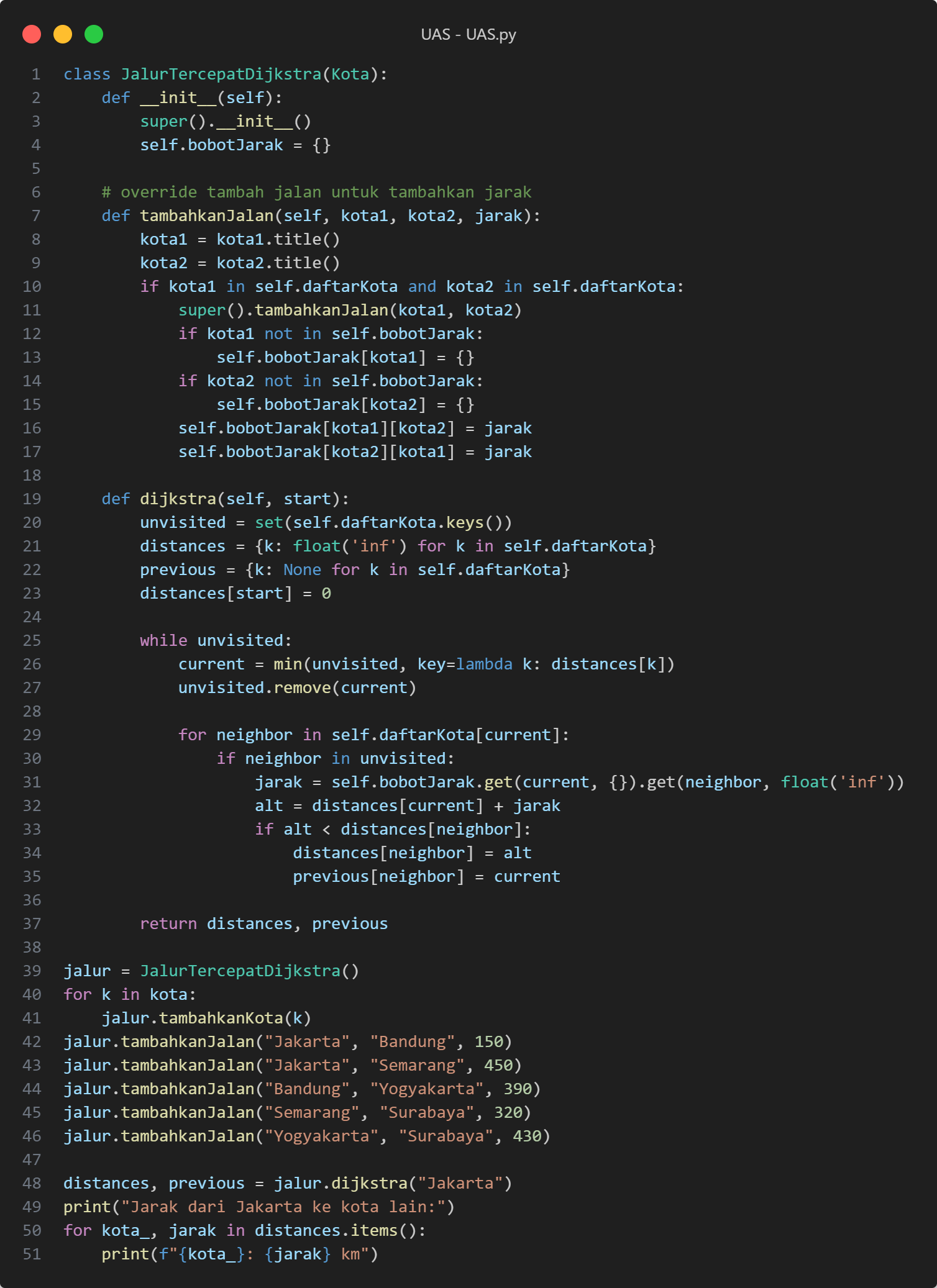
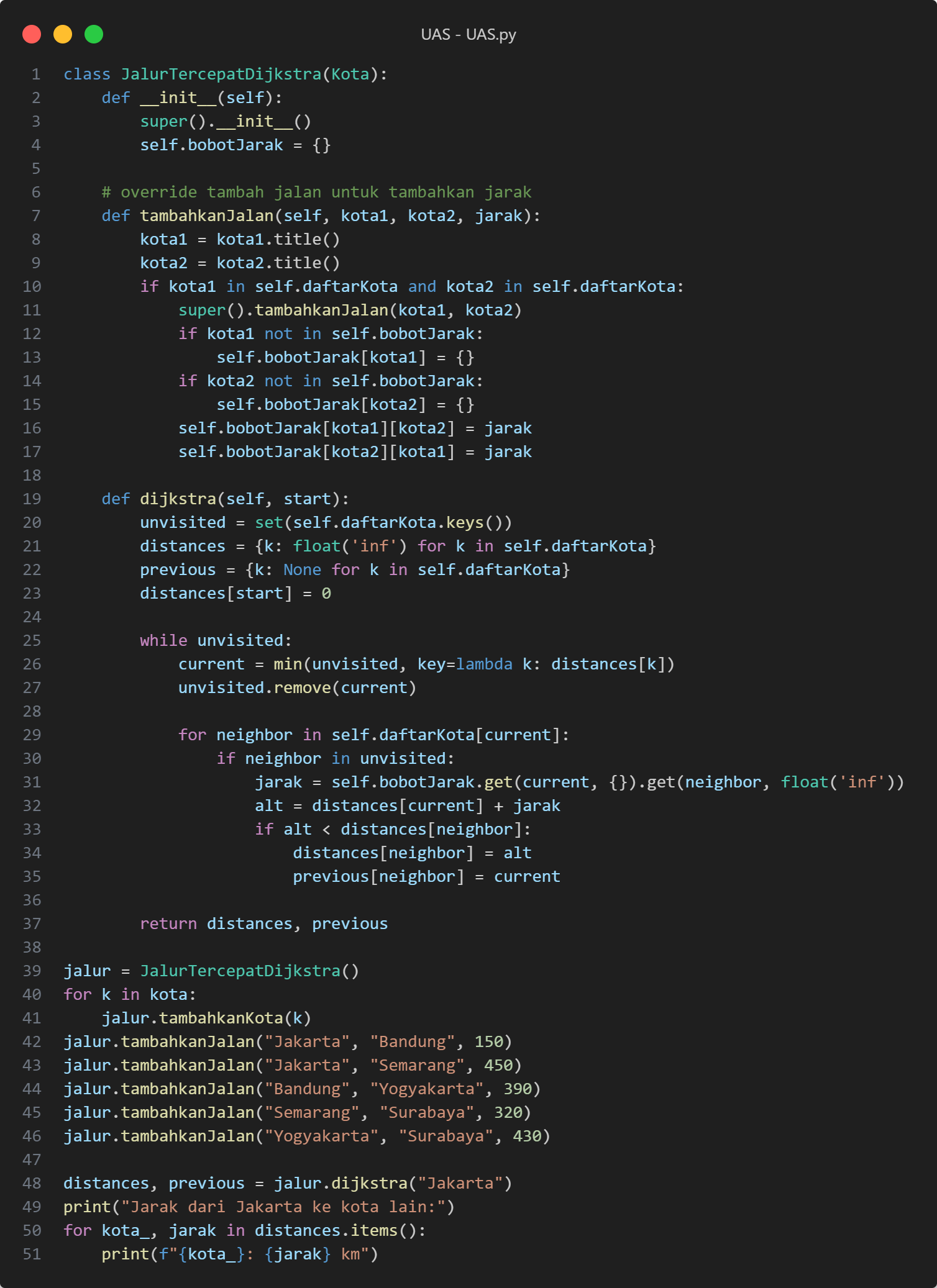
Kelas JalurTercepatDijkstra

