پاسخ سوال ۳۰ — دایجسترا با وزن‌های کوچک

صورت سوال

فرض کنید می‌خواهیم الگوریتم دایجسترا را روی گرافی اجرا کنیم که وزن یال‌های آن اعداد صحیح در بازه $\{0, 1, \ldots, W\}$ هستند، که در آن W عدد نسبتاً کوچکی است.

(الف) نشان دهید چگونه می‌توان الگوریتم دایجسترا را به گونه‌ای پیاده‌سازی کرد که در زمان

$$O(W|V| + |E|)$$

اجرا شود.

(ب) یک پیاده‌سازی جایگزین دهید که زمان اجرای آن برابر

$$O((|V| + |E|) \log W)$$

باشد.

پاسخ

نقش صف اولویت در الگوریتم دایجسترا

الگوریتم دایجسترا برای محاسبهٔ کوتاه‌ترین مسیر از یک رأس مبدأ به تمام رأس‌های گراف، همواره نیاز دارد رأسی را انتخاب کند که در میان تمام رأس‌های پردازش‌نشده کمترین فاصلهٔ موقت را از مبدأ دارد.
برای انجام این کار از یک *Priority Queue* (صف اولویت) استفاده می‌شود. در این صف، هر رأس با کلید (*key*) برابر با فاصلهٔ موقت فعلی آن از مبدأ نگهداری می‌شود. مهم‌ترین عملیاتی که روی این ساختار انجام می‌شود عبارت‌اند از:

- *Extract-Min*: استخراج رأسی که کمترین فاصلهٔ موقت را دارد.
- *Insert*: اضافه کردن یک رأس جدید به صف.
- *Decrease-Key*: کاهش مقدار کلید یک رأس هنگامی که در عملیات *Relax* مسیر کوتاه‌تری برای آن پیدا می‌شود.

در پیاده‌سازی استاندارد دایجسترا معمولاً از *Binary Heap* برای صف اولویت استفاده می‌شود. در این حالت هر عملیات *Extract-Min* و *Decrease-Key* در زمان

$$O(\log |V|)$$

انجام می‌شود و در نتیجه زمان اجرای کل الگوریتم برابر

$$O((|V| + |E|) \log |V|)$$

خواهد بود.
اما در این مسئله از این ویژگی استفاده می‌کنیم که وزن تمام یال‌ها اعداد صحیح کوچکی در بازهٔ

$$\{0, 1, \ldots, W\}$$

هستند. این محدودیت باعث می‌شود بتوان صف اولویت را با ساختارهای ساده‌تری نسبت به هیپ پیاده‌سازی کرد و زمان اجرای الگوریتم را به

$$O(W|V| + |E|)$$

یا حتی

$$O((|V| + |E|) \log W)$$

کاهش داد.
ابتدا یک مشاهدهٔ مهم در مورد الگوریتم دایجسترا داریم.
اگر در یک مرحله رأسی را با فاصلهٔ مینیمم D از صف اولویت خارج کنیم، تمام رأس‌های دیگری که در همان لحظه داخل صف هستند یا در این مرحله به‌روزرسانی می‌شوند، فاصله‌ای حداکثر برابر با $D + W$ خواهند داشت. علت این موضوع آن است که بزرگ‌ترین وزن هر یال برابر W است و هر فاصلهٔ جدید از طریق افزودن وزن یک یال به فاصلهٔ یک رأس پردازش‌شده به دست می‌آید.
بنابراین در هر لحظه اختلاف بین کمترین فاصله و بیشترین فاصله در میان رأس‌های پردازش‌نشده حداکثر W است.

الف)

برای این قسمت از ایدهٔ *Bucket Sort* استفاده می‌کنیم.

ساختار داده

به جای استفاده از یک *Min-Heap* معمولی، یک آرایه از باکت‌ها (لیست‌های پیوندی) ایجاد می‌کنیم.
از آنجا که طولانی‌ترین مسیر ممکن در گراف حداکثر شامل $|V| - 1$ یال است و هر یال حداکثر وزن W دارد، بیشترین فاصلهٔ ممکن از مبدأ از مرتبهٔ

$$W|V|$$

خواهد بود.
بنابراین آرایه‌ای شامل

$$W|V| + 1$$

باکت در نظر می‌گیریم.
باکت شمارهٔ i شامل تمام رأس‌هایی است که فاصلهٔ موقت آن‌ها از مبدأ دقیقاً برابر i است.

