

Nama : Andika Anantyo

NIM : 5311421017

## **TEKNIK PENCARIAN BLIND SEARCH**

1. Tentukan bagaimana algoritma BFS di atas dapat menentukan node ke 8, 6, dan 7.
2. Ubahlah method static void main sehingga bentuk tree seperti Gambar 4.5 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 5.
3. Ubahlah method static void main sehingga bentuk tree seperti Gambar 4.6 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 9.
4. Ubahlah kode program di atas sehingga bentuk tree seperti Gambar 4.7 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node C.

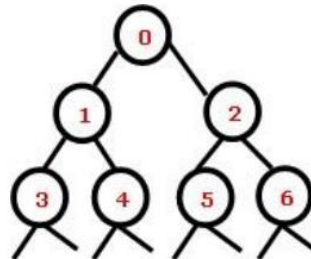
### **Jawab**

1. Node 8, 6, dan 7 ditemukan dalam urutan tertentu berdasarkan jarak (distance) dari node awal yang ditentukan, yaitu node 3. Algoritma BFS mengunjungi node-node dalam urutan lebar, yang berarti akan lebih dulu mengunjungi node yang lebih dekat dengan node awal dan kemudian bergerak ke node yang lebih jauh.
  - a. Pertama, BFS dimulai dari simpul awal n3 (dengan jarak 0) dan mengeksplorasi semua simpul yang dapat dijangkau dari n3.
  - b. Di level pertama pencarian, BFS akan mengeksplorasi simpul-simpul yang terhubung langsung ke n3, yaitu n4 dan n2.
  - c. Di level kedua, BFS akan mengeksplorasi simpul-simpul yang terhubung ke simpul level pertama, yaitu n5 dan n6 (dari n4) serta n1 (dari n2).
  - d. Di level ketiga, BFS akan mengeksplorasi simpul-simpul yang terhubung ke simpul level kedua, yaitu n7 (dari n5 dan n6) dan n8 (dari n6).
  - e. Setelah selesai menjelajahi semua simpul yang terjangkau, BFS akan mengakhiri eksekusi.

Hasil output program:

```
Output - tugas1 (run) ×
run:
(3, d=0) (4, d=1) (2, d=1) (5, d=2) (6, d=2) (1, d=2) (7, d=3) (8, d=3) BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Merubah method static void agar hasil tree seperti gambar 4.5. Algoritma BFS dapat menentukan node 5 dengan tahapan sebagai berikut.



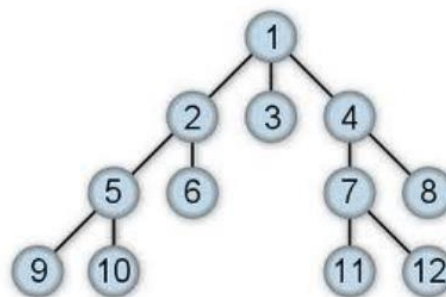
Gambar 4.5. Tree 1

- Pertama, BFS dimulai dari simpul awal n0 (dengan jarak 0) dan mengeksplorasi semua simpul yang dapat dijangkau dari n0.
- Di level pertama pencarian, BFS akan mengeksplorasi simpul-simpul yang terhubung langsung ke n0, yaitu n1 dan n2.
- Di level kedua, BFS akan mengeksplorasi simpul-simpul yang terhubung ke simpul level pertama, yaitu n3 dan n4 (dari n1) serta n5 dan n6 (dari n2).
- Setelah selesai menjelajahi semua simpul yang terjangkau, BFS akan mengakhiri eksekusi.

Hasil output program:

```
Output ×
Debugger Console × tugas1 (run) ×
run:
(0, d=0) (1, d=1) (2, d=1) (3, d=2) (4, d=2) (5, d=2) (6, d=2) BUILD SUCCESSFUL (total time: 0 seconds)
```

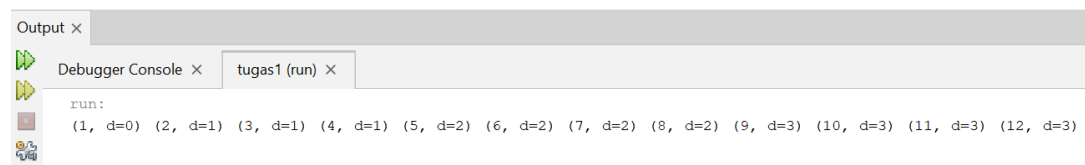
3. Merubah method static void agar hasil tree seperti gambar 4.6. Algoritma BFS dapat menentukan node 9 dengan tahapan sebagai berikut.



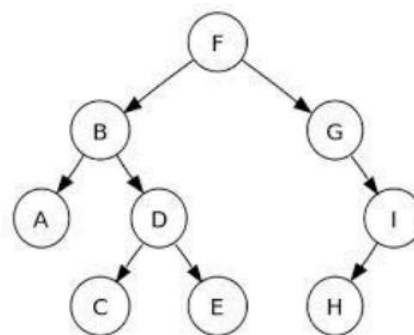
Gambar 4.6. Tree 2

- Pertama, BFS dimulai dari simpul awal n1 (dengan jarak 0) dan mengeksplorasi semua simpul yang dapat dijangkau dari n1.
- Di level pertama pencarian, BFS akan mengeksplorasi simpul-simpul yang terhubung langsung ke n0, yaitu n2, n3, dan n4.
- Di level kedua, BFS akan mengeksplorasi simpul-simpul yang terhubung ke simpul level pertama, yaitu n5 dan n6 (dari n2) serta n7 dan n8 (dari n4).
- Di level ketiga, BFS akan mengeksplorasi simpul-simpul yang terhubung ke simpul level kedua, yaitu n9 dan n10 (dari n5) serta n11 dan n12 (dari n7).
- Setelah selesai menjelajahi semua simpul yang terjangkau, BFS akan mengakhiri eksekusi.

Hasil output program:



- Merubah method static void agar hasil tree seperti gambar 4.7. Pada program yang saya buat, saya menggunakan angka untuk mewakili abjad (A = 1, B =2, dst) karena saya belum dapat merubah kodingnya untuk menampilkan abjad. Algoritma BFS dapat menentukan node C (node 3) dengan tahapan sebagai berikut.

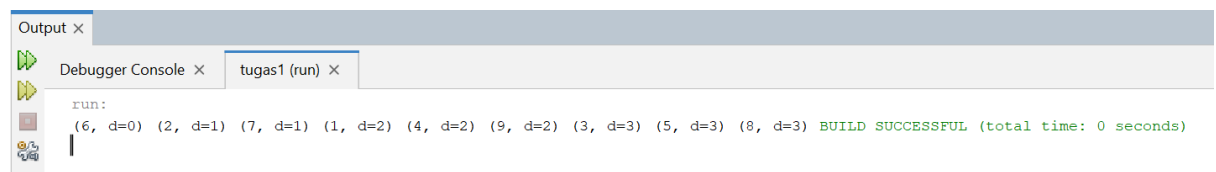


Gambar 4.7. Tree 3

- Pertama, BFS dimulai dari simpul awal n6 (nF) (dengan jarak 0) dan mengeksplorasi semua simpul yang dapat dijangkau dari n6 (nF).
- Di level pertama pencarian, BFS akan mengeksplorasi simpul-simpul yang terhubung langsung ke n6 (nF), yaitu n2 (nB) dan n7 (nG).

- c. Di level kedua, BFS akan mengeksplorasi simpul-simpul yang terhubung ke simpul level pertama, yaitu n1 (nA) dan n4 (nD) (dari nB) serta n9 (nI) (dari nG).
- d. Di level ketiga, BFS akan mengeksplorasi simpul-simpul yang terhubung ke simpul level kedua, yaitu n3 (nC) dan n5 (nE) (dari nD) serta n8 (nH) (dari nI).
- e. Setelah selesai menjelajahi semua simpul yang terjangkau, BFS akan mengakhiri eksekusi.

### Hasil output program



