1 - Pengenalan Desain dan Analisis Algoritma

[KOMS120403]

Desain dan Analisis Algoritma (2022/2023)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 1 (March 2023)

Kontrak perkuliahan

- **Kredit:** 3 SKS
- **Pengampu:** Dewi Sintiari
  - email: nld.sintiari@gmail.com
- **Evaluasi:**
  - Presence ($\geq 75\%$) + attitude: 20%
  - Quiz / Take-home assignments (theoretical & programming): 40%
  - Midterm exam (written): 20%
  - Final exam (written): 20%
  - Bonus: writing an article, writing in wikipedia?
  - Grade = 20% Presence + 40% Assignments + 20% Midterm + 20% Final + Bonus

Bagian 2. Apa itu algoritma, mengapa algoritma dibutuhkan?

Apa itu algoritma, mengapa algoritma dibutuhkan?

Contoh algoritma sederhana

Recipe of Indomie goreng

Ingredients

Steps

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....

Result

Apa itu algoritma, mengapa algoritma dibutuhkan?

Contoh algoritma sederhana

Recipe of Teh celup

Ingredients

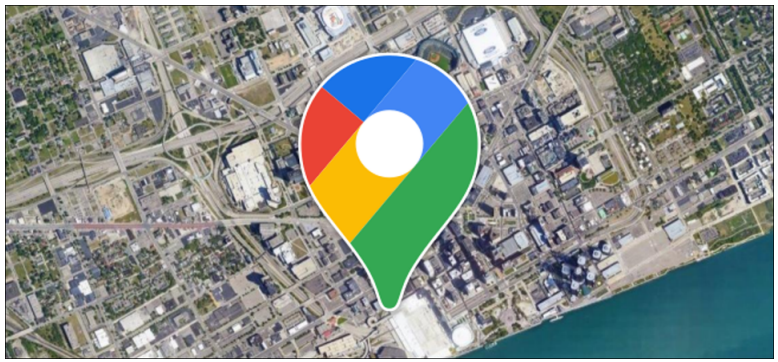
Steps

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....

Result

Apa itu algoritma, mengapa algoritma dibutuhkan?

Audio & video compression algorithms allow transmitting live video across internet



Route findings algorithms are used to find a route between two cities

Optimization and scheduling algorithms are used to arrange the schedule of airlines in the world

Apa itu algoritma, mengapa algoritma dibutuhkan?

Algoritma adalah prosedur

Definisi

Algoritma adalah urutan terbatas dari instruksi yang terdefinisi dengan baik, biasanya digunakan untuk memecahkan kelas masalah tertentu atau untuk melakukan komputasi.

- Dalam Ilmu Komputer, suatu algoritma memberi komputer serangkaian instruksi khusus, yang memungkinkan komputer melakukan segalanya.
- Program komputer = algoritma yang ditulis dalam bahasa pemrograman yang dapat dimengerti oleh komputer.
- Sebuah algoritma dijabarkan dalam bentuk pseudocode.

Algorithmic thinking adalah kemampuan untuk menentukan langkah-langkah yang jelas untuk menyelesaikan suatu masalah. Ini memungkinkan kita untuk memecah masalah dan membuat konsep solusi.

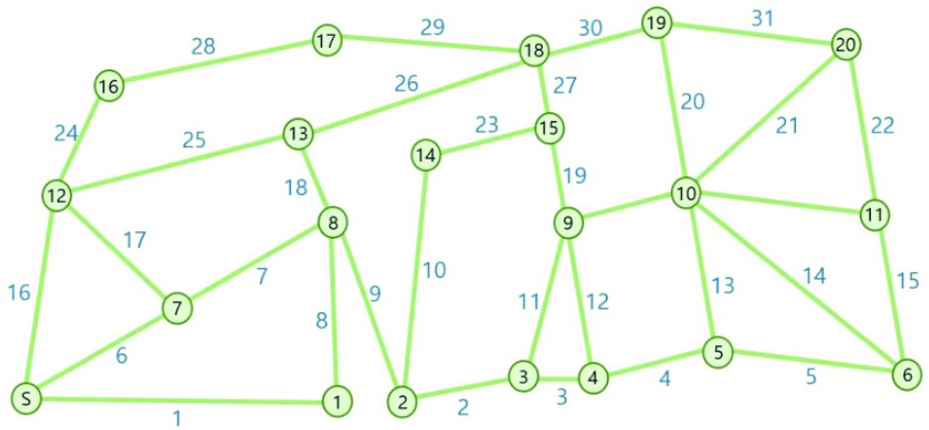
Komponen penting dalam algoritma

- **Initial situation**/condition/input (*Initial state*): sebanyak nol atau lebih masukan (input).
- **Final situation**/condition/output (*Output state*): setidaknya menghasilkan sebuah output
- **Definiteness**: setiap langkah harus jelas, terdefinisi dengan baik dan tepat, tidak ambigu.
- **Finiteness**: harus memiliki jumlah langkah yang terbatas dan harus berakhir setelah waktu yang terbatas.
- **Effectiveness**: setiap langkah harus sederhana dan harus memakan waktu yang terbatas.
- **Constraints** diberikan di awal dan selama menyusun algoritma (*batasan dan asumsi*)

