Perbandingan strategi Bruteforce, Greedy, Breadth First Search dan Depth First Search dalam Optimisasi Grocery Shopping

Suhendi 13516048
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516048@std.stei.itb.ac.id

***Abstract*—Kegiatan Grocery Shopping adalah kegiatan berbelanja kebutuhan sehari-hari. Kegiatan tersebut cepat atau lambat pasti harus dilakukan dan tidak dapat dihindari. Untuk mahasiswa yang suka berhemat tentu ingin mengeluarkan biaya minimum untuk Grocery Shopping. Oleh karena itu perlu diformulasikan strategi agar dapat meminimumkan biaya Grocery Shopping.**

Keywords—grocery shopping; minimize; optimization; strategy; algorithm

I. PENDAHULUAN

Di zaman modern ini, segala sesuatu telah tersedia. Seiring perkembangan zaman yang marak ini, tingkat kenyamanan juga meningkat. Contohnya saja ojek online saat ini telah mempermudah transportasi. Selain itu, ojek online juga menyediakan layanan untuk membeli makanan dan sebagainya. Salah satu layanan yang termasuk juga adalah *Grocery Shopping.*

Grocery Shopping adalah kegiatan berbelanja kebutuhan sehari-hari. Kebutuhan sehari-hari yang dimaksud seperti sabun, shampoo, odol, sikat gigi dan sebagainya dan ada juga bahan masakan bagi yang memasak sendiri. Kegiatan tersebut pasti tidak dapat dihindari siapapun, cepat atau lambat pasti harus dilakukan.

Untuk mahasiswa yang sedang menempuh pendidikan tentu saja ingin berhemat. Hal ini dapat dilihat dari pola makan mahasiswa yang senang makan di warung tegal dengan harga sepuluh ribu rupiah. Salah satu cara untuk membantu mahasiswa berhemat adalah dengan memilih tempat belanja yang paling murah.

Mahasiwa dihadapkan dengan berbagai toko yang memiliki harga setiap barang yang berbeda-beda. Mungkin saja di toko pertama harga shampoo lebih murah dari toko kedua dan sebaliknya harga sabun lebih murah di toko kedua dibandingkan dengan toko pertama. Oleh karena adanya hal ini, kita dapat memformulasikan strategi untuk menghemat uang.

II. LANDASAN TEORI

A. *Exhaustive Search*

Exhaustive search atau bruteforce search adalah strategi pemecahan masalah dengan mengenumerasi setiap solusi yang mungkin dan memilih solusi yang memenuhi keinginan. Algoritma yang bruteforce pada umumnya simpel dan pasti akan menemukan solusi yang diharapkan hanya saja, sumberdaya yang diperlukan untuk menjalankan algoritma bruteforce umumnya akan meningkat pesat seiring dengan meningkatnya solusi yang mungkin. Hal ini menyebabkan penggunaan algoritma bruteforce tidak optimal untuk permasalahan besar.

Algoritma umum untuk exhaustive search adalah sebagai berikut:

1. Menghasilkan kandidat solusi pertama.
2. Mengecek apabila kandidat solusi tersebut sesuai harapan.
3. Bila tidak, hasilkan kandidat solusi berikutnya dan ulangi langkah 2.
4. Bila iya maka solusi telah ditemukan.

Contoh permasalahan yang dapat diselesaikan dengan *exhaustive search* adalah pencarian dua titik terdekat pada himpunan titik 2 dimensi. Pendekatan *exhaustive search* dapat dilakukan dengan menghasilkan pasangan 2 titik dan mencari pasangan dengan titik terdekat.

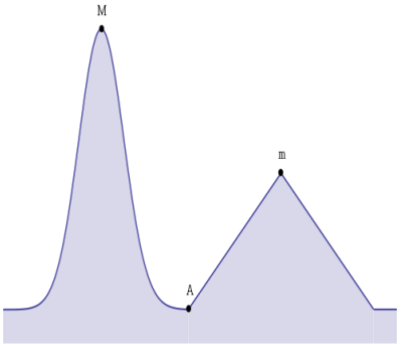
Contoh kasus permasalahan closest pair. Misalkan terdapat himpunan titik v = {(0, 5), (2, 2), (3, 2), (6,4)}. Perhitungan jarak dapat dilakukan dengan Euclidean Distance.

