PENERAPAN ALGORITMA RUNUT-BALIK DALAM PENCARIAN SOLUSI PERMAINAN *PEG SOLITAIRE*

Yudha Adiprabowo - 13506050

Program Studi Teknik Informatika
Institut Teknologi Bandung
Jl. Ganesha 10 Bandung
e-mail: if16050@students.if.itb.ac.id

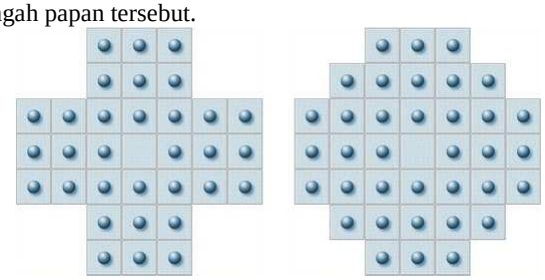
ABSTRAK

Makalah ini membahas mengenai algoritma runut-balik (*backtracking)* dan penerapannya dalam pencarian solusi permainan *Peg Solitaire.* Algoritma runut-balik adalah algoritma yang berbasis pada DFS (*Depth First Search*) untuk mencari solusi persoalan secara sistematis di antara semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan sehingga waktu pencarian dapat dihemat. *Peg Solitaire* merupakan permainan klasik yang menggunakan papan berlubang yang diisi penuh dengan pasak (*peg*) kecuali satu lubang di tengahnya. Cara memenangkan permainan ini adalah dengan mengosongkan papan dari pasak-pasak tersebut sampai hanya tersisa satu pasak saja. Makalah ini hanya membahas mengenai *Peg Solitaire* versi Inggris.

Kata kunci: *Peg Solitaire, backtracking*.

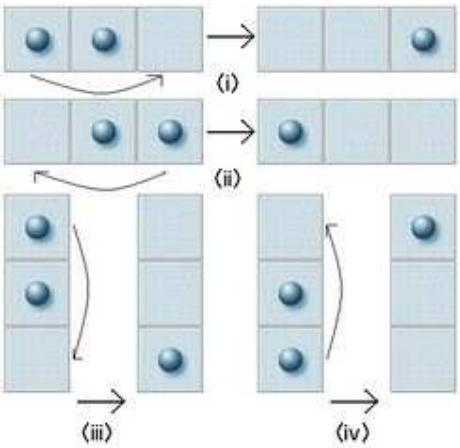
1. PENDAHULUAN

Peg Solitaire merupakan permainan yang menggunakan sebuah papan berlubang-lubang berbentuk seperti tanda tambah. Lubang di papan tersebut berjumlah 33 untuk versi Inggris, dan berjumlah 37 untuk versi Eropa. Pada versi standardnya, pasak-pasak dalam permainan ini memenuhi seluruh papan kecuali satu lubang di tengah-tengah papan tersebut.



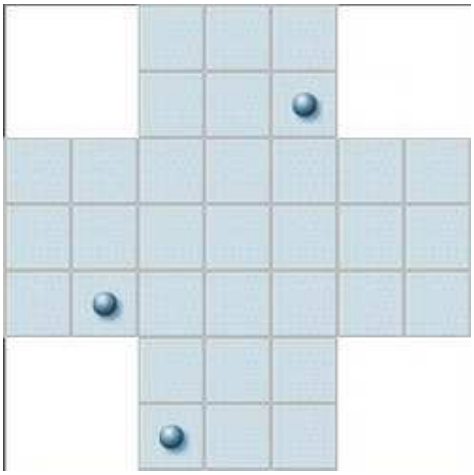
Gambar 1. Contoh *Peg Solitaire* standard, kiri : versi Inggris (yang dibahas dalam makalah ini), kanan : versi Eropa

Cara memainkan permainan ini adalah dengan memindahkan sebuah pasak secara horizontal / vertikal melompati sebuah pasak lain (berjarak satu kotak) menuju sebuah lubang berjarak dua kotak lalu membuang pasak yang dilompati tersebut. Langkah ini terus dilakukan sampai pasak di seluruh papan hanya bersisa satu.