Bagian 3. Contoh masalah algoritmik klasik

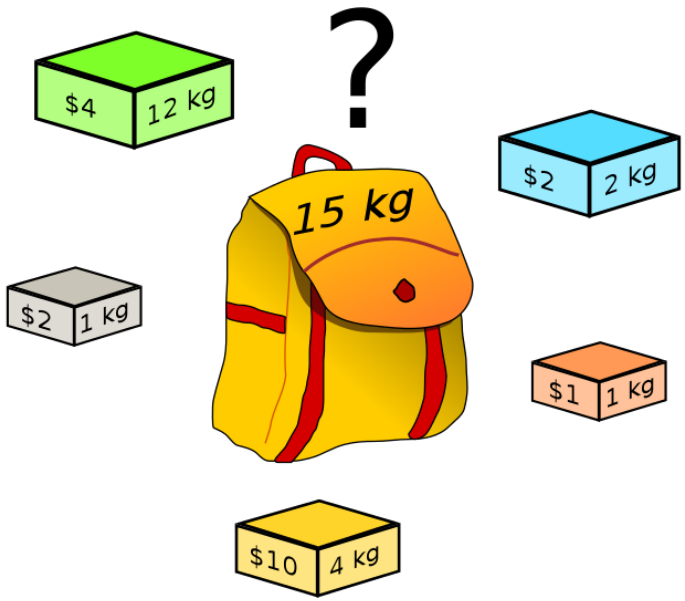
1. Traveling Salesman Problem (TSP)

Diberikan daftar kota dan jarak antara setiap pasangan kota, berapakah rute terpendek yang mengunjungi setiap kota tepat satu kali dan kembali ke kota asal?



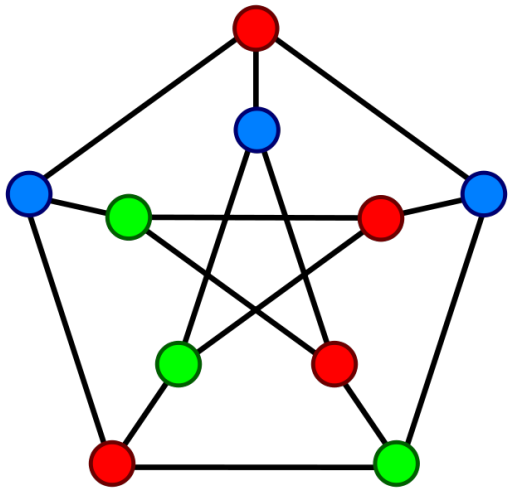
2. Integer Knapsack Problem

Diberikan satu set item, masing-masing dengan bobot dan nilai, tentukan jumlah setiap item yang termasuk dalam koleksi sehingga bobot totalnya kurang dari atau sama dengan batas yang diberikan dan nilai totalnya sebesar mungkin.



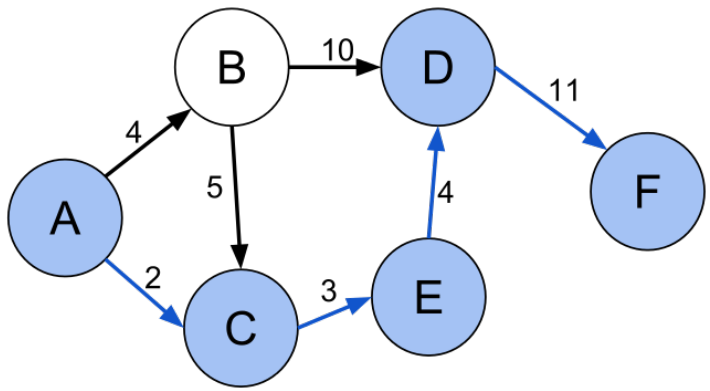
3. Masalah pewarnaan graf

Diberikan graf dengan n simpul, tentukan jumlah minimum warna berbeda yang diperlukan untuk mewarnai simpul sedemikian rupa sehingga tidak ada dua simpul yang bertetangga memiliki warna yang sama. Kita juga ingin menemukan cara untuk mewarnai graf.



