امیرحسین نجفی زاده 9831065 – دومین تشریحی

فرض کنید یک جدول درهم را پیاده سازی می‌کنیم و می‌خواهیم بین `open addressing` و `chaining` یکی را برای پیاده سازی جدول انتخاب کنیم. برای دو استراتژی از یک *hash function* و *hash table size* برابر استفاده می‌کنیم. به سوالات زیر پاسخ دهید:

الف) تصادف اولیه (*primary collision*) زمانی رخ می‌دهد که دو کلید به یک *slot* توسط تابع درهم ساز نگاشته (*map*) شوند. (در واقع *hash function* یک مقدار یکسان به ازای دو ورودی مختلف می‌دهد.) به ازای کدام یک از دو روش پیاده سازی تعداد *primary collision* کمتری داریم؟

ب) یکی از اشکالات استفاده از `chaining` و `linear search` ، یک لیست پیوندی برای پیدا کردن یک *record* است که هزینه آن زیاد است. آیا استفاده از `open addressing` و کاوش خطی جستجو را بهینه‌تر می‌کند؟ (در صورتی که بدانیم تعداد کم ولی قابل توجه‌ای *collision* داریم.)

ج) اگر از کاوش درجه دو استفاده کنیم چطور؟

د) به طور کلی چه زمانی از `open addressing` و چه زمانی از `chaining` استفاده می‌کنیم؟

پاسخ :

الف : در اولین استفاده از تابع Hash Function برای هر دو روش خروجی معمولا یکی است. امکان primary collision در روش chaining بیشتر است. به دلیل اینکه در این روش ما برای یک کلید ورودی به Hash Function تنها یک خروجی را بررسی می کنیم و در صورت collision تغییری بر روی کلید Hash شده نمی دهیم. اما در open addressing معمولا تنها Hash Function ها بر روی Hash شدن کلید تاثیر نمی گذارند و بر اساس تعریف کردن توابع probe function می توان به هنگام collision گزینه های بیشتری را بررسی کنیم. اما در شروع کار چون probe function در تلاش 0 ام خود می باشد نتیجه Hash کردن ها در هر دو روش یکسان می باشد ، پس در شروع کار با فرض Hash Function های یکسان و probe function فاقد جمله ی ثابت ، هر دو روش احتمال collision یکسانی دارند و ادامه فرآیند می باشد که open addressing را بهینه می کند.

ب : در روش chaining ما به ازای یک کلید ورودی تنها یک خروجی Hash شده را بررسی می کنیم ، این کار احتمال collision را بیشتر می کند. بنابر این تعداد اعضا در لیست های پیوندی بیشتر می شود (با توجه به افزایش load factor). به همین جهت برای جست و جو باید درگیر سرچ کردن در لیست های پیوندی و جابه جایی با pointer ها بشویم که نسبتا هزینه زیادی می برد. اما با استفاده از تکنیک open addressing دیگر نیازی به وجود لیست های پیوندی و جست و جوی خطی نمی باشد. با تعریف کردن probe function خوب می توانیم خانه های مختلفی را بررسی کنیم که همه در یک آرایه هستند پس دسترسی به آن ها از پیچیدگی زمانی O(1) می باشد و در اینجا فقط یک سری محاسبات عددی نیاز داریم. از این جهت تکنیک open addressing جست و جو را بهینه می کند.

ج : استفاده از کاوش درجه دو به مراتب از کاوش درجه اول بهتر است ، زیرا طبق رابطه hash(key,i) = h(key) + p(i) اگر فرض کنیم مقدار h(key) برای یک کلید همواره ثابت است ، خطی یا غیر خطی بودن رابطه را p(i) مشخص می کند. هرچه رابطه ما از خطی بودن به غیرخطی بودن تبدیل بشود عملیات open addressing بهتر می شود ، زیرا در رابطه های خطی معمولا در یک بازه یک مسیر پله پله ایی با ترتیب مشخص را برای یافتن slot خالی پیمایش می کنیم ، این عملیات در مقادیر کوچک تعداد slot ها مناسب است اما در وضعیت هایی با تعداد slot زیاد به دلیل خطی بودنش زمان زیادی را برای یافتن یک slot خالی سپری می کند. اما رابطه غیر خطی (در اینجا درجه 2) به مراتب گام های بیشتری را در هر مرحله افزایش i طی می کند و امکان بررسی خانه های مختلف را برای ما فراهم می کند. در این صورت ما برای یافتن خانه ی خالی معمولا ترتیب خاصی را در پیش نمیگیریم و خانه های مختلف را برای وجود slot خالی را بررسی می کنیم که زمان یافتن slot خالی به ویژه در مقادیر تعداد بالای slot ها کاهش می دهد.

د : برای نتیجه گیری کلی ابتدا جدول زیر را بررسی می کنیم.

OPEN ADDRESSING	CHAINING
نیاز به محاسبات عددی دارد	تنها به hash کردن کلید دارد
دسترسی به خانه های مختلف از مرتبه O(1) است	برای دسترسی به یک خانه معلوم به پارامتر های loading factor و تنوع خروجی hash function وابسته است
می تواند تعداد محدودی عنصر را در خود ذخیره کند	محدودیتی در ذخیره سازی تعداد عناصر ندارد (به دلیل وجود لیست های پیوندی)
به هنگام استفاده فضای اضافی در حافظه اشغال نمی کند و تمامی SLOT ها را پر می کند	ممکن است با توجه به تعریف hash function خانه های خالی اضافی در حافظه ایجاد کند
تمامی SLOT ها حتی اگر خالی باشند استفاده می شوند و SLOT اضافی نداریم	بعضی slot ها ممکن است اصلا استفاده نشوند

با توجه به جدول فوق می توان این نتیجه را گرفت که ، وقتی از تعداد عناصر خبر نداریم و همچنین نمی دانیم بین کلیدهای داده شده رابطه ایی وجود دارد یا خیر از chaining استفاده می کنیم ، اما زمانی که از تعداد کل عناصر مطلع باشیم یا کران بالای تعداد عناصر را بدانیم ، استفاده از open addressing بهتر می باشد. همچنین اگر بدانیم بین کلید های ورودی یک رابطه ایی بر قرار است استفاده از open addressing بهتر است زیرا در این حالت مقادیر hash شده به مراتب دارای collision کمتری می باشند. البته این را هم در نظر بگیرید که در مقادیر اندک slot ها معمولا open addressing کارایی ندارد چون مقادیر اندکی از عناصر را می تواند ذخیره کند ، تنها در شرایطی که بحث حافظه و تعداد عناصر مطرح است chaining نسبت به open addressing برتری دارد. اما در مواقعی که سرعت دسترسی به داده ها هدف است open addressing روش بهینه می باشد.