11 - Backtracking

[KOMS120403]

Desain dan Analisis Algoritma (2023/2024)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 11 (May 2024)

Daftar isi

Bagian 1. Prinsip dasar strategi *bactracking*

Prinsip dasar Backtracking

- Dengan teknik *exhaustive search*, kita mendaftar semua kandidat solusi, kemudian mengidentifikasi sebuah solusi yang memenuhi sifat tertentu yang diharapkan.
- Algoritma backtracking memberikan perbaikan pada exhaustive search.
- Dalam exhaustive search, semua kemungkinan solusi dieksplorasi dan dievaluasi satu per satu.
- Dalam backtracking, kita tidak memeriksa semua kemungkinan, hanya kemungkinan yang mengarah pada solusi. Simpul lain yang tidak mengarah ke solusi akan dipangkas.

Ide pokok:
Percabangan pada state-space tree dapat dipangkas segera setelah kita dapat menyimpulkan bahwa itu tidak dapat mengarah pada solusi.

Prinsip dasar Backtracking

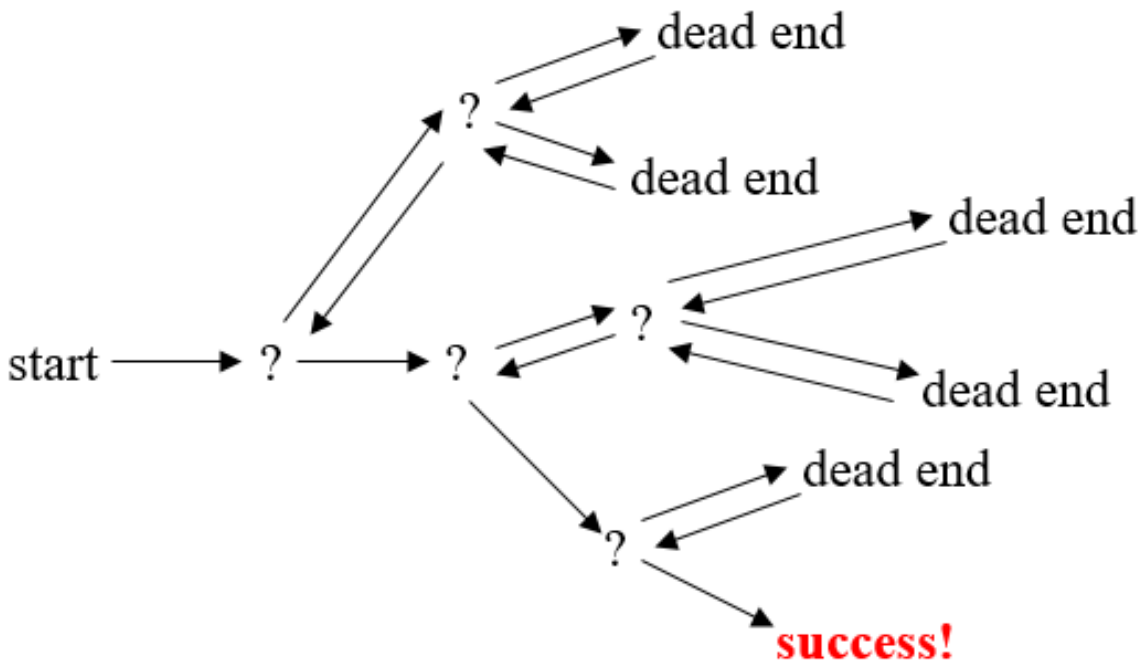


Figure: Ilustrasi algoritma backtracking (sumber: `https://miro.medium.com/`)

Representasi solusi

- Representasi solusi: sebuah output dapat dituliskan sebagai *n-tuple* $(x_1, x_2, \ldots, x_n)$ dimana setiap koordinat x_i adalah elemen dari beberapa "himpunan terbatas yang diurutkan secara linear (*finite linearly ordered set*)" (dilambangkan dengan S_i).

- Tuples: semua tupel solusi dapat memiliki panjang yang sama (misal pada masalah *n-queens* dan masalah *sirkuit Hamilton*) dan dapat pula memiliki panjang yang berbeda (misal pada masalah *subset-sum*).