4. Shortest path problem

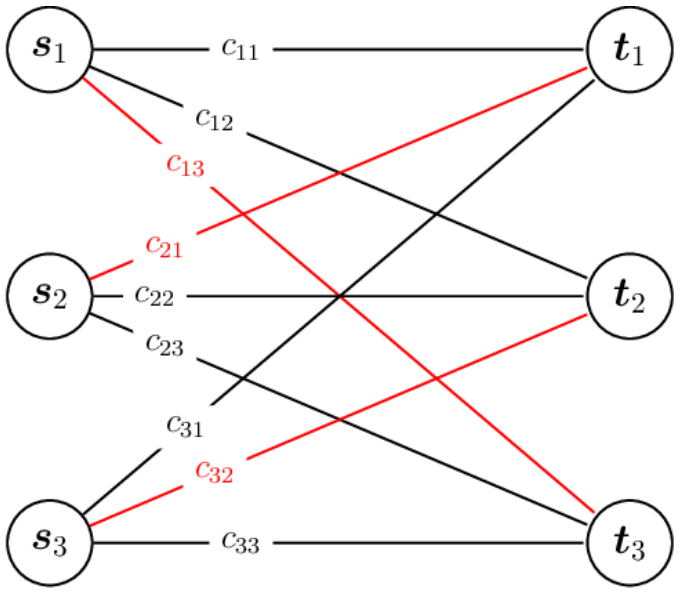
Diberi graf dengan n simpul, temukan lintasan antara dua simpul (atau simpul) dalam graf sedemikian rupa sehingga jumlah bobot sisi penyusunnya diminimalkan.



5. Assignment problem

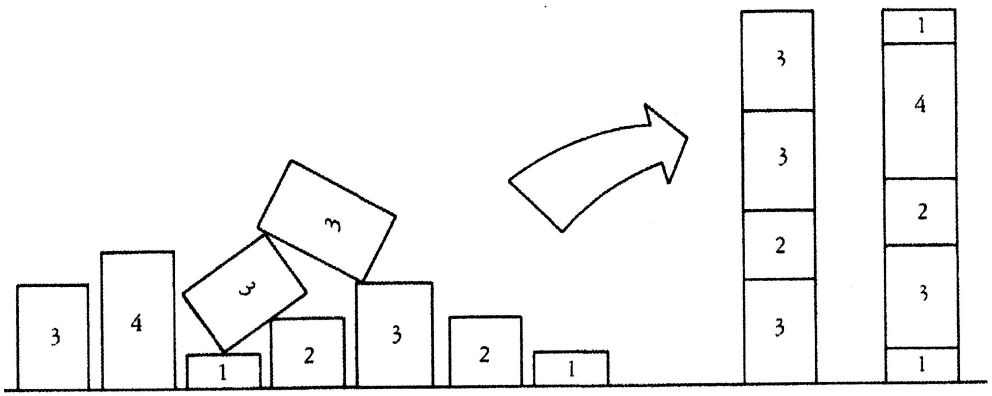
Diberi n staf dan n tugas. Siapa pun dapat ditugaskan untuk melakukan tugas apa pun, menimbulkan biaya $c(i, j)$ yang dapat bervariasi tergantung pada pemasangan staf(s_i)-tugas(t_j).

Diperlukan untuk melakukan tugas sebanyak mungkin dengan menugaskan satu staf untuk setiap tugas, sedemikian rupa sehingga total biaya $\sum c(i, j)$ dari penugasan tersebut diminimalkan.



6. Partition problem

Diberi multiset S bilangan bulat positif, putuskan apakah S dapat dipartisi menjadi dua himpunan bagian S_1 dan S_2 sehingga jumlah angka di S_1 sama dengan jumlah angka di S_2.



Bagian 4. Karakteristik algoritma

Tujuan studi tentang algoritma

1. Memahami ide dasar atau **alur permasalahan**.
2. Menemukan **pendekatan pemecahan masalah**. Desain yang baik dapat menghasilkan solusi yang baik.
3. Memahami **prinsip dasar** perancangan algoritma.
4. **Membandingkan** kinerja algoritma dengan menerapkan berbagai teknik.
5. **Meningkatkan efisiensi** teknik yang ada.
6. **Metode deskripsi** terbaik tanpa menjelaskan detail implementasi.
7. **Mengukur perilaku (atau kinerja)** metode dalam semua kasus (kasus terbaik, kasus terburuk, kasus rata-rata)
8. Dapat **mengukur dan menganalisis kompleksitas (waktu dan ruang)** dari masalah mengenai ukuran input tanpa mengimplementasikan dan menjalankannya; itu akan mengurangi biaya desain.

Tujuan studi tentang algoritma

Peran DAA:

- Perancangan algoritma untuk memecahkan masalah di Ilmu Komputer.
- Merancang dan menganalisis logika tentang cara kerja program sebelum mengembangkan kode sebenarnya.
- Mengklasifikasikan algoritma berdasarkan tujuan/desain/kompleksitasnya/dll..

Coba Anda telaah maksud dari setiap poin di atas.

Tujuan studi tentang algoritma

Desain algoritma: metode atau proses matematis untuk pemecahan masalah dan algoritma rekayasa.

Prosedur:

1. Langkah 1: Deskripsikan masalah. Langkah ini jauh lebih sulit daripada yang terlihat.
2. Langkah 2: Analisis masalahnya.
3. Langkah 3: Kembangkan algoritma tingkat tinggi.
4. Langkah 4: Sempurnakan algoritma dengan menambahkan lebih banyak detail.
5. Langkah 5: Tinjau algoritma.

