Dimas Angga Saputra 13510046
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13510046@std.stei.itb.ac.id

***Abstract*— makalah ini membahas bagaimana teori pohon diimplementasikan menjadi sebuah struktur data dalam pemrograman, studi kasus pada makalah ini adalah permainan otak *rush hour*.**

***Index Terms*—pohon, struktur data, rush hour, if-then-else.**

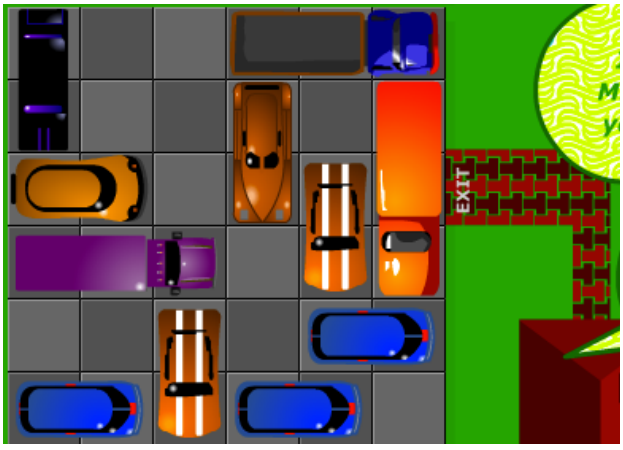
I. PENDAHULUAN

1.1 Permainan *Rush Hour*

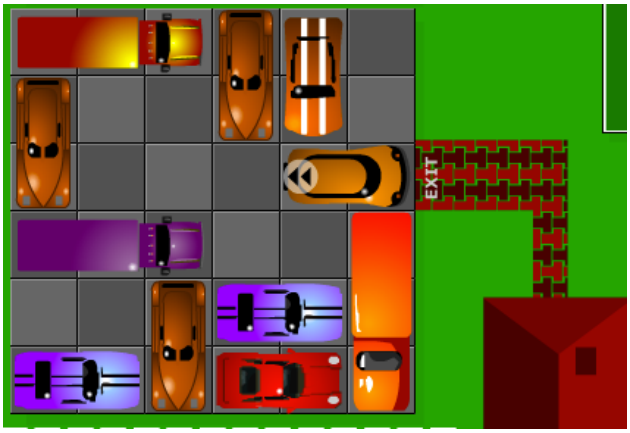
Rush Hour merupakan sebuah *game* otak (*brain game*) di mana Pemain harus mengeluarkan mobil pemain pada jam-jam pulang kantor (*rush hour*) dari tempat parkir yang penuh dengan mobil yang diparkir secara tidak teratur (melintang, lurus, dan sebagainya). Pintu keluar hanya satu dan pemain berlagak seperti tukang parkir dengan mendorong mobil-mobil maju atau mundur sesuai arahnya. Pemain tidak boleh mengubah arah mobil lain. *Rush Hour* adalah teka-teki blok geser (*sliding bliock*) yang diciptakan oleh Nob Yoshigahara di akhir 1970-an dan pertama dijual di Amerika Serikat pada tahun 1996. Area parkir berukuran grid $m \times n$, dan setiap mobil bisa memakai (*occupy*) minimal 2 buah *grid*..

Secara umum *game* ini adalah *game puzzle* yang menggunakan keputusan-keputusan dari pemain sabagai faktor utama penentu kemenangan. Karenanya, memetakan keputusan yang diambil baik sudah diambil, akan diambil, atau tidak jadi diambil sangat penting dalam *game* ini.

Jika diilustrasikan maka *initial state* dan *final state* adalah :



Gambar *initial state* permainan, tujuan permainan adalah mengeluarkan mobil sedan jingga yang sejajar dengan jalur *exit* melewati garis panah dengan memindah-mindahkan mobil-mobil lain sesuai orientasi masing-masing mobil.



Gambar *final state* permainan, terlihat bahwa objektif telah berhasil dilakukan dan pemain dapat melanjutkan ke tingkat kesulitan yang lebih tinggi.