Gambar 2. Langkah-langkah valid dalam *Peg Solitaire*, (i) lompat ke kanan, (ii) lompat ke kiri, (iii) lompat ke bawah, dan (iv) lompat ke atas

Sebenarnya, dalam menyelesaikan permainan ini tidak ada algoritma yang khusus. Meskipun demikian, dengan algoritma runut-balik (*backtracking*), kemungkinan untuk menemukan solusi akan semakin besar karena solusi untuk permainan ini termasuk sulit untuk ditemukan. Setelah jumlah pasak jauh berkurang, pemain lebih sering bertemu keadaan "buntu", yaitu pasak-pasak yang tersisa di papan terletak berjauhan dan hanya berjumlah satu sehingga tidak dapat melakukan lompatan lagi.



Gambar 3. Contoh kondisi "buntu"

2. METODE

Metode yang digunakan dalam pencarian solusi permainan *Peg Solitaire* ini adalah metode algoritma runut-balik (*backtracking*).

2.1 Sekilas Tentang Algoritma Runut-Balik

Properti metode runut-balik:

a. Solusi persoalan
Solusi dinyatakan sebagai vektor dengan n-tuple :

$X = (x_1,x_2,...,x_n)$, x adalah elemen berhingga dari S

b. Fungsi Pembangkit nilai x_k
Dinyatakan sebagai :

$T(k)$

T(k) membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.

c. Fungsi Pembatas
Dinyatakan sebagai :

$B(x_1,x_2,...,x_k)$

Fungsi pembatas menentukan apakah $(x_1,x_2,...,x_k)$ mengarah ke solusi. Jika ya, maka pembangkitan nilai untik x_k-1 dilanjutkan, tapi jika tidak, maka $(x_1,x_2,...,x_k)$ dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi. Fungsi pembatas tidak selalu dinyatakan sebagai fungsi matematis, dapat juga dinyatakan sebagai fungsi berpredikat *true* / *false*, atau dalam bentuk lain yang ekivalen.

Langkah-langkah pencarian solusi :

a. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah metode pencarian mendalam atau DFS. Simpul-simpul yang sudah dibangkitkan disebut simpul hidup, Simpul hidup yang sedang diperluas disebut simpul-E. Simpul dinomori dari atas ke bawah sesuai urutan kelahirannya.
b. 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. Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas. Simpul yang sudah mati tidak akan pernah diperluas lagi.
c. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan runut-balik ke simpul hidup terdekat (simpul orangtua). Selanjutnya simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun kenmbali sampai lintasan tersebut membentuk solusi.
d. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

2.1 Algoritma Runut-Balik Untuk Solusi *Peg Solitaire*

Metode algoritma runut-balik (*backtracking*) yang digunakan di sini adalah metode rekursif. Properti algoritma yang digunakan adalah sebagai berikut. Solusi persoalan dinyatakan dengan sebuah kelas *Move* yang menyimpan seluruh pergerakan pasak sampai mencapai solusi. Fungsi pembangkit dinyatakan dengan fungsi *findStep*(*int level*) yang akan membangkitkan semua kemungkinan pergerakan pasak. Sementara, fungsi pembatas dinyatakan dengan fungsi *solved*() yang bernilai *true* jika jumlah pasak dalam papan hanya tinggal satu.

Berikut ini adalah implementasi algoritma runut-balik dalam bahasa C++ untuk menyelesaikan permainan *Peg Solitaire*. Dalam algoritma ini terdapat setidaknya dua kelas, yaitu kelas *Backtrack* dan kelas *Solitaire* yang merupakan turunan kelas *Backtrack*. Kelas *Backtrack* memberikan *pseudocode* dasar bagaimana algoritma ini bekerja sementara kelas *Solitaire* lebih dalam implementasi pergerakan pasak-pasak dalam permainan.

