan koneksi (*edge*) antara *node*.

Gambar 2.6
Contoh *Flow Graph*



Berdasarkan gambar 2.6, setiap *node* pada *flow graph* diidentifikasi berdasarkan angka, sedangkan pada gambar 2.7, setiap *edge* diidentifikasi berdasarkan huruf. Masukan huruf pada matriks, dibuat berdasarkan pasangan antara dua *node*. Sebagai contoh, *node* 3 dihubungkan dengan *node* 4 oleh *edge* b.

Gambar 2.7
Contoh *Graph Matrix*



Pada tahap ini, *graph matrix* tidak lain adalah representasi tabular *flow graph*. Dengan menambahkan *link weight* pada setiap data matriks, *graph matrix* dapat menjadi alat yang andal untuk mengevaluasi struktur kontrol program selama pengujian. *Link weight* memberikan informasi tambahan mengenai alur kontrol. Pada bentuk sederhana, *link weight* bernilai 1 (ada hubungan) atau 0 (tidak ada hubungan). Tetapi *link weight* dapat bekerja lain, seperti pada properti berikut:

a) Kemungkinan bahwa sebuah *link* (*edge*) akan dijalankan.
b) Waktu pemrosesan yang dikerjakan selama *traversal* sebuah *link*.
c) *Memory* yang dibutuhkan selama *traversal* sebuah *link*.
d) *Resource* yang dibutuhkan selama *traversal* sebuah *link*.
e) Control Structure Testing
f) *Basis path* yang dibahas sebelumnya adalah salah satu cara untuk pengujian *control structure*. Walaupun pengujian *basis path* sederhana dan efektif, namun belum cukup. Oleh karena itu, cakupan pengujian perlu diperluas agar meningkatkan kualitas pengujian *white-box*.
g) Condition testing
h) *Condition testing* adalah metode perancangan *test case* yang menguji kondisi logikal yang terdapat dalam modul program. Kondisi sederhana adalah variabel *boolean* atau ekspresi relasional, yang mungkin diawali dengan satu operator NOT (¬).
i) Data flow testing
j) Metode pengujian *data flow* memilih jalur pengujian program berdasarkan lokasi dari definisi dan penggunaan variabel pada program.
k) Loop testing
l) *Loop testing* adalah salah satu cara pengujian *white-box* yang berfokus pada kesesuaian implementasi *loop*. Empat kelas berbeda dari *loop* bisa didefinisikan: *simple loops*, *concatenated loops*, *nested loops*, dan *unstructured loops*.
m) Menurut Galin (2004:197), kelebihan *white-box testing* adalah:
n) Pemeriksaan langsung *statement* per *statement* kode program yang dapat menentukan kebenaran program seperti yang dituliskan pada jalur proses, termasuk apakah algoritma didefinisikan dan di-*coding* dengan benar.
o) Meningkatkan kinerja pelaksanaan *line coverage* (menerapkan paket piranti lunak khusus) yang menyediakan penguji dengan sejumlah daftar baris kode yang belum pernah dieksekusi. Penguji kemudian dapat membuat *test case* untuk mencakup baris kode tersebut.
p) Memastikan kualitas dari hasil *coding* dan sesuai dengan standar *coding*.

q) Kekurangan *white-box testing* adalah:
r) Menggunakan sumber daya lebih banyak dibanding *black-box testing* pada pengujian program yang sama.
s) Tidak dapat menguji kinerja piranti lunak dalam arti, ketersediaan (waktu respon), keandalalan, daya tahan

beban, dan kelas pengujian lain yang berhubungan dengan operasi, revisi, dan faktor transisi.

Karakteristik *white-box testing* adalah membatasi penggunaannya pada modul piranti lunak yang mempunyai resiko sangat tinggi dan rawan kesalahan, sehingga menjadi sangat penting untuk diidentifikasi dan membenarkan sebanyak mungkin *errors* tersebut.

Online Judge

Menurut Wikipedia, *online judge* adalah sistem *online* untuk menguji program dalam lomba pemrograman. *Online judge* juga digunakan untuk berlatih menghadapi perlombaan.

Sistem ini dapat mengkompilasi dan mengeksekusi kode program, dan mengujinya dengan data yang telah dibuat. Kode yang terkirim dapat dijalankan dengan pembatasan, termasuk pembatasan waktu, pembatasan memori, pembatasan keamanan, dan sebagainya. Keluaran kode akan ditangkap oleh sistem dan dibandingkan dengan keluaran standar. Sistem kemudian akan memberikan hasil penilaian. Jika kesalahan ditemukan pada standar keluaran, maka penilaian kembali menggunakan metode yang sama harus dilakukan.