Kelas ini mewarisi Kota dan menambahkan kemampuan menghitung jalur terpendek berdasarkan jarak.

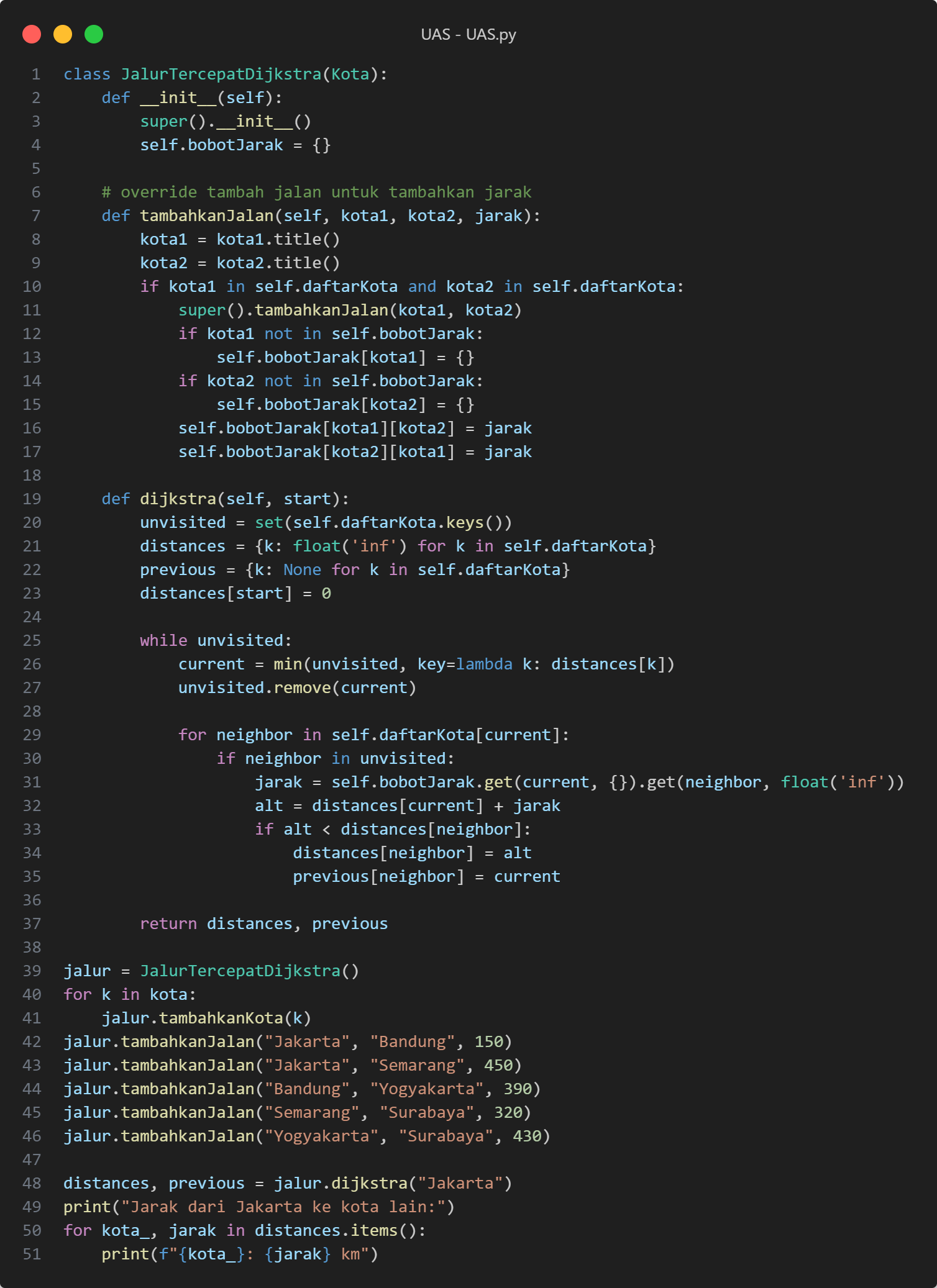


Pada bagian ini, kelas JalurTercepatDijkstra dibuat dengan mewarisi kelas Kota. Dengan demikian, seluruh properti dan metode yang dimiliki oleh kelas Kota, seperti tambahkanKota() dan struktur daftarKota, secara otomatis tersedia di dalam JalurTercepatDijkstra.

