Анализ на задача calculations

Тагове: матрично умножение, binary lifting, графи, битови маски

Рекурентната връзка за намиране на елементите на редицата работи по модул 2, което означава, че събирането е еквивалентно на побитово изключващо или (XOR), а умножението – на побитово И (AND).

Тъй като рекурентната зависимост е линейна от ред k, стандартният подход за намиране на n-тия член е чрез повдигане на преходна матрица в степен. Ако дефинираме състоянието в стъпка i като стълб-вектор от предходните k елемента:

$$A^{(i)} = \begin{pmatrix} a_{i-k} \\ a_{i-k+1} \\ \vdots \\ a_{i-1} \end{pmatrix}$$

то матрицата на прехода T има размер $k \times k$, в която първите $k-1$ реда реализират отместване с единици по наддиагонала, а последният ред съдържа коефициентите m:

$$T = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ m_1 & m_2 & m_3 & \dots & m_k \end{pmatrix}$$

Преходът между две последователни състояния се задава с умножението:

$$A^{(i+1)} = T \cdot A^{(i)} = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ m_1 & m_2 & m_3 & \dots & m_k \end{pmatrix} \begin{pmatrix} a_{i-k} \\ a_{i-k+1} \\ \vdots \\ a_{i-2} \\ a_{i-1} \end{pmatrix} = \begin{pmatrix} a_{i-k+1} \\ a_{i-k+2} \\ \vdots \\ a_{i-1} \\ \bigoplus_{j=1}^{k} (m_j \wedge a_{i-k+j-1}) \end{pmatrix}$$

Тогава състоянието в стъпка n се получава директно от началния вектор $A^{(k+1)} = \begin{pmatrix} a_1 & a_2 & \dots & a_k \end{pmatrix}^T$ чрез:

$$A^{(n)} = T^{n-k} \cdot A^{(k+1)}$$

а търсеният отговор a_n е последният елемент на вектора $A^{(n)}$.

При T заявки директното пресмятане на степени на матрица за всяка заявка отнема $O(T \cdot k^3 \log n)$ време. Това би било твърде бавно за пълния брой точки, тъй като T достига 5×10^5.

Първа оптимизация е предварително да се сметнат степените на матрицата на прехода за степени на двойката: $T^{2^0}, T^{2^1}, \dots, T^{2^{29}}$. При всяка заявка векторът от началните k елемента може да се умножава последователно само по съответните предварително изчислени матрици, вместо да се преумножават самите матрици помежду си. Това свежда сложността на всяка заявка до векторно-матрични умножения, или $O(k^2 \log n)$.

За пълно решение може да се използва друго ключово наблюдение: $k \leq 20$. Това означава, че състоянието на всеки k последователни елемента се определя от двоична маска с k бита, като общият брой възможни състояния е 2^k (най-много 2^{20}, което е около 10^6).

Всеки преход от дадено състояние към следващото е детерминиран и зависи единствено от стойностите на коефициентите m. Тъй като от всяко състояние излиза точно едно ребро, можем да разглеждаме преходите като функционален граф.

След като намерим директния наследник за всяка от 2^k маски, можем да приложим двоично повдигане:

- `jmp[0][mask]` пази състоянието след 1 стъпка;
- `jmp[l][mask] = jmp[l - 1][jmp[l - 1][mask]]` пази състоянието след 2^l стъпки.

Предварителното изчисление отнема $O(2^k \cdot (k + \log n))$ време и памет. След това всяка от T-те заявки се обработва за $O(\log n)$ чрез преминаване по двоичния запис на $n - k$.

Струва си да се отбележи и алтернативното оптимално решение, което отново използва функционалния граф, но прави линеен прекомпют спрямо неговия размер. Вместо двоично повдигане, можем да обходим графа от 2^k върха. Тъй като от всеки връх излиза точно едно ребро, графът е съставен от компоненти, всяка от които представлява цикъл с евентуално вливащи се в него дървета. С едно обхождане в дълбочина за време $O(2^k)$ можем да намерим всички цикли, техните дължини, както и разстоянието от всеки връх до съответния цикъл. След тази предварителна обработка, всяка от T-те заявки се отговаря за $O(1)$ време – ако n е по-малко от разстоянието до цикъла, взимаме върха по пътя към него, а в противен случай използваме дължината на цикъла за позициониране чрез модулна аритметика. Така общата сложност пада до $O(2^k + T)$, което напълно елиминира логаритмичния множител.

Допълнителни ресурси

- CP-Algorithms: Linear Recurrence Relations and Matrix Exponentiation
- CP-Algorithms: Finding cycles and Successor Graphs
- CSES: Planets Queries I
- CSES: Planets Queries II
- Matrix Exponentiation Codeforces Gym

Автор: Иван Лупов