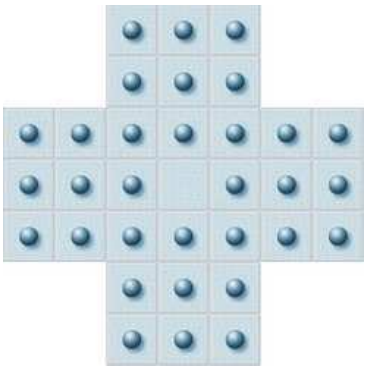
Contoh *Source Code* dalam bahasa C++

```
bool Backtrack::findStep(int level)
{
  assert(level >= 1);
  if (solved())
  {
    saveLevel(level - 1);
    return true;
  }
  Step *p_step = firstStep();
  if (p_step)
  {
    do
    {
      doStep(p_step);
      traceStep(p_step, level);
      if (findStep(level + 1))
      {
        saveStep(p_step, level);
        return true;
      }
      undoStep(p_step);
      traceStep(p_step, level, true);
    }
    while (!m_abort && nextStep(p_step));
    delete p_step;
  }
  return false;
}

bool Backtrack::solve()
{
  m_abort = false;
  const bool c_ret = findStep(1);
  assert(!(m_abort && c_ret));
  return c_ret;
}

void Backtrack::saveLevel(int level)
{
  assert(level >= 0);
}

void Backtrack::traceStep(const Step
*pc_step, int level, bool undo)
{
  assert(pc_step);
  assert(level >= 1);
}

bool Solitaire::solved() const
{
  return (m_numPegs == 1);
}
```

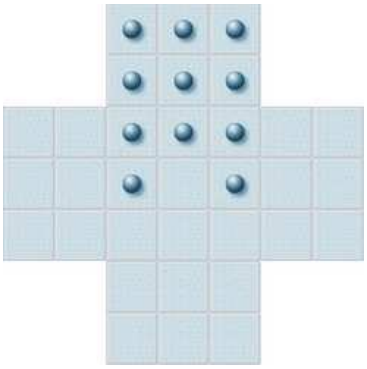
Penerapannya dalam masalah ini dapat dijabarkan sebagai berikut. Pertama, program akan mencari langkah pertama yang mungkin dilakukan dari kondisi papan saat ini. Kedua, jika setelah langkah pertama solusi belum ditemukan, maka program akan melakukan pengulangan untuk melakukan langkah tersebut. Ketiga, program lalu melakukan rekursif untuk mencari langkah selanjutnya, lalu menyimpan seluruh langkah yang telah dilakukan agar bisa runut-balik bisa dilakukan. Pengulangan berhenti jika tidak ada lagi langkah yang bisa dilakukan. Keempat, jika tidak ada langkah yang bisa dilakukan lagi tetapi solusi belum ditemukan, maka program akan melakukan runut-balik.

3. ANALISIS

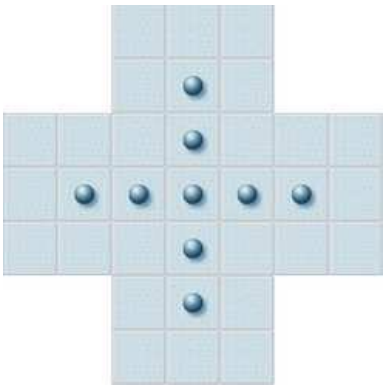
Dengan menggunakan algoritma pada bagian sebelumnya, dilakukan percobaan untuk menyelesaikan beberapa contoh papan *Peg Solitaire*:



Gambar 4. Kondisi standard



Gambar 5. *Fireplace*



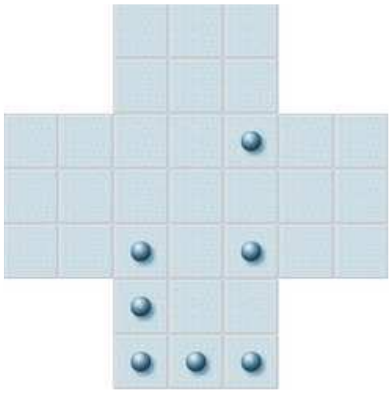
Gambar 6. *Plus*

Pada papan *Fireplace* algoritma menemukan solusi dalam 10 langkah, dengan jumlah runut-balik sangat

sedikit (penulis tidak dapat menghitung jumlahnya karena komputer penulis terlalu cepat memproses pergerakan). Sementara, pada papan *Plus*, solusi ditemukan dalam 8 langkah , dengan tidak melakukan runut-balik sama sekali. Terakhir, pada papan kondisi standard, solusi ditemukan dalam > 200 langkah dan runut-balik yang dilakukan lebih dari 100 kali.

Pencarian solusi yang panjang pada papan kondisi standard disebabkan oleh beberapa hal, yakni:

1. Jumlah pasak yang terlalu banyak, sehingga mengakibatkan besarnya pohon solusi yang dibangkitkan.
2. Kondisi mendekati akhir yang agak rumit, seperti bisa dilihat pada gambar 7, di mana kondisi ini justru tidak akan melahirkan solusi. Akibatnya, program harus melakukan runut-balik sampai ke beberapa simpul orangtua di atas, setelah melewati kondisi ini berkali-kali.



Gambar 7. Kondisi mendekati "buntu"

Berikut ini diberikan tabel kemungkinan posisi pasak (KPP) dalam papan kondisi standard setelah *n* lompatan:

Tabel 1 Jumlah kemungkinan posisi pasak dalam papan kondisi standard

n lompatan	KPP	Kemungkinan buntu
1	1	0
2	2	0
3	8	0
4	39	0
5	171	1
6	719	0
7	2.757	0
8	9.751	0
9	31.312	0
10	89.927	1
11	229.614	1
12	517.854	0
13	1.022.224	5
14	1.753.737	10
15	2.598.215	7
16	3.312.423	27
17	3.626.632	47
18	3.413.313	121
19	2.765.623	373
20	1.930.324	925
21	1.160.977	1.972
22	600.372	3.346
23	265.865	4.356
24	100.565	4.256
25	32.250	3.054
26	8.688	1.715
27	1.917	665
28	348	182
29	50	39
30	7	6
31	2	2

Dari tabel dapat dilihat kemungkinan penyebab algoritma runut-balik yang kita gunakan banyak melakukan runut-balik itu sendiri. Setelah lompatan ke-22 (pada lompatan ke-23), terdapat 265.865 kemungkinan letak pasak pada papan dan 4.356 dari jumlah tersebut akan mengarah ke kondisi buntu. Jika sejak awal algoritma tidak masuk ke jalan yang tepat, maka runut-balik akan terus dilakukan.

4. KESIMPULAN

Algoritma runut-balik (*backtracking*) cukup baik untuk mencari solusi permainan *Peg Solitaire* walaupun untuk beberapa papan *Peg Solitaire* dengan jumlah pasak sangat banyak algoritma ini akan banyak memakan waktu untuk menemukan solusinya. Tentu saja, algoritma ini masih jauh lebih baik daripada algoritma *brute force* karena kita tidak harus melewati jutaan kemungkinan untuk menemukan solusi. Meskipun begitu, algoritma runut-balik mungkin bukanlah penyelesaian terbaik untuk mencari solusi permainan *Peg Solitaire*. Penggunaan algoritma lain bisa saja menghasilkan pencarian solusi yang lebih cepat daripada algoritma runut-balik.

REFERENSI

[1] Munir, Rinaldi. "Diktat Kuliah IF2251 Strategi Algoritmik". Institut Teknologi Bandung. 2007.

[2] *Interactive Mathmatics Miscellany and Puzzles*. *Peg Solitaire*. http://www.cut-the-knot.org/proofs/pegsolitaire.shtml diakses pada hari Senin, 19 Mei 2008.

[3] Wikipedia. *Peg Solitaire*. http://en.wikipedia.org/wiki/Peg_solitaire diakses pada hari Senin, 19 Mei 2008.