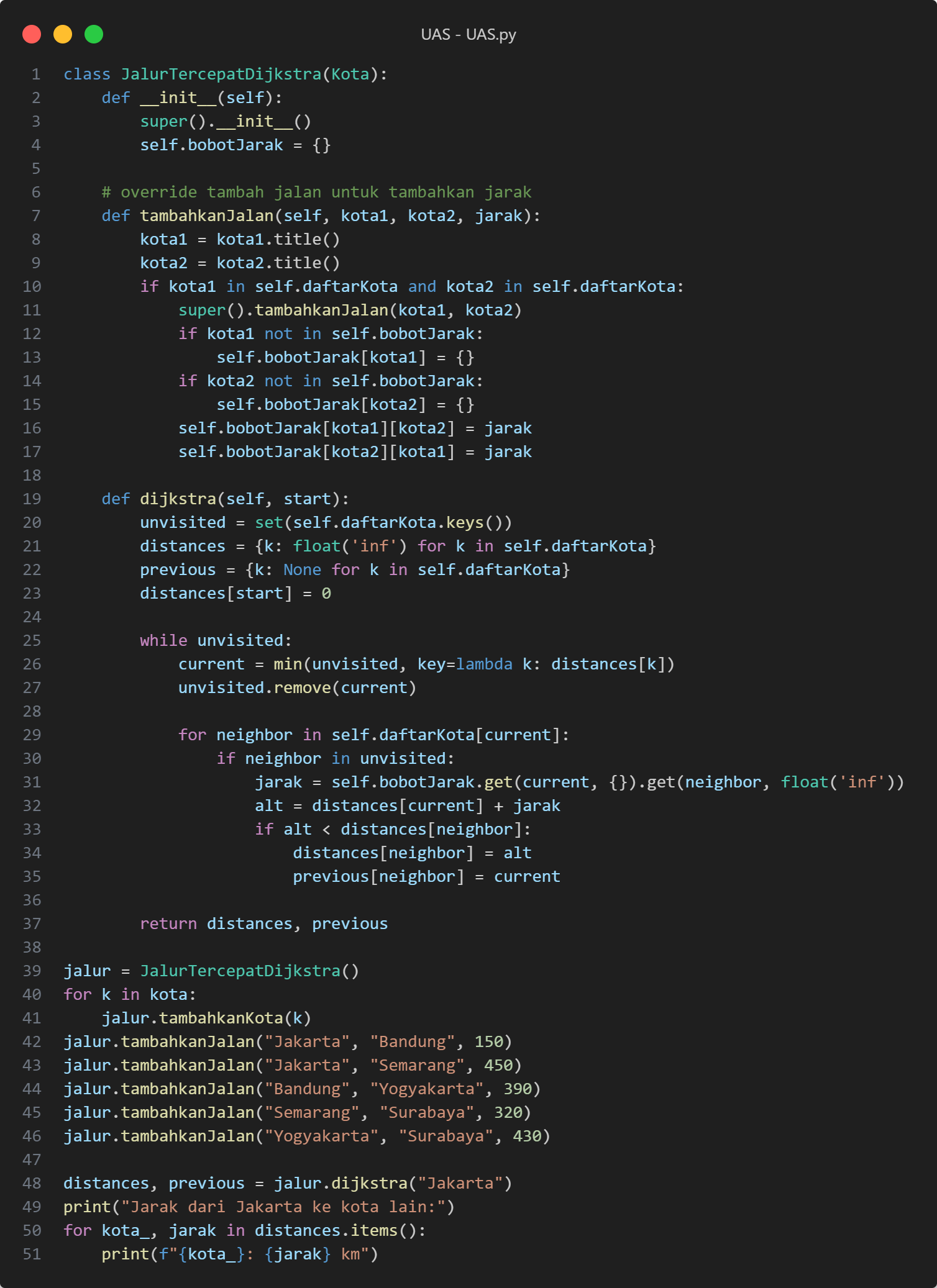
Tambahan properti bobotJarak digunakan untuk menyimpan informasi jarak antar kota dalam bentuk dictionary bersarang. Struktur ini berbentuk bobotJarak[kota1][kota2] = jarak.