Backtracking dalam DFS

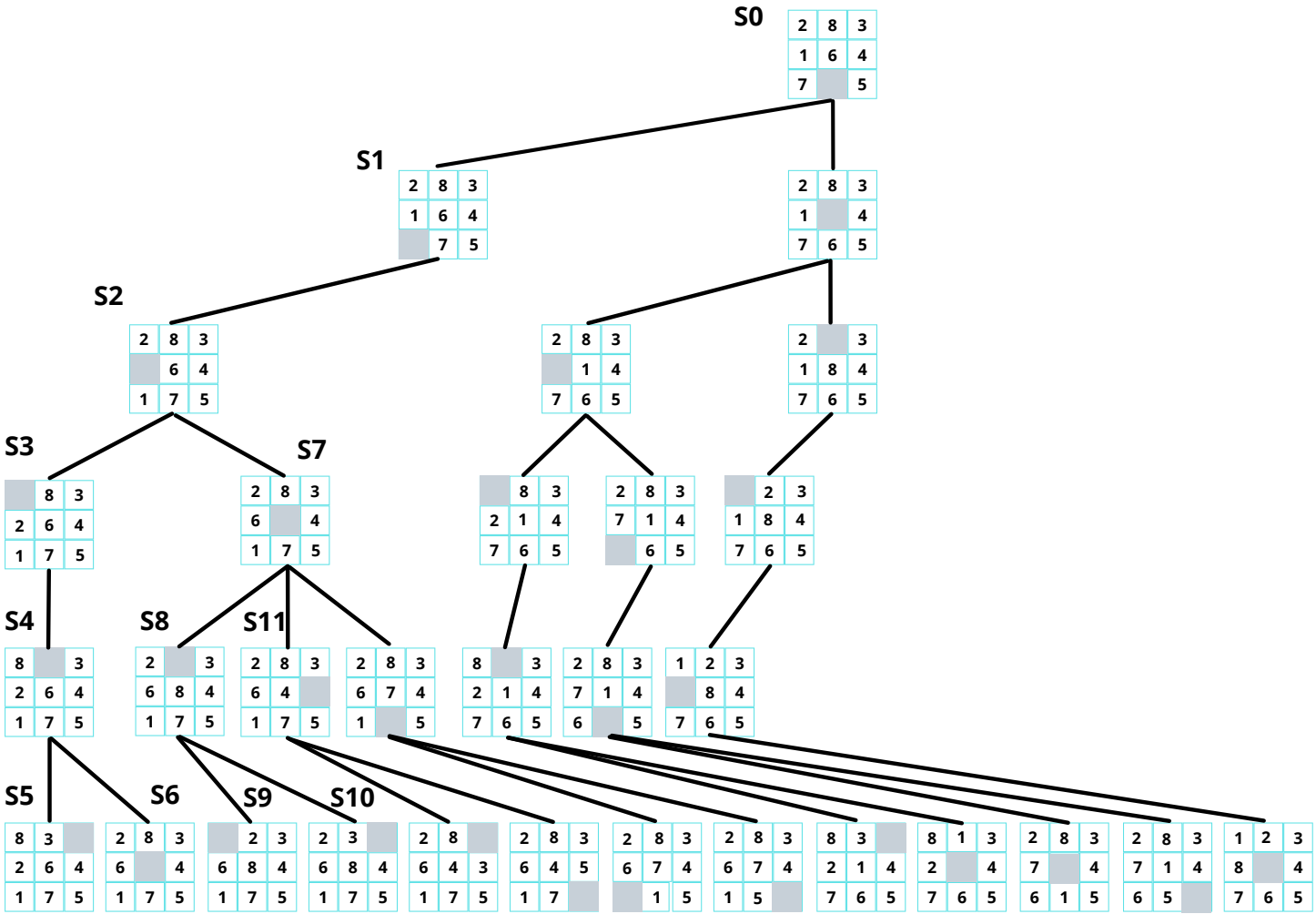
Backtracking dalam DFS digunakan dalam pencarian solusi masalah yang memiliki banyak kemungkinan penyelesaian.

Solusinya diperoleh dengan melihat pendekatan depth-first.

- Anda tidak memiliki cukup informasi untuk mengetahui langkah selanjutnya.
- Setiap keputusan mengarahkan Anda ke beberapa/banyak pilihan baru.
- Beberapa urutan pilihan mungkin menjadi solusi masalah.

Dalam DFS, backtracking digunakan sebagai cara metodologis untuk mencoba beberapa urutan keputusan.

Contoh backtracking dalam DFS



Bagian 2. State-space tree

State-space tree (1)

Backtracking dapat dilihat sebagai pencarian pada graf pohon, yang dimulai dari akar hingga ke daun (simpul solusi).

