ANALISIS PENERAPAN ALGORITMA RUNUT-BALIK DALAM PENCARIAN SOLUSI PERSOALAN "LOMPATAN KUDA"

R. Raka Angling Dipura (13505056)

Jurusan Teknik Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
E-mail: if15056@students.if.itb.ac.id

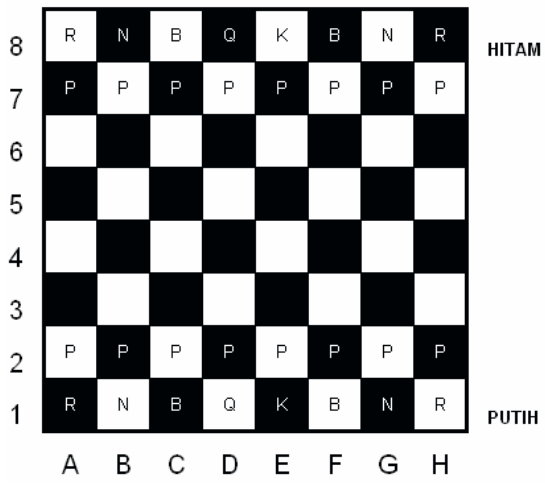
ABSTRAK

Algoritma runut-balik merupakan algoritma berbasis *DFS* yang penggunaannya lebih mangkus daripada algoritma *brute force*. Penggunaan algoritma runut-balik sangat luas, mulai dari permainan (seperi sudoku, *tic-tac-toe*, dan catur) hingga penelusuran graf. Dalam makalah ini dibahas pencarian solusi dari persoalan "lompatan kuda". Yaitu tantangan untuk menjelajahi tiap petak dalam setengah papan catur (4x8) tanpa menginjak petak yang sama lebih dari sekali.

Kata Kunci : algoritma, runut-balik, *brute force*, lompatan kuda, DFS.

1. PENDAHULUAN

Persoalan "lompatan kuda" merupakan kasus yang cukup erat dengan permainan catur. Dalam permainan catur, bidak-bidak disusun sedemikian rupa pada papan 8x8 sesuai dengan aturan. Sedangkan persoalan "lompatan kuda" hanya memanfaatkan sifat kuda dalam catur yang dapat berpindah sejauh 2 kolom dan 1 baris, atau 2 baris dan 1 kolom.

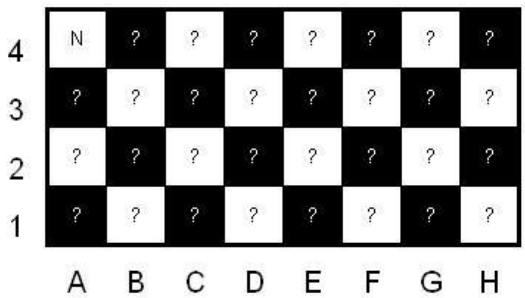


Gambar 1. Papan Catur

Keterangan :
P = *Pawn*, biasa disebut pion
R = *Rook*, biasa disebut benteng
N = *Knight*, biasa disebut kuda
B = *Bishop*, biasa disebut gajah atau peluncur
Q = *Queen*, biasa disebut menteri atau ratu
K = *King*, biasa disebut raja

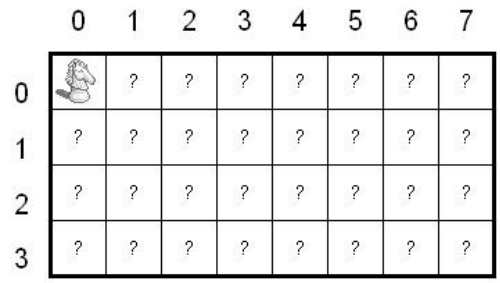
2. DESKRIPSI MASALAH

Dalam persoalan "lompatan kuda", papan catur yang digunakan hanya setengah dari papan catur normal (4x8). Sebuah kuda diletakkan di petak pojok kiri-atas, sementara bidak lainnya disusun acak di petak lainnya.



Gambar 2. Susunan "lompatan kuda"

Untuk mempermudah pencarian solusi, maka persoalan "lompatan kuda" akan diimplementasikan ke dalam matriks 4x8. Penelusuran dilakukan sampai didapatkan solusi pertama atau runut-balik telah habis (solusi tidak ditemukan).



Gambar 3. Matriks "lompatan kuda"

3. ALGORITMA RUNUT BALIK

Istilah runut-balik pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Selanjutnya R.J. Walker, Golomb,dan Baumert menyajikan uraian umum tentang runut-balik dan penerapannya pada berbagai persoalan.

Runut-balik (sering juga disebut *backtrack*) merupakan perbaikan dari algoritma *bruteforce*. Runut-balik menelusuri pohon solusi dengan prinsip DFS (*Depth First Search*).