1.2 Teori Pohon

Dalam konteks ini, pohon adalah graf tak berarah, terhubung, yang tidak memiliki sirkuit.
Secara umum teorema mengenai definisi pohon adalah :

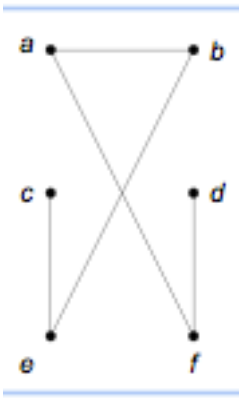
Teorema : Misalkan $G = (V, E)$ adalah graf tak-berarah sederhana dan jumlah simpulnya n. Maka, semua pernyataan di bawah ini adalah ekivalen:

1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.

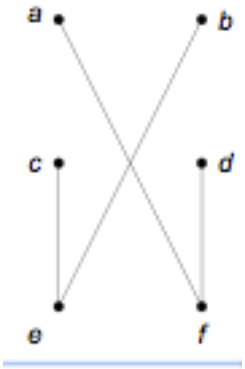
3. *G* terhubung dan memiliki $m = n - 1$ buah sisi.
4. *G* tidak mengandung sirkuit dan memiliki $m = n - 1$ buah sisi.
5. *G* tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.

G terhubung dan semua sisinya adalah jembatan.

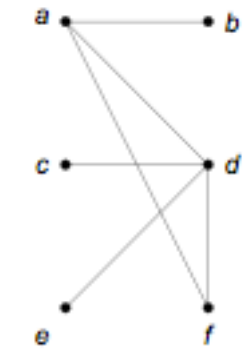
Teorema tersebut juga merupakan definisi lain pohon. Ilustrasi contoh pohon dan bukan pohon adalah :



i.Pohon (terhubung, tidak ada sirkuit)



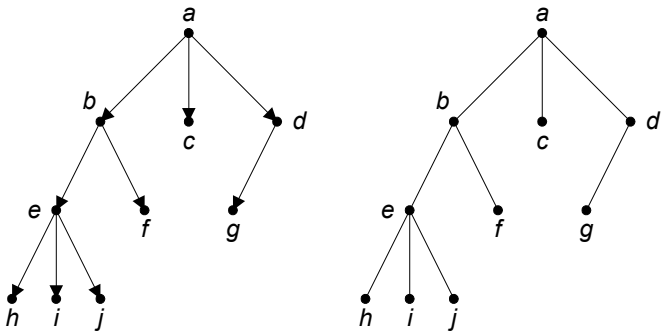
ii. Bukan pohon, ada bagian yang tidak terhubung



iii. Bukan pohon, ada sirkuit a-d-f-a

Dalam teori ini, ada pula pohon berakar yang memiliki definisi :

> Pohon yang satu buah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah dinamakan **pohon berakar** (*rooted tree*).



Kiri : Pohon berakar
Kanan : Sesuai perjanjian, arah dalam garis pohon boleh dibuang.

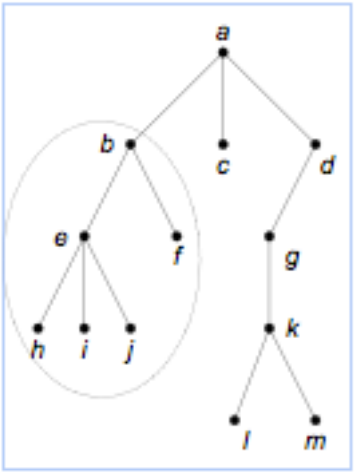
Terminologi dalam pohon berakar :

1. *Child* (anak) dan *Parent* (Orang tua)
   Dalam contoh pohon berakar diatas, b adalah *child* dari a, dan a adalah *parent* dari b. begitupun h adalah *child* dari e, dan e adalah *parent* dari h.

2. *Path* (lintasan)
   Dalam contoh pohon berakar diatas, lintasan dari a ke i adalah a – b – e – i . dengan panjang lintasan adalah 3.

3. *Sibling* (saudara sekandung)
   Dalam contoh pohon berakar diatas, e adalah *sibling* f. namun e bukan *sibling* g.

4. *Sub-Tree* (upa pohon)
   Jika dicontohkan dengan gambar :



Bisa ditarik kesimpulan *sub-tree* adalah pohon didalam pohon atau pohon bagian yang lebih kecil didalam pohon yang lebih besar

5. *Degree* (derajat)
   Dalam contoh pohon berakar diatas, derajat a adalah 3, derajat b adalah 2. Bisa ditarik kesimpulan, derajat adalah jumlah anak dihitung dari simpul yang dimaksud.