Tujuan studi tentang algoritma

Analisis algoritma:

- Kebenaran
  - Apakah hubungan input/output cocok dengan persyaratan algoritma?
- Jumlah pekerjaan yang dilakukan (*kompleksitas* algoritma)
  - Jumlah operasi dasar untuk melakukan tugas
- Jumlah ruang yang digunakan
  - Memori digunakan
- Kesederhanaan, kejelasan
  - Verifikasi dan implementasi
- Optimal
  - Apakah mungkin untuk melakukan yang lebih baik?

Tujuan studi tentang algoritma

Apa yang membuat algoritma bagus?

Dua poin penting dalam membangun algoritma

- Correctness (kebenaran)
- Efficiency (efisiensi)

Tujuan studi tentang algoritma

Analisis algoritma: Kebenaran

Definisi

Algoritma untuk menyelesaikan masalah P dikatakan benar jika dan hanya jika untuk semua contoh masalah $i \in I$, algoritma tersebut berhenti dan menghasilkan output yang benar $o \in O$.

Pembuktian kebenaran dilakukan dengan menggunakan formal matematic proof:

- Counterexample (bukti *tidak langsung*)
- Induksi (bukti *langsung*)
- Invariansi *loop*

Pendekatan lain: dibuktikan dengan kasus/pencacahan, dengan rantai iffs, dengan kontradiksi, dengan kontrapositif.

Tujuan studi tentang algoritma

Analisis algoritma: Kompleksitas ruang/waktu

Dua parameter mendasar yang menjadi dasar kita dapat menganalisis **efisiensi** suatu algoritma:

- Space Complexity: jumlah ruang (penyimpanan) yang diperlukan oleh algoritma untuk dijalankan hingga selesai.
- Time Complexity: fungsi ukuran input n yang mengacu pada jumlah waktu yang dibutuhkan oleh suatu algoritma untuk berjalan hingga selesai.
- atau sumber daya lain yang diperlukan untuk menjalankannya

Coba Anda bandingkan kedua algoritma berikut

Bagaimana cara menghitung fpb dari dua angka m dan n?

Prosedur sekolah menengah:

1. Tentukan faktor prima dari m.
2. Temukan faktor prima dari n.
3. Identifikasi semua faktor persekutuan dalam dua perluasan utama yang ditemukan di Langkah 1 dan 2. (Jika p adalah faktor persekutuan yang terjadi p_m dan p_n kali dalam m dan n, masing-masing, itu harus diulang $\min\{p_m, p_n\}$ kali.)
4. Hitung produk dari semua faktor umum dan kembalikan sebagai fpb dari angka yang diberikan.

Coba Anda bandingkan kedua algoritma berikut

Bagaimana cara menghitung fpb dari dua angka m dan n?

Algoritma Euclid

1. Tetapkan nilai $\min\{m, n\}$ ke t.
2. Bagi m dengan t. Jika sisa pembagian ini adalah 0, lanjutkan ke Langkah 3; jika tidak, lanjutkan ke Langkah 4.
3. Bagi n dengan t. Jika sisa pembagian ini adalah 0, kembalikan nilai t sebagai jawaban dan hentikan; jika tidak, lanjutkan ke Langkah 4.
4. Turunkan nilai t sebanyak 1. Lanjutkan ke Langkah 2.

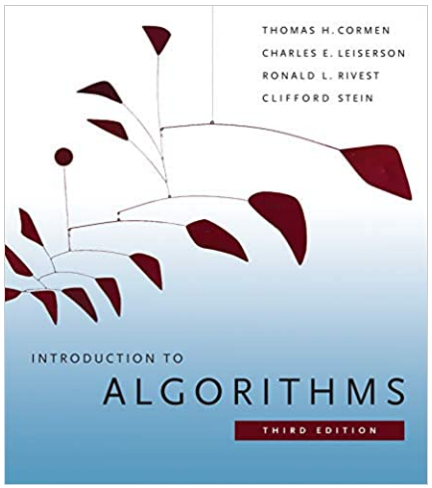
Bagian 5. Rencana pembelajaran semester, rujukan dan ringkasan

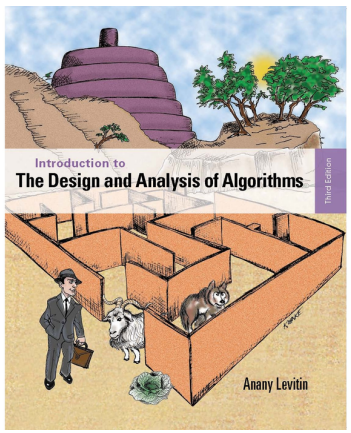
Outline of the semester

1. Pengantar desain dan analisis algoritma
2. Kompleksitas analisis algoritma
3. Algoritma Brute Force
4. Algoritma Rekursif
5. Algoritma *Divide-and-conquer*, *Decrease-and-conquer*, dan Transform-and-conquer
6. Algoritma BFS dan DFS, Backtracking, Branch & Bound
7. Pemrograman dinamis
8. Algoritma *Greedy*
9. Algoritma pada struktur graf
10. Teori kompleksitas komputasional (P, NP, NP-C)

1. Rujukan

- Introduction to Algorithms (Thomas H. Cormen, C. E. Leiserson, R. Rivest, C. Stein), The MIT Press, 1989.
- Introduction to the Design and Analysis of Algorithms (Anany Levitin), Pearson, 2012.