```
Output x
Debugger Console x  tugas1 (run) x
run:
(6, d=0) (2, d=1) (7, d=1) (1, d=2) (4, d=2) (9, d=2) (3, d=3) (5, d=3) (8, d=3) BUILD SUCCESSFUL (total time: 0 seconds)
```

## TEKNIK HEURISTIC SEARCH

1. Pelajari class EightPuzzleSearch, EightPuzzleSpace, dan Node.
2. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya Gambar 5.8. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state. Analisa dan bedakan dengan solusi pada point 1.
3. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya Gambar 5.9. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state. Analisa dan bedakan dengan solusi pada point 1 dan 2.
4. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya Gambar 5.10. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state. Analisa dan bedakan dengan solusi pada point 1, 2, dan 3.
5. Ubahlah initial dan goal state dari program dan class-class di atas sehingga bentuk initial dan goal statenya Gambar 5.11. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state.

### Jawab

1. Penjelasan class sebagai berikut:
  - a. Class EightPuzzleSearch
    - Kelas ini merupakan inti dari algoritma pencarian untuk menyelesaikan masalah puzzle delapan ubin.
    - Kelas ini memiliki fungsi untuk mengatur dan menjalankan pencarian serta menentukan parameter penting seperti heuristik yang digunakan.
    - Melakukan inisialisasi node awal (root) dan node tujuan (goal), serta mengatur ruang terbuka (open) dan ruang tertutup (closed) yang digunakan selama pencarian.
    - Menyediakan metode untuk menghitung biaya heuristik (h1 dan h2), memilih heuristik, memilih node terbaik untuk dieksplorasi, dan mengelola informasi pencarian.

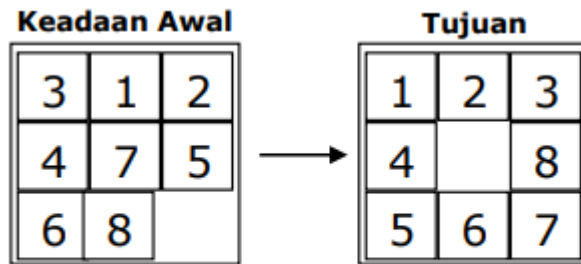
b. Class EightPuzzleSpace

- Program berisi metode untuk menghitung biaya heuristik (h1 dan h2), memilih heuristik, memilih node terbaik untuk dieksplorasi, dan mengelola informasi pencarian.
- Program berisi metode untuk menghitung biaya heuristik (h1 dan h2), memilih heuristik, memilih node terbaik untuk dieksplorasi, dan mengelola informasi pencarian.
- Program juga berisi metode untuk mendapatkan daftar node suksesor yang dapat dicapai dari suatu node tertentu, sehingga memungkinkan pencarian bergerak dari satu keadaan ke keadaan lain dalam ruang pencarian.

c. Class Node

- Kelas ini merepresentasikan node dalam konteks pencarian puzzle delapan ubin.
- Setiap node memiliki properti berupa konfigurasi ubin dalam bentuk array 1 dimensi, biaya pencarian (cost), referensi ke node parent, dan daftar node suksesor yang dapat dicapai dari node ini.
- Node ini memiliki metode untuk mencetak dirinya dalam bentuk string, membandingkan dua node untuk menentukan kesamaan konfigurasi ubin, serta untuk mengembalikan jalur dari root hingga node ini.
- Digunakan dalam penyimpanan informasi tentang keadaan di ruang pencarian dan mengelola pergerakan antar-keadaan selama pencarian.

2. Initial dan goal state yang diinginkan seperti gambar 5.8. Hal yang perlu dilakukan adalah merubah urutan angka pada initial dan goal state sesuai dengan urutan gambar 5.8 dari kiri atas ke kanan → turun dari kiri ke kanan lagi dan seterusnya. Hasil yang didapatkan dengan menunggu waktu running 200 menit hanya menunjukkan keadaan awal saja, tidak menunjukkan goal statenya. Berbeda dengan koding awal program yang menunjukkan initial state, kemudian riwayat pencarian, dan goal state nya juga dengan waktu 0 detik. Saya menggunakan goal state yang tidak rumit dengan mayoritas angka urut dari awal ke akhir, dan hasil yang muncul sesuai yang diinginkan dengan waktu yang singkat.



Gambar 5.8. Initial dan Goal state pada 8-puzzle 2

Hasil program awal (defaultnya):