2.2. Tinjauan Organisasi/Objek Penelitian

IBII pada awalnya adalah singkatan nama yayasan, yaitu Institut Bisnis Indonesia, yang mengelola lembaga pendidikan di bidang bisnis. Lembaga yang didirikan pada tahun 1987 ini menyelenggarakan program pendidikan setara S1 dengan gelar BBA (Bachelor of Business Administration). Para pendiri yayasan yang juga penyelenggara pendidikan ini, Kwik Kian Gie dan praktisi-praktisi bisnis yang berprestasi dalam bidangnya, yaitu, yaitu Kaharudin Ongko dan Djoenaedi Joesoef, tidak hanya mencerminkan tapi juga menyakinkan masyarakat bahwa seluk beluk bisnis adalah fokus utama pendidikan di IBII.

Pada tahun 1993 status IBII berubah menjadi Sekolah Tinggi Ilmu Ekonomi (STIE). STIE IBII menyelenggarakan pendidikan jenjang S1 (program sarjana) yaitu Program Studi Manajemen dan Program Studi Akuntansi. Pada Program Studi Manajemen terdapat konsentrasi: Manajemen Keuangan, Manajemen Pemasaran, dan Manajemen Kewirausahaan. Pada Program Studi Akuntansi terdapat konsentrasi: Akuntansi Manajeman, Pemeriksaan Akuntansi (Auditing), dan Akuntansi Perpajakan. Mulai tahun ini pula STIE IBII menyelenggarakan pendidikan jenjang S2 (program magister) dengan membuka Program Studi Magister Manajemen konsentrasi Manajemen Keuangan dan Manajemen Pemasaran.

Mulai tahun 2004 STIE IBII melengkapi pelayanannya dengan membuka pendidikan S3 (program doktor) ilmu manajemen dengan konsentrasi: Manajemen Keuangan, Manajemen Pemasaran, Manajemen Strategik, dan Akuntansi Manajemen.

Sejalan dengan tuntutan perkembangan zaman, pada tahun 2005 STIE IBII berubah menjadi Institut Bisnis dan Informatika Indonesia (IBII) dengan menambah empat program studi baru jenjang S1 yaitu: Sistem Informasi (SI), Teknik Informatika (TI), Ilmu Komunikasi (IlKom), dan Ilmu Administrasi Bisnis (IAB). Selain itu, Program Studi Manajemen IBII juga membuka konsentrasi baru yaitu Manajemen Kewirausahaan. Selain menambah empat program studi baru dan menambah satu konsentrasi di bidang Manajemen untuk program studi jenjang S1, pada tahun itu juga IBII membuka penyelenggaraan pendidikan jenjang S2 untuk Program Studi Magister Akuntansi dengan konsentrasi Jasa Keuangan Internal dan Jasa Keuangan Eksternal.

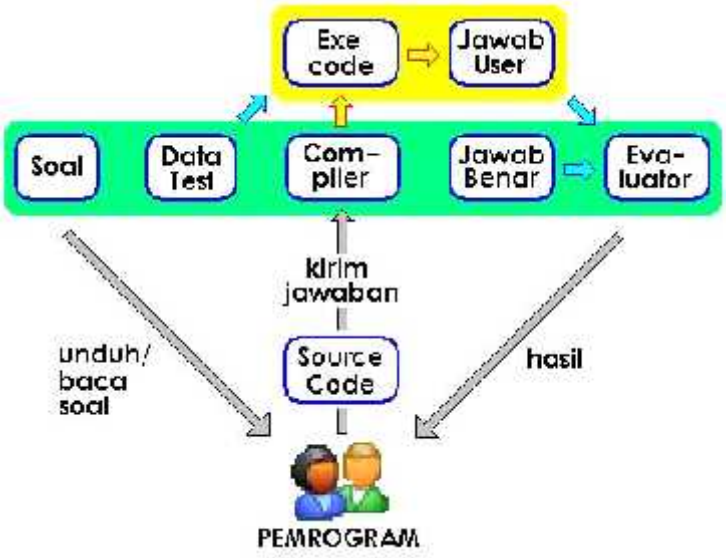
Untuk menunjang kegiatan belajar mahasiswa jurusan informatika, IBII menyediakan fasilitas IBII Online Judge. Dilatarbelakangi oleh karena dosen

pengajar mempunyai keterbatasan waktu untuk memeriksa kebenaran program komputer karya mahasiswa, padahal penguasaan mahasiswa terhadap mata kuliah algoritma, struktur data, dan pemrograman sangat ditentukan oleh frekuensi mahasiswa tersebut menulis program, IBII Online Judge dibuat untuk menggantikan peran dosen di dalam memeriksa kebenaran program komputer yang ditulis mahasiswa. Dengan demikian dapat memberi kesempatan kepada mahasiswa untuk melatih diri menjadi seorang pemrogram yang andal.

