Kombinasi Tiga Algoritma untuk Puzzle Shikaku

Yusuf Adrianysah – 13507120

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jalan Ganeca 10, Bandung 40175
ysf4m1c@yahoo.co.id

ABSTRAK

Permainan shikaku adalah permainan *logic puzzle* dimana kemampuan visual dan geometris dibutuhkan. Bagi manusia lebih mudah, karena manusia memproses informasi visual dengan cepat dan tidak langkah-demi-langkah seperti yang dikerjakan komputer. Namun, bagaimana caranya mengimplementasikan algoritma untuk menyelesaikan shikaku pada komputer, yang tidak punya kemampuan visual dan harus bekerja langkah demi langkah seperti itu?

Terdapat beberapa pendekatan penyelesaian masalah. Makalah ini akan membahas brute-force, greedy, BFS, dan runut balik bila diimplementasikan sendiri-sendiri. Terakhir, akan disusun algoritma "smart" yang menggabungkan konsep greedy, BFS, dan runut balik dengan harapan penyelesaian shikaku menjadi lebih efisien lagi.

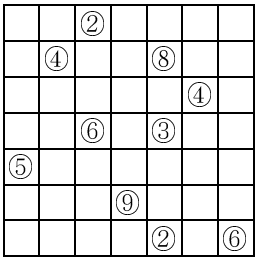
Kata kunci: Shikaku, Puzzle, Brute force, Greedy, BFS, Runut balik, Backtracking.

1. PENDAHULUAN

Shikaku (四角) dalam bahasa Jepang berarti *segi empat*. Permainan ini berjenis puzzle. Pembuatnya adalah Nikoli, yaitu perusahaan penerbit yang juga membuat puzzle lain seperti sudoku, kakuro, dan hitori.

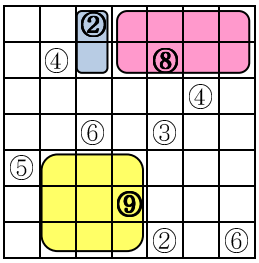
1.1. Cara bermain shikaku

Papan shikaku terdiri dari n × n petak. Di petak-petak tersebut, terdapat angka-angka yang lokasinya acak. Contohnya seperti gambar berikut:



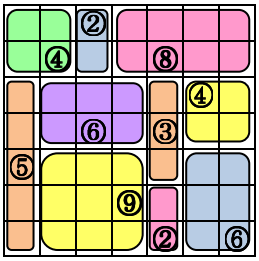
Gambar 1. Matriks shikaku

Pemain harus membagi petak n × n tersebut menjadi petak-petak yang lebih kecil, dimana setiap subpetak hanya boleh "mengantongi" satu angka dan luas subpetak harus sama dengan angka itu juga. Contohnya:



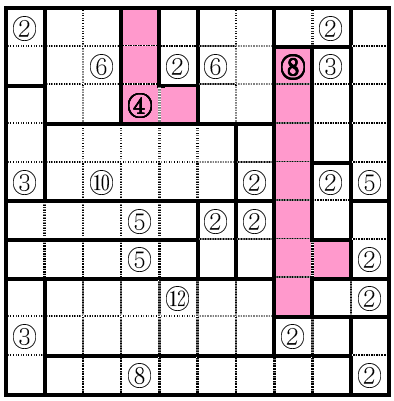
Gambar 2. Matriks shikaku dalam progres

Permainan berakhir bila semua angka sudah mendapat partisi masing-masing.



Gambar 3. Matriks shikaku yang sudah selesai

Partisi yang diperbolehkan hanyalah partisi berbentuk persegi atau persegi panjang. Dengan kata lain, pembagian petak seperti yang ditandai dengan warna merah ini tidak dibenarkan, meskipun luasnya benar.



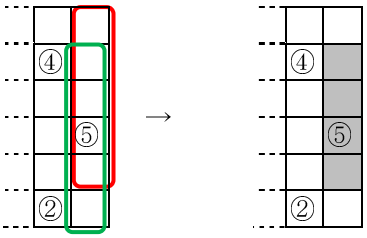
Gambar 4. Pembagian yang tidak sah

2. ALGORITMA PENYELESAIAN SHIKAKU

2.1. Oleh Manusia

Manusia menyelesaikan shikaku secara *trial and error*, metode coba-coba dengan pensil. Ada beberapa patokan yang dipegang oleh orang yang bermain shikaku, yaitu:

1. Luasan partisi adalah perkalian faktor dari angka yang dikurungnya. Misalnya,
   ③ adalah 1 × 3;
   ④ adalah 1 × 4 atau 2 × 2;
   ⑥ adalah 1 × 6 atau 2 × 3;
   ⑧ adalah 1 × 8 atau 2 × 4;
   ⑨ adalah 1 × 9 atau 3 × 3;
   ⑫ adalah 1 × 12, 2 × 6, atau 3 × 4;
   dan seterusnya.
2. Dahulukan mempartisi angka yang berada di sudut, kemudian pinggir matriks.
3. Dahulukan angka ganjil dan prima. Misalnya ⑤ pasti 1 × 5, tidak mungkin kombinasi yang lain. Partisi semacam ini pasti bentuknya panjang.
4. Amankan petak-petak kemungkinan partisi yang beririsan. Contohnya, ⑤ di sini punya dua kemungkinan, amankan 4 petak yang saling beririsan terlebih dulu.



Gambar 5. Penjelasan poin keempat

2.2. Oleh Komputer

2.2.1. Algoritma brute force

Penyelesaian dengan brute force mirip dengan algoritma untuk suudoku, namun kita perlu bantuan tabel (*array*). Ambil titik mulai dari salah satu sudut, bebas. Misalnya kita ambil mulai dari sudut kiri atas.

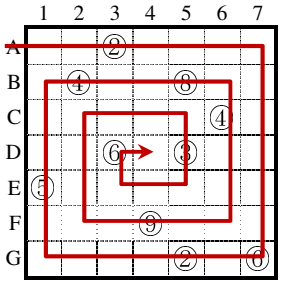
- Tahap observasi
  - Berjalanlah ke kanan sampai ditemukan angka. Antrikan angka tersebut (dan ingatlah posisinya) ke dalam tabel.
  - Bila tidak ditemukan angka lagi sampai ujung baris, mulai lagi dari paling kiri baris bawahnya.
- Tahap eksekusi
  - Proses mulai dari awal tabel. Baca elemen tabel. Menurut angka dalam elemen tabel tersebut, coba satu kemungkinan bentuk partisi yang sah pada matriks shikaku.
  - Bila partisi itu ternyata menabrak partisi lain, atau ikut mengantongi angka di petak lain, maka partisi itu tidak dibenarkan. Coba kemungkinan partisi yang lain.
  - Apabila tidak ada kemungkinan partisi yang sah, maka mundurlah kembali ke angka sebelumnya (elemen tabel sebelumnya) lalu ganti bentuk partisi di angka tersebut.