Pair		**Jarak**
Titik 1	**Titik 2**	
(0, 5)	(2, 2)	3.6055
(0, 5)	(3, 2)	4.2426
(0, 5)	(6, 4)	6.0827
(2, 2)	(3, 2)	1.0
(2, 2)	(6, 4)	4.4721
(3, 2)	(6, 4)	3.6055

Dari hasil perhitungan jarak antar pasangan titik tersebut, dapat didapatkan pasangan titik terdekat yaitu (2, 2) dan (3, 2) dengan jarak 1.

B. Greedy

Algoritma *greedy* merupakan algoritma yang memilih solusi yang 'tampak' terbaik. Algoritma *greedy* biasanya dijalankan melalui langkah-langkah dimana setiap langkah akan ada pilihan dimana algoritma tersebut akan memilih pilihan terbaik pada saat itu. Penggunaan algoritma *greedy* tidak menjamin bahwa solusi optimum akan ditemukan. Hal ini dikarenakan prinsip algoritma *greedy* yang mengutamakan pilihan yang terlihat baik dapat mengakibatkan algoritma hanya mencapai maksimum lokal.

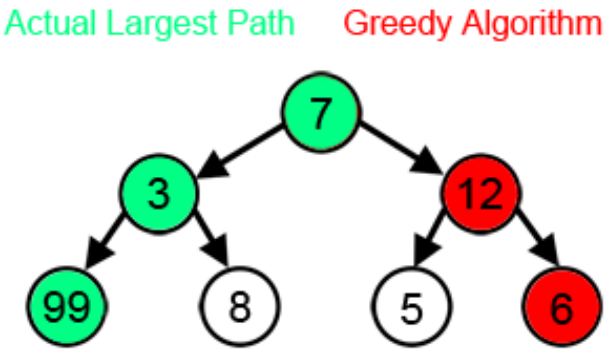


Gambar 1. Ilustrasi kasus yang mencapai maksimum lokal.
Sumber:
https://en.wikipedia.org/wiki/File:Greedy_Glouton.svg

Pada gambar diatas, dapat dilihat bahwa titik M merupakan maksimum global sedangkan m merupakan maksimum lokal tetapi karena slope dari titik A ke m pada awalnya lebih curam akibatnya algoritma *greedy* akan memilih ke arah kanan dibandingkan dengan arah kiri sehingga mencapai titik m yang merupakan maksimum lokal dan hanya mendapatkan solusi sub-optimum.

Pada umumnya, kerangka algoritma *greedy* sebagai berikut:
1. Hasilkan kandidat sub-solusi.
2. Pilih kandidat sub-solusi yang paling optimum.
3. Ulangi langkah 1 hingga solusi ditemukan

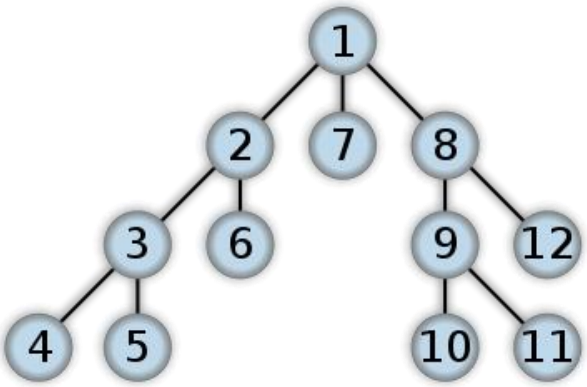
Walaupun pada kerangka diatas pada langkah 2 algoritma memilih sub-solusi optimum, bukan berarti solusi keseluruhan juga optimum. Seperti yang telah dijelaskan diatas, algoritma *greedy* hanya mengevaluasi sebagian solusi dan solusi optimum sebuah permasalahan mungkin memiliki sub-solusi yang tidak optimum.



Gambar 2. Ilustrasi kasus dimana algoritma *greedy* sub-optimum.
Sumber: https://brilliant.org/wiki/greedy-algorithm/

C. Depth First Search

Depth First Search adalah strategi penelusuran struktur data graf yang menelusuri graf secara 'mendalam'. Proses *DFS* bermula dari suatu simpul dan akan 'mengunjungi' simpul tersebut. *DFS* mencari seluruh simpul keluar dari simpul yang terakhir kali dikunjungi yang masih memiliki simpul keluar yang belum dikunjungi. Setelah semua simpul telah dikunjungi maka akan kembali ke simpul sebelumnya dan mengunjungi simpul keluar lainnya. Proses ini akan berulang hingga seluruh simpul yang dapat dicapai dari simpul sumber telah dikunjungi. Bila masih ada simpul yang belum dikunjungi maka akan memilih salah satu simpul dan proses *DFS* akan diulangi.