State-space tree

yakni merupakan graf pohon yang mewakili semua keadaan yang mungkin (solusi atau non-solusi) dari masalah, dari simpul akar (sebagai keadaan awal) ke simpul daun (sebagai keadaan terminal/akhir).

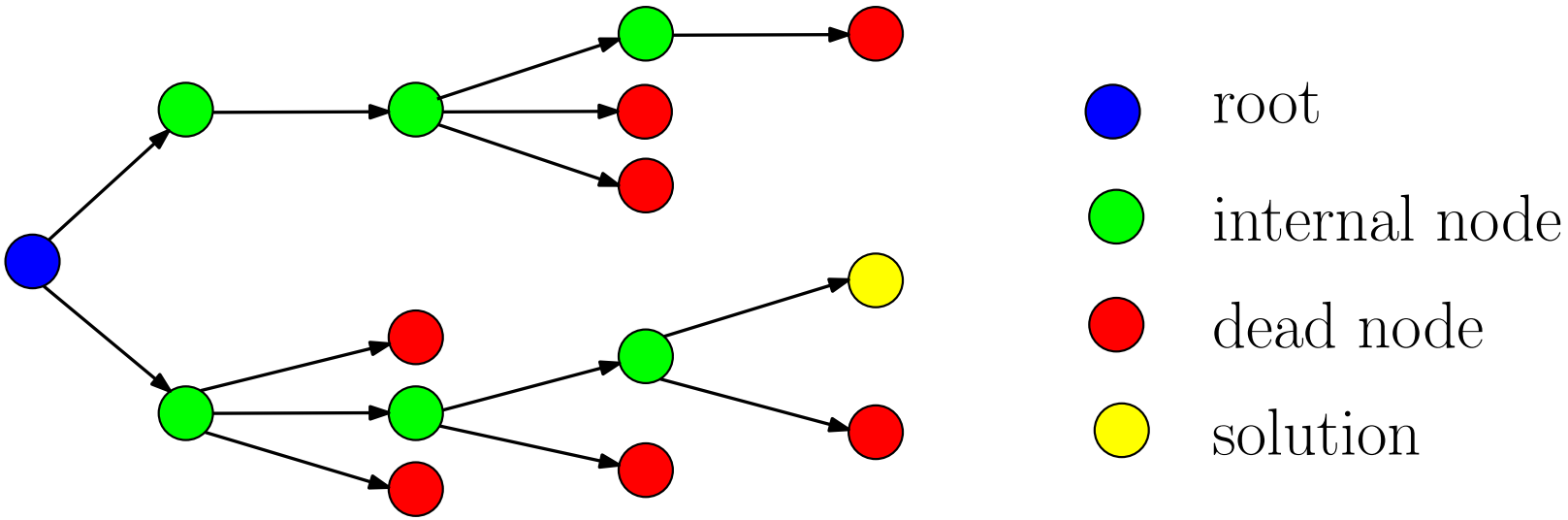
Sebuah algoritma backtracking menghasilkan (secara eksplisit atau implisit) **state-space tree**:

- simpulnya mewakili tupel yang dibangun sebagian oleh i koordinat pertama yang ditentukan oleh tindakan pada tahap sebelumnya;
- jika tupel $(x_1, x_2, \ldots, x_i)$ adalah kandidat solusi, maka algoritma menemukan elemen berikutnya dalam ruang S_{i+1} yang konsisten dengan nilai $(x_1, x_2, \ldots, x_i)$ dan fungsi kendala masalahnya, kemudian menambahkannya ke tupel sebagai koordinat yang ke-$(i+1)$;
- jika tidak ada elemen yang bisa diambil di ruang S_{i+1}, algoritma mundur untuk mempertimbangkan nilai berikutnya dari x_i, dan demikian seterusnya.

State-space tree (2)

- Akar (*root*) mewakili keadaan awal sebelum pencarian dimulai;
- Simpul internal
  - simpul dari level pertama dalam pohon mewakili pilihan yang dibuat untuk komponen pertama dari solusi;
  - simpul tingkat kedua mewakili pilihan untuk komponen kedua;
  - dan seterusnya...;
- Simpul daun (*leaves*) mewakili jalan buntu yang tidak menjanjikan atau solusi lengkap yang ditemukan oleh algoritma.