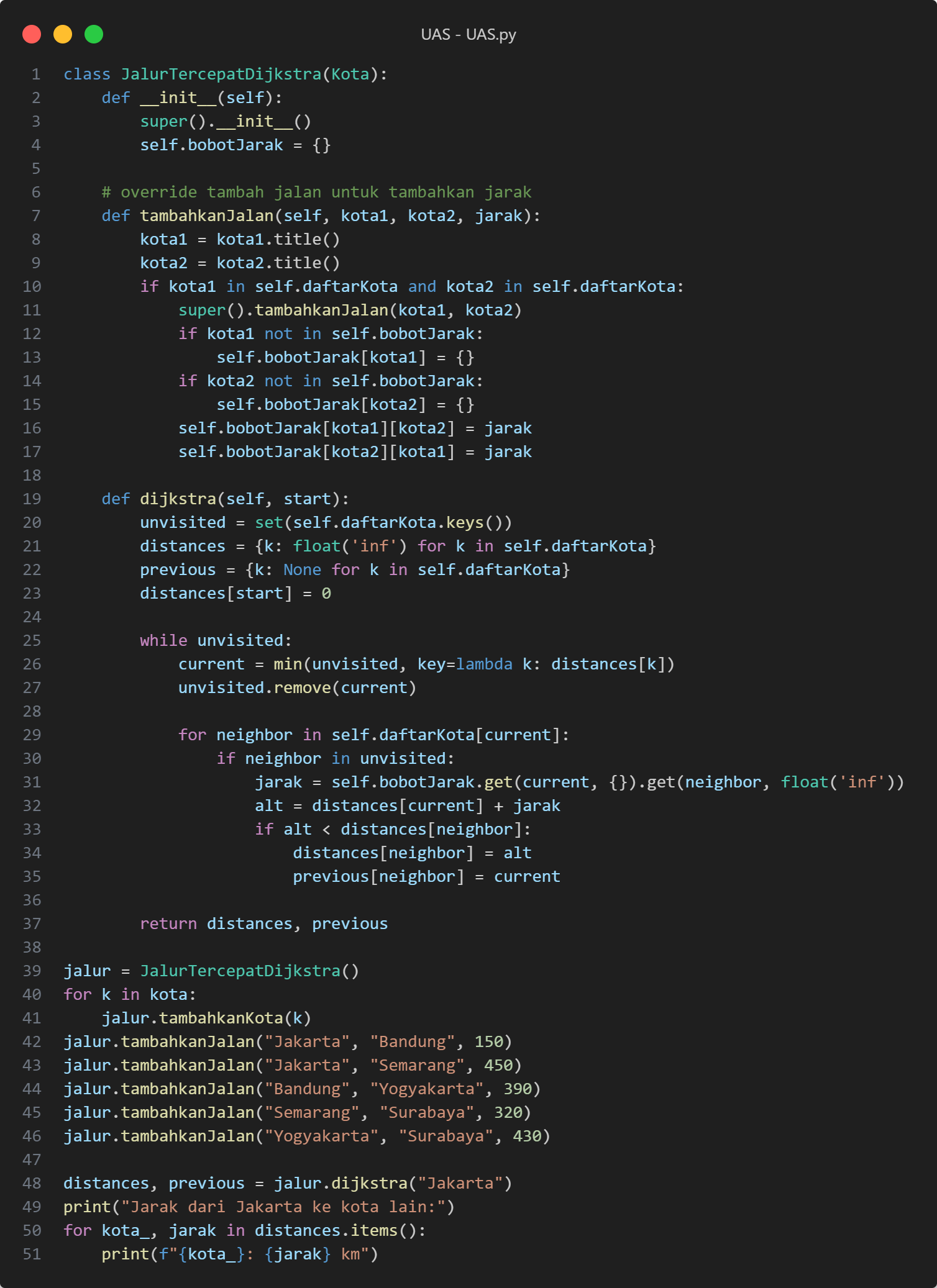
Fungsi ini merupakan override (penimpaan) dari metode tambahkanJalan di kelas Kota. Perbedaannya adalah fungsi ini juga menerima parameter tambahan berupa jarak, yang menunjukkan bobot jalur antara dua kota. Fungsi ini memastikan bahwa setiap hubungan dua arah antar kota disimpan baik di struktur daftarKota (graf tak berbobot), maupun di bobotJarak (graf berbobot). Dengan demikian, relasi antar kota dapat diakses untuk keperluan navigasi maupun perhitungan bobot.

 Bagian awal dari metode dijkstra() digunakan untuk menginisialisasi struktur data yang dibutuhkan:

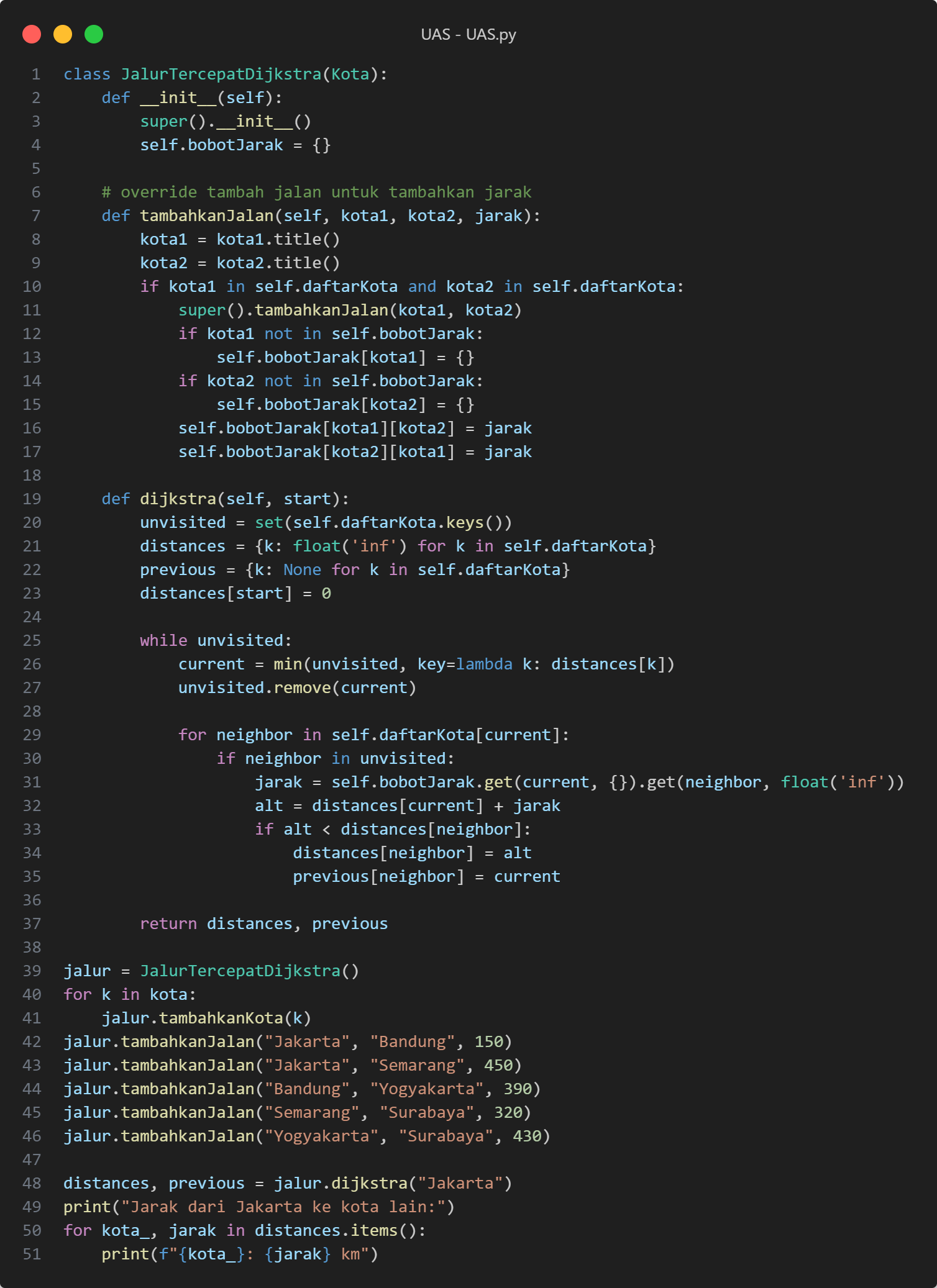
* unvisited: himpunan kota yang belum dikunjungi.
* distances: menyimpan jarak terpendek dari kota awal ke kota lainnya, diinisialisasi dengan nilai tak hingga (inf).
* previous: menyimpan rute kota sebelumnya untuk pelacakan jalur terpendek.
* Jarak dari kota asal (start) ke dirinya sendiri diatur menjadi 0 karena tidak ada perjalanan.