نحوهٔ اجرا

از اندیس صفر شروع کرده و اولین باکت غیرخالی را پیدا می‌کنیم. رأس‌های موجود در آن باکت را پردازش می‌کنیم و در صورت انجام عملیات *Relax*، رأس مقصد را به باکت متناظر با فاصلهٔ جدیدش منتقل می‌کنیم.

از آنجا که فواصل در الگوریتم دایجسترا به صورت صعودی استخراج می‌شوند، هرگز لازم نیست به اندیس‌های قبلی بازگردیم.

تحلیل زمان اجرا

- پیمایش باکت‌ها در کل اجرای الگوریتم حداکثر

$$O(W|V|)$$

زمان می‌برد.

- هر یال دقیقاً یک بار بررسی می‌شود و هر جابه‌جایی بین باکت‌ها در زمان ثابت انجام می‌شود؛ بنابراین هزینهٔ بررسی یال‌ها برابر

$$O(|E|)$$

است.

در نتیجه زمان اجرای کل الگوریتم برابر است با

$$O(W|V| + |E|).$$

ب)

در روش قسمت (الف) مشکل اصلی این بود که برای پیدا کردن کوچک‌ترین فاصله ممکن بود تعداد زیادی باکت خالی پیمایش شوند. برای حذف این هزینه از ساختار *Radix Heap* استفاده می‌کنیم.

همان‌طور که در ابتدا مشاهده کردیم، اگر کوچک‌ترین فاصلهٔ موجود در صف اولویت برابر D باشد، تمام فاصله‌های دیگر در بازهٔ

$$[D, D + W]$$

قرار دارند. بنابراین اختلاف فاصلهٔ هیچ رأسی با مینیمم فعلی بیشتر از W نیست.

ساختار داده

در *Radix Heap* به جای داشتن یک باکت برای هر مقدار فاصله، باکت‌ها بر اساس اندازه‌های توانی از دو ساخته می‌شوند. به بیان دیگر، باکت‌ها بازه‌هایی با طول‌های

$$1,\ 2,\ 4,\ 8,\ \ldots$$

را پوشش می‌دهند.

از آنجا که بیشترین اختلاف ممکن بین یک فاصله و مینیمم فعلی برابر W است، تنها

$$O(\log W)$$

باکت برای پوشش دادن تمام این اختلاف‌ها کافی خواهد بود.

هر رأس بر اساس مقدار اختلاف فاصلهٔ خود با مینیمم فعلی در یکی از این باکت‌ها قرار می‌گیرد.

تحلیل زمان اجرا

در الگوریتم دایجسترا هر رأس حداکثر یک بار از صف اولویت استخراج می‌شود، بنابراین در مجموع $|V|$ عملیات *Extract-Min* داریم.

همچنین هر یال دقیقاً یک بار در عملیات *Relax* بررسی می‌شود. هر عملیات *Relax* ممکن است باعث کاهش فاصلهٔ رأس مقصد شود و در نتیجه یک عملیات *Decrease-Key* یا جابه‌جایی در ساختار *Radix Heap* انجام گیرد. بنابراین تعداد کل این عملیات‌ها از مرتبهٔ $|E|$ است.

در *Radix Heap* هر رأس در طول اجرای الگوریتم تنها بین تعداد محدودی از سطوح جابه‌جا می‌شود و تعداد این سطوح برابر $O(\log W)$ است. در نتیجه هزینهٔ سرشکن عملیات‌های مربوط به هر رأس یا هر کاهش فاصله از مرتبهٔ

$$O(\log W)$$

خواهد بود.
بنابراین:

$$|V|$$

عملیات استخراج رأس و

$$|E|$$

عملیات احتمالی کاهش فاصله داریم که هر کدام هزینه‌ای از مرتبهٔ

$$O(\log W)$$

دارند. در نتیجه زمان اجرای کل برابر است با

$$O((|V| + |E|) \log W).$$

بنابراین با استفاده از *Radix Heap*، بدون نیاز به پیمایش تعداد زیادی باکت خالی، زمان اجرای دایجسترا به

$$O((|V| + |E|) \log W)$$

کاهش می‌یابد.