- Kuliah pengantar Strategi Algoritma (Rinaldi Munir ITB)
- e-Modul Struktur Data dan Analisis Algoritma (Made Windu A. Kesiman, PTI Undiksha)

2. Analisis kompleksitas

Menentukan fungsi kompleksitas waktu dan kompleksitas ruang. algoritma dikatakan *efisien* ketika nilai fungsi ini kecil, atau tumbuh lambat dibandingkan dengan pertumbuhan ukuran input.

- n: ukuran masukan
- $T(n)$: jumlah perhitungan/langkah (perbandingan, operasi aritmatika, mengakses array, dll.)
- $S(n)$: memori/ruang penyimpanan

- Mengukur kompleksitas: best-case, worst-case, dan average-case
- Kita biasanya memperkirakan kompleksitas *asimptotik*, yaitu, untuk memperkirakan fungsi kompleksitas untuk input besar yang berubah-ubah. Kita notasi menggunakan Big O (*upper-bound*), Big-omega (*lower-bound*) dan Big-theta (*tight-bound*).

3. Algoritma Brute Force/Exhaustive search

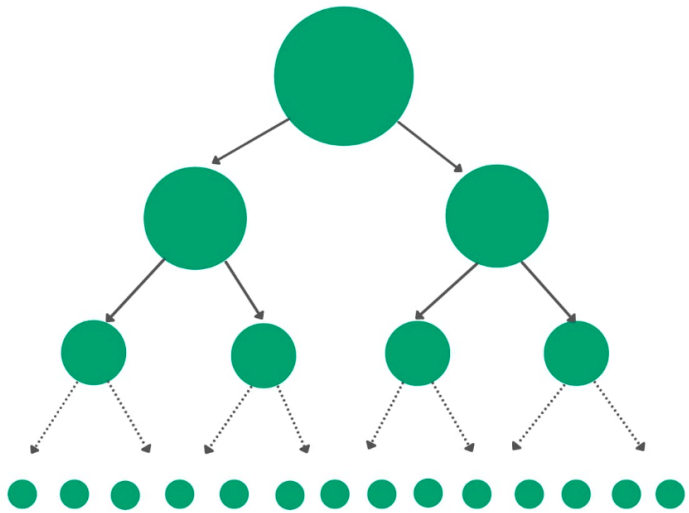
Ini adalah jenis algoritma yang paling dasar dan paling sederhana. Ini secara sistematis menghitung semua kandidat yang mungkin untuk solusi dan memeriksa apakah setiap kandidat adalah solusi dari pernyataan masalah.

Contoh: Diberi kunci PIN 4 digit, dimana digit yang akan dipilih dari 0-9. Brute force akan mencoba semua kemungkinan kombinasi satu per satu seperti 0001, 0002, 0003, 0004, dan seterusnya sampai kita mendapatkan PIN yang tepat (paling banyak ada 10.000 percobaan).

4. Algoritma Rekursif

Dalam Ilmu Komputer, *recursion* adalah metode penyelesaian masalah di mana solusinya bergantung pada solusi untuk kasus yang lebih kecil dari masalah yang sama.

Ini adalah algoritma yang menyebut dirinya sendiri dengan nilai input "lebih kecil (atau lebih sederhana)", dan yang memperoleh hasil untuk input saat ini dengan menerapkan operasi sederhana ke nilai yang dikembalikan untuk input yang lebih kecil (atau lebih sederhana).



5. Algoritma divide-and-conquer

Ini bekerja dengan secara rekursif memecah masalah menjadi dua/lebih sub-masalah dari jenis yang sama/terkait, sampai ini menjadi cukup sederhana untuk diselesaikan secara langsung. Solusi dari sub-masalah tersebut kemudian combined untuk memberikan solusi dari masalah aslinya.

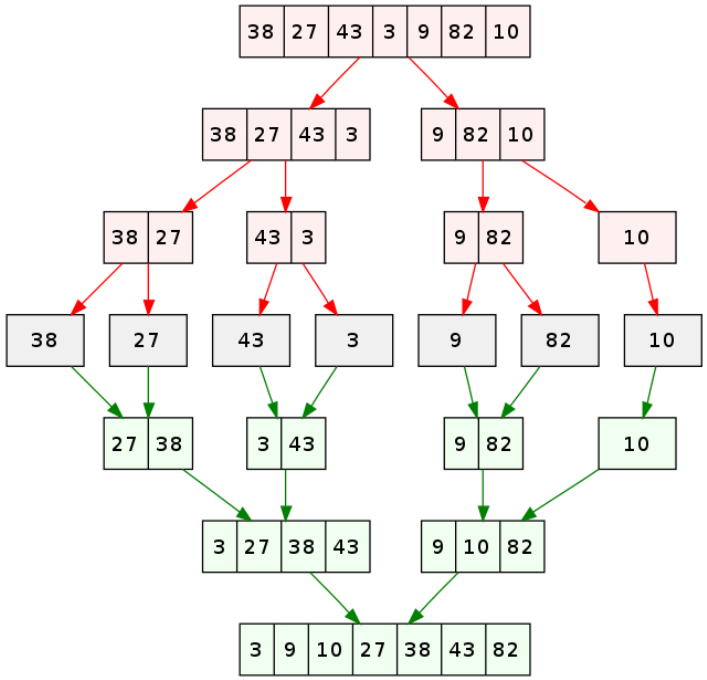


Figure: Divide-and-conquer algorithm to sort a sequence of numbers

6. Algoritma decrease-and-conquer

Tiga langkah utama:

- Kurangi (*decrease*) *instance* masalah menjadi instance yang lebih kecil dari masalah yang sama dan perluas solusi.
- Selesaikan (*conquer*) masalah dengan menyelesaikan contoh masalah yang lebih kecil.
- Perluas (*extend*) solusi dari instance yang lebih kecil untuk mendapatkan solusi dari masalah awal.

Ide dasar: mengeksploitasi hubungan antara solusi untuk contoh masalah tertentu dan solusi untuk contoh yang lebih kecil.

7. Algoritma transform-and-conquer

Teknik yang digunakan dalam merancang algoritma yang akan mentransformasikan (to transform) suatu kasus menjadi bentuk lain, kemudian menentukan solusi (untuk menaklukkan) dari bentuk baru kasus tersebut.

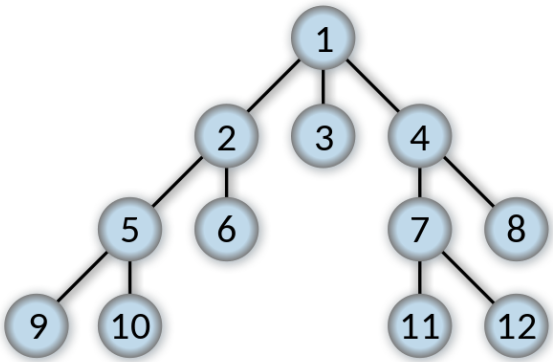
1. Tahap pertama melibatkan transformasi ke masalah lain yang lebih bisa menerima solusi.
2. Tahap kedua melibatkan pemecahan masalah baru di mana masalah baru yang diubah diselesaikan. Kemudian solusi diubah kembali ke masalah awal.

Contoh: mengalikan dua bilangan sederhana XII dan IV (dalam sistem bilangan Romawi).

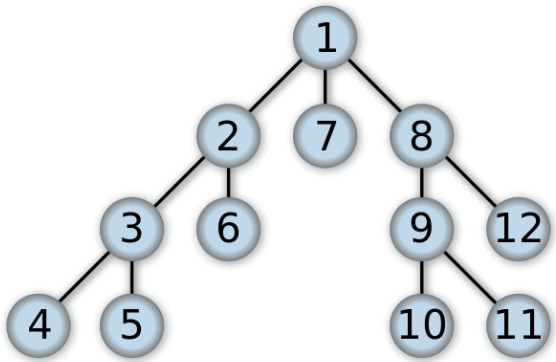
- Tahap 1: bilangan XII dan IV diubah menjadi soal lain sebesar 12×4.
- 2nd stage: perkalian sebenarnya dilakukan dengan 48, kemudian hasilnya diubah menjadi Angka Romawi sebagai XLVIII.

8. Algoritma BFS and DFS

Breadth-first search (BFS) adalah algoritma traversal graf yang mulai melintasi graf dari simpul akar dan menjelajahi semua simpul tetangga. Sedangkan untuk Depth-first search (DFS) dimulai dari root node dan menjelajah sejauh mungkin sepanjang setiap cabang sebelum melakukan backtracking.