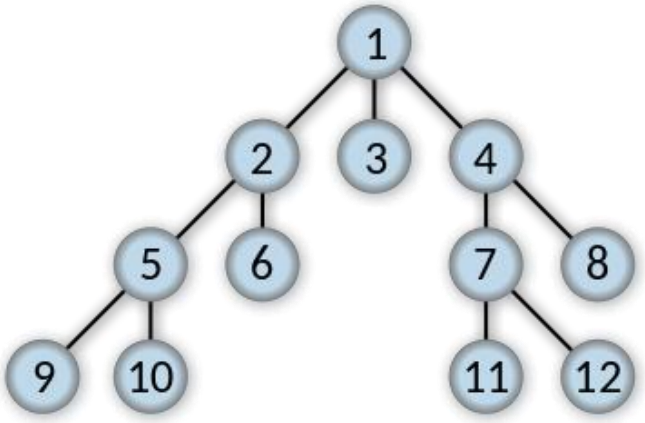
Gambar 3. Urutan simpul yang dikunjungi DFS.
Sumber: https://en.wikipedia.org/wiki/File:Depth-first-tree.svg

Dari ilustrasi diatas dapat dilihat bahwa dari simpul akar, akan menelusuri simpul keluar hingga tidak ditemukan simpul keluar lagi kemudian akan kembali dan menelusuri simpul keluar lainnya.

D. Breath First Search

Breadth First Search adalah strategi penelusuran struktur data graf yang menelusuri graf secara 'melebar'. Proses *BFS* sama dengan *DFS* bermula dari suatu simpul. Perbedaan mendasar yaitu *DFS* akan mengunjungi dari simpul yang

terbaru sedangkan *BFS* akan mengunjungi seluruh simpul keluar dari simpul yang setingkat.



Gambar 4. Urutan simpul yang dikunjungi BFS.
Sumber: https://en.wikipedia.org/wiki/File:Breadth-first-tree.svg

Dari ilustrasi diatas dapat dilihat bahwa dari simpul akar, akan menelusuri setiap tingkatan simpul kemudian akan berlanjut ke tingkatan berikutnya.

III. Strategi Pengehematan Uang

Permasalahan yang kita hadapi adalah memilih toko untuk membeli setiap barang dari daftar barang dari beberapa toko yang diketahui harga setiap barangnya agar memerlukan uang yang minimum. Permasalahan tersebut berupa permasalahan minimisasi.

Setiap strategi akan disertakan potongan kode dalam bahasa pemrogramman Python 3. Penjelasan struktur data yang akan digunakan untuk potongan kode dibawah adalah sebagai berikut:

- items = list dari nama barang yang akan dibeli
- shops = list dari dictionary yang memili informasi harga barang berdasarkan nama barang
- result = list dari indeks toko pada list *shops* yang dipilih untuk setiap barang

A. *Exhaustive Search*

Dengan strategi *Exhaustive Search*, yang kita lakukan adalah mencari kombinasi setiap toko dimana kita akan membeli barang tersebut dan memilih kombinasi dengan total harga barang yang minimum.

Algoritma *Exhaustive Search*:

1. Menghasilkan *list* berisi nilai 0 sebanyak jumlah barang sebagai solusi pertama.
2. Hitung harga barang solusi pertama.
3. Jadikan solusi pertama sebagai solusi terbaik.
4. Mencari dari belakang angka yang bukan berupa indeks terakhir dari toko dari solusi sekarang.
5. Bila indeks ada, tambahkan nilai pada indeks tersebut dengan nilai 1.
6. Ubah seluruh angka setelah indeks tersebut menjadi 0.
7. Hitung harga barang solusi sekarang.
8. Bila harga barang solusi sekarang lebih kecil dari harga minimum makan jadikan solusi sekarang sebagai solusi terbaik.
9. Ulangi langkah 4.
10. Bila indeks tidak ada atau berupa -1 (out of range) maka solusi terbaik telah ditemukan.