```

Output ×
Debugger Console × job5 (run) ×

run:

Root: 3 1 2 4 7 5 6 8 0

Solution found
 3 1 2 4 7 5 6 8 0
 3 1 2 4 7 5 6 0 8
 3 1 2 4 0 5 6 7 8
 3 1 2 0 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

BUILD SUCCESSFUL (total time: 0 seconds)

```

Hasil program setelah dirubah:

```

Output ×
Debugger Console × job5 (run) ×

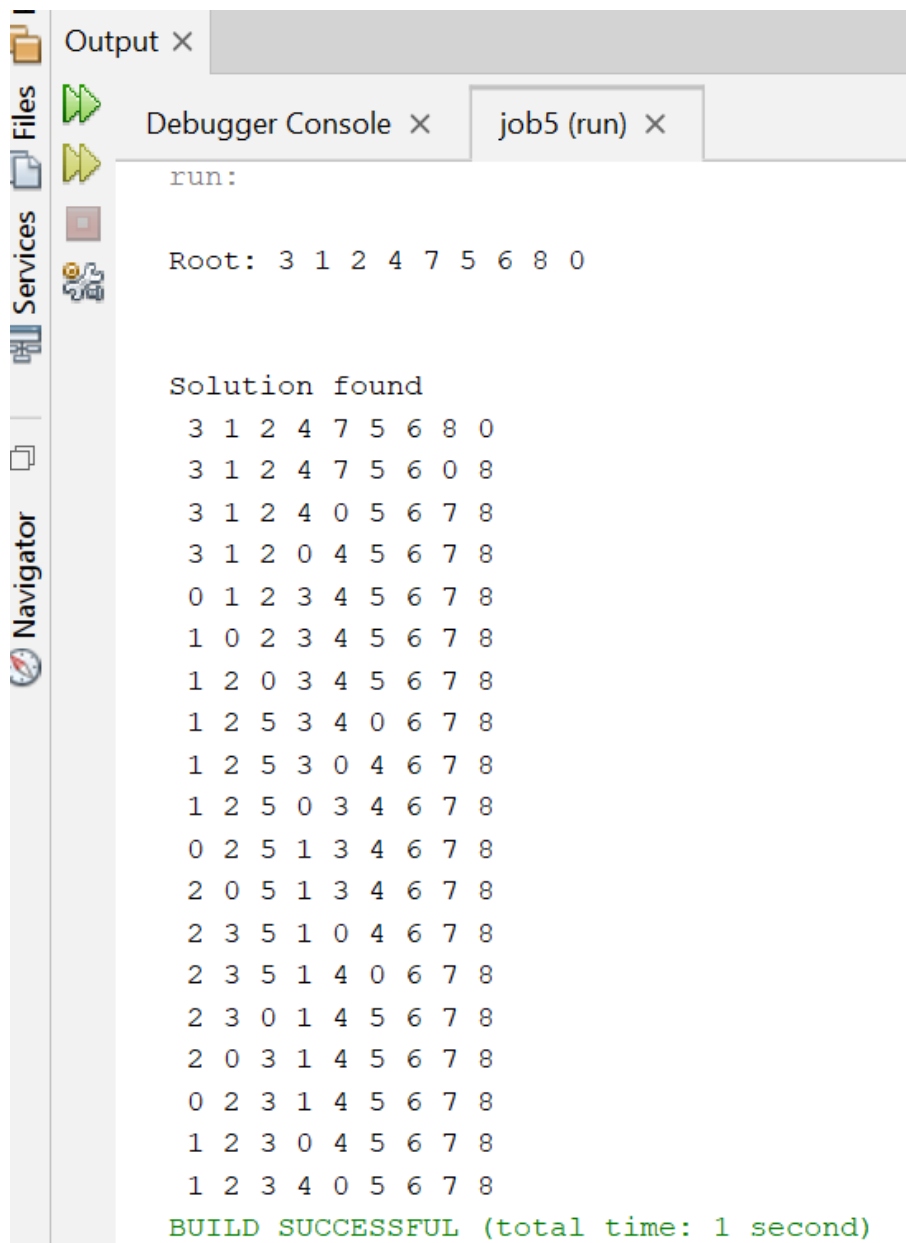
run:

Root: 3 1 2 4 7 5 6 8 0

BUILD SUCCESSFUL (total time: 209 minutes 7 seconds)

```

Hasil program dengan membuat goalnya tidak rumit:



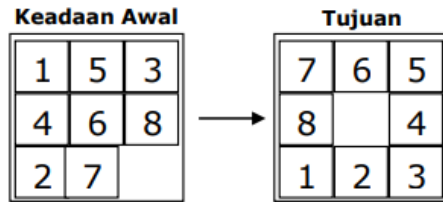
```
run:
Root: 3 1 2 4 7 5 6 8 0

Solution found
3 1 2 4 7 5 6 8 0
3 1 2 4 7 5 6 0 8
3 1 2 4 0 5 6 7 8
3 1 2 0 4 5 6 7 8
0 1 2 3 4 5 6 7 8
1 0 2 3 4 5 6 7 8
1 2 0 3 4 5 6 7 8
1 2 5 3 4 0 6 7 8
1 2 5 3 0 4 6 7 8
1 2 5 0 3 4 6 7 8
0 2 5 1 3 4 6 7 8
2 0 5 1 3 4 6 7 8
2 3 5 1 0 4 6 7 8
2 3 5 1 4 0 6 7 8
2 3 0 1 4 5 6 7 8
2 0 3 1 4 5 6 7 8
0 2 3 1 4 5 6 7 8
1 2 3 0 4 5 6 7 8
1 2 3 4 0 5 6 7 8

BUILD SUCCESSFUL (total time: 1 second)
```

3. Initial dan goal state yang diinginkan seperti gambar 5.9. Hal yang perlu dilakukan adalah merubah urutan angka pada initial dan goal state sesuai dengan urutan gambar 5.9 dari kiri atas ke kanan → turun dari kiri ke kanan lagi dan seterusnya. Hasil pertama yang didapat dari running program selama 87 menit hanya menunjukkan initial state saja, tidak ada goal statenya. Hal ini sama seperti soal nomor 2 dengan hasil hanya initial state. Kemudian pada running kedua dengan waktu 100 menit sudah benar dan mendapatkan hasil sesuai gambar 5.9.





Gambar 5.9. Initial dan Goal state pada 8-puzzle 3

Hasil program pertama:

```

Output ×
Debugger Console × job5 (run) ×
run:
Root: 1 5 3 4 6 8 2 7 0

BUILD SUCCESSFUL (total time: 87 minutes 18 seconds)

```

Hasil program kedua:

```

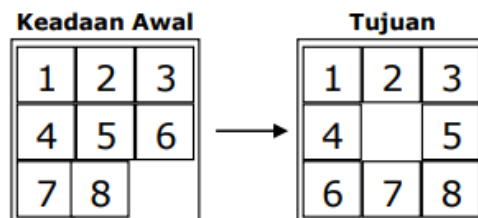
Output ×
Debugger Console × job5 (run) ×
Root: 1 5 3 4 6 8 2 7 0

Solution found
1 5 3 4 6 8 2 7 0
1 5 3 4 6 0 2 7 8
1 5 0 4 6 3 2 7 8
1 0 5 4 6 3 2 7 8
1 6 5 4 0 3 2 7 8
1 6 5 4 7 3 2 0 8
1 6 5 4 7 3 2 8 0
1 6 5 4 7 0 2 8 3
1 6 0 4 7 5 2 8 3
1 0 6 4 7 5 2 8 3
1 7 6 4 0 5 2 8 3
1 7 6 0 4 5 2 8 3
0 7 6 1 4 5 2 8 3
7 0 6 1 4 5 2 8 3
7 6 0 1 4 5 2 8 3
7 6 5 1 4 0 2 8 3
7 6 5 1 0 4 2 8 3
7 6 5 1 8 4 2 0 3
7 6 5 1 8 4 0 2 3
7 6 5 0 8 4 1 2 3
7 6 5 8 0 4 1 2 3