Pada *Online judge* tersedia:

a) kumpulan soal.
b) *data test* untuk masing-masing soal.
c) jawaban untuk masing-masing soal.

Gambar 3.1
Proses Kerja IBII Online Judge



Sumber: http://ti.ibii.ac.id/fasilitas/online-judge

Mahasiswa yang mempunyai hak akses menggunakan situs *online judge* dengan:

a) membaca soal.
b) membuat jawaban (*source code* program).
c) mengirim *source code* program kepada *online judge.*
d) menerima hasil evaluasi on-line judge dalam hitungan detik.

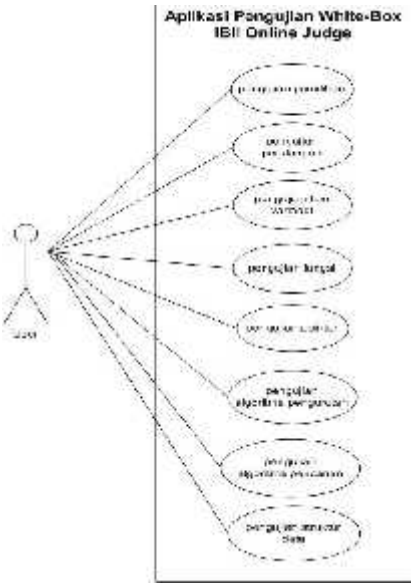
Oleh *online judge source code* program mahasiswa tersebut di-*compile*. Apabila tidak terjadi kesalahan sintaks maka akan terbentuk *executable code* (kode bahasa mesin). Kode bahasa mesin dijalankan (*run*) dengan mengggunakan data masukan yang telah disiapkan. Jawaban program mahasiswa dibandingkan dengan jawaban yang benar. Hasil perbandingkan ini dilaporkan kepada mahasiswa. Dengan cara ini mahasiswa dapat melakukan perbaikan program jika masih salah dan mengirim ulang *source code* kepada *online judge* untuk dievaluasi.

4. HASIL DAN PEMBAHASAN

Berikut ini adalah rancangan Aplikasi Pengujian White-Box IBII Online Judge

A. Use Case Diagram

Aplikasi Pengujian White-Box IBII Online Judge



Gambar 4.1 *Use Case Diagram*

Activity diagram di atas menggambarkan proses untuk melakukan pengujian. Pertama pengguna harus memasukkan lokasi berkas kode program. Kemudian pengguna menentukan aturan pengujian pada salah satu kategori. Aturan pengujian dapat ditambahkan dari kategori yang berbeda. Setelah penentuan aturan selesai terdapat pilihan untuk menyimpan aturan yang telah dibuat. Setelah itu dapat dimulai proses eksekusi untuk memulai pengujian kode program. Jika tidak ada data yang ingin diuji, pengguna dapat keluar melalui menu yang di sediakan.

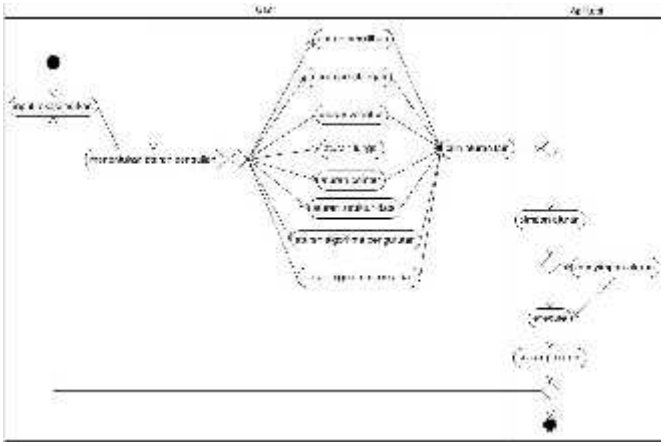
4.3. Implementasi Sistem

Berikut ini adalah penjelasan mengenai Aplikasi Pengujian White-Box IBII Online Judge yang mencakup aplikasi program yang dihasilkan, panduan instalasi, panduan Pada tampilan utama aplikasi merangkum semua fungsi untuk pengujian *white-box*. Terdapat *textbox* untuk memasukkan lokasi berkas kode program yang akan diuji, *menu*

4.1. Rancangan Sistem

berbasis Java dengan menggunakan bantuan diagram *Unified Modelling Language* (UML).