Kode program untuk strategi *Exhaustive Search*:

```
def exhaustive(items, shops):
  def gen_next(prev):
    # Menyalin solusi sebelumnya.
    result = list(prev)
    # Cari toko berikutnya yang dapat diubah.
    i = len(prev) - 1
    while i >= 0 and result[i] == len(shops) - 1:
      i -= 1
    if i > -1:
      # Ubah toko untuk barang yang belum mencoba
semua kemungkinan.
      result[i] += 1
      # Mengubah toko untuk barang setelahnya
menjadi 0.
      for j in range(i + 1, len(items)):
        result[j] = 0
      return result
    else:
      # Tidak ada kandidat solusi lainnya.
      return None
  def evaluate_price(solution):
    price = 0
    for i in range(0, len(items)):
      if items[i] in shops[solution[i]]:
        price += shops[solution[i]][items[i]]
      else:
        # Toko tidak menjual barang tertentu.
        return math.inf
    return price
  # Menghasilkan solusi pertama.
  best_solution = [0 for _ in range(0,
len(items))]
  min_price = evaluate_price(best_solution)
  next_solution = gen_next(best_solution)
  while next_solution:
    price = evaluate_price(next_solution)
    if price < min_price:
      # Ditemukan solusi yang lebih baik.
      min_price = price
      best_solution = next_solution
    # Ulangi untuk kandidat solusi berikut.
    next_solution = gen_next(next_solution)
```

```
    return min_price, best_solution
```

B. Greedy

Strategi greedy yaitu untuk setiap barang, memilih toko yang memiliki harga minimum untuk barang tersebut.

Algoritma:

1. Terdefinisikan parameter *items* dan *shops*.
2. Mulai dengan list kosong sebagai *result*.
3. Mulai dengan variable *price* dengan nilai 0.
4. Untuk setiap barang, tambahkan toko yang memiliki harga minimum barang kepada list *result*.
5. Untuk setiap barang, tambahkan juga harga barang pada toko tersebut pada variable *price*.
6. Kembalikan *price* dan *result*.

Kode program untuk strategi Greedy:

```
def greedy(items, shops):
  price = 0
  buy_at_list = []
  i = 0
  for item in items:
    # Mencari harga minimum untuk setiap barang.
    min_price = math.inf
    buy_at_list.append(-1)
    for si in range(0, len(shops)):
      shop = shops[si]
      if item in shop:
        t_price = shop[item]
        if t_price < min_price:
          min_price = t_price
          buy_at_list[i] = si
    price += min_price
    i += 1
  return price, buy_at_list
```

C. Depth First Search

Dengan strategi *Depth First Search*, yang kita lakukan adalah mengganggap pemilihan suatu toko sebagai sisi graf dan daftar toko yang dipilih sebagai simpul.

Algoritma:

1. Terdefinisikan parameter *items* dan *shops*.
2. Mulai dengan list kosong untuk *result* sebagai parameter tambahan optional.

   Basis:
3. Bila panjang list *result* = panjang list *items*, kembalikan total harga dan list *result*.

   Rekurens:
4. Untuk setiap toko, ulangi langkah 3 dengan list *result* yang telah ditambahkan dan simpan hasil untuk setiap toko.
5. Pilih hasil dengan harga yang minimum dan kembalikan.

Kode program untuk strategi *Depth First Search*:

```
def dfs(items, shops, result=[]):
  if len(result) < len(items):
    # Percabangan, menghitung harga untuk setiap
toko yang berbeda.
    min_price = math.inf
    min_list = None
    for i in range(0, len(shops)):
      if items[len(result)] in shops[i]:
        # Menambahkan setiap toko pada vector toko
        candidate = list(result)
        candidate.append(i)
        # Rekursif hingga didapatkan list toko
sesuai dengan panjang daftar barang.
        price, final_list = dfs(items, shops,
candidate)
        if price < min_price:
          # Memilih kemungkinan yang paling murah.
          min_price = price
          min_list = final_list
    return min_price, min_list
  else:
    # Hitung harga total.
    price = 0
    for i in range(0, len(items)):
      price += shops[result[i]][items[i]]
    return price, result
```

D. Breadth First Search

Strategi *Breadth First Search* yaitu sama seperti *DFS*, mendefnisikan pemilihan toko suatu barang sebagai sisi dan daftar tokoh yang telah dipilih sebagai simpul.

Algoritma:

1. Terdefinisikan parameter *items* dan *shops*.
2. Dua list kosong yaitu untuk simpul sekarang dan untuk simpul berikut.
3. Selama ada simpul di list simpul sekarang, tambahkan seluruh simpul keluar ke list simpul berikut.
4. Bila list simpul sekarang kosong, tukarlah isi list simpul sekarang dengan list simpul berikut. Ulangi langkah 3.
5. Bila list simpul berikut kosong maka seluruh simpul telah dikunjungi.