Dalam pengambilan keputusan penelusuran, runut-balik hanya mempertimbangkan pilihan-pilihan yang mengarah kepada ditemukannya solusi. Hal ini merupakan keunggulan algoritma runut-balik atas algoritma *bruteforce*.

Ciri khas dari algoritma runut-balik adalah adanya penelusuran balik ke simpul (dalam representasi pohon) sebelumnya (apabila penelusuran saat itu mengalami kebuntuan) lalu melanjutkan penelusuran ke kemungkinan solusi lain.

4. PENCARIAN SOLUSI

Algoritma runut-balik sangat membantu dalam mencari solusi dari persoalan ”lompatan kuda” karena persoalan ”lompatan kuda” ini menuntut penelusuran yang sistematis.

Berikut ini uraian langkah yang dilakukan dalam algoritma runut balik:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Simpul yang sudah dilahirkan dinamakan simpul hidup
2. Jika lintasan yang diperoleh dari perluasan simpul hidup tidak mengarah ke solusi, maka simpul itu akan menjadi simpul mati dimana simpul itu tidak akan diperluas lagi
3. Jika posisi terakhir ada di simpul mati, maka dilakukan runut-balik ke simpul orang tua lalu simpul hidup yang baru dibangkitkan sebagai simpul anak
4. Pencarian dihentikan jika kita telah ditemukan solusi atau justru tidak ditemukan solusi (akar menjadi simpul mati)

Langkah-langkah di atas memanfaatkan pembangkitan pohon untuk menemukan solusi. Posisi awal kuda adalah akar dari pohon, sedangkan petak yang dipilih selanjutnya akan menjadi simpul anak. Penelusuran dilakukan secara DFS. Translasi langkah-langkah di atas sebagai petunjuk teknis untuk mencari solusi persoalan ”lompatan kuda” adalah sebagai berikut:

1. Kuda ada di petak (0, 0)
2. Kuda bergerak ke petak lain sesuai aturan langkah kuda (di petak yang dituju harus ada bidak lain untuk ”dimakan”)
3. Lakukan langkah 2 selama masih memungkinkan
4. Jika kuda kehabisan ruang gerak (misal pada petak B), lakukan penelusuran balik ke petak sebelumnya (misal petak A)
5. Hapus petak B dari himpunan kemungkinan gerakan kuda saat di kotak A
6. Ulangi langkah 3 sampai 5 hingga solusi ditemukan atau penelusuran kembali ke petak (0, 0) dan semua kemungkinan gerak pada petak (0, 0) telah habis

Langkah-langkah teknis tersebut kemudian menjadi dasar perancangan *pseudo-code* prosedur pencarian solusi persoalan ”lompatan kuda”.

```
procedure lompatanKuda
{ mencari solusi pertama untuk ”memakan”
  seluruh     bidak di papan catur 4x8 dengan
  sebuah kuda
}
Deklarasi
  M : matriks[4][8] integer {M adalah matriks
  yang memiliki 4 baris dan 8 kolom berisi
  integer}
  bar : integer
  kol : integer

Algoritma:
  for bar←0 to 3 do
    for kol←0 to 7 do
      M[bar][kol]←0
    endfor
  endfor
  M[0][0]←1
  while (solusi belum ditemukan) do
    if  (mungkinLompat(M)) then
      maju()
    endif
    else
      mundur()
    endif
  endwhile
```

Papan catur direpresentasikan sebagai matriks 4x8. Pertama, matriks M diisi dengan 0 (belum dijelajahi). Lalu M[0][0] diisi dengan 1, karena petak tersebut merupakan posisi awal kuda.

Selanjutnya diadakan pengulangan selama solusi belum ditemukan. Pemeriksaan dilakukan terhadap matriks M. Jika dari petak kuda berada memungkinkan kuda untuk bergerak, maka prosedur maju() akan ”menggerakkan” kuda dan memperbarui status matriks. Jika kuda kehabisan ruang gerak, dijalankan mundur() (runut-balik) untuk memeriksa kemungkinan solusi lain.

5. IMPLEMENTASI DALAM JAVA

Untuk membuktikan ketepatan *pseudo-code* yang telah dirancang, maka dibuatlah sebuah program dalam bahasa JAVA sebagai implementasi *pseudo-code*. Program ini terdiri dari 3 kelas, *stack*, *langkahKuda*, dan *utama*.