State-space tree (3)



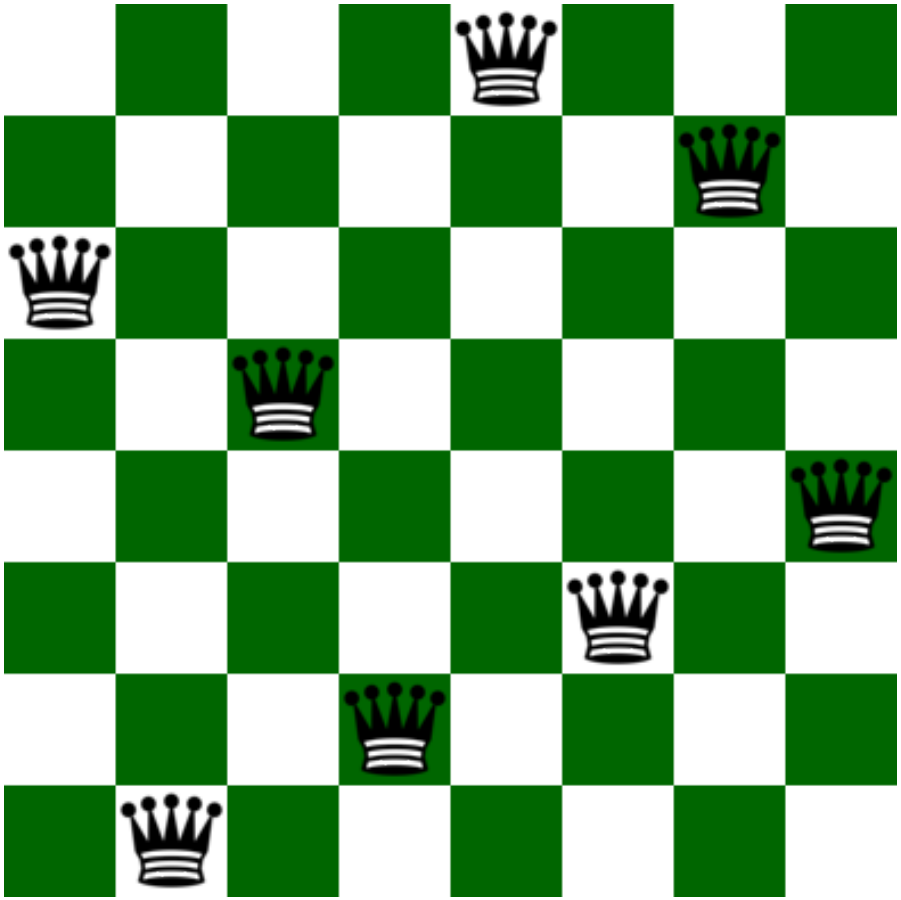
Tipe simpul state-space tree

- Promising node: sesuai dengan solusi yang dibangun sebagian yang mungkin masih mengarah pada solusi lengkap;
- Non-promising node: simpul mati (*dead node*)

State-space tree (4)

- Solusi dicari dengan membangkitkan simpul status (*state nodes*), sehingga menghasilkan lintasan dari akar ke daun;
- Untuk membangun simpul, aturan DFS diikuti;
- Simpul yang dihasilkan disebut simpul aktif (*live node*);
- Simpul aktif yang sedang diperluas disebut simpul perluasan (*expand-node*);
- Setiap kali expand-node diperluas, lintasan yang dihasilkan menjadi lebih panjang;
- Fungsi yang digunakan untuk "memangkas" sebuah expand-node disebut bounding function;
- Ketika sebuah simpul dimatikan, maka secara otomatis semua simpul anaknya akan dipangkas;
- Jika pembuatan jalur berakhir dengan simpul mati, maka dilakukan runut balik (backtrack) ke simpul induk;
- Simpul orang tua ini menjadi simpul perluasan baru;
- Pencarian dihentikan jika solusi ditemukan.

Bagian 3. *n*-Queens problem

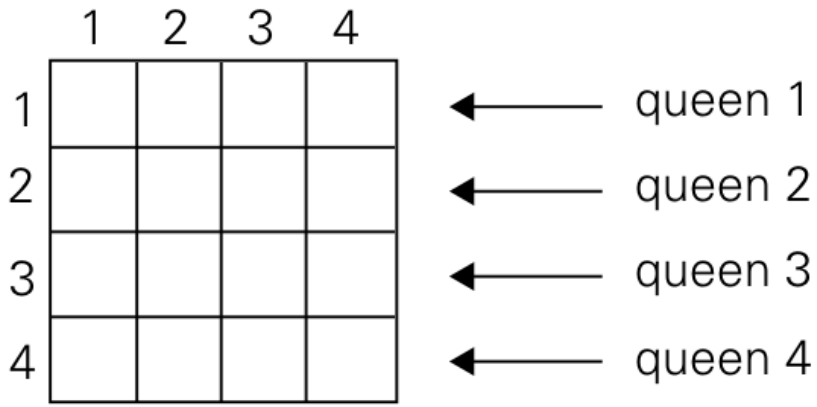


Masalah n-ratu (n-Queens problem)

Permasalahan

Tempatkan n ratu di papan catur berukuran $n \times n$, sehingga tidak ada dua ratu yang saling menyerang (dengan berada di baris yang sama, di kolom yang sama, atau di diagonal yang sama).