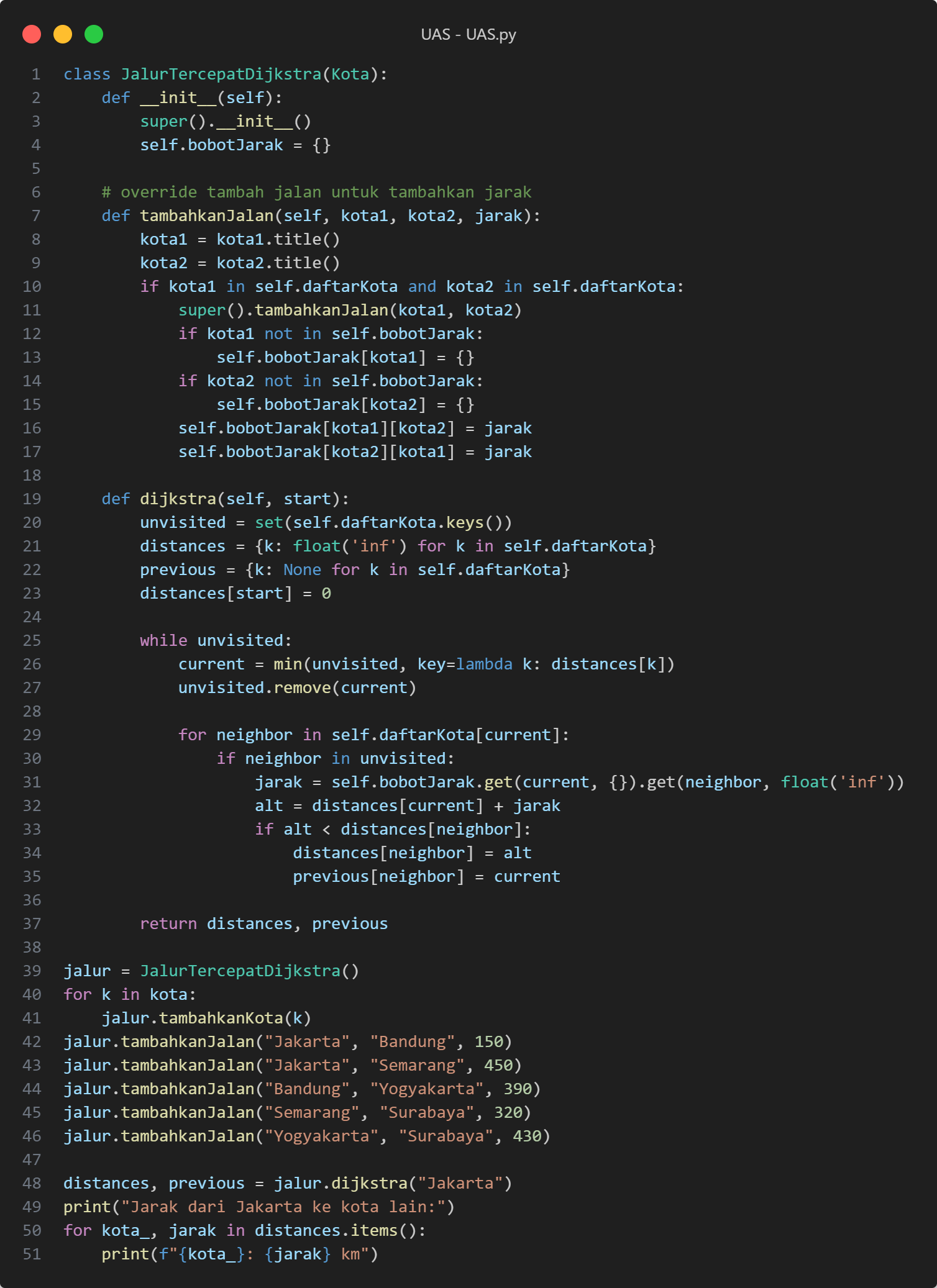
Selama masih ada kota yang belum dikunjungi, algoritma akan mencari kota dengan jarak terpendek dari kota asal. Kota tersebut dijadikan kota saat ini (current) dan dikeluarkan dari unvisited.