5.1 Kelas *langkahKuda*

Kelas *langkahKuda* pada dasarnya adalah sebuah matriks dua dimensi. Berikut ini keterangan prosedur serta fungsi yang ada:

- prosedur *jalan(int i)* dengan *i* adalah tipe jalan yang akan dilakukan ”kuda”. Ada 8 tipe jalan ”kuda” yang didefinisikan
- prosedur *jalan2(int i)* adalah prosedur balikan dari *jalan(int i)*
- fungsi *isValid()* memeriksa status elemen matriks. Apakah boleh diisi atau tidak
- prosedur *printMatriks()* berfungsi untuk menampilkan matriks
- fungsi *masihAda(int x, langkahKuda A)* memeriksa apakah *jalan(int i)* memungkinkan untuk dilakukan (paling tidak sekali) untuk *i*=*x* sampai dengan *i*=8

```
class langkahKuda {
 public int baris;
 public int kolom;
 public int langkah;
 public int[][] mat;
 public langkahKuda(){
  baris=0;
  kolom=0;
  langkah=1;
  mat = new int[4][8];
  for (int i=0;i<4;i++){
   for (int j=0;j<8;j++){
    mat[i][j] = 0;
   }
  }
  mat[baris][kolom] = 1;
 }
 public void jalan(int i){
  if(i==1){
   baris=baris-1;
   kolom=kolom+2;
   langkah=1;
  } else if(i==2){
   baris=baris+1;
   kolom=kolom+2;
   langkah=2;
  }else if(i==3){
   baris=baris+2;
   kolom=kolom+1;
   langkah=3;
  }else if(i==4){
   baris=baris+2;
   kolom=kolom-1;
   langkah=4;
  }else if(i==5){
   baris=baris+1;
   kolom=kolom-2;
   langkah=5;
  }else if(i==6){
   baris=baris-1;
   kolom=kolom-2;
   langkah=6;
  }else if(i==7){
   baris=baris-2;
   kolom=kolom-1;
   langkah=7;
  }else if(i==8){
   baris=baris-2;
   kolom=kolom+1;
   langkah=8;
  }
 }
 public void jalan2(int i){
  if(i==1){
   baris=baris+1;
   kolom=kolom-2;
  } else if(i==2){
   baris=baris-1;
   kolom=kolom-2;
  }else if(i==3){
   baris=baris-2;
   kolom=kolom-1;
  }else if(i==4){
   baris=baris-2;
   kolom=kolom+1;
  }else if(i==5){
   baris=baris-1;
   kolom=kolom+2;
  }else if(i==6){
   baris=baris+1;
   kolom=kolom+2;
  }else if(i==7){
   baris=baris+2;
   kolom=kolom+1;
  }else if(i==8){
   baris=baris+2;
   kolom=kolom-1;
  }
 }
 public boolean isValid(){
  if ((kolom>-1) && (baris>-1) && (kolom<8) &&
(baris<4)){
   return (mat[baris][kolom]==0);
  } else {
   return false;
  }
 }
 public void printMatriks() {
  for (int i=0;i<4;i++) {
   for (int j=0;j<8;j++) {
    System.out.print (mat[i][j]+" ");
   }
   System.out.println("");
  }
 }
 public boolean masihAda(int x, langkahKuda A){
  for(int i=x;i<=8;i++){
   A.jalan(i);
   if(A.isValid()){
    A.jalan2(i);
    return true;
   }
   A.jalan2(i);
  }
  return false;
 }
}
```

5.2 Kelas *stack*

Kelas *stack* pada dasarnya sama dengan *stack* umumnya. Hanya saja kelas ini telah disesuaikan sehingga masing-masing tingkat dari *stack* menyimpan 3 informasi (baris dari kelas *langkahKuda*, kolom dari kelas *langkahKuda*, jenis langkah yang digunakan).

- prosedur *push(int x, int y, int z)* merupakan prosedur untuk memasukkan *x* (nilai baris dari suatu elemen matriks kelas *langkahKuda*), *y* (nilai kolom dari suatu elemen matriks kelas *langkahKuda*), dan *z* (tipe langkah yang telah digunakan untuk mencapai baris dan kolom tersebut) ke dalam *stack*.
- prosedur *pop()* berfungsi untuk mengeluarkan elemen teratas dari *stack*
- fungsi *isEmpty()* memeriksa apakah *stack* kosong
- fungsi *isFull()* memeriksa apakah *stack* telah terisi penuh

```
class stack{
 public int[][] data;
 public int top;
 public stack(){
  data = new int[33][3];
  top = 0;
 }
 public void push(int x, int y, int z){
  if(isFull()==false){
   top++;
   data[top][0]=x;
   data[top][1]=y;
   data[top][2]=z;
  }
 }
 public void pop(){
  if(isEmpty()==false){
   top--;
  }
 }
 public boolean isEmpty(){
  return(top==0);
 }
 public boolean isFull(){
  return(top==32);
 }
}
```

