



Brute Force

Tim Olimpiade Komputer Indonesia

Pendahuluan

Melalui dokumen ini, kalian akan:

- Mempelajari konsep *brute force*.
- Mampu mengerjakan persoalan dengan pendekatan *brute force*.



Konsep

- **Brute force** bukan suatu algoritma khusus, melainkan suatu strategi penyelesaian masalah.
- Sebutan lainnya adalah *complete search* dan *exhaustive search*.
- Prinsip dari strategi ini hanya satu, yaitu...



Konsep (lanj.)

coba semua kemungkinan!



Sifat Brute Force

- *Brute force* **menjamin** solusi pasti benar, karena seluruh kemungkinan dijelajahi.
- Akibatnya, umumnya *brute force* bekerja dengan lambat.
- Terutama ketika banyak kemungkinan solusi yang perlu dicoba.



Soal: Subset Sum

- Diberikan N buah bilangan $\{a_1, a_2, \dots, a_N\}$ dan bilangan K .
- Apakah terdapat subhimpunan sedemikian sehingga jumlahan dari elemen-elemennya sama dengan K ?
- Bila ya, maka keluarkan "YA". Selain itu keluarkan "TIDAK".

Batasan:

- $1 \leq N \leq 15$
- $1 \leq K \leq 10^9$
- $1 \leq a_i \leq 10^9$



Solusi

- Untuk setiap elemen, kita memiliki 2 pilihan, yaitu memilih elemen tersebut atau tidak memilihnya.
- Kita akan menelusuri semua kemungkinan pilihan.
- Jika jumlahan dari elemen-elemen yang dipilih sama dengan K , maka terdapat solusi.
- Hal ini dapat dengan mudah diimplementasikan secara rekursif.



Performa?

- Terdapat 2^N kemungkinan konfigurasi "pilih/tidak pilih".
- Kompleksitas solusi adalah $O(2^N)$.
- Untuk nilai N terbesar, $2^N = 2^{15} = 32.768$.
- Masih jauh di bawah 100 juta, yaitu banyaknya operasi komputer perdetik pada umumnya.



Implementasi

SOLVE(i, sum)

1 **if** $i > N$

2 **return** ($sum == K$)

3 $option1 = \text{SOLVE}(i + 1, sum + a_i)$ // Pilih elemen a_i

4 $option2 = \text{SOLVE}(i + 1, sum)$ // Tidak pilih elemen a_i

5 **return** $option1$ or $option2$

SOLVESUBSETSUM()

1 **return** SOLVE(1, 0)



Optimisasi

- Bisakah solusi tersebut menjadi lebih cepat?
- Perhatikan kasus ketika nilai sum telah melebihi K .
- Karena semua a_i bernilai positif, maka sum tidak akan mengecil.
- Karena itu, bila sum sudah melebihi K , **dipastikan** tidak akan tercapai sebuah solusi.



Solusi Teroptimisasi

SOLVE(i, sum)

1 **if** $i > N$

2 **return** ($sum == K$)

3 **if** $sum > K$

4 **return** *false*

5 $option1 = \text{SOLVE}(i + 1, sum + a_i)$ // Pilih elemen a_i

6 $option2 = \text{SOLVE}(i + 1, sum)$ // Tidak pilih elemen a_i

7 **return** $option1$ or $option2$



Pruning

Hal ini biasa disebut sebagai **pruning** (pemangkasan).

Pruning

Merupakan optimisasi dengan mengurangi ruang pencarian dengan cara menghindari pencarian yang sudah pasti salah.



Pruning (lanj.)

- Meskipun mengurangi ruang pencarian, *pruning* umumnya tidak mengurangi kompleksitas solusi.
- Sebab, biasanya terdapat kasus yang mana *pruning* tidak mengurangi ruang pencarian secara signifikan.
- Pada kasus ini, solusi dapat dianggap tetap bekerja dalam $O(2^N)$.



Soal: Mengatur Persamaan

- Diberikan sebuah persamaan: $p + q + r = 0$.
- Masing-masing dari p , q , dan r harus merupakan anggota dari himpunan bilangan yang unik $\{a_1, a_2, \dots, a_N\}$
- Berapa banyak triplet (p, q, r) berbeda yang memenuhi persamaan tersebut?

Batasan:

- $1 \leq N \leq 2.000$
- $-10^5 \leq a_i \leq 10^5$



Solusi Sederhana

COUNTTRIPLETS()

```
1  count = 0
2  for i = 1 to N
3      for j = 1 to N
4          for k = 1 to N
5               $p = a_i$ 
6               $q = a_j$ 
7               $r = a_k$ 
8              if  $(p + q + r) == 0$ 
9                   $count = count + 1$ 
10 return count
```



Solusi Sederhana (lanj.)

- Kompleksitas waktu solusi ini adalah $O(N^3)$.
- Tentunya terlalu besar untuk nilai N mencapai 2.000.
- Ada solusi yang lebih baik?



Observasi

- Jika kita sudah menentukan nilai p dan q , maka nilai r haruslah $-(p + q)$.
- Jadi cukup tentukan nilai p dan q , lalu periksa apakah nilai $-(p + q)$ ada pada bilangan-bilangan yang diberikan.
- Pemeriksaan ini dapat dilakukan dengan *binary search*.
- Kompleksitas solusi menjadi $O(N^2 \log N)$



Solusi Lebih Baik

COUNTTRIPLETSFAST()

```
1  count = 0
2  for i = 1 to N
3      for j = 1 to N
4          p = ai
5          q = aj
6          r = -(p + q)
7          if EXISTS(r)
8              count = count + 1
9  return count
```

dengan EXISTS(r) adalah algoritma *binary search* untuk memeriksa keberadaan r di $\{a_1, a_2, \dots, a_N\}$ (tentunya setelah diurutkan).



Solusi Lebih Baik (lanj.)

- Kompleksitas $O(N^2 \log N)$ sudah cukup untuk N yang mencapai 2.000.
- Dari sini kita belajar bahwa optimisasi pada pencarian kadang diperlukan, meskipun ide dasarnya adalah *brute force*.



Penutup

- Ide dari *brute force* biasanya sederhana: Anda hanya perlu menjelajahi seluruh kemungkinan solusi.
- Biasanya merupakan ide pertama yang didapatkan saat menghadapi masalah.
- Lakukan analisis algoritma, jika kompleksitasnya cukup, maka *brute force* saja :)
- Bila tidak cukup cepat, coba lakukan observasi.
- Bisa jadi kita dapat melakukan *brute force* dari "sudut pandang yang lain" dan lebih cepat.
- Bila tidak berhasil juga, baru coba pikirkan strategi lainnya.