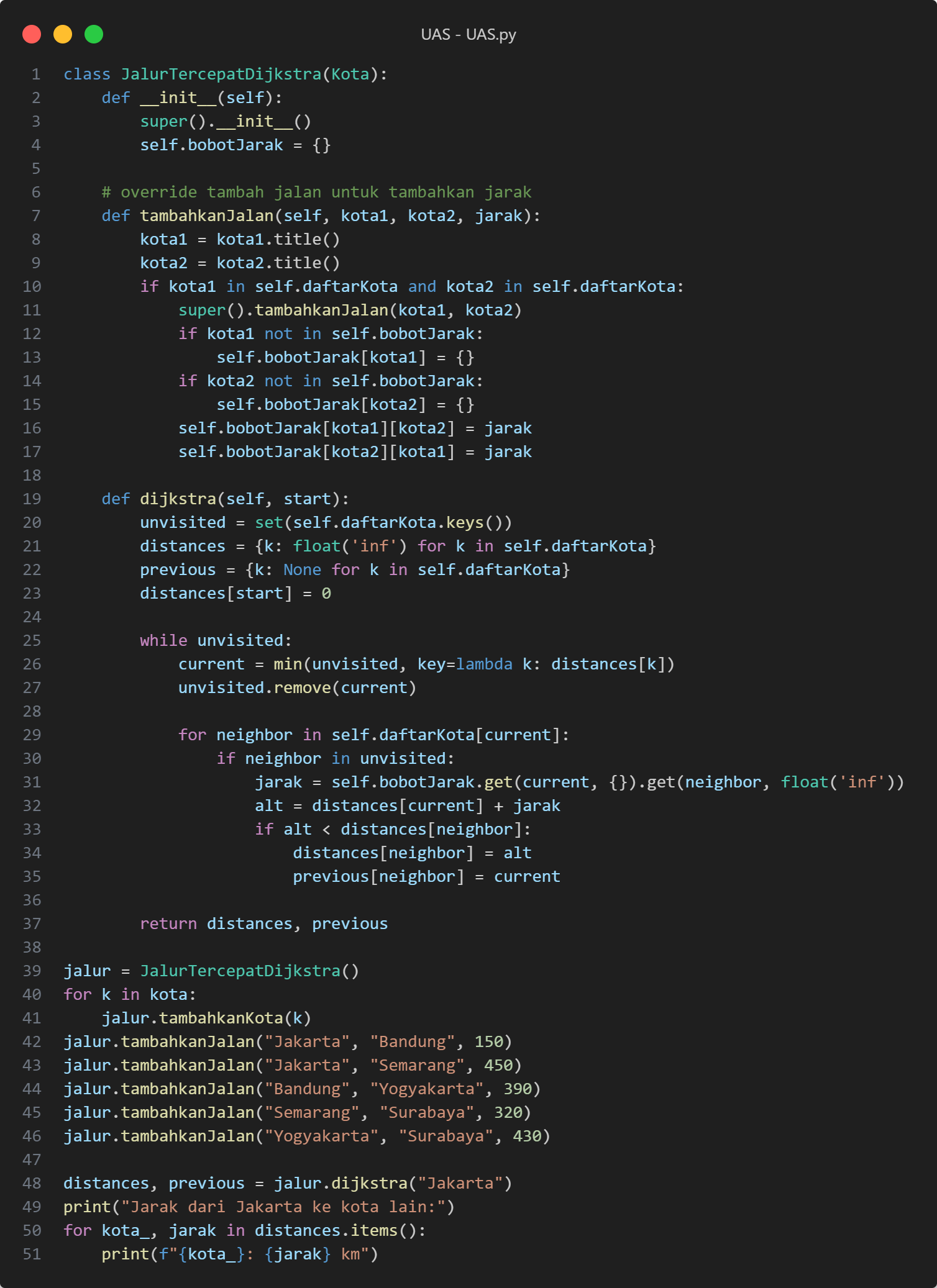
Pada bagian ini semua tetangga dari current akan dicek. Jika tetangga tersebut belum dikunjungi, jarak dari kota asal ke tetangga dihitung melalui current. Jika rute tersebut lebih pendek dibandingkan jarak yang sudah tersimpan sebelumnya, maka distances dan previous diperbarui.



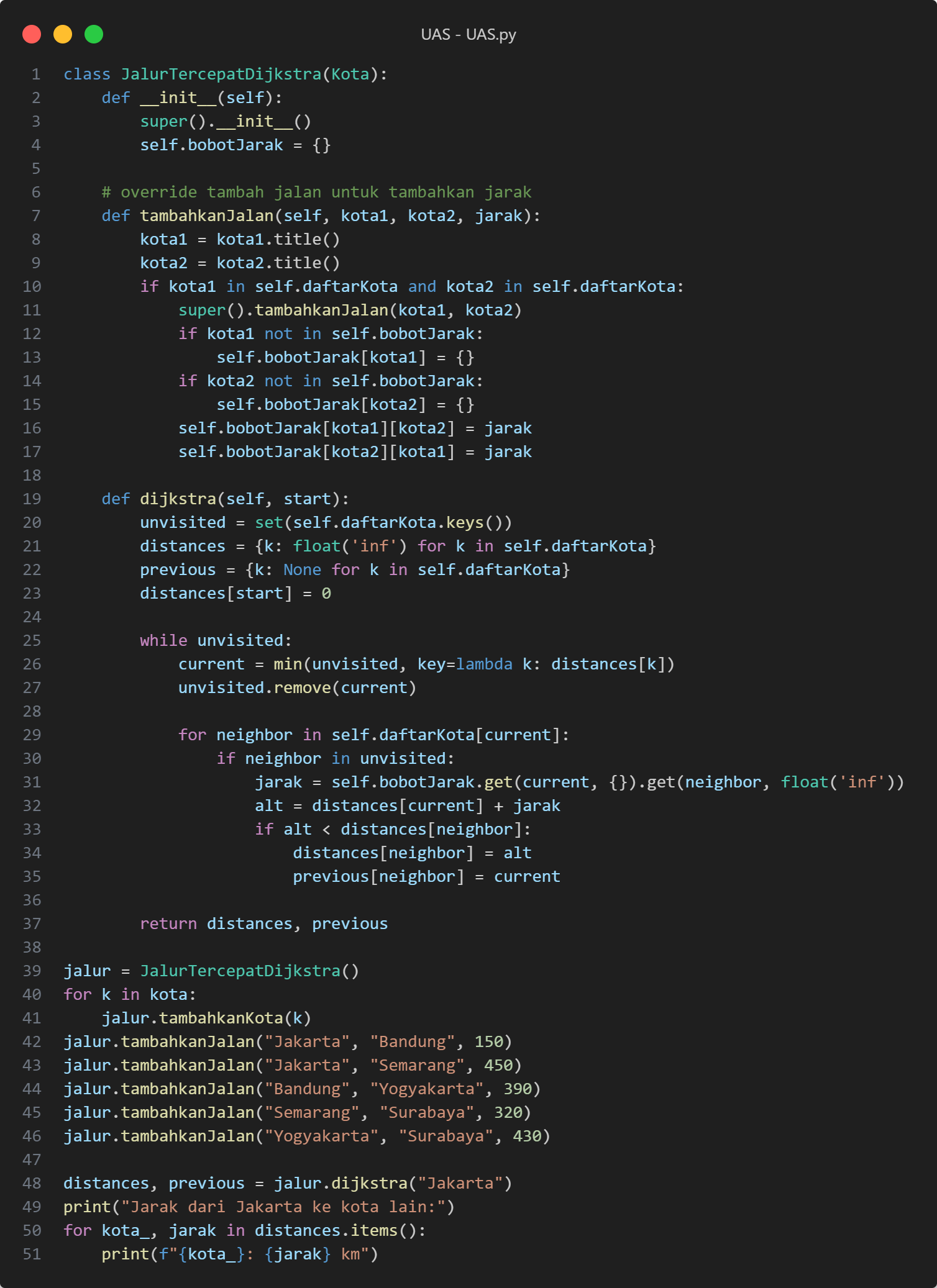
Setelah semua kota dihitung, fungsi mengembalikan: Distances untuk jarak terpendek dari kota asal ke setiap kota lain. Dan Previous untuk kota sebelumnya di jalur tercepat, berguna untuk merekonstruksi rute.



