Анализ на задача machine

Тагове: конструктивна задача, префикси, сегментно дърво, разделяй и владей

В обобщение, задачата запознава състезателя със силно ограничен абстрактен модел на изчисление, в който могат да се пишат програми (или по-удобно за осмисляне да се създават графи), съдържащи единствено аритметичните операциите събиране и изваждане. На тази програма ще се даде двумерна таблица и ще се очаква като изход тя да върне префиксните суми на тази двумерна таблица. Целим да ползваме, колкото се може по-малко върхове и графа, който създадем да е колкото се може по-"нисък" (да няма дълги верижки от върхове).

По-лесно е да се мисли за тези програми като графи (а и така ги въвежда условието на задачата), заради това ще говорим със съответната терминология от върхове и ребра в графи. С $A_{i,j}$ ще означаваме върха, който съдържа j-тото число от i-тия ред на таблицата. С $u = v \oplus w$ ще означаваме, че създаваме нов връх, който е получен чрез операция $\oplus \in \{+, -\}$, приложена върху v и w.

Идеи, основани на префиксни суми

Почти рудиментарно, очевидно задачата ще може да се започне с някакви идеи, сходни на начините по които човек би си сметнал префиксните суми на матрица в `C++`. За тази цел има две популярни идеи:

$$p_{i,j} = p_{i-1,j} + p_{i,j-1} - p_{i-1,j-1} + a_{i,j}$$

тук с p означаваме префиксната матрица, а с a – входната матрица. Ако наивно "навържем" трите операции една след друга ще получим по-високо, от колкото е нужно. По-оптимално е първо да съберем първите два аргумента в сумата, да извадим от третия аргумент четвъртия и накрая от първата сума да извадим разликата. Алгебрически, оптимизацията, за която говорим би била:

$$p_{i,j} = ((p_{i-1,j} + p_{i,j-1}) - p_{i-1,j-1}) + a_{i,j} \quad (1)$$
$$p_{i,j} = (p_{i-1,j} + p_{i,j-1}) - (p_{i-1,j-1} - a_{i,j}) \quad (2)$$

да извършим равенство (2), вместо (1), защото там няма вложен скоби.

Друга популярна стратегия за същия проблем е да направим първо префиксни суми, а после префиксни суми отделно за всяка колона. Що се отнася за решенията в тази секция, това е най-оптималното – "най-лошия" връх ще бъде този в последната колона и последния ред, който ще седи над $N + M - 2$ други върха. Допълнително, този подход не се нуждае от два "междинни" върха за всяка сметка (но пък първите префиксни суми също не са супер полезни, освен в по-късните сметки).

Идеи, основани на сегментни дървета

Всъщност друг бонус на последното решение от предходната секция е, че то много гладко ни води към още по-добри идеи. Въпреки че обикновено сегментното дърво / дървото на Фенуик биха били overkill за простата задача за изчисляване на префиксни суми (че и статични на всичкото отгоре), тука са много уместни именно защото извършват малък брой "вложени" изчисления (същата причина, поради която по-рано предпочетохме (2) пред (1)).

Единственият по-интересен момент тук е как от едно построено сегментно дърво може да се извлече префиксна сума в текущия модел на изчисление. Ще направим напълно нормална заявка за интервал в сегментното дърво, с единствената модификация, че при извикването `query(node, l, r, ql, qr)`[1] ще ни връща връх в графа, който строим, съдържащ сума-

[1] Тук $[l, r]$ е интервала, чиято сума е във връх $node$, а $[ql, qr]$ е интервала, чиято сума търсим

та на интервала $[\max(ql, l), \min(qr, r)]$. В случай, че този интервал е празен, ще означим това със специален символ (примерно ще държим връх 0 празен), а сливането на две заявки, тоест върховете получени от тях, се получава сравнително интуитивно чрез създаване на нов връх.

Въпреки че не става ясно защо (поне дотук в анализа), първо да построим сегментни дървета по редовете на матрицата и след това да построим такива върху колоните е решение за 100 точки. Височината на получения граф е $\log_2 N + \log_2 M = \log N \times M$.

Идеи, основани на разделяй и владей метод

Друг (и доста по-досаден) начин да изкараме 100 точки на задачата е да извършим "разделяй и владей" върху таблицата. На всяка стъпка ще разделяме подматрица $[u, d, l, r]$ на четири горе-долу равни части и ще сливаме решенията от тях. Наистина няма нещо прекалено интересно в това решение (освен прекалено писане на код), заради това няма да влизаме в допълнителни детайли (такива няма).

Доказателство за оптималност

Очевидно ограничението по височина е по-важното и по-трудното за изкарване. Ще се фокусираме върху него – достатъчно ниската бройка на върховете ще дойде като страничен фактор.

Нека разгледаме последната префиксна сума – тя задължително ще съдържа сумата на всички $N \times M$ върха под себе си. Да допуснем обаче, че съществува решение, което намира тази сума във връх, който е на височина $k < \log_2 N \times M$. Този връх обаче може да има най-много 2^k деца в основата (които съответно трябва да са входните числа в матрицата). Това обаче означава, че сме получили сумата на цялата матрица, сумирайки $2^k < 2^{\log_2 N \times M} = N \times M$ числа от входа. Това обаче е невъзможно. Така чрез противоречие показахме, че оптималното решение трябва да строи граф с височина поне $\log_2 N \times M$ (това е *долна граница*). Ние имаме и алгоритъм, който постига тази стойност (съществуването на алгоритъма ни дава *горна граница*), следователно можем да сме уверени, че това е именно оптималната височина на граф, решаващ задачата.

Допълнително

Описаният в задачата абстрактен модел, в който програмите се конструират и разглеждат като графи от математически операции, е класически пример за изчислителен граф (computational graph). Те намират практически ползи в машинното обучение чрез автоматично диференциране, оптимизацията на компилатори и паралелните изчисления.

За по-задълбочено изучаване на алгоритмите зад изчислителните графи и техниките за тяхното оптимизиране, можете да разгледате следните ресурси и задачи:

- Книгата "Deep Learning" (Ian Goodfellow, Yoshua Bengio, Aaron Courville): Глава 6.5 е конкретно се занимава с изчислителните графи и ролята им в *backpropagation*;
- Проектът "micrograd" на Andrej Karpathy (GitHub/YouTube): Практическо ръководство, което показва как да се програмира от нулата базов engine за изчислителни графи и автоматично диференциране.

Автор: Иван Лупов