Algoritma Bruteforce

(disarikan dari diktat Strategi Algoritma, Rinaldi Munir)

wijanarto

Definisi

- Dikenal juga sebagai exhaustive search, hanya berbeda pada masalah yang akan di pecahkan
- Disebut juga naïve algorithm
- Suatu pendekatan langsung (straightfoward) untuk memecahkan masalah, yang di dasarkan pada pernyataan masalahnya dan konsep yang terlibat di dalamnya
- Sederhana, langsung dan jelas

Contoh

- Menghitung a^n ($a > 0$, n adalah bilangan bulat tak-negatif)
- $a^n = a \times a \times \ldots \times a$ (sebanyak n kali),
  - jika $n > 0$
- $a^n = 1$,
  - jika $n = 0$
- Kompleksitas O(n)

Algoritma

- kalikan 1 dengan *a* sebanyak *n* kali

```
function pangkat(input a, n : integer)→integer
{ Menghitung a^n, a > 0 dan n bilangan bulat tak-
negatif
  Masukan: a, n
  Keluaran: nilai perpangkatan.
}
Deklarasi
  k, hasil : integer

Algoritma:
  hasil←1
  for k←1 to  n do
     hasil←hasil * a
  endfor
  return hasil
```

contoh

- Menghitung $n!$ (n bilangan bulat tak-negatif)
- $n! = 1 \times 2 \times 3 \times \ldots \times n$, jika $n > 0$
- $= 1$, jika $n = 0$
- Kompleksitas O(n)

Algoritma

- kalikan *n* buah bilangan, yaitu 1, 2, 3, …, *n*, bersama-sama

```
function faktorial(input n : integer)→integer
{ Menghitung n!, n bilangan bulat tak-negatif
  Masukan: n
  Keluaran: nilai faktorial dari n.
}
Deklarasi
  k, fak : integer

Algoritma:
  fak←1
  for k←1 to  n do
     fak←fak * k
  endfor
  return fak
```

Contoh

- Mengalikan dua buah matrik yang berukuran $n \times n$
- Misalkan $C = A \times B$ dan elemen-elemen matrik dinyatakan sebagai c_{ij}, a_{ij}, dan b_{ij}

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

- *Kompleksitas $O(n^2)$*

Algoritma

- hitung setiap elemen hasil perkalian satu per satu, dengan cara mengalikan dua vektor yang panjangnya *n*.

```
procedure PerkalianMatriks(input A, B : Matriks,
                           input n : integer,
                           output C : Matriks)
{          Mengalikan matriks A dan B yang berukuran n × n, menghasilkan matriks C
yang juga berukuran n × n
  Masukan: matriks integer A dan B, ukuran matriks n
  Keluaran: matriks C
}


Deklarasi
  i, j, k : integer

Algoritma
   for i←1 to n do
     for j←1 to n do
        C[i,j]←0    { inisialisasi penjumlah }
        for k ← 1 to n do
           C[i,j]←C[i,j] + A[i,k]*B[k,j]
        endfor
     endfor
   endfor
```

Contoh

- Menemukan semua faktor dari bilangan bulat *n* selain dari 1 dan *n* itu sendiri. Definisi faktor dari sebuah bilangan adalah sebagai berikut:
- Definisi: Bilangan bulat *a* adalah faktor dari bilangan bulat *b* jika *a* habis membagi *b*.

Algoritma

- Tentukan kompleksitasnya ???

```
procedure CariFaktor(input n : integer)
{ Mencari faktor dari bilangan bulat n selain 1 dan n itu sendiri.
  Masukan: n
  Keluaran: setiap bilangan yang menjadi faktor n dicetak.
}
Deklarasi
  k : integer

Algoritma:
  k←1
  ketemu ← false
  for k←2 to n - 1 do
    if n mod k = 0 then
      write(k)
    endif
  endfor
```

Karakteristik

- Tidak Cerdas dan Boros, buktinya adalah kompleksitas algoritmanya, yaitu membutuhkan jumlah langkah yang besar dalam penyelesainnya khususnya untuk data berukuran besar
- Mudah di terapkan pada sebagian masalah yang ada, jadi di lihat dari *kemudahan* dia lebih efisien.
- Beberapa pekerjaan mendasar pada komputer hanya dapat di selesaikan dengan metode ini, misal menghitung jumlah data, mencari min dan max dan seabagainya

Latihan dan Tugas

- Carilah kompleksitas algoritma dalam :
  - menentukan nilai min dan max
  - Pengurutan secara sekensial
  - Uji Keprimaan
  - Menghitung nilai polinom
- Buat implementasi Algoritma String Matching dan Closest Pair (Jarak titik terdekat)

Strength and Weakness

- Kekuatan
  - Wide applicable
  - Sederhana dan mudah
  - Layak untuk sebagian masalah (sorting,matrik,string matching)
  - Menjadi standar untuk tugas komputasi (menentkan min max, penjumlahan n bilangan)
- Kelemahan
  - Tidak efisien
  - Lamban
  - Tidak kreatif