(a) BFS

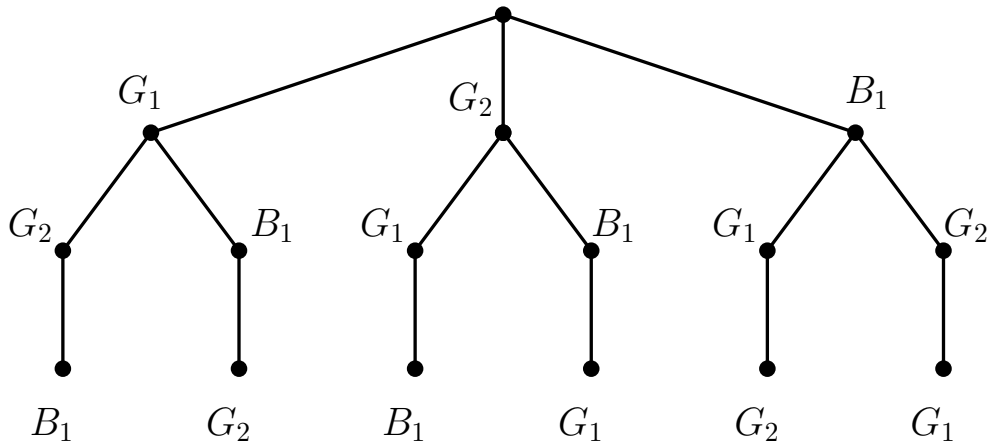


(b) DFS

9. Algoritma Backtracking

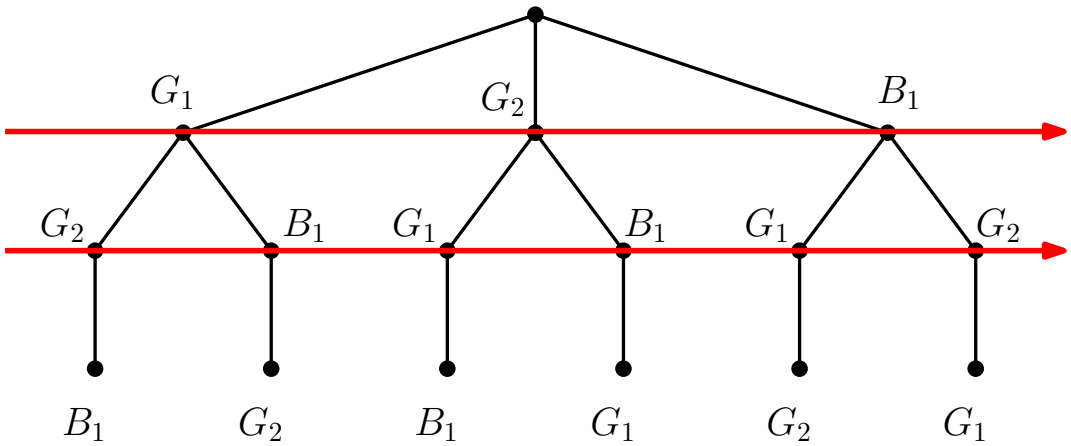
Backtracking adalah teknik algoritmik dimana tujuannya adalah untuk mendapatkan semua solusi dari suatu masalah dengan menggunakan pendekatan brute force.

Contoh: Kita ingin menyebutkan cara yang berbeda untuk memesan 2 anak perempuan dan laki-laki berturut-turut.



10. Algoritma Branch and bound

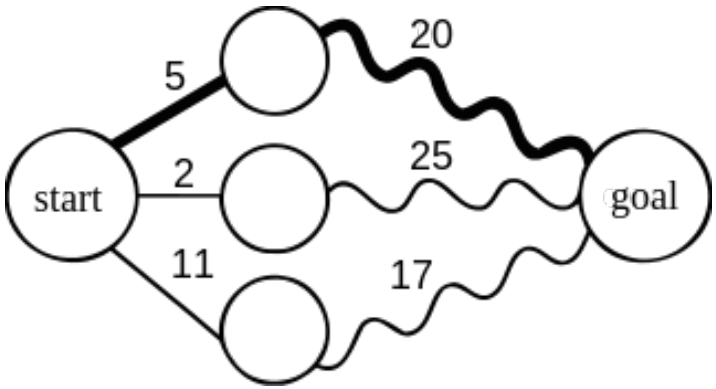
Mirip dengan backtracking karena juga menggunakan state space tree. Ini digunakan untuk memecahkan masalah optimasi dan masalah minimisasi. Jika kita telah memberikan masalah maksimisasi maka kita dapat mengubahnya menggunakan teknik Branch and bound dengan hanya mengubah masalah tersebut menjadi masalah maksimisasi.



11. Dynamic programming

Ini adalah teknik yang membantu memecahkan kelas masalah secara efisien yang memiliki submasalah yang tumpang tindih dan properti substruktur yang optimal.

Pemrograman dinamis sebagian besar diterapkan pada algoritma rekursif.



12. Algoritma Greedy

Ini adalah paradigma algoritmik yang membangun solusi bagian demi bagian, dengan selalu memilih, pada setiap langkah, bagian berikutnya yang menawarkan manfaat paling jelas dan langsung (yaitu pilihan optimal secara lokal).

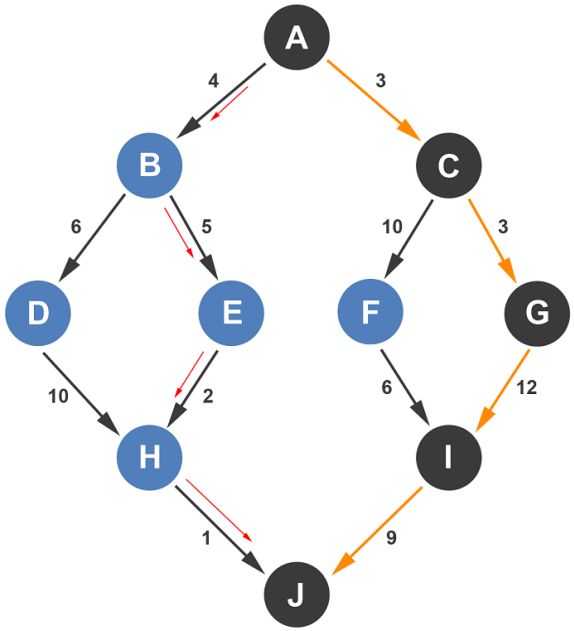


Figure: Greedy algorithm to find a path from A to J

13. Algoritma Sorting

Ini adalah algoritma yang menempatkan elemen daftar ke dalam urutan.