5.3 Kelas *utama*

Kelas *utama* merupakan *main* dari program ini. K berpindah dengan menjalankan *jalan(i)* dengan *i* dimulai dari 1. Jika *isValid()*, maka *stack* melakukan *push(K.baris, K.kolom, K.langkah)* untuk mencatat informasi petak tempat kuda berada (baris dan kolom) serta langkah apa yang diambil untuk mencapai petak tersebut. Setelah itu *loop* diteruskan. Jika tidak *isValid()*, maka *jalan2(i)* dilakukan untuk mengembalikan status K lalu *i* ditambah 1 (*increment*). Selanjutnya dilakukan *jalan(i)*. Proses ini dilakukan berulang kali hingga *i* mencapai 9, *stack* penuh, atau runut-balik habis. Jika *i* mencapai 9, maka dijalankan runut-balik (*stack* melakukan *pop()* untuk mengeluarkan informasi tentang langkah terakhir yang diambil, lalu *jalan2(i)* dilakukan dengan *i* adalah tipe langkah yang diapat dari *pop()*). Jika *stack* penuh, maka solusi ditemukan dan matriks ditampilkan. Jika runut-balik habis, maka solusi tidak ditemukan.

```
class utama {
 public static void main(String argv[]){
  stack S = new stack();
  langkahKuda K = new langkahKuda();
  K.printMatriks();
  int i=1;
  int j=2;
  S.push(0, 0, 1);
  do{
   K.jalan(i);
   if(K.isValid()){
    K.mat[K.baris][K.kolom]=j;
    S.push(K.baris, K.kolom, K.langkah);
    i=1;
    j++;
   }else{
    K.jalan2(i);
    i++;
    if(i==9){
     do{
      K.mat[S.data[S.top][0]][S.data[S.top][1]]
      =0;
      j--;
      K.jalan2(S.data[S.top][2]);
      S.pop();
     }while(S.data[(S.top)+1][2]==8 ||
K.masihAda(S.data[(S.top)+1][2]+1, K)==false);
      i=S.data[(S.top)+1][2]+1;
    }
   }
  }while(S.isFull()==false || S.data[1][2]!=8);
  if (S.isFull()){
   System.out.println("Solusi ditemukan");
   K.printMatriks();
  }else{
   System.out.println("Solusi tidak ditemukan");
  }
 }
}
```

6. SOLUSI

Algoritma runut-balik untuk mencari solusi persoalan ”lompatan kuda” di atas telah diimplementasikan ke dalam sebuah program dalam bahasa JAVA.

Persoalan ”lompatan kuda” ternyata memiliki penyelesaian. Pohon solusi yang ditemukan terlalu panjang sehingga tidak mungkin untuk ditampilkan. Berikut ini adalah rute jalur sesuai dengan pohon solusi yang telah ditemukan:

	0	1	2	3	4	5	6	7
0	mulai	selesai	8	19	2	25	10	21
1	15	18	1	24	9	20	3	26
2	30	7	16	13	28	5	22	11
3	17	14	29	6	23	12	27	4

Gambar 4. Solusi rute kuda

Petak *mulai* adalah petak posisi awal kuda. Setelah itu petak-petak lain mengandung angka yang menaik sebagai penunjuk petak mana yang harus ditempati selanjutnya untuk mencapai solusi. Petak *selesai* adalah petak terakhir yang ditempati oleh kuda.

7. KESIMPULAN

Algoritma runut-balik adalah algoritma yang memiliki karakteristik khas yaitu penelusuran DFS yang sistematik serta kemampuan untuk menelusuri balik ke simpul orangtua (atau bahkan lebih tinggi) jika penelusuran dengan memperluas simpul anak mengalami kebuntuan. Setelah melakukan runut-balik, akan dibangkitkan simpul anak lain yang mengarah kepada solusi.

Penerapan algoritma runut-balik dalam mencari solusi persoalan ”lompatan kuda” cukup tepat. Karena pemain kerap kali tidak memiliki informasi yang cukup untuk mengetahui petak mana yang harus dipilih selanjutnya. Setiap pilihan yang diambil mengarah pada sekumpulan pilihan yang baru. Oleh karena itu runut-balik cukup efisien untuk meneliti kembali kemungkinan pilihan lain apabila pilihan yang telah dilakukan tidak dapat mencapai solusi.

Namun pada prakteknya, algoritma runut-balik hampir mustahil untuk dilakukan secara sempurna tanpa bantuan (misalnya kertas dan alat tulis). Hal ini disebabkan kesulitan untuk mengingat langkah apa saja yang telah ditempuh. Oleh karena itu pemain lebih sering melakukan langkah acak (*bruteforce*) untuk mencari solusi dari persoalan ”lompatan kuda”.

REFERENSI

[1] Munir, Rinaldi, “Strategi Algoritmik”, ITB, 2007.
[2] http://en.wikipedia.org/ Tanggal akses: 20 Mei 2007 pukul: 15.00