B.***Activity Diagram***



Gambar 4.2. *Activity Diagram* Aplikasi Pengujian White-Box IBII Online Judge

pemakaian, spesifikasi program aplikasi, dan evaluasi. Aplikasi Komputer yang Dihasilkan

Berikut adalah tampilan layar dari Aplikasi Pengujian White-Box IBII Online Judge beserta penjelasan singkat mengenai beberapa fiturnya:

Gambar 4.3. Tampilan Layar Menu Awal

tab untuk mengatur aturan-aturan pengujian, dan hasil pengujian akan ditampilkan di label *result* di pojok kanan bawah.

Gambar 4.4 Tampilan Layar Menu *File*

Gambar 4.5 Tampilan Layar Menu *About*

Pada menu *file*, terdapat *load rules* berfungsi untuk membaca berkas aturan, *save rules* berfungsi untuk menyimpan berkas aturan, *reset form* berfungsi untuk mengulang pengisian *form*, dan *exit* berfungsi untuk keluar dari aplikasi. Pada menu *about*, terdapat informasi pembuat aplikasi.

Tombol *run batch* digunakan untuk menampilkan kotak dialog yang membaca lokasi *folder* berkas kode program, baturan, Pada *tab algorithm* berfungsi untuk melakuk pengujian algoritma pengurutan dan pencarian yang digunakan. Algoritma pengurutan yang dapat diuji antara lain *bubble sort*, *selection sort*, *insertion sort*, *quicksort*, *mergesort*. Pada *tab data structure* berfungsi untuk melakukan pengujian struktur data. Pengujian yang dapat dilakukan antara lain *linked list* (*singly*, *doubly*, *circular*), *stack* representasi

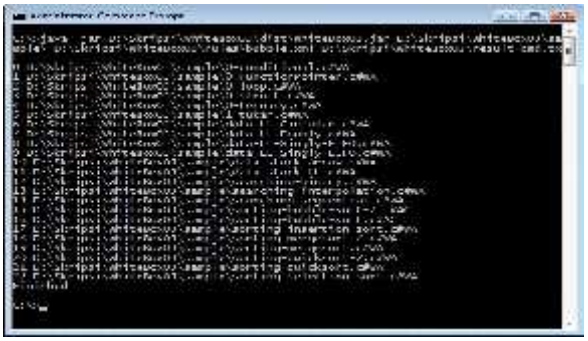
Gambar 4.7. Tampilan Layar *Batch Process Command Line*

dan berkas keluaran hasil pengujian. Hasil pengujian akan ditulis ke berkas keluaran hasil pengujian yang telah ditentukan.

Gambar 4.6 Tampilan Layar Menu *Tab Algorithm*

Algoritma pencarian yang dapat diuji antara lain *sequential search*, *binary search*,dan *interpolation search*.

list dan representasi larik, *queue* representasi *list* dan representasi larik, dan *tree* (*binary tree*, *red black tree*, *AVL tree*).



Berikut adalah contoh tampilan pengujian banyak berkas melalui *command line.* Aplikasi menerima tiga parameter yaitu lokasi berkas pengujian, lokasi berkas aturan, dan lokasi berkas keluaran hasil pengujian.

5.SIMPULAN

Berdasarkan penelitian dan pengembangan Aplikasi Pengujian White-Box IBII Online Judge, maka dapat ditarik simpulan bahwa aplikasi pengujian *white-box* IBII Online Judge dapat melakukan pengujian pada jawaban kode program pengguna terhadap persyaratan soal dalam penggunaan algoritma dan strukur data. Pengujian yang dapat dilakukan antara lain:

1. Aplikasi pengujian *white-box* dapat menguji pembatasan jumlah *if, switch, ternary conditional*.
2. Aplikasi pengujian *white-box* dapat menguji pembatasan jumlah *for*, *do-while*, *while*.
3. Aplikasi pengujian *white-box* dapat menguji pembatasan penggunaan tipe variabel, *struct*, larik.
4. Aplikasi pengujian *white-box* dapat menguji pembatasan penggunaan *function return type*, *parameter by value*, *parameter by reference*.
5. Aplikasi pengujian *white-box* dapat menguji pembatasan penggunaan *pointer*, alokasi memori secara dinamik, *function pointer*.
6. Aplikasi pengujian *white-box* dapat menguji penggunaan algoritma pengurutan: *bubble sort*, *selection sort*, *insertion sort*, *quicksort*, dan *mergesort*.
7. Aplikasi pengujian *white-box* dapat menguji penggunaan algoritma pencarian: *sequential search*, *binary search*, dan *interpolation search*.
8. Aplikasi pengujian *white-box* dapat menguji penggunaan *linked list*: *singly-linked list*, *doubly-linked list*, *circular list*.
9. Aplikasi pengujian *white-box* dapat menguji penggunaan struktur data *stack*: representasi dengan larik, representasi dengan *list*.
10. Aplikasi pengujian *white-box* dapat menguji penggunaan struktur data *queue*: representasi dengan larik, representasi dengan *list*.
11. Aplikasi pengujian *white-box* dapat menguji penggunaan struktur data *tree*: *binary search tree*, *AVL tree*, dan *red black tree*.

6.REKOMENDASI

Penulis memberikan saran untuk pengembangan pada kemudian hari berdasarkan hasil penelitan yang telah dilakukan:

1. Aplikasi pengujian dapat ditambahkan fasilitas analisis *static checking*, pengujian plagiasi, bobot penilaian (*scoring*) untuk jawaban yang mendekati benar, dan laporan kesalahan.
2. Aplikasi pengujian dapat ditambahkan aturan pengujian algoritma dan struktur data di luar materi bahasan.
3. Aplikasi pengujian dapat ditambahkan pemeriksaan rekursi, *infinite loop*, *typedef*, *preprocessor*, dan operator *bitwise*.
4. Pengembangan modul pengujian *white-box* karena masih menggunakan pencarian pola secara *brute-force* dan terdapat keterbatasan pola yang dapat dikenali.
5. Proses *tokenizing* dapat dioptimalkan karena standar *class* StreamTokenizer yang digunakan masih kurang akurat dalam melakukan *tokenizing* terhadap kode program.
6. Pengembangan untuk pengujian bahasa pemrograman lain selain bahasa C

7.DAFTAR PUSTAKA

[1] Deitel, P. J., Deitel, H. M. (2007), *C How To Program*, Edisi ke-5, New Jersey: Pearson Education Inc.

[2] Galin, Daniel, (2004), *Software Quality Assurance: From Theory to Implementation*, Pearson Education Inc.

[3] Kendall, Kenneth E., Julie E. Kendall (2010), *Systems Analysis and Design*, Edisi ke-8, New Jersey: Pearson Education, Inc.

[4] Knuth, Donald E. (1997), *The Art of Computer Programming vol 1/ Fundamental Algorithms*, Edisi ke-3, Addison Wesley.
[5] Knuth, Donald E. (1998), *The Art of Computer Programming vol 3/ Sorting and Searching*, Edisi ke-2, Addison Wesley.
[6] Levitin, Anany (2003), *Introduction to The Design & Analysis of Algorithm*, Addison Wesley.
[7] McCabe, T., "A Software Complexity Measure," IEEE Trans. Software Engineering, vol. SE-2, Desember 1976, halaman 308-320.
[8] Myers, Glenford J., Corey Sandler, dan Tom Badgett (2011), *The Art of Software Testing*, Edisi ke-3, John Wiley & Sons.
[9] Pressman, Roger S. (2010), *Software Engineering: A Practitioner's Approach*, Edisi ke-7, Boston: McGraw Hill.
[10] Sebesta, Robert W. (2003), *Concepts of Programming Languages*, Edisi ke-6, Boston: Addison Wesley.
[11] Sedgewick, Robert (1998), *Algorithm in C Parts 1-4. Fundamentals, data structures, sorting, searching*, Edisi ke-3, Addison-Wesley.
[12] Weiss, Mark Allen (1997), *Data Strucure And Algorithm Analysis In C*, Edisi ke-2, Addison Wesley Longman.
[13] _____ (2011), *Online Judge*, sumber: http://en.wikipedia.org/wiki/Online_judge (diakses 15 Mei 2011).
_____ (2009), *Online Judge*, sumber: http://si.ibii.ac.id/fasilitas/76-online-judge/105-online-judge (diakses 28 September 2011).