6. *Leaf* (daun)
   Merupakan simpul-simpul yang derajatnya 0, atau tidak memiliki *child.* Dalam contoh pohon berakar diatas, *leaf* adalah c,f , g, h ,i, j

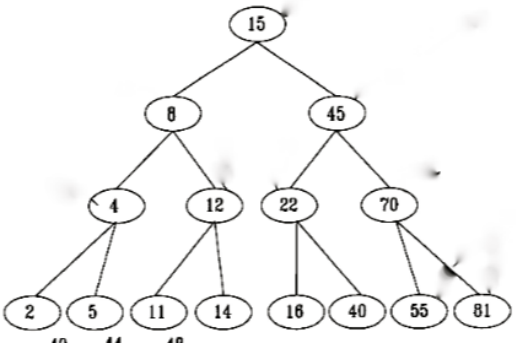
7. *Internal nodes* (simpul dalam)
   Simpul yang mempunyai anak adalah simpul dalam. Dalam contoh pohon berakar diatas, b, d ,e adalah *internal nodes.*

8. *Depth* (kedalaman)
   Merupakan jumlah tingkat dengan akar adalah tingkat 0, jika memiliki satu anak, kedalaman bertambah 1. Dalam contoh pohon berakar diatas, kedalamannya adalah 3.

II. POHON PADA PERMAINAN RUSH HOUR

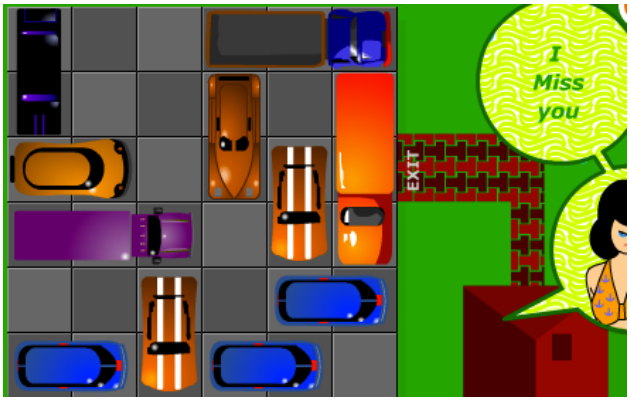
Dalam permainan Rush Hour yang berfokus pada keputusan dan solusi, pohon bisa digunakan sebagai representasi sturuktur data dari kemungkinan-kemungkinan solusi yang mungkin dilakukan.

Jika diilustrasikan dengan gambar maka pohon akan menjadi seperti :

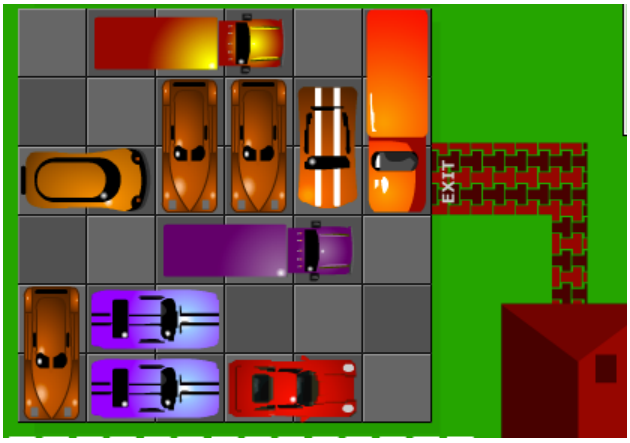


Dimana 15 merepresentasikan *initial-state.* Dan setiap daun merepresentasikan *final state* dengan dua kemungkinan, permainan berhasil atau gagal diselesaikan.