Exhaustive Search

- Problem Solving dengan brute force untuk masalah yang melibatkan pencarian elemen dengan sifat khusus
  (kombinatorik→permutasi, kombinasi, himpunan)

Langkah dalam ES

- Enumerasi
  - Mendaftar setiap solusi yang mungkin secara sistematis
- Evaluasi
  - Mengevaluasi setiap kemungkinan solusi, yang layak di simpan dan yang tidak di buang
- Solusi umum tercapai bila langkah berakhir

Travelling Salesperson Problem

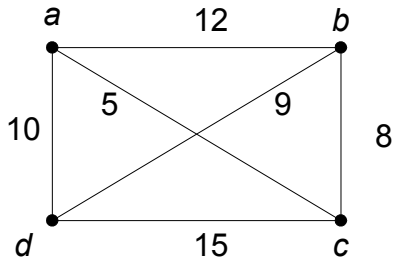
- Diberikan *n* buah kota serta diketahui jarak antara setiap kota satu sama lain. Temukan perjalanan (*tour*) terpendek yang melalui setiap kota lainnya hanya sekali dan kembali lagi ke kota asal keberangkatan

Algo TSP dengan ES

- Enumerasikan (*list*) semua sirkuit Hamilton dari graf lengkap dengan n buah simpul.
- Hitung (evaluasi) bobot setiap sirkuit Hamilton yang ditemukan pada langkah 1.
- Pilih sirkuit Hamilton yang mempunyai bobot terkecil.

Contoh untuk n=4

- Misalkan simpul *a* adalah kota tempat dimulainya perjalanan (*starting city*).



Rute perjalanan (*tour*)	Bobot
$a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$	10+12+8+15 = 45
$a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$	12+5+9+15 = 41
$\boldsymbol{a \rightarrow c \rightarrow b \rightarrow d \rightarrow a}$	10+5+9+8 = **32**
$a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$	12+5+9+15 = 41
$\boldsymbol{a \rightarrow d \rightarrow b \rightarrow c \rightarrow a}$	10+5+9+8 = **32**
$a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$	10+12+8+15 = 45

TSP

- Untuk 4 kota, terdapat 6 buah kemungkinan rute perjalanan (atau sirkuit Hamilton). Rute perjalananan terpendek adalah $a \rightarrow c \rightarrow b \rightarrow d \rightarrow a$ atau $a \rightarrow d \rightarrow b \rightarrow c \rightarrow a$ dengan bobot = 32.

TSP

- Karena perjalanan berawal dan berakhir pada simpul yang sama, maka untuk *n* buah simpul semua rute perjalanan yang mungkin dibangkitkan dengan permutasi dari *n* – 1 buah simpul. Permutasi dari *n* – 1 buah simpul adalah (*n* – 1)!. Pada contoh di atas, untuk *n* = 6 akan terdapat

  $(4 - 1)! = 3! = 6$

buah rute perjalanan

Analisa

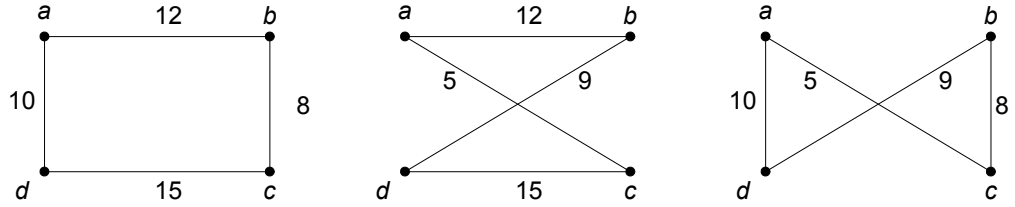
- Jika persoalan TSP diselesaikan dengan metode *exhaustive search*, maka kita harus mengenumerasi sebanyak $(n - 1)!$ buah sirkuit Hamilton, menghitung setiap bobotnya, dan memilih sirkuit Hamilton dengan bobot terkecil. Untuk menghitung bobot setiap sirkuit Hamilton dibutuhkan waktu $O(n)$, maka kompleksitas waktu algoritma *exhaustive search* untuk persoalan TSP sebanding dengan $(n - 1)!$ dikali dengan waktu untuk menghitung bobot setiap sirkuit Hamilton. Dengan kata lain, kompleksitas waktu algoritma *exhaustive search* untuk persoalan TSP adalah $O(n \cdot n!)$.

Analisa

- Kita dapat menghemat jumlah komputasi dengan mengamati bahwa setengah dari rute perjalanan adalah hasil pencerminan dari setengah rute yang lain, yakni dengan mengubah arah rute perjalanan
- 1 dan 6
- 2 dan 4
- 3 dan 5

Analisa

- maka dapat dihilangkan setengah dari jumlah permutasi (dari 6 menjadi 3). Ketiga buah sirkuit Hamilton yang dihasilkan adalah seperti gambar di bawah ini:



- Dengan demikian, untuk graf dengan *n* buah simpul, kita hanya perlu mengevaluasi sirkuit Hamilton sebanyak $(n - 1)!/2$.