Algoritma brute force adalah algoritma yang sedapat mungkin dihindari, karena boros waktu dan tenaga. Maka dari itu, penyelesaian shikaku dengan brute force tidak diilustrasikan. Ilustrasi brute force mengakibatkan panjang makalah ini menjadi 12 halaman.

2.2.2. Algoritma greedy

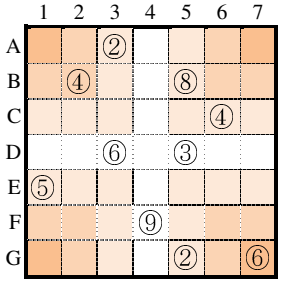
Greedy yang digunakan di sini adalah *greedy by corner* atau *greedy by edge*. Kerjakan angka-angka yang berada di ujung atau pinggir terlebih dahulu.

Untuk *greedy by edge* (pinggir lebih dulu), tahap observasi bisa dilakukan dengan lintasan spiral seperti ini:



Gambar 6. Greedy by edge

Sedangkan *greedy by corner*, program bisa memeriksa petak-petak terujung, atau 2 × 2 petak terujung, atau 3 × 3 petak terujung, tergantung kebutuhan.

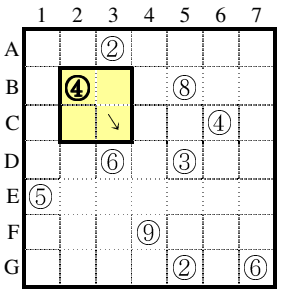


Gambar 7. Greedy by corner

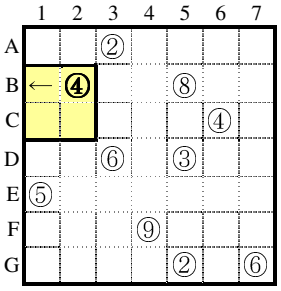
Setelah petak-petak "sudut" didapatkan, program lanjut dengan memeriksa pinggir terlebih dulu. Terakhir adalah area tengah yang belum tersentuh.

Tahap eksekusi menurut algoritma greedy sama dengan brute force, namun ada sedikit penambahan optimasi:

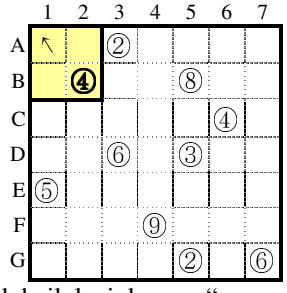
> *Sebisa mungkin tarik partisi ke luar.*
> (ke sudut jika mungkin, ke pinggir jika sudut tak mungkin)



Pemilihan partisi untuk angka ④ begini dianggap buruk, karena "mengarah" ke dalam.



Seperti ini lebih baik, ada usaha "mengarah" ke luar.



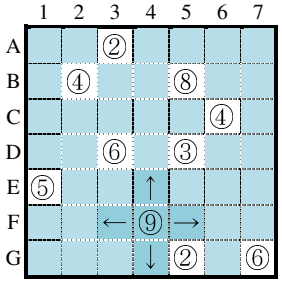
Seperti ini lebih baik lagi, karena "mengarah" ke ujung.

Gambar 8. "Sebisa mungkin tarik partisi ke luar"

2.2.3. Algoritma BFS + *bounding function*

BFS atau *breadth first search* adalah algoritma pencarian graf yang dimulai dari satu simpul kemudian melebar ke sekitarnya. Dalam shikaku, setiap petak berangka menjadi akar dari pohon pencarian BFS, sedangkan petak-petak sekitarnya menjadi simpul tetangga.

BFS pada tahap observasi tidak membantu. BFS yang dimaksud ada pada tahap eksekusi. Contoh berikut menggambarkan BFS yang dieksekusi atas petak ⑨, dan petak yang diwarnai biru adalah "ruang gerak" petak ⑨ ini.



Gambar 9. BFS atas petak ⑨ di F4

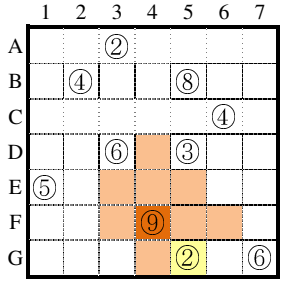
BFS seperti ini tidak ada gunanya. Untuk apa petak G6 diakses, sudah jelas tidak mungkin ada partisi valid yang melingkupi petak G6 tadi.

Maka dari itu, dibuatlah sebuah fungsi pembatas atau *bounding function* untuk menentukan apakah simpul tetangga ini layak dikunjungi atau tidak. *Bounding function* ini secara informal dinyatakan sebagai:

Ketika mengunjungi petak x, maka periksa:

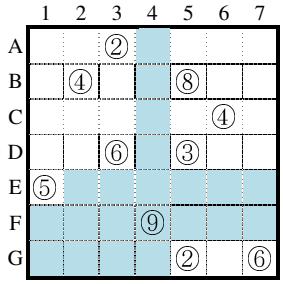
1. *Bila petak x sudah menjadi milik partisi lain, maka petak x dan keturunannya tidak layak.*
2. *Bila x bukan milik partisi manapun (petak bebas), maka periksa persegi panjang yang dibentuk dari petak x ke petak awal. Bila persegi panjang ini berisi minimal **dua** petak angka **atau** menabrak partisi lain, maka petak x dan keturunannya tidak layak.*

Agar lebih jelas, berikut ilustrasinya:



Gambar 10. BFS atas petak ⑨ di F4 (baru sebagian)

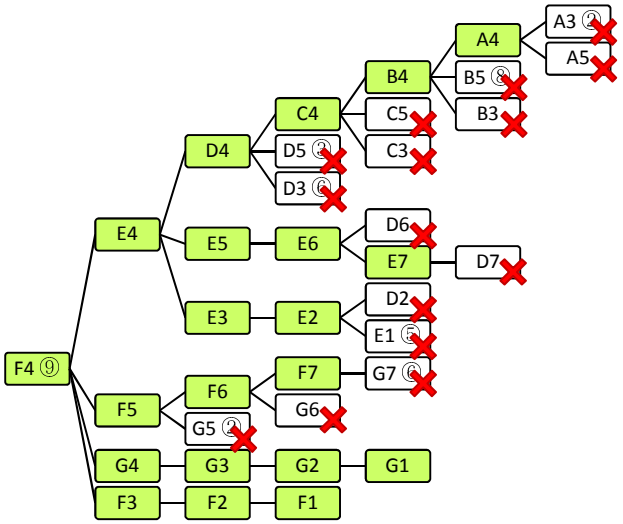
Sejauh ini, BFS mengunjungi petak-petak sesuai urutan: E4 → F5 → G4 → F3 → D4 → E5 → E3 → F6 → G5. Semua petak di atas kecuali G5 memenuhi syarat fungsi pembatas. Begitu simpul G5 dikunjungi, ternyata persegi yang dibentuk dari petak F4 sampai G5 mengandung dua petak berangka (yaitu ② dan ⑨). Maka, simpul G5 beserta keturunannya dibunuh/dimatikan. Hasil dari BFS+ ini selengkapnya adalah:



Gambar 11. BFS atas petak ⑨ di F4 dengan fungsi pembatas

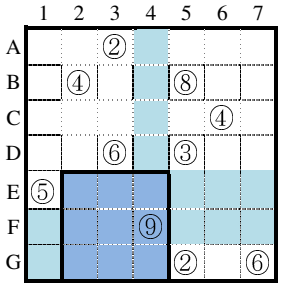
Petak yang diwarnai biru diatas adalah "ruang gerak" dari petak ⑨. Partisi yang melingkupi ⑨ pun pasti berada di wilayah biru ini.

Berikutnya, program akan mencoba kemungkinan partisi yang sesuai. Kemungkinan untuk angka ⑨ ialah: 9 × 1 (horizontal panjang), 1 × 9 (vertikal panjang), dan 3 × 3.



Gambar 12. Pohon BFS yang dibentuk dengan memperhatikan *bounding function*. Simpul menyatakan koordinat petak, dan simpul yang dimatikan ditandai dengan ✕ silang merah.

Untuk 9 × 1 dan 1 × 9 sudah jelas tidak mungkin, karena ukuran papan shikaku dalam contoh ini hanya 7 × 7. Satu-satunya yang mungkin adalah 3 × 3:



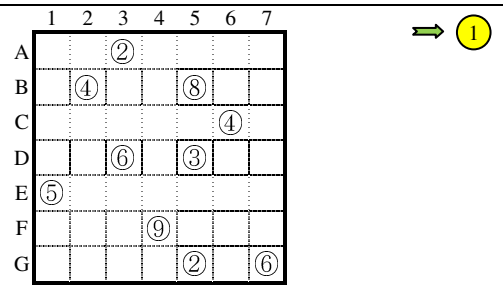
Gambar 13. Partisi untuk ⑨ berhasil ditempatkan.

2.2.4. Algoritma runut balik

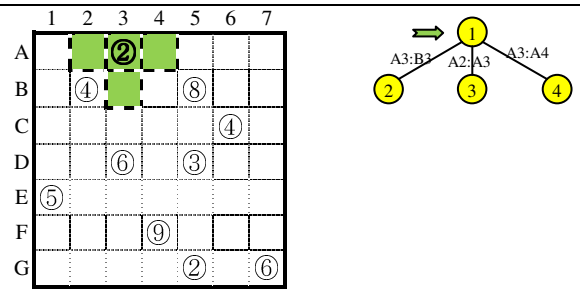
Algoritma runut balik menggunakan konsep yang berbeda. Di sini, pohon yang dibentuk adalah pohon *state*, bukan pohon koordinat petak. Secara umum, algoritma runut balik untuk shikaku adalah sebagai berikut:

- Pada awalnya, baru ada satu simpul saja (yaitu simpul akar) dan kursor penunjuk berada di petak kiri atas.
- Jalan ke kanan sampai ditemukan angka. Himpun semua kemungkinan bentuk partisi sambil memperhatikan fungsi pembatas.
- Gambar partisi di matriks shikaku sesuai kemungkinan yang pertama ditemukan.
- Himpunan solusi yang mungkin ini ditambahkan ke dalam pohon ruang status sebagai simpul anak dari simpul saat ini.
- Namun bila himpunan solusi saat ini kosong, maka lakukan runut balik ke kakek dari simpul saat ini, lalu coba simpul anak lainnya yang belum tersentuh. Pindahkan juga posisi kursor penunjuk ke petak angka yang bersesuaian dengan simpul kakek tadi.
    - *"Mencoba simpul anak lainnya yang belum tersentuh" berarti mengubah bentuk partisi di papan shikaku.*
- Jalan lagi ke kanan sampai ditemukan angka berikutnya. Bila kursor penunjuk sudah tiba di ujung baris, ulangi lagi dari petak terkiri baris bawahnya.

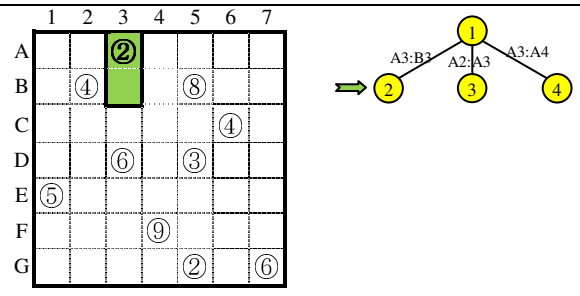
Supaya mudah, saya akan menggunakan notasi Microsoft Excel untuk menggambarkan sebuah bentuk persegi panjang. Contohnya notasi **A1:C3** adalah persegi yang sudutnya dibentuk dari petak A1, A3, C1, dan C3.



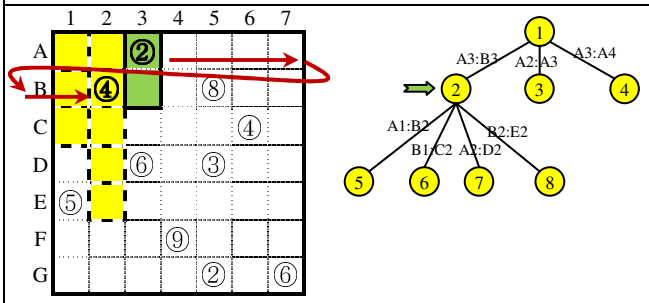
Keadaan awal. Belum ada partisi dan baru ada satu *state*.



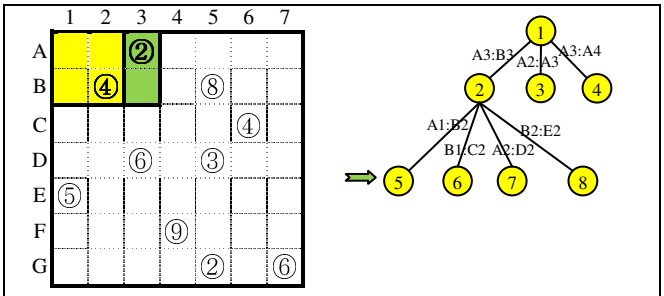
Jalan dari petak terkiri ke kanan. Ditemukan angka ②. Ada tiga kemungkinan partisi yang bisa melingkupi ② ini.



