

Динамизация на структури от данни част 1

15.01.2021

Проблема, който ще разгледаме тук е абстрактен.

Дефиниция: Имаме три множества T_1 , T_2 и T_3 и имаме проблем за търсене: т.е.

функция $Q : T_1 \times 2^{T_2} \rightarrow T_3$. В този проблем няма никакви конкретни типове или

query

обекти, за които да си мислим или някаква конкретна заявка.

Примери:

1) $T_1 = T_2$; $T_3 = \{\text{true}, \text{false}\}$

$Q(x, A) \Leftrightarrow x \in A$, x е елемент от T_1 , а A е множество от елементи на T_1 (с някакво свойство)

2) $T_1 = T_2 = \mathbb{N}$; $T_3 \in \mathbb{N}$

$Q(x, A) = \text{succ}_A(x) = \min\{y \in A \mid y \geq x\}$ или

$Q(x, A) = \text{pred}_A(x) = \max\{y \in A \mid y \leq x\}$

Постановка:

Предположение: Имаме статично решение за проблема Q , т.е. ако е дадено $A \subseteq T_2$ може да индексираме A , $n = |A|$:

1) за време $\mathcal{P}_s(n)$ $\mathcal{P} \equiv \text{Procedure}$

2) с памет $\mathcal{S}_s(n)$ $\mathcal{S} \equiv \text{Space}$

3) отговор на заявка $Q(x, A)$ за време $\mathcal{Q}_s(n)$

(Втория елемент A е фиксиран през цялото време и затова решението се нарича статично)

Дефиниция: (Полу)динамична структура от данни за Q е структура от данни \mathcal{D} ,
dynamic

за която може да:

1) отговаряме на заявки от вида $Q(x, A)$ за време $\mathcal{Q}_{\mathcal{D}}(|A|)$ $Q \equiv \text{Query}$

2) изисква памет $\mathcal{S}_{\mathcal{D}}(|A|)$ $S \equiv \text{Space}$

3) да добавяме елемент $a \notin A$ към A ($I(a, \mathcal{D})$) за време $\mathcal{I}_{\mathcal{D}}(|A|)$ $I \equiv \text{Insert}$

4) да изтриваме елемент $a \in A$ ($D(a, \mathcal{D})$) за време $\mathcal{E}_{\mathcal{D}}(|A|)$ $D \equiv \text{Delete}$

Когато поддържаме само 1) и 3), ние говорим за полудинамична структура от данни, а когато поддържаме 1), 3) и 4) - говорим за динамична структура от данни.

Дефиниция: Наричаме проблема за търсене $Q : T_1 \times 2^{T_2} \times T_3$ разложим, ако има функция $\sqcup : T_3 \times T_3 \rightarrow T_3$ със следните две свойства:

1) за всеки две множества $A, B \subseteq T_2$ и $x \in T_1$, ако $A \cap B = \emptyset$, то

$Q(x, A \cup B) = \sqcup(Q(x, A), Q(x, B))$

2) \sqcup да може да се изчислява за време $O(1)$ (константно време)

Свойства на \sqcup :

1) \sqcup е комутативна и асоциативна за всяко фиксирано x .

Ако $T_{3,x} = \{Q(x, A) \mid A \subseteq T_2\}$ са възможните резултати от заявките $Q(x, A)$, то

тогава ако $A, B, C : \begin{cases} A \cap B = \emptyset \\ B \cap C = \emptyset \\ C \cap A = \emptyset \end{cases}$ са дизюнктни множества:

$$Q(x, A \cup B) = \sqcup (Q(x, B), Q(x, A))$$

$$Q(x, A \cup B \cup C) = \sqcup \left(Q(x, A), \sqcup (Q(x, B), Q(x, C)) \right) = Q(x, (A \cup B) \cup C) = \\ = \sqcup \left(\sqcup (Q(x, A), Q(x, B)), Q(x, C) \right)$$

2) $Q(x, \emptyset)$ играе ролята на единичен елемент в $T_{3,x}$ с операцията \sqcup .