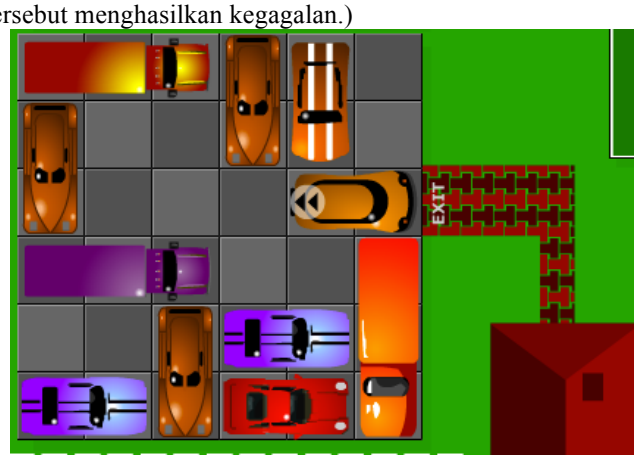
Setiap *node* pada pohon merepresntasikan langkah yang diambil misal :



Adalah *node* 15



Sementara gambar ini adalah *node* 14 (misal *node* tersebut menghasilkan kegagalan.)

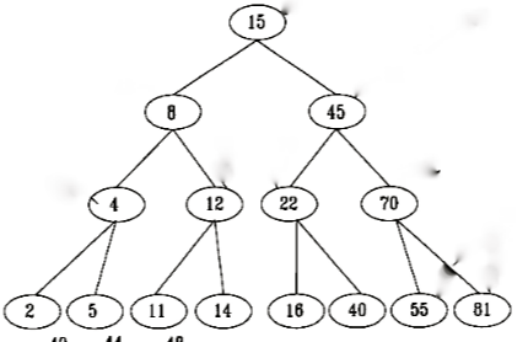


Adalah *node* 40 (misal *node* tersebut menghasilkan keberhasilan.

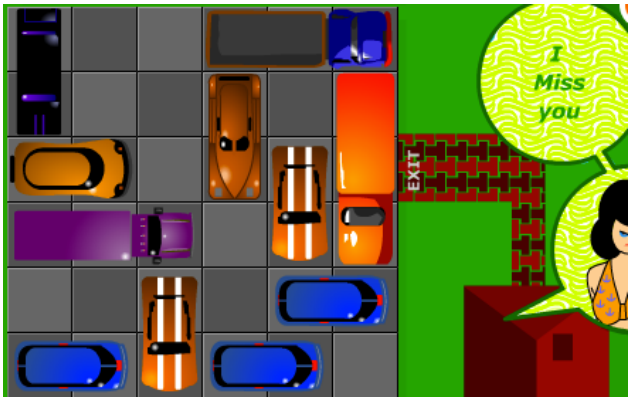
Dalam implementasinya, bisa saja *node* 14, 15 atau lebih (lebih dari satu *node*) menghasilkan kegagalan, baik dengan kondisi yang sama, ataupun berbeda. Begitupun isi 2 *node* yang berbeda mungkin sama namun cara mencapainya berbeda.

III. ANALISIS DAN PEMBAHASAN

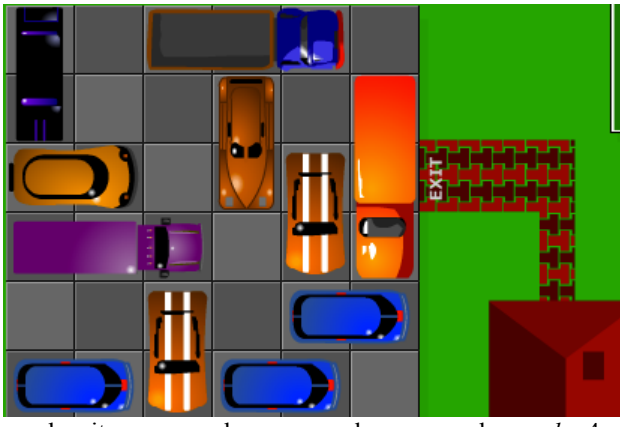
Dalam pemrograman, metode pencarian solusi dengan data struktur *tree* sangatlah berguna, jika dijabarkan langkah-langkah pembangunan algoritma pencarian solusi tersebut dengan ilustrasi, maka bisa dicontohkan dengan :