- Untuk $n = 1$, masalahnya memiliki solusi trivial
- Untuk $n = 2, 3$, masalah tidak memiliki solusi
- Bagaimana jika $n = 4$?



Algoritma

START

1. mulai dari kolom paling kiri
2. jika semua ratu ditempatkan, *return* TRUE atau PRINT konfigurasinya
3. periksa semua baris di kolom saat ini
    1. jika ratu ditempatkan dengan aman, tandai baris dan kolom; dan secara rekursif periksa apakah kita mendekati konfigurasi saat ini, apakah kita mendapatkan solusi atau tidak
    2. jika penempatan menghasilkan solusi, kembalikan TRUE
    3. jika penempatan tidak menghasilkan solusi, pangkas tanda dan coba baris lain
4. jika semua baris dicoba dan solusi tidak diperoleh, *return* FALSE dan lakukan *backtrack*

END

Masalah *n*-Ratu

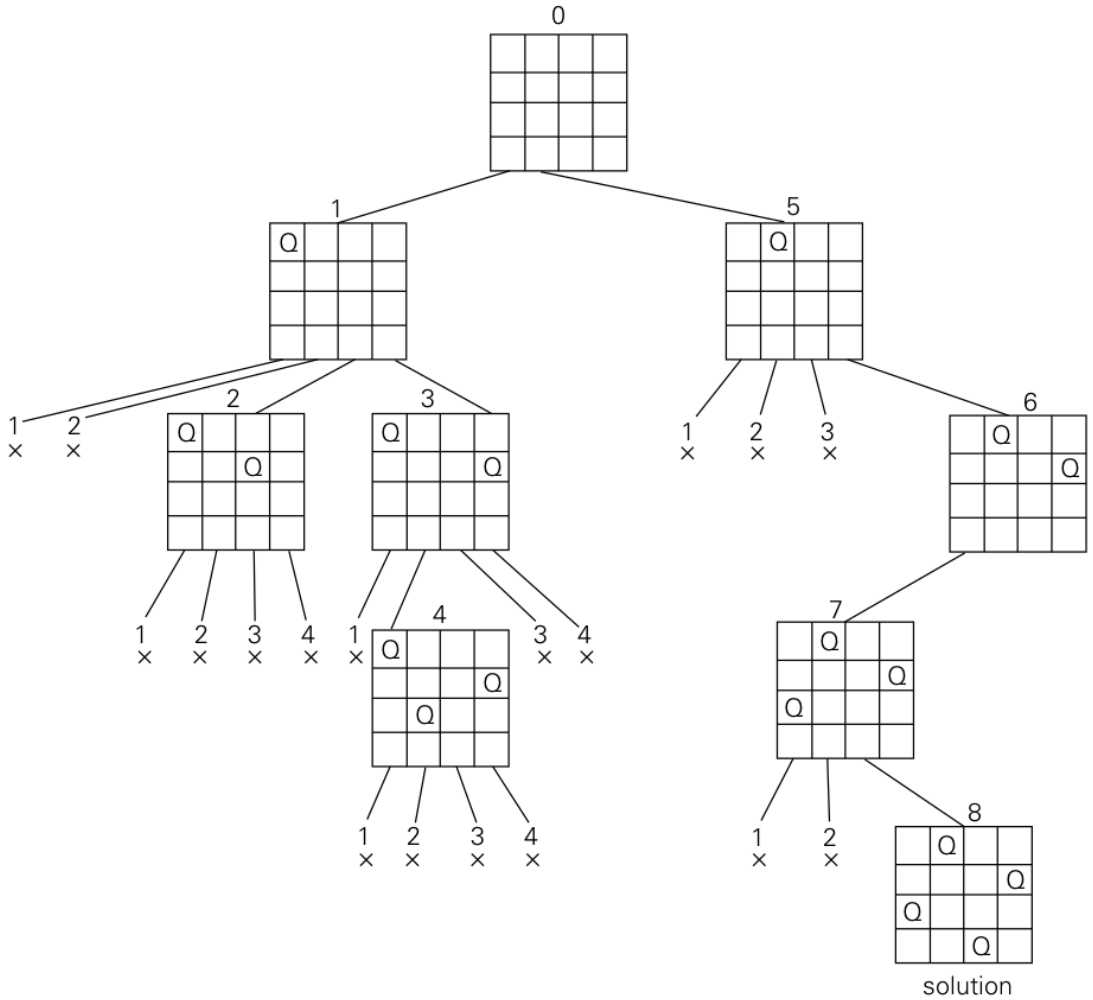


Figure: State-space tree untuk memecahkan masalah 4-ratu dengan backtracking. Tanda $\times$ menunjukkan usaha yang gagal untuk menempatkan sebuah ratu di kolom yang ditunjukkan. Angka di atas simpul menunjukkan urutan pembangkitan simpul.

Masalah n-Ratu

1. Kita mulai dengan papan kosong dan kemudian tempatkan ratu 1 di posisi pertama yang mungkin dari barisnya, yaitu di kolom 1 dari baris 1.
2. Lalu kita tempatkan ratu 2, setelah gagal mencoba kolom 1 dan 2, di posisi pertama yang dapat diterima untuk itu, yaitu kuadrat $(2, 3)$, kuadrat di baris 2 dan kolom 3.
3. Ini terbukti menjadi jalan buntu karena tidak ada posisi yang dapat diterima untuk ratu 3.
4. Jadi, algoritma mundur dan menempatkan ratu 2 di posisi berikutnya yang memungkinkan di $(2, 4)$.
5. Kemudian ratu 3 ditempatkan di $(3, 2)$, yang terbukti menjadi jalan buntu lainnya.
6. Algoritma kemudian mundur ke ratu 1 dan memindahkannya ke $(1, 2)$.
7. Ratu 2 lalu pergi ke $(2, 4)$, ratu 3 menjadi $(3, 1)$, dan ratu 4 menjadi $(4, 3)$, yang merupakan solusi untuk masalah tersebut.

Bagian 4. Contoh permasalahan lain

Masalah sirkuit Hamilton

Permasalahan

Diberikan graf terhubung G, temukan sirkuit Hamilton di G. (Ingat bahwa sirkuit Hamilton adalah sirkuit yang mengunjungi semua simpul G tepat satu kali dan kembali ke titik awal.)