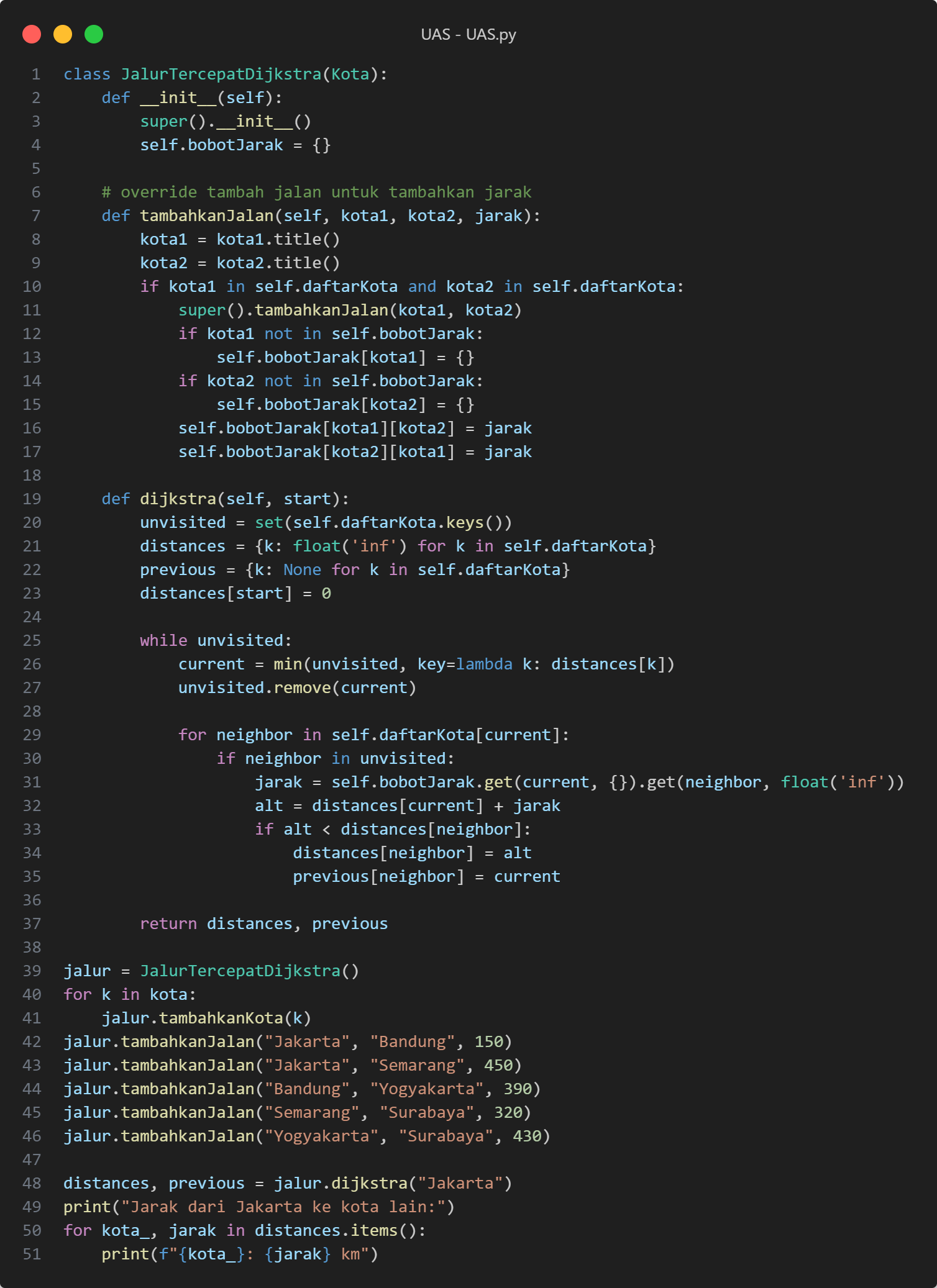
Pada baris ini, membuat objek jalur dari kelas JalurTercepatDijkstra, yang memungkinkan mengelola kota dan jalan antar kota serta menghitung jalur terpendek. Lalu, dilakukan iterasi terhadap list kota yang berisi nama kota. Setiap nama kota k ditambahkan ke dalam objek jalur menggunakan metode tambahkanKota(k), yang menyimpan kota-kota dalam atribut daftarKota.



Di bagian ini, menambahkan jalan antar kota beserta jaraknya. Setiap kali memanggil metode tambahkanJalan(kota1, kota2, jarak), memastikan bahwa dua kota tersebut terhubung dalam graf dengan jarak yang sesuai. Ini mengupdate baik daftar kota yang terhubung dalam daftarKota maupun jarak antar kota dalam bobotJarak.



Fungsi ini akan menjalankan algoritma Dijkstra untuk menghitung jarak terpendek dari kota Jakarta ke semua kota lain yang ada di dalam graf.



Bagian ini menampilkan hasil dari perhitungan algoritma Dijkstra. Setiap kota yang ada dalam dictionary distances akan ditampilkan bersama jarak terpendek yang ditemukan oleh algoritma.

