الگوریتم اصلی برنامه:

الگوریتم کد برای مدلسازی و مسیریابی در گراف با استفاده از الگوریتم Dijkstra به صورت زیر است:

1. ایجاد کلاس‌ها و تعاریف اولیه:
- ایجاد کلاس `Node` برای نگه‌داری نودها با ویژگی‌های شناسه و نوع.
- ایجاد کلاس `Edge` برای نگه‌داری یال‌ها با ویژگی هزینه.
- ایجاد کلاس `Universe` برای نگه‌داری نودهای یک کهکشان.
- ایجاد کلاس `Galaxy` برای نگه‌داری نودها و یال‌ها در یک کهکشان.

2. تابع `shortestPath`:
- ایجاد تابع `shortestPath` برای پیدا کردن مسیر کوتاه‌ترین هزینه بین دو نود با استفاده از الگوریتم Dijkstra.
- استفاده از صف اولویت برای پیمایش بهینه گراف.
- ذخیره کردن مسیر و هزینه‌ها در دیکشنری‌های مناسب.

3. تابع `main`:
- ایجاد تابع `main` برای اجرای برنامه.
- ایجاد نودها و اضافه کردن آنها به کهکشان و گراف.
- دریافت دستورات کاربر و انجام اقدامات مختلف:
  - ایجاد نود جدید.
  - ایجاد یال بین نودها.
  - مسیریابی داخل یک کهکشان.
  - مسیریابی بین کهکشان‌ها.

توضیح کد:

1. `Node` و `Edge` Classes:
- این دو کلاس به تعریف نودها و یال‌ها به منظور ایجاد ساختار گراف کمک می‌کنند.

2. `Universe` Class:
- کلاسی برای نگه‌داری نودها در یک کهکشان. دارای تابع `addNode` برای اضافه کردن نودها به کهکشان است.

3. `Galaxy` Class:

- کلاسی برای نگه‌داری نودها و یال‌ها در یک کهکشان. دارای تابع `addEdge` برای اضافه کردن یال‌ها به گراف کهکشانی است.

4. `shortestPath` Function:

- الگوریتم Dijkstra را پیاده‌سازی می‌کند تا کوتاه‌ترین مسیر و مجموع هزینه بین دو نود را پیدا کند.
- دریافت مقادیر گراف، نود شروع و نود مقصد.
- ایجاد متغیرهای نگه‌دارنده فاصله‌ها و نودهای قبلی.
- استفاده از یک صف اولویت برای پیمایش گراف و به‌روزرسانی مسیر کوتاه‌تر.
- نهایتاً مسیر کوتاه‌تر و مجموع هزینه را چاپ می‌کند.

5. `main` Function:

- ایجاد کهکشان و نودهای آن.
- ایجاد یال‌هایی بین نودها.
- دریافت دستورات کاربر و اجرای موارد مختلف.
- دستورات 3 و 4 که کار با گراف و پیدا کردن کوتاه‌ترین مسیر را انجام می‌دهند، در اینجا پیاده‌سازی می‌شوند.
- حذف نودها برای جلوگیری از خرابی حافظه.

6. **Case 1: ایجاد نود جدید**

- در این قسمت، یک نود جدید با استفاده از دستور `CREATE` ایجاد می‌شود.
- ابتدا اطلاعات مربوط به نود از ورودی خوانده می‌شود، شامل شناسه و نوع نود.
- سپس یک نمونه از کلاس `Node` با اطلاعات ورودی ایجاد و به کهکشان (`Galaxy`) اضافه می‌شود.

7. **Case 2: ایجاد یال بین نودها**

- در این قسمت، یک یال بین دو نود موجود با استفاده از دستور `CREATE` ایجاد می‌شود.
- ابتدا اطلاعات مربوط به نودها و ویژگی‌های یال از ورودی خوانده می‌شود، شامل نود ابتدایی، نود پایانی و هزینه یال.
- سپس یک نمونه از کلاس `Edge` با اطلاعات ورودی ایجاد و به کهکشان (`Galaxy`) اضافه می‌شود.

8. **Case 3: مسیریابی درون یک کهکشان**

- ابتدا دستور مسیریابی به صورت `FIND source->destination` خوانده می‌شود، که `source` و `destination` نودهای مورد نظر هستند.

- تابع `shortestPath` با استفاده از الگوریتم Dijkstra فراخوانی می‌شود تا مسیر کوتاه‌ترین مسیر و هزینه آن بین دو نود محاسبه شود.

- مسیر و هزینه محاسبه شده نمایش داده می‌شود.

9. **Case 4**: مسیریابی بین کهکشان‌ها

- ابتدا دستور مسیریابی بین کهکشان‌ها به صورت `FIND source->destination` خوانده می‌شود.

- برای مسیریابی بین کهکشان‌ها، نود مربوط به کهکشان‌ها باید ایجاد شده باشند.

- تابع `shortestPath` با استفاده از الگوریتم Dijkstra فراخوانی می‌شود تا مسیر کوتاه‌ترین مسیر و هزینه آن بین دو نود (که در واقع نودهای مرتبط با کهکشان‌ها هستند) محاسبه شود.

- مسیر و هزینه محاسبه شده نمایش داده می‌شود.

این گزارش کامل نشان می‌دهد که هر تابع و هر خط از کد چه کارهایی را انجام می‌دهد تا اجرای موارد مختلف پروژه را انجام دهد.