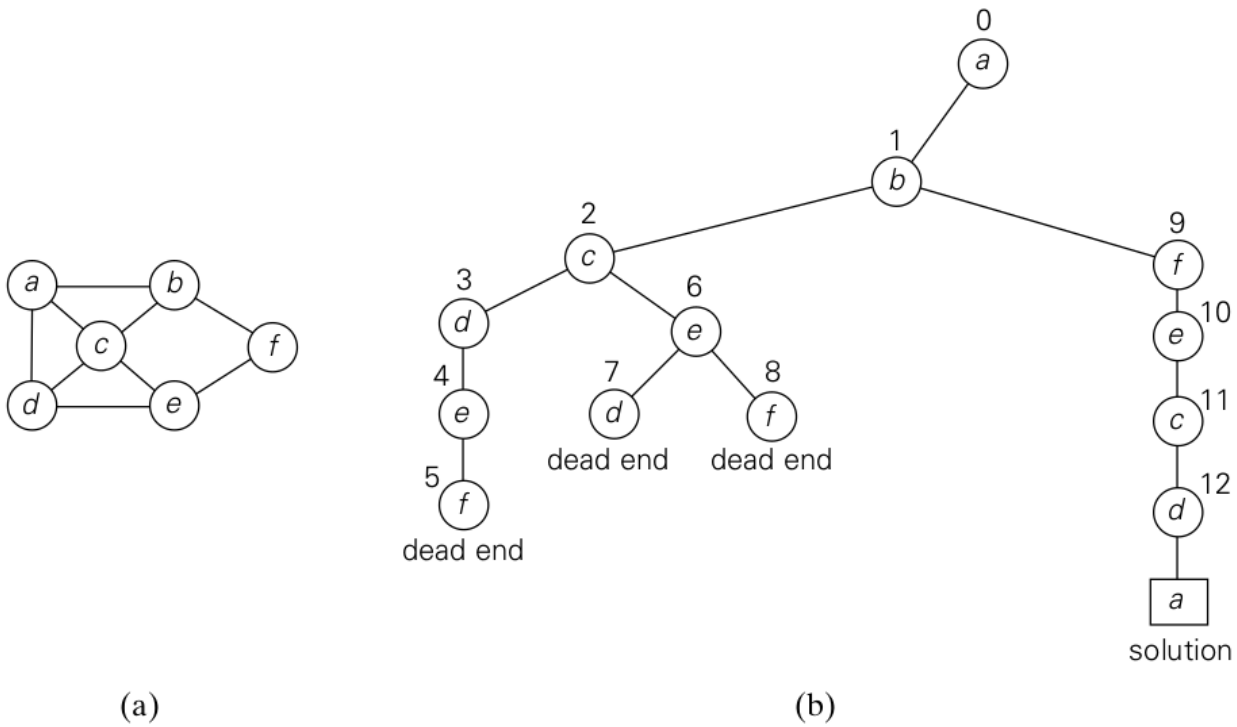


Figure: (a) Graf. (b) State-space tree untuk menemukan sirkuit Hamilton. Angka di atas simpul graf pohon menunjukkan urutan pembangkitan simpul.

Subset-sum problem (1)

Permasalahan

Diberikan suatu himpunan n bilangan bulat positif $A = \{a_1, \ldots, a_n\}$ *dan bilangan bulat positif d. Temukan subhimpunan dari A yang jumlahnya sama dengan d.*

Contoh 1: Diberikan $A = \{1, 2, 5, 6, 8\}$, $d = 9$, solusinya adalah: $\{1, 2, 6\}$ dan $\{1, 8\}$.

Contoh 2: Diberikan $A = \{3, 5, 6, 7\}$, $d = 15$, solusinya adalah: $\{3, 5, 7\}$.

Subset-sum problem (2)