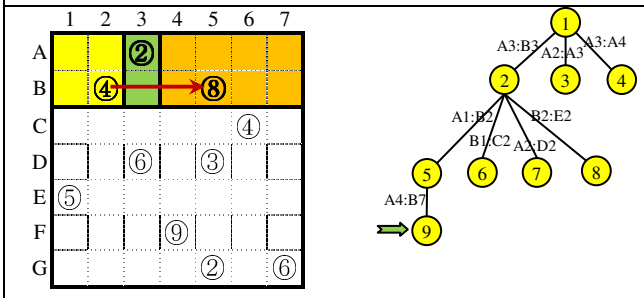
Ambil kemungkinan pertama, yaitu partisi A3:B3.



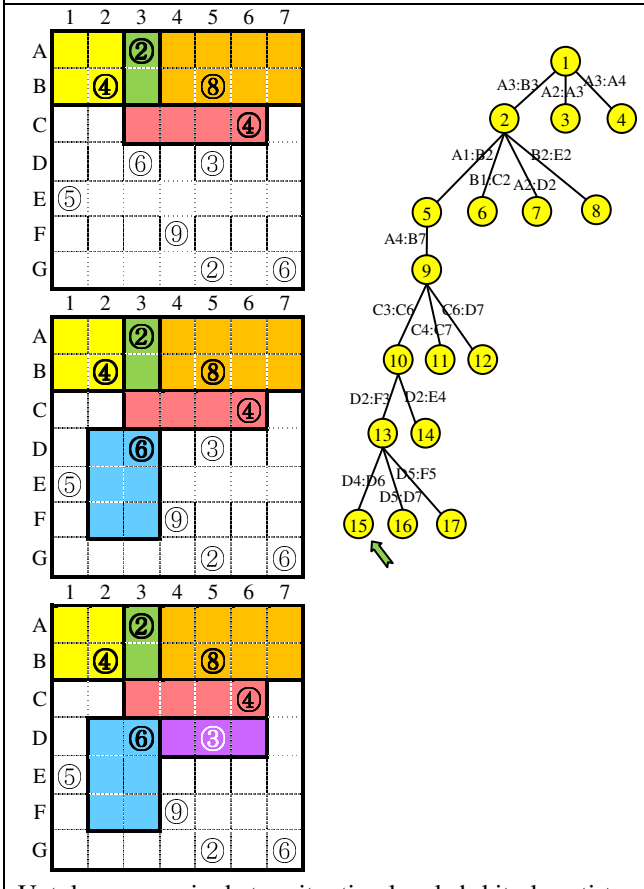
Jalan lagi ke kanan sampai pindah baris bawahnya. Ditemukan angka ④. Didapat 4 kemungkinan.



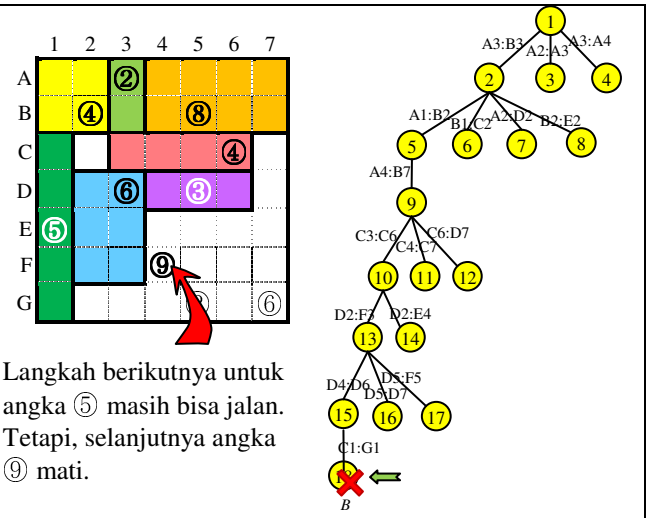
Ambil kemungkinan pertama, yaitu A1:B2



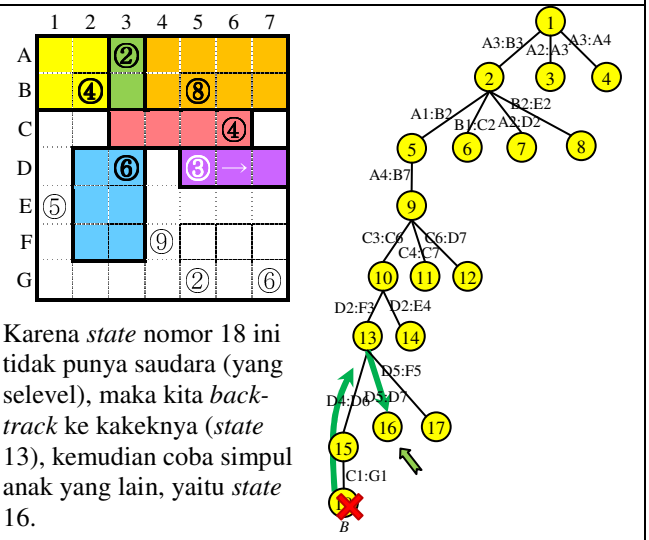
Jalan lagi ke kanan. Ditemukan angka ⑧. Hanya satu inilah kemungkinan yang ada.



Untuk mempersingkat cerita, tiga langkah kita lewati tanpa masalah.



Langkah berikutnya untuk angka ⑤ masih bisa jalan. Tetapi, selanjutnya angka ⑨ mati.



Karena *state* nomor 18 ini tidak punya saudara (yang selevel), maka kita *backtrack* ke kakeknya (*state* 13), kemudian coba simpul anak yang lain, yaitu *state* 16.

… dan seterusnya.

Gambar 14. Ilustrasi penyelesaian shikaku dengan algoritma runut balik

Dalam contoh ini, bisa diprediksi *state* 16 dan 17 akhirnya juga mati. Jelas, karena secara visual sudah terlihat bahwa partisi yang melingkupi angka ⑥ sudah salah. Pada akhirnya, terlalu banyak runut balik yang dilakukan.

3. KOMBINASI ALGORITMA UNTUK PENYELESAIAN SHIKAKU

Dalam bab ini akan disusun algoritma kombinasi yang menggabungkan ide dari algoritma greedy, BFS+, dan runut balik, dengan harapan penyelesaian shikaku menjadi lebih efisien lagi.