missal pohon ini adalah pohon solusi,

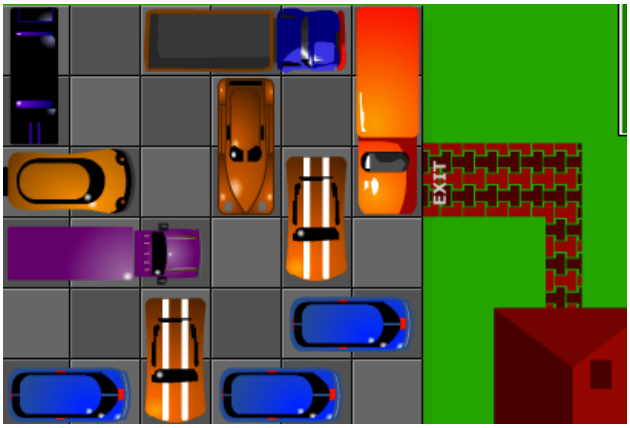


State ini adalah *node* 15 sebagai permulaan.
setelah itu algoritma akan mencoba *node* 8 yang missal berisi solusi :

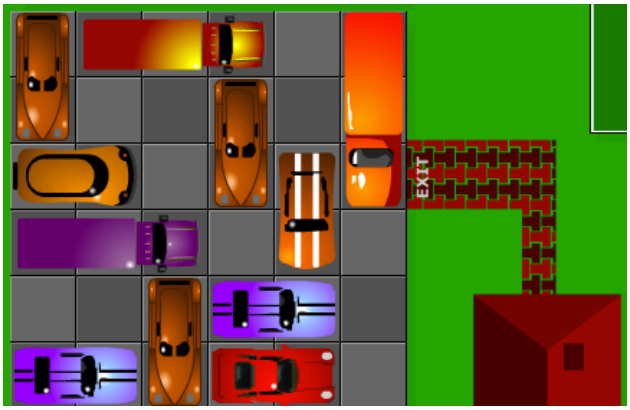


Dan algoritma mencoba meneruskan mencoba *node* 4 yang missal berisi solusi :

s

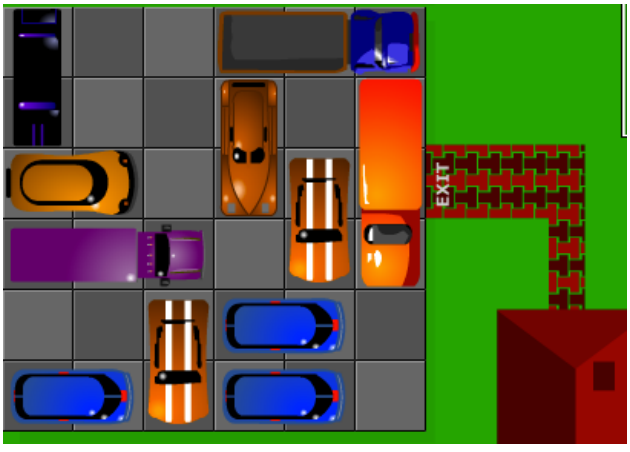


Setelah itu asumsikan *node* 2 adalah *node* akhir (karena sebenarnya kemungkinan solusinya sangat banyak, bisa mencapai ribuan, untuk mempersingkat, digunakan asumsi dan permisalan) dan *node* 2 menemukan kegagalan dengan solusi :

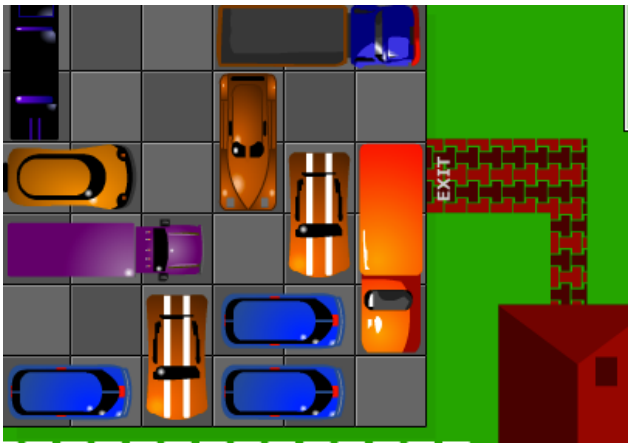


Maka algoritma akan mencoba solusi di *node* 5, jika masih ditemui kegagalan, algoritma akan mencoba secara berurutan *node* 12 – 11 – 14 (karena *node* 4 sudah lebih dulu dicoba) dengan asumsi ketika missal solusi ditemukan di *node* 11, maka pencarian dihentikan untuk menghemat waktu.