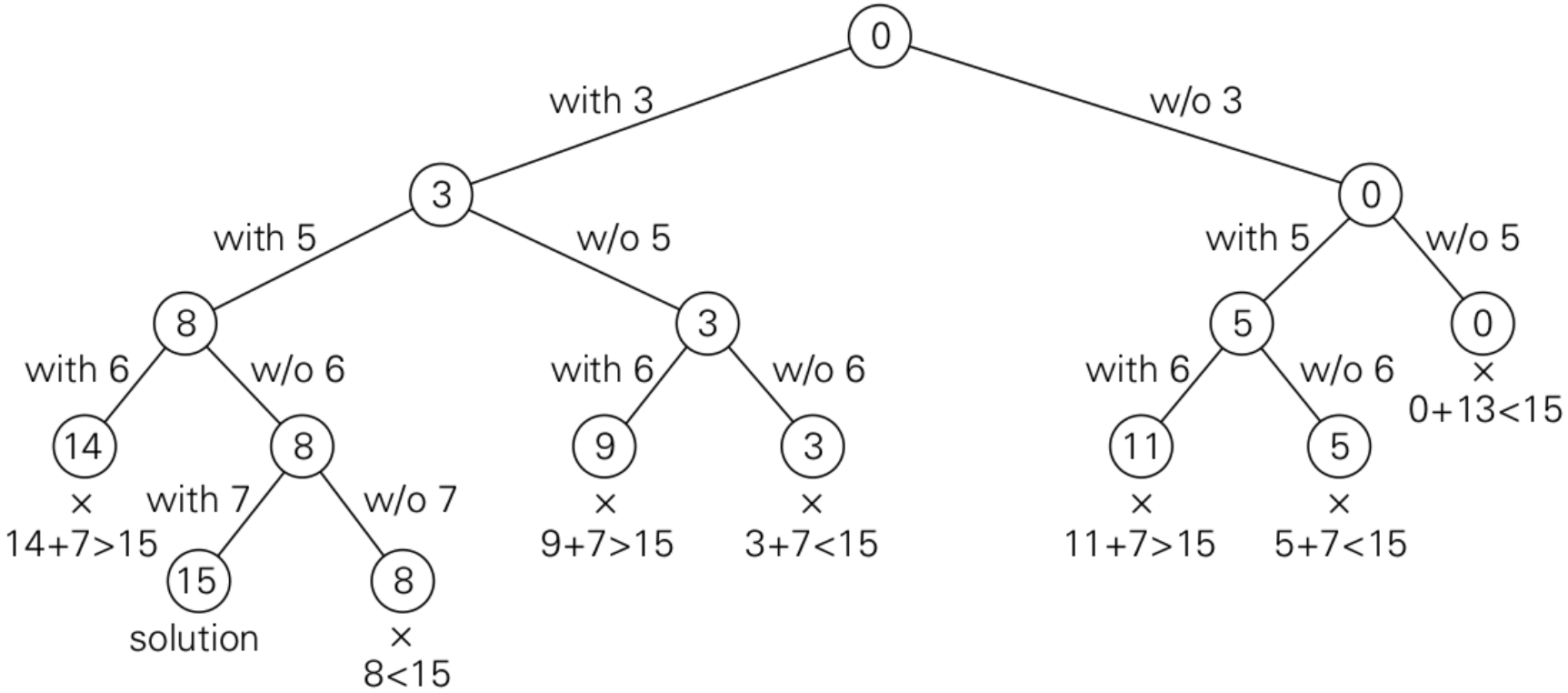


Figure: Pohon ruang-status lengkap dari algoritma backtracking yang diterapkan pada instance $A = \{3, 5, 6, 7\}$ dan $d = 15$ dari masalah Subset-sum. Angka di dalam sebuah simpul menunjukkan jumlah dari elemen yang sudah termasuk dalam himpunan bagian yang diwakili oleh simpul tersebut. Ketidaksetaraan di bawah daun menunjukkan alasan penghentiannya (*prunning the node*).

Subset-sum problem (3)

Sebuah lintasan dari akar ke simpul pada level ke-i dari pohon menunjukkan i nomor pertama yang dimasukkan atau tidak dimasukkan dalam himpunan bagian yang diwakili oleh simpul tersebut.

Kita mencatat nilai s, jumlah dari angka-angka ini, pada simpul.

- Jika s sama dengan d, kita memiliki solusi untuk masalah tersebut. Kita dapat melaporkan hasil ini dan berhenti atau,
- Jika semua solusi perlu ditemukan, lanjutkan dengan menelusuri kembali ke simpul induk.
- Jika s tidak sama dengan d, kita dapat mengakhiri simpul sebagai tidak menjanjikan jika salah satu dari dua ketidaksetaraan berikut terpenuhi:

$$s + a_{i+1} > d \text{ (jumlah } s \text{ terlalu besar)}$$

$$s + \sum_{j=i+1}^{n} a_j < d \text{ (jumlah } s \text{ terlalu kecil)}$$

Bagian 5. Kerangka algoritma Backtracking

Kerangka algoritma backtracking

Algorithm 1 Backtracking

```
1:  procedure BACKTRACK(X[1..i])
2:      input: X[1..i]: the first i promising components of a solution
3:      output: all the tuples representing the problem's solution
4:      if X[1..i] is a solution then
5:          write (X[1..i])
6:      else
7:          for each x ∈ S_{i+1} consistent with X[1..i] and the constraints do