Algoritma ini membagi penyelesaian menjadi dua tahap, yaitu observasi dan ekseksusi. Pencatatan petak angka mana saja yang akan diproses beserta urutannya, dilakukan pada tahap observasi menggunakan algoritma *greedy by*

corner. Tahap eksekusi menggabungkan ide dari 3 algoritma ini:

- Penentuan batas-batas lokasi partisi memakai BFS+.
- Enumerasi bentuk-bentuk partisi yang sah akan membentuk pohon DFS runut balik.
- Penentuan partisi terbaik saat ini, meminjam ide dari optimasi eksekusi greedy.

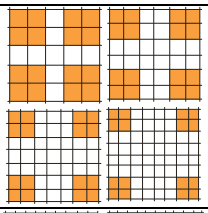
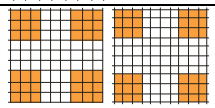
3.1. Tahap observasi

Diberikan shikaku $n \times n$, dimana n bilangan asli ≥ 5. Kita akan memeriksa $m \times m$ petak terujung, dimana $m < n$. Saya menyarankan ukuran m sesuai rumus:

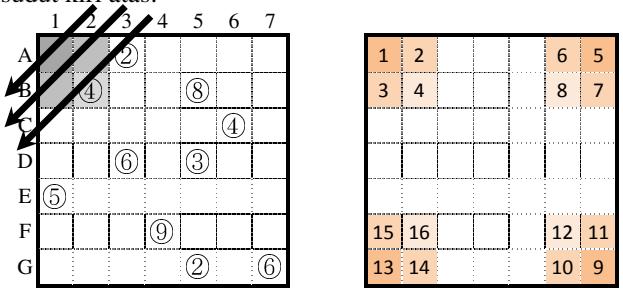
$$m = \left\lceil \frac{n}{4} \right\rceil \qquad (1)$$

m adalah pembulatan keatas (fungsi ceiling) dari $\frac{n}{4}$.

Tabel 1. Ukuran papan shikaku dan seberapa luas area yang dianggap sudut

Ukuran matriks (n)	Jumlah petak terujung (m)	Ilustrasi
5×5 sampai 8×8	2×2	
9×9 sampai 12×12	3×3	 *(selanjutnya terlalu kecil untuk ditampilkan)*
…dan seterusnya.		

Untuk setiap area yang dianggap sudut, urutan pengecekan dilakukan miring (dari yang "paling" ujung ke yang paling "tidak" ujung). Contohnya, untuk papan 7 × 7 pada sudut kiri atas:



Gambar 15. Urutan ketika mengunjungi petak area sudut

Tabel 2. Tabel sementara hasil observasi di empat sudut

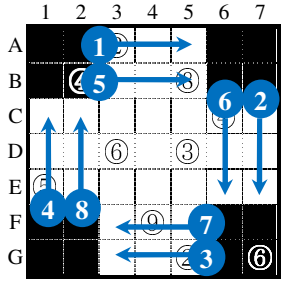
Angka	④	⑥								
Posisi	B2	G7								

Setelah angka-angka pada area sudut didapatkan, berikutnya periksa area pinggir. Yang didefinisikan sebagai daerah "pinggir" adalah:

sudut	pinggir	sudut
pinggir	tengah	pinggir
sudut	pinggir	sudut

Gambar 16. Definisi area sudut, pinggir, dan tengah

Sama seperti pemeriksaan petak-petak sudut, pemeriksaan petak-petak pinggir juga dilakukan dari yang terpinggir ke yang ter"tidak" pinggir. Dalam gambar ini, area sudut dihitamkan sebagai tanda sudah pernah dikunjungi:



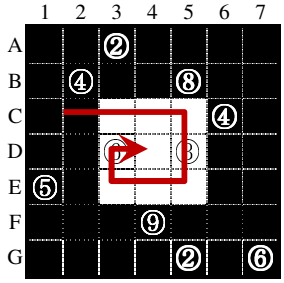
Gambar 17. Urutan ketika mengunjungi petak area pinggir

Kita berkeliling dari lintasan/*track* terluar, dilanjutkan *track* dalam. Bila matriks shikaku adalah 9 × 9 sehingga area sudut menjadi 3 × 3, maka area pinggir memiliki tebal tiga *track*.

Tabel 3. Tabel sementara hasil tahap observasi

Angka	④	⑥	②	②	⑤	⑧	④	⑨		
Posisi	B2	G7	A3	G5	E1	B5	C6	F4		

Langkah terakhir dalam tahap observasi adalah memeriksa area tengah. Urutan terbaik untuk mengunjungi petak-petak area ini adalah lintasan spiral.



Gambar 18. Urutan ketika mengunjungi petak area tengah

Sekarang kita telah selesai menyusun tabel antrian yang nantinya akan digunakan di tahap eksekusi:

Tabel 4. Tabel final hasil tahap observasi

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3

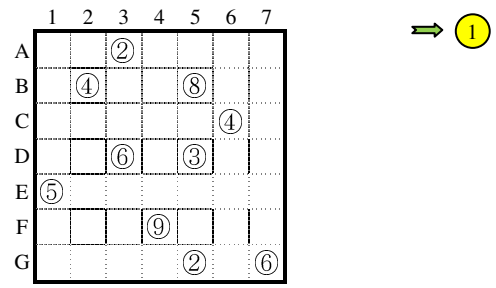
3.2. Tahap eksekusi

Tahap eksekusi dilakukan mulai dari elemen tabel pertama. Urutan eksekusi menurut algoritma combo adalah:

- Laksanakan BFS+ atas petak angka menurut elemen tabel.
- Enumerasi kemungkinan bentuk partisi yang mungkin atas angka itu pada daerah hasil BFS. Tetap perhatikan prinsip optimasi, sedapat mungkin tarik ke luar.
- Kemungkinan terbaik (sesuai prinsip optimasi) didaftarkan sebagai simpul anak pertama dari *state* saat ini. Kemungkinan terbaik kedua sebagai simpul anak kedua, dan seterusnya.
- Ambil kemungkinan pertama yang disebut terbaik itu. Gambar partisi yang dimaksud pada matriks shikaku. Kita pindah ke simpul anak yang diambil ini.
- Bila tidak ada partisi yang sah untuk simpul saat ini,
  - Hapus partisi milik simpul saat ini beserta orang tuanya dari papan shikaku.
  - Lakukan runut balik ke kakeknya simpul saat ini (berarti mundur 2 elemen tabel).
  - Coba simpul anak yang lain.
  - Gambar partisi yang baru di papan shikaku.
- Setelah partisi untuk simpul saat ini berhasil dibentuk, pindah ke elemen tabel berikutnya.
- Tahap eksekusi selesai bila semua elemen tabel sudah selesai diproses.