Presentasi

**Presentasi Kode: Mencari Jalur Tercepat Menggunakan Algoritma Dijkstra**

Pada bagian pertama, kita akan memulai dengan pengenalan kelas **Kota**, yang berfungsi untuk mengelola daftar kota dan jalan-jalan yang menghubungkan kota-kota tersebut.

**1. Kelas Kota:**

* Di sini, kita memiliki kelas **Kota** yang menyimpan data kota dan jalan antar kota.
* Ketika objek dari kelas **Kota** dibuat, pada bagian **\_\_init\_\_()** (constructor), kita menyiapkan sebuah dictionary kosong yang disebut **daftarKota**. Dictionary ini akan digunakan untuk menyimpan kota-kota dan kota tetangganya.
* Fungsi **tambahkanKota(kota)** memungkinkan kita untuk menambahkan kota ke dalam graf kota jika kota tersebut belum ada. Nama kota dipastikan dalam format yang benar menggunakan **kota.title()** (misalnya: "jakarta" menjadi "Jakarta").
* Fungsi **tambahkanJalan(kota1, kota2)** menambahkan jalan dua arah antara dua kota. Kita memeriksa apakah kedua kota ada di dalam **daftarKota**, kemudian menambahkan kota satu sama lain sebagai tetangga.
* Fungsi **tampilkanKota()** akan menampilkan semua kota yang ada beserta tetangga-tetangganya, memberikan gambaran bagaimana graf kota kita terhubung.

**Alur Kelas Kota:**

* Dengan kelas **Kota** ini, kita bisa membuat dan mengelola graf kota dan hubungan antar kota. Setiap kota memiliki daftar kota tetangga yang terhubung dengannya melalui jalan.

**2. Kelas JalurTercepatDijkstra:**

Setelah kelas **Kota**, kita perlu kelas yang lebih spesifik untuk menghitung jalur terpendek antar kota menggunakan algoritma **Dijkstra**. Inilah fungsi kelas **JalurTercepatDijkstra**.

* **JalurTercepatDijkstra** adalah turunan dari kelas **Kota**. Jadi, objek **JalurTercepatDijkstra** akan mewarisi semua fungsi dari **Kota** (seperti menambahkan kota dan jalan antar kota).
* Dalam **\_\_init\_\_()**, kita juga menyiapkan atribut baru **bobotJarak** yang akan menyimpan jarak antar kota. Ini adalah dictionary di mana setiap kota akan menyimpan informasi jarak ke kota tetangganya.
* Fungsi **tambahkanJalan(kota1, kota2, jarak)** di sini sedikit berbeda. Selain menambahkan hubungan antar kota, kita juga menyertakan jarak antara kedua kota tersebut dalam atribut **bobotJarak**.

**Kenapa kita perlu menambahkan jarak?**

* Ini karena tujuan kita adalah mencari **jalur terpendek** antara kota-kota. Jarak inilah yang akan digunakan oleh algoritma Dijkstra untuk menghitung jalur dengan biaya terendah.

**3. Implementasi Algoritma Dijkstra:**

Sekarang mari kita bahas inti dari kode ini, yaitu algoritma **Dijkstra** untuk menghitung jalur terpendek.

* Fungsi **dijkstra(start)** akan menerima parameter **start** (kota awal), dan bertujuan untuk menghitung jarak terpendek dari kota awal ini ke semua kota lainnya dalam graf.

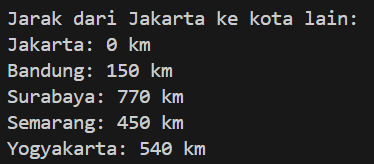
**Prosesnya adalah sebagai berikut**:

1. **Inisialisasi**:
   * Kita menyiapkan sebuah set **unvisited** yang berisi semua kota yang belum dikunjungi. Kemudian, **distances** diinisialisasi untuk menyimpan jarak dari kota awal ke kota-kota lainnya, dimulai dengan **infinity** untuk semua kota, kecuali kota awal yang bernilai 0.
   * **previous** diinisialisasi untuk menyimpan kota sebelumnya dalam jalur terpendek, meskipun pada bagian kode ini tidak digunakan untuk menampilkan jalurnya.
2. **Mengunjungi Kota**:
   * Selama masih ada kota yang belum dikunjungi, kita memilih kota yang jaraknya paling pendek dari kota awal. Ini dilakukan dengan memilih kota dengan nilai **distances[k]** terkecil di antara kota-kota yang belum dikunjungi.
   * Setelah memilih kota tersebut, kita mengunjungi kota-kota tetangga dan menghitung jarak baru yang lebih pendek.
   * Jika jarak baru ini lebih kecil daripada jarak sebelumnya, kita perbarui jarak di **distances** dan juga simpan kota tersebut sebagai kota sebelumnya dalam **previous**.
3. **Proses Berulang**:
   * Proses ini berlanjut hingga semua kota telah dikunjungi dan jarak terpendek ke setiap kota telah ditemukan.

**Hasilnya**:

* Fungsi **dijkstra(start)** akan mengembalikan dua dictionary:
  + **distances** yang berisi jarak terpendek dari kota awal ke setiap kota lainnya.
  + **previous** yang berisi informasi kota sebelumnya dalam jalur terpendek (meskipun tidak digunakan di sini untuk menampilkan jalur).
*  Di bagian akhir ini, kita membuat objek **jalur** dari kelas **JalurTercepatDijkstra** dan menambahkan beberapa kota ke dalam graf, serta menambahkan jalan beserta jaraknya antar kota.
*  Setelah itu, kita menjalankan **dijkstra("Jakarta")** untuk menghitung jarak terpendek dari **Jakarta** ke semua kota lain.

Hasil dari fungsi **dijkstra** akan ditampilkan dalam bentuk jarak terpendek dari Jakarta ke kota lainnya, seperti:



Dari hasil ini, kita bisa melihat bahwa jarak terpendek dari **Jakarta** ke **Bandung** adalah 150 km, sementara jarak ke **Surabaya** adalah 770 km, dan seterusnya.u0