Примери:

1) $T_1 = T_2$; $T_3 = \{\text{true}, \text{false}\}$

$$Q(a, A) \Leftrightarrow x \in A$$

$$A \cap B = \emptyset$$

$$x \in A \cup B \Leftrightarrow x \in A \text{ или } x \in B$$

$$Q(x, A \cup B) = Q(x, A) \vee Q(x, B) \\ \equiv \sqcup$$

2) $T_1 = T_2 = \mathbb{N}$; $T_3 = \mathbb{N}$.

$$Q(x, A) = \text{succ}_A(x)$$

$$\text{Нека } A \cup B = \emptyset$$

$$\text{succ}_{A \cup B}(x) = \min (\text{succ}_A(x), \text{succ}_B(x)) \\ \equiv \sqcup$$

Основното, което ще използваме е асоциативността и възможността да разбием нашето множество на няколко дизюнктни (непресичащи се) множества.

Забележка: Ако имаме няколко множества $M = M_1 \cup M_2 \cup \dots \cup M_n$, $M_i \cap M_j = \emptyset$, за $i \neq j$, то $Q(x, M) = Q(x, \emptyset) \sqcup Q(x, M_1) \sqcup \dots \sqcup Q(x, M_n)$, т.е. подковата ни дава средство да агрегираме отделните резултати.

Твърдение: Ако Q е разложим проблем за търсене с характеристики:

$\mathcal{Q}_s(n)$ - намаляваща

Колкото повече елемента има в нашето множество - заявката не може да стане по-лесна.

$\mathcal{P}_s(n) : \frac{\mathcal{P}_s(n)}{n}$ - намаляваща

Поне трябва да погледнем елементите в това множество, т.е. средно на елемент трябва да отделяме повече време, когато имаме повече елементи.

$\mathcal{S}(n) : \frac{\mathcal{S}_s(n)}{n}$ - намаляваща

За повече елементи най-вероятно ще ни е необходима повече памет за запазването на информация, която свързва текущ елемент с останалите.

то тогава има полудинамична структура \mathcal{D} за проблема Q със следните характеристики:

- 1) $\mathcal{Q}_{\mathcal{D}} \in O(Q_s(n) \times \log(n))$ - т.е. заявката в новата структура от данни се влошава с фактор от логаритъм.
- 2) Паметта не се влошава: $\mathcal{S}_{\mathcal{D}}(n) \in O(\mathcal{S}_{\mathcal{D}}(n))$
space
- 3) $\overline{\mathcal{F}_{\mathcal{D}}}(n) \in O\left(\frac{\mathcal{P}_s(n)}{n} \times \log(n)\right)$ - амортизираното време за n операции
(амортизирано)
по добавяне.

Динамизацията идва на цената на допълнителен фактор от $\log(n)$ за n операции.

Доказателство:

Във всеки момент от време, ако M е множеството което представяме и $|M| = n$, то това което правим е да представим n в двоична бройна система: $n = \sum_{i=0}^k a_i 2^i$ (и ако

се абстрахираме от първия момент, в който $n = 0$, може да смятаме, че $a_k = 1$ за $n > 0$ и $k = 0$ за $n = 0$). Представяме нашето множество M като обединение на $k + 1$ множества със следното свойство:

$$M = \bigcup_{i=0}^k M_i, \text{ така че:}$$

- 1) $M_i \cap M_j = \emptyset, \forall i \neq j$ (дизюнктни)
- 2) $|M_i| = a_i 2^i$, т.е. броя на елементите в множеството M_i или ще бъде равен на 0 или ще бъде равен на i -тата степен на двойката (0 ще бъде, ако съответния бит в двоичното представяне на n е 0, в противен случай ще бъде 2^i)

Това което поддържае е следното:

За всяко $i = \overline{0, k}$ ($\forall i = 0, 1, 2, \dots, k$):

- 1) S_i - статична структура за множеството M_i
- 2) L_i - списък от елементите на M_i