8:              X[i + 1] ← x
9:              BACKTRACK(X[1..i + 1])
10:         end for
11:     end if
12: end procedure
```

Kompleksitas waktu

Backtracking pada dasarnya adalah pencarian lengkap yang dilakukan di atas ruang pencarian. Jadi kompleksitas waktu dari algoritma backtracking **ditentukan oleh ukuran ruang pencarian**.

Misalnya, dalam masalah n-queens dan Hamiltonian, ukuran ruang pencarian adalah sekitar $\mathcal{O}(n!)$.

> *Secara intuitif, ratu pertama memiliki n penempatan, ratu kedua tidak boleh berada di kolom yang sama dengan yang pertama, sehingga ratu kedua memiliki kemungkinan* $n-1$*, dan seterusnya, dengan kompleksitas waktu sebesar* $\mathcal{O}(n!)$*.*

Kelebihan & kekurangan

Kelebihan

- Biasanya diterapkan pada masalah kombinatorial yang sulit dimana tidak ada algoritma yang efisien untuk menemukan solusi yang tepat.
- Berbeda dengan pendekatan pencarian lengkap, backtracking setidaknya memiliki harapan untuk menyelesaikan beberapa contoh ukuran tak-trivial dalam jumlah waktu yang dapat diterima (terutama untuk masalah optimasi).
- Bahkan jika backtracking tidak menghilangkan elemen apa pun dari ruang keadaan masalah dan akhirnya menghasilkan semua elemennya, ia menyediakan teknik khusus untuk melakukannya.

Kekurangan

- Backtracking **bukan** teknik yang efisien (walaupun berhasil diimplementasikan pada masalah sebelumnya).
- Pada kasus terburuk, mungkin harus menghasilkan semua kandidat yang mungkin dalam ruang keadaan masalah yang tumbuh secara eksponensial (atau lebih cepat).

end of slide...