Kode program untuk strategi *Breadth First Search*:

```
def bfs(items, shops):
  def evaluate_price(solution):
    price = 0
    for i in range(0, len(items)):
      if items[i] in shops[solution[i]]:
        price += shops[solution[i]][items[i]]
      else:
```

```
        # Toko tidak menjual barang tertentu.
        return math.inf
    return price
open_nodes = []
next_nodes = []
# Minimum values.
min_price = math.inf
best_sol = None
# Initial node.
next_nodes.append([])
while next_nodes:
    open_nodes = next_nodes
    next_nodes = []
    for n in open_nodes:
      if len(n) == len(items):
        price = evaluate_price(n)
        if price < min_price:
          min_price = price
          best_sol = n
      else:
        # Adds all out nodes.
        for i in range(0, len(shops)):
          next_node = list(n)
          next_node.append(i)
          next_nodes.append(next_node)
return min_price, best_sol
```

IV. Pengujian dan Analisis Kinerja

Pengujian dilakukan dengan testcase sebagai berikut

Daftar barang yang akan dibeli:

No.	Nama
1	sabun
2	odol
3	shampoo
4	galon air
5	tisu 100
6	kangkung
7	bayam
8	indomie goreng
9	indomie kari

Daftar harga barang:

No.	Nama	Toko 1	Toko 2	Toko 3
1	sabun	11000	10000	10000
2	odol	12000	-	-
3	shampoo	20000	21000	21000
4	galon air	16000	16500	16500
5	tisu 50	4000	3000	-
6	tisu 100	6000	-	5000
7	kangkung	7000	6000	6000
8	bayam	-	7000	6000
9	indomie goreng	2300	2200	2100
10	indomie kari	2300	2300	2200

Hasil yang diperoleh:

Strategi	Harga	Hasil
Exhaustive Search	79300	[1, 0, 0, 0, 2, 1, 2, 2, 2]
Greedy	79300	[1, 0, 0, 0, 2, 1, 2, 2, 2]
DFS	79300	[1, 0, 0, 0, 2, 1, 2, 2, 2]
BFS	79300	[1, 0, 0, 0, 2, 1, 2, 2, 2]

Berikut adalah data hasil rata-rata waktu eksekusi dari 10 percobaan.

Strategi	Rata-rata waktu (detik)	Optimal
Exhaustive Search	0.05196475982666015	✔
Greedy	0.00001475811004638	✔
DFS	0.01026959419250488	✔
BFS	0.05220761299133301	✔

Dari data diatas, dapat dilihat bahwa eksekusi strategi *greedy* optimal untuk kasus ini. Dari keempat strategi yang diterapkan, strategi *greedy* memiliki waktu eksekusi yang paling cepat.

Dari hasil eksperimen diatas, terlihat bahwa algoritma *greedy* memiliki waktu eksekusi yang paling cepat. Pada dasarnya, algoritma *greedy* ialah algoritma yang cepat tetapi tidak terjamin optimal. Kasus penghematan uang merupakan salah satu kasus dimana algoritma *greedy* optimal.

V. Kesimpulan

Permasalahan optimisasi penghematan uang *Grocery Shopping* ini pada dasarnya merupakan permasalahan yang sederhana dan simpel. Tetapi permasalahan yang sederhana ini dapat digunakan untuk menggambarkan dan membandingkan 4 strategi algoritma yakni *exhaustive search*, *greedy*, *depth first search* dan *breadth first search*.

Untuk kasus penghematan uang pada *Grocery Shopping* ini, strategi yang dapat diterapkan adalah dengan memilih toko dimana setiap barang yang akan dibeli minimum. Algoritma yang paling cocok untuk kasus ini adalah dengan strategi *greedy*. Hal ini dikarenakan strategi *greedy* yang relatif cepat dan dapat menghasilkan solusi optimum.

Ucapan Terimakasih

Pertama-tama penulis mengucapkan syukur kepada Tuhan sehingga penulis dapat menyelesaikan penulisan makalah ini. Penulis berterima kasih kepada keluarga penulis yang telah mendukung penulis dalam menempuh pendidikan di program studi Teknik Informatika. Penulis juga berterima kasih kepada Dr. Ir. Rinaldi Munir, M.T. sebagai dosen mata kuliah IF2211 Strategi Algoritma yang telah mengajarkan berbagai jenis strategi algoritma yang dapat diterapkan pada makalah ini. Penulis juga berterimakasih kepada Nella Zabrina yang telah

membantu, mendukung dan memberi semangat dalam persiapan makalah ini.

REFERENCES

[1] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). Introduction to Algorithms (Second ed.). MIT Press and McGraw–Hill. pp. 595–601. ISBN 0-262-03293-7.

[2] Goodrich, Michael T.; Tamassia, Roberto (2001), Algorithm Design: Foundations, Analysis, and Internet Examples, Wiley, ISBN 0-471-38365-1

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Mei 2018

Suhendi - 13516048