BUILD SUCCESSFUL (total time: 101 minutes 8 seconds)

```

- Initial dan goal state yang diinginkan seperti gambar 5.10. Hal yang perlu dilakukan adalah merubah urutan angka pada initial dan goal state sesuai dengan urutan gambar 5.10 dari kiri atas ke kanan → turun dari kiri ke kanan lagi dan seterusnya. Setelah running program dalam 0 detik, didapatkan hasil seperti yang diinginkan yaitu initial state, riwayat pencarian, dan goal state yang sesuai dengan gambar 5.10. Berbeda dengan nomor 2 yang tidak menampilkan goal state. Untuk waktunya lebih cepat dibandingkan nomor 3 karena goal state nya tidak acak.



Gambar 5.10. Initial dan Goal state pada 8-puzzle 4

Hasil program:

```

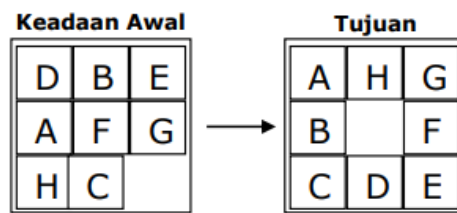
Output ×
Debugger Console × job5 (run) ×
run:
Root: 1 2 3 4 5 6 7 8 0

Solution found
1 2 3 4 5 6 7 8 0
1 2 3 4 5 6 7 0 8
1 2 3 4 0 6 7 5 8
1 2 3 4 6 0 7 5 8
1 2 0 4 6 3 7 5 8
1 0 2 4 6 3 7 5 8
0 1 2 4 6 3 7 5 8
4 1 2 0 6 3 7 5 8
4 1 2 6 0 3 7 5 8
4 1 2 6 5 3 7 0 8
4 1 2 6 5 3 0 7 8
4 1 2 0 5 3 6 7 8
0 1 2 4 5 3 6 7 8
1 0 2 4 5 3 6 7 8
1 2 0 4 5 3 6 7 8
1 2 3 4 5 0 6 7 8
1 2 3 4 0 5 6 7 8
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Initial dan goal state yang diinginkan seperti gambar 5.11. Hal yang perlu dilakukan adalah merubah urutan angka pada initial dan goal state sesuai dengan urutan gambar 5.11 dari kiri atas ke kanan → turun dari kiri ke kanan lagi dan seterusnya. Kemudian

pada koding lain dirubah karena untuk memunculkan abjad berbeda dengan memunculkan angka. Pada nomor ini saya belum paham cara untuk merubah angka menjadi abjadnya.



Gambar 5.11. Initial dan Goal state pada 8-puzzle 5

Hasil program: belum berhasil masih error

---

## CATATAN

Pada percobaan **Teknik Heuristic Search** yang sudah dilakukan, terdapat beberapa hal yang saya temukan:

- Initial state dengan urutan acak maupun tidak, tidak mempengaruhi waktu untuk mencapai goal state.
- Goal state sangat berpengaruh terhadap waktu pencarian. Pada percobaan yang sudah saya lakukan pada nomor 2 goal state yang acak, waktu pencariannya lama hingga 200 menit dan hasilnya hanya initial state saja. Sedangkan pada nomer 4 dengan goal state yang mayoritas urut hanya angka 0 yang diubah, waktu algoritma mencapai goal state sangat cepat dan hasilnya seperti yang diinginkan.
- Saya melakukan percobaan dengan memakai goal state dengan urutan angka yang urut, hanya dirubah letak 0 nya saja. Pada nomor 2 dapat dilihat, algoritma membutuhkan waktu yang cepat dan hasil yang sesuai. Berbeda dengan goal state yang sesuai dengan soal yang urutannya acak.
- Percobaan kedua pada nomor 3 berhasil mendapatkan hasil yang sesuai dimana sebelumnya hanya menghasilkan initial state saya, menurut saya walau awalnya tidak bisa, dapat di run lagi sampai mendapatkan hasil yang sesuai walaupun menguras waktu.
- Saya juga sudah mencoba dengan bahasa Python di Google Collabs, dan hasilnya juga masih sama saja dengan waktu yang lama juga.

...

Root: 1 5 3 4 6 8 2 7 0

Executing (2h 19m 58s) <cell line: 143> > run() > get\_previous\_cost()