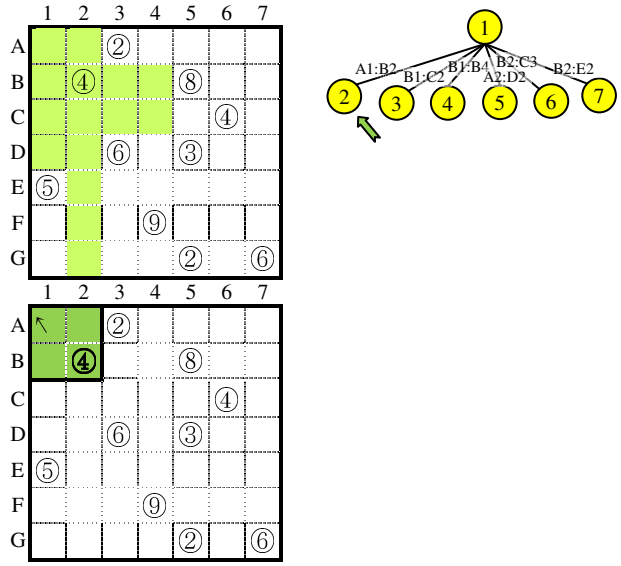
Berikut akan dicontohkan penyelesaian shikaku 7 × 7 yang sama dengan contoh sebelumnya, berdasarkan algoritma combo.

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3



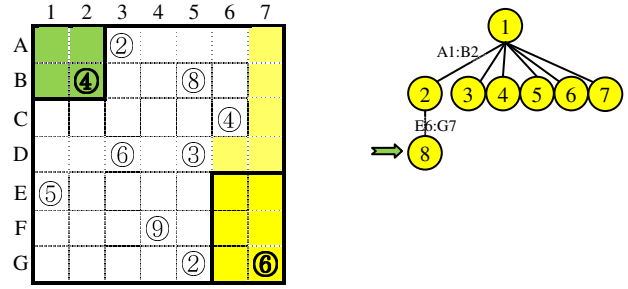
Keadaan awal.

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3



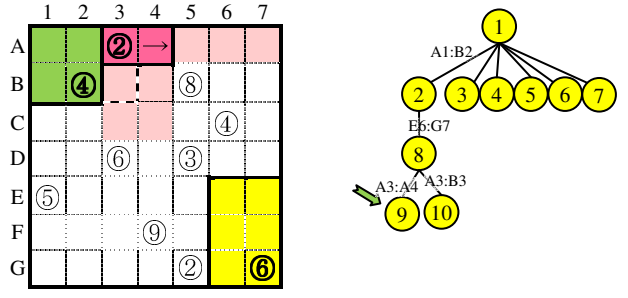
Langkah 1.
Gambar atas: Eksekusi BFS+ atas petak ④.
Gambar bawah: Pemilihan partisi terbaik (mengarah ke ujung) sesuai prinsip optimasi greedy.

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3



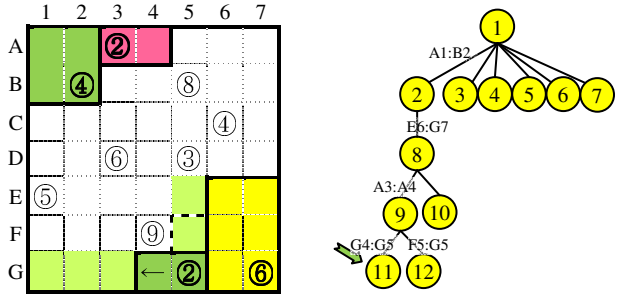
Langkah 2. Maju ke elemen tabel berikutnya. Hanya ada satu kemungkinan untuk ⑥. (untuk menghemat ruang, daerah hasil BFS+ dan bentuk partisi yang dipilih dijadikan satu gambar)

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3



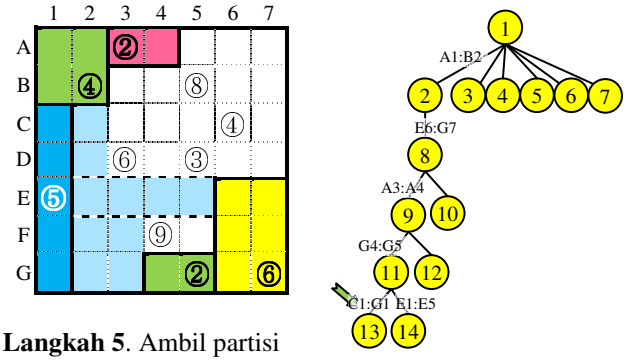
Langkah 3. Prinsip optimasi menghasilkan partisi yang mengarah ke pinggir.

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3



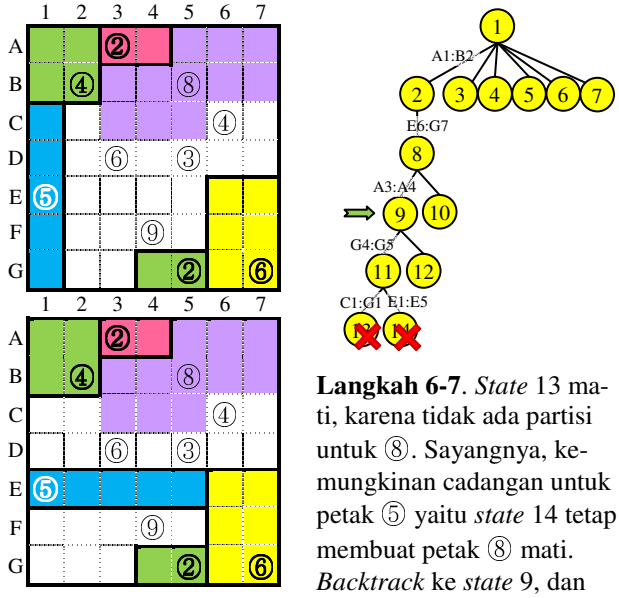
Langkah 4.

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3



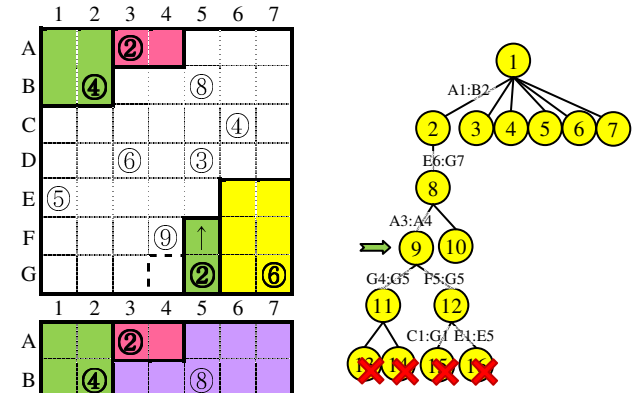
Langkah 5. Ambil partisi yang menempel di pinggir.