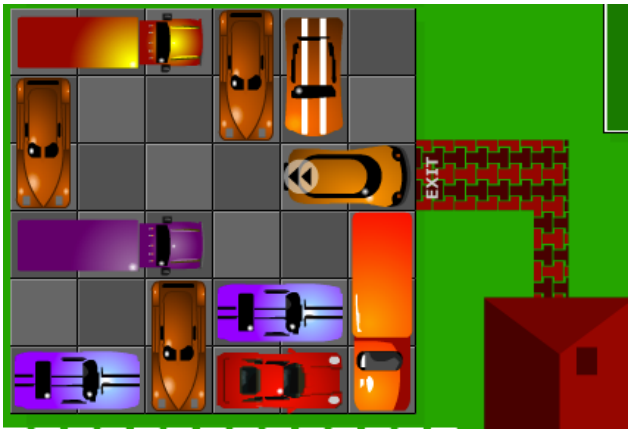
Ketika algoritma belum menemukan solusi di tingkat tersebut, maka akan dicari ke tingkat yang lebih atas yaitu *node* 45 yang missal berisi solusi :



dilanjutkan mencoba solusi yang tersimpan di *node* 22 yang misal berisi :



Asumsikan beberapa *node* setelah *node* 22 (dengan urutan pencarian seperti yang dilakukan sebelumnya) solusi ditemukan :



Pencarian pun dihentikan dengan mengeluarkan solusi posisi-posisi mobil.

Ada pula kemungkinan ketika seluruh solusi sudah dipetakan, namun tidak ada satupun *node* yang berisi keberhasilan maka algoritma akan memberitahukan solusi tidak ditemukan.

Singkatnya, metode pencarian solusi di pohon adalah :

```
Begin
    Mulai /* mengecek state akar */
    while (solusi belum ditemukan or belum semua
    daun dicek)
        while (belum mencapai daun)
            Cek anak terkiri
        endwhile
        if (solusi ditemukan) then
            stop
        else
            cek sibling
        endif

        if (solusi tidak ditemukan di sibling)
        then
            cek sibling parent
        else
            stop
        endif
    endwhile
    End.
```

Jadi algoritma pertama mengecek terus anak terkiri sampai mencapai daun, ketika solusi tidak ditemukan, akan dicek *sibling* dari daun tersebut, ketika tidak ditemukan juga, akan dicek *sibling* dari *parent* dari daun tersebut sampai ke anak-anaknya, jika masih tidak ditemukan akan naik ke *sibling* dari *parent* ke tingkat yang lebih atas.

Dengan algoritma tersebut urutan pencarian dari pohon contoh adalah :

15 – 8 – 4 – 2 – 5 -12- 11 – 14 -45 – 22 – 16 – 40 – 70 – 55 – 81

Dengan asumsi solusi ditemukan di *node* 81 atau solusi tidak ditemukan setelah mengecek semua *node*.

IV. KESIMPULAN

Setelah melakukan studi pustaka maka kesimpulan yang dapat diambil adalah :

1. Konsep pohon dapat diimplementasikan menjadi sebuah struktur data
2. Dalam pemetaan solusi, pencarian dengan struktur data ini dapat membantu jika harus melakukan runut – balik
3. Implementasi konsep pohon dapat digunakan untuk pemecahan masalah yang berbasiskan keputusan atau solusi.
4. Konsep if-then-else sangat erat kaitannya dengan struktur data ini.

DAFTAR REFERENSI

[1]. Munir, Rinaldi. (2006). B ahan Kuliah IF2153 Matematika Diskrit. Departemen Teknik Informatika, Institut Teknologi Bandung.

[2]. http://www.agame.com/game/rush_hour_2.html Tanggal akses : selasa 18 desember 2012 pukul 18.05

[3]. http://www.freepatentsonline.com/6567815-0-large.jpg tanggal akses : selasa 18 desember 2012 pukul 19.24

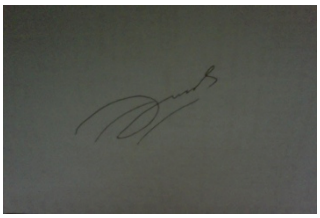
[4]. www.thinkfun.com/microsite/**rushhour**/nob tanggal akses : selasa 18 desember 2012 pukul 08.42

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, 18 Desember 2010

Ttd



Dimas Angga Saputra
13510046