Output dari setiap algoritma pengurutan harus memenuhi dua syarat:

1. Outputnya dalam urutan monoton (setiap elemen tidak lebih kecil/lebih besar dari elemen sebelumnya, sesuai dengan urutan yang diperlukan).
2. Outputnya adalah permutasi (pengaturan ulang, namun mempertahankan semua elemen asli) dari input.

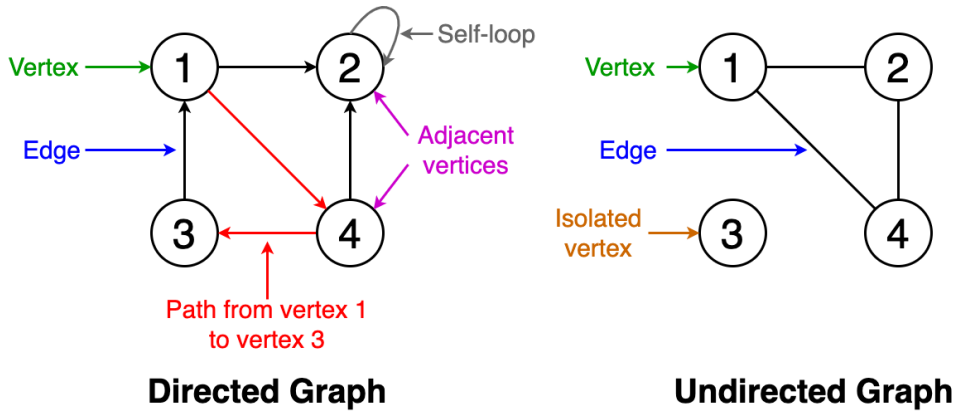
Contoh:

$$\text{COMPUTERSCIENCE} \rightarrow \text{CCCEEEIMNOPRSTU}$$

13. Algoritma pada struktur graf

Graf adalah notasi abstrak yang digunakan untuk mewakili hubungan antara pasangan objek. Sebuah graf terdiri dari:

- **Titik/Verteks/Simpul:** objek yang saling berhubungan dalam sebuah graf.
- **Sisi:** link yang menghubungkan simpul-simpul.



Example: masalah jalur terpendek, pohon rentang minimum, deteksi siklus, komponen yang terhubung kuat, pewarnaan graf, aliran maksimum

14. Teori kompleksitas komputasional (P, NP, NPC)

Berfokus pada mengklasifikasikan masalah komputasi menurut *penggunaan sumber daya*, dan *hubungan* antar kelas.

Permasalahan komputasional adalah tugas yang diselesaikan oleh komputer.

Kompleksitas komputasi atau *complexity* dari suatu algoritma adalah jumlah sumber daya (*time* dan *memori*) yang diperlukan untuk menjalankannya.

Kompleksitas masalah adalah kompleksitas dari algoritma **terbaik** yang memungkinkan pemecahan masalah.

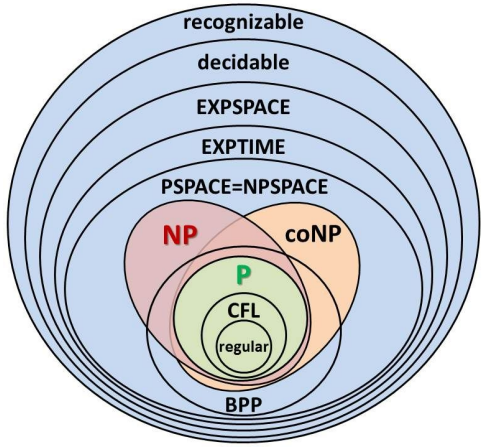


Figure: Computational complexity classes

Klasifikasi algoritma berdasarkan strategi

1. Solusi langsung: brute-force, greedy
2. Space-state base: backtracking, branch and bound
3. Solusi top-down: divide-and-conquer, decrease-and-conquer, transform-an-conquer, dynamic programming, BFS & DFS
4. Solusi bottom-up: dynamic programming

Contoh desain algoritma (1)

Algoritma untuk menjumlahkan dua bilangan bulat

Tahapan:

1. BEGIN
2. Deklarasikan tiga integer a, b, dan c
3. Definisikan nilai a dan b
4. Jumlahkan nilai a dan b
5. Simpal hasil dari langkah 4 di c
6. Print c
7. END

Contoh desain algoritma (1)

Pseudocode:

Algorithm 1 Penjumlahan dua bilangan bulat

1: **procedure** ADD(a, b)
2: get values of a and b from user
3: $c \leftarrow a + b$
4: **print** c
5: **end procedure**

end of slide...