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3



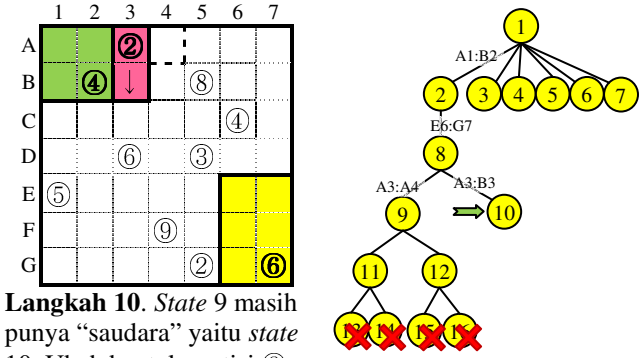
Langkah 6-7. *State* 13 mati, karena tidak ada partisi untuk ⑧. Sayangnya, kemungkinan cadangan untuk petak ⑤ yaitu *state* 14 tetap membuat petak ⑧ mati. *Backtrack* ke *state* 9, dan mundur dua elemen tabel.

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3



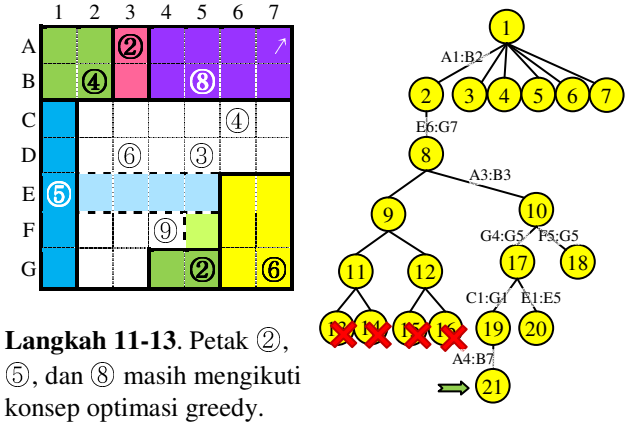
Langkah 8-9. Memasuki *state* 12, kita ganti partisi ② menjadi F5:G5. Namun akhirnya ⑤ membuat ⑧ mati lagi. *Backtrack* lagi ke *state* 9.

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3



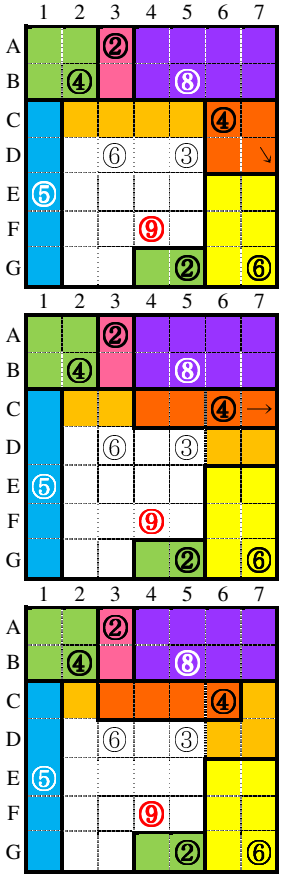
Langkah 10. *State* 9 masih punya "saudara" yaitu *state* 10. Ubah bentuk partisi ② di petak A3.

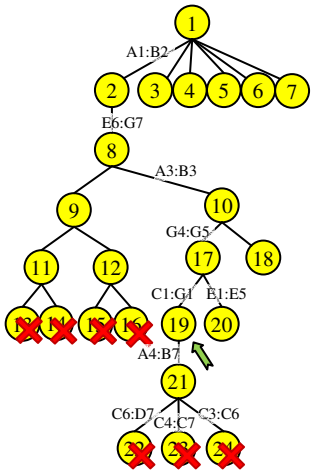
Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3



Langkah 11-13. Petak ②, ⑤, dan ⑧ masih mengikuti konsep optimasi greedy.

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	E1	B5	C6	F4	D5	D3

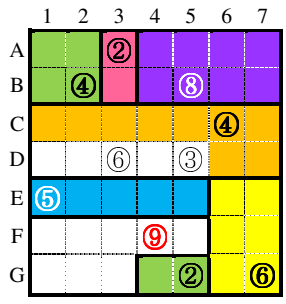


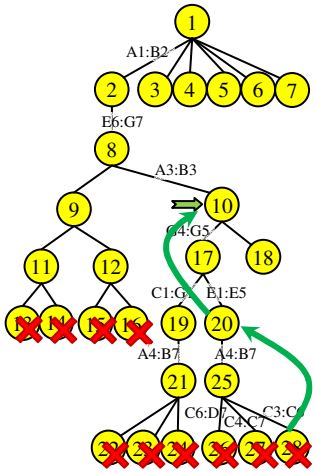


Langkah 14-16. Petak ④ ini masih bisa jalan, tetapi semua kemungkinan untuk ④ menyebabkan angka ⑨ berikutnya mati.

Setelah sampai di *state* 24 dan kehabisan akal, kembali *backtrack* ke state 19. Mundur 2 elemen tabel, proses angka ⑤.

Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	G5	**E1**	**B5**	C6	F4	D5	D3

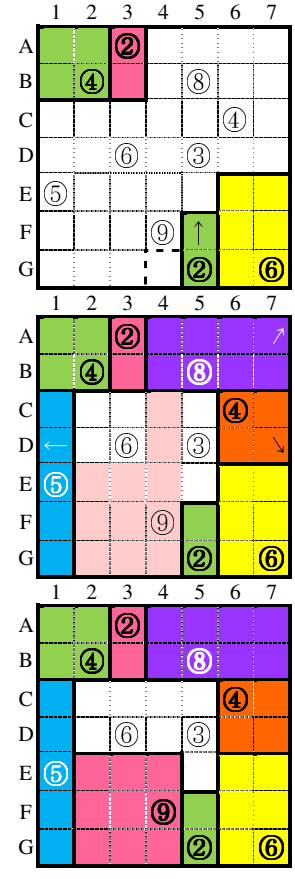


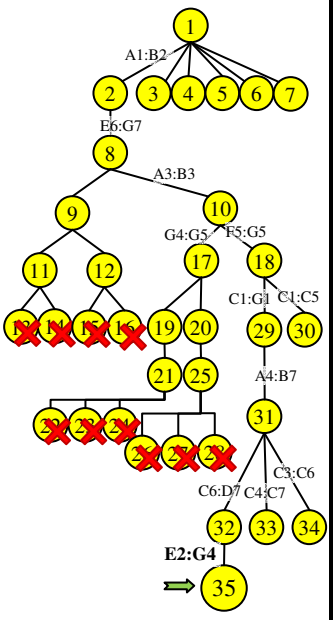


Langkah 17-19. Kita mencoba *state* 20 dengan mengubah partisi ⑤. Tetapi sama saja, semua kemungkinan partisi untuk ④ masih belum bisa membebaskan partisi untuk ⑨.

Setibanya di *state* 28 dan mati, kembali *backtrack* ke *state* 20. Rupanya *state* 19 dan 20 tidak punya saudara lagi yang selevel. Jadi, *backtrack* lagi ke kakeknya yaitu *state* 10.

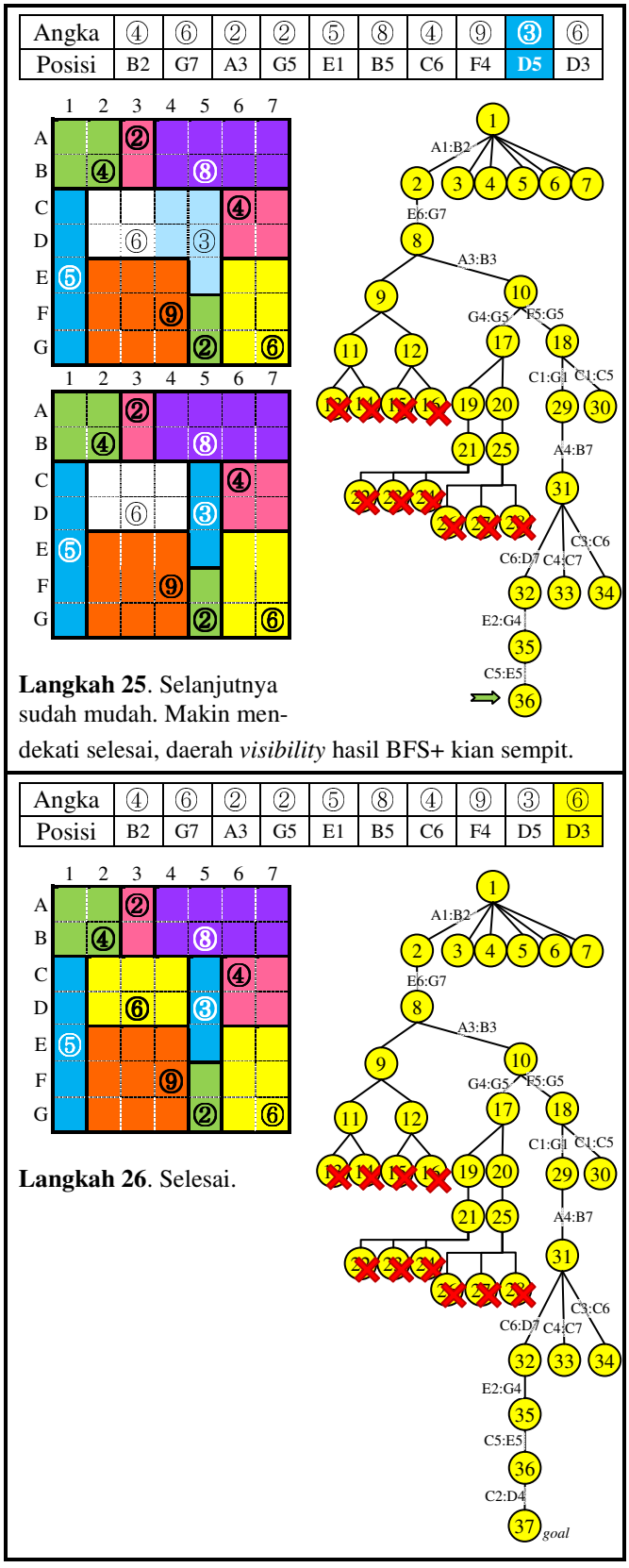
Angka	④	⑥	②	②	⑤	⑧	④	⑨	③	⑥
Posisi	B2	G7	A3	**G5**	E1	B5	C6	**F4**	D5	D3





Langkah 20-24. Begitu ki-ta masuk *state* 18, partisi ② di petak G5 yang menjadi sumber masalah akhirnya terselesaikan.

Pada langkah 21, 22, 23; angka ⑤, ⑧, dan ④ segera mendapat partisi yang mengarah ke ujung karena tunduk pada prinsip optimasi greedy. Dan akhirnya, BFS+ atas petak ⑨ membuahkan hasil, dimana ⑨ boleh mendapat partisi yang sesuai.



Langkah 25. Selanjutnya sudah mudah. Makin mendekati selesai, daerah *visibility* hasil BFS+ kian sempit.

Langkah 26. Selesai.

Gambar 19. Ilustrasi penyelesaian shikaku dengan algoritma combo

4. KESIMPULAN

Kebetulan menurut contoh shikaku 7 × 7 ini, terdapat 5x *backtrack* yang terjadi. Untuk matriks shikaku yang lain, hasilnya akan berbeda. Contohnya, geser saja petak ③ ke bawah, dari semula D5 menjadi E5. Bentuk partisi yang sesuai untuknya tetap sama, yaitu C5:E5. Bedanya, Anda hanya perlu 3x *backtrack*. Jadi, jumlah *backtrack* yang dijadikan indikator seberapa banyak "usaha sia-sia", sangat bergantung pada papan shikaku-nya. Meskipun demikian, umumnya algoritma ini masih lebih efisien daripada algoritma runut balik murni, disebabkan penghitungan secara *greedy* di kedua tahapannya.

Sejauh makalah ini dicetak, saya masih belum menemukan referensi tentang algoritma untuk puzzle shikaku. Entah saya salah memasukkan *keyword* di google, atau memang belum ada yang menelitinya. Bila memang belum pernah ada orang yang mengembangkan algoritma untuk shikaku, maka saya boleh-boleh saja memberi nama algoritma combo ini dengan sebutan "algoritma Ryan".

Memang shikaku masih kalah tenar dari suudoku, belum banyak orang yang mengetahui permainan ini. Semoga di kemudian hari ada orang yang menemukan algoritma yang lebih baik lagi.

REFERENSI

[1] Rinaldi Munir, "*Strategi Algoritma*", Program Studi Teknik Informatika ITB, 2009.

[2] http://www.nikoli.co.jp/en/puzzles/shikaku/

[3] Erik D. Demaine, Robert A. Hearn, "*Playing Games with Algorithm, Algorithmic Combinatorial Game Theory*". (http://erikdemaine.org/papers/AlgGameTheory_GONC3/paper.pdf), halaman 19-20.

[4] http://www.puzzle-shikaku.com/.

[5] http://homepage2.nifty.com/warai_kamosika/sikaku.htm