ОПЕРАЦИИ:

1) **Търсене:**

Вход: $x \in T_1$

$r \leftarrow Q_s(x, \emptyset)$

for $i = 0$ **to** k **do**

if $L_i \neq \emptyset$ **then**

$r \leftarrow r \sqcup Q_s(x, S_i)$

done

return r

Времева сложност: Очевидно имаме най-много $\log n$ итерации за цикъла и

$$O\left(\log_2(n) + \sum_{i=1}^k Q_s(|S_i|)\right) \leq O\left(\log_2(n) + \sum_{i=1}^k Q_s(n)\right) = O(\log_2(n) \times Q_s(n)),$$

тъй като k отново е от порядъка на $\log_2 n$.

2) **Добавяне** на елемента $a \notin M$ към M ($a \in T_2$).

Добавянето на елемент ще съответства на това към числото n да добавим едно битче (една единичка):

$$\begin{array}{r} 10101111 \\ + 1 \\ \hline 10110000_{(2)} = m \end{array}$$

Т.е. това съответства на намирането на първото множество M_i , което е празно и да побединим елементите на всички множества, които са след тази намерена позиция i , да добавим новия елемент към новообразуваното множество и да нулираме множествата сформитаци новото множество.

Ще намерим първия индекс ($\xleftarrow{\text{посока}}$), за който $a_i = 0$ и $L_i = \emptyset$. След това ще подадем съответните елементи на нашата процедура P_s , която индексира статично множество.

В този списък трябва да попадне новодобавения елемент a , затова го инициализираме с него:

```

L ← {a}
i ← 0
while Li ≠ ∅ do
    премахваме Si
    L ← L ∪ Li
    Li ← ∅ (въпрос на имплементация)
    i = i + 1
done
Si ← Ps(L)
Li ← L

```

Времева сложност: Ако сме създали ново множество M_i , то времето което сме похарчили е $1 + i + P_s(2^i)$. За да направим **амортизиран анализ** е необходимо да анализираме колко често ще ни се наложи да създаваме ново множество M_i .

Създаването на ново множество с индекс i се случва при добавянето на $m^{-\text{ти}}$, $m \leq n$ елемент когато $m \equiv 0 \pmod{2^i}$ и $m \equiv 1 \pmod{2^{i+1}}$ ($2^i \mid m$ и $2^{i+1} \nmid m$).

Т.е. когато добавяме n елемента, ситуацията в които ще се добави конкретно i -то множество ще бъдат точно $\left\lfloor \frac{n}{2^i} \right\rfloor - \left\lfloor \frac{n}{2^{i+1}} \right\rfloor$ на брой.

$$\left\lfloor \frac{n}{2^i} \right\rfloor - \left\lfloor \frac{n}{2^{i+1}} \right\rfloor \leq \frac{n}{2^i}, \text{ откъдето общото време за } n \text{ добавяния е}$$

$$n \overline{\mathcal{F}}_{\mathcal{D}}(n) \leq \sum_{i=0}^k \frac{n}{2^i} (1 + i + P_s(2^i)) = \sum_{i=0}^k \frac{n(1+i)}{2^i} + \underbrace{\sum_{i=0}^k \frac{n P_s(2^i)}{2^i}}_B = A$$

За B : Припомняме, че $\frac{P_s(n)}{n}$ е намаляваща. Това означава, че т.к. $2^i \leq n$, то дробта $\frac{P_s(2^i)}{2^i} \leq \frac{P_s(n)}{n} \Rightarrow A \leq \sum_{i=0}^k \frac{n(1+i)}{2^i} + \sum_{i=0}^k \frac{n P_s(n)}{n} = \sum_{i=0}^k \frac{n(1+i)}{2^i} + (\log_2(n) + 1) P_s(n)$

$$\Rightarrow \overline{\mathcal{F}}_{\mathcal{D}}(n) \leq \sum_{i=0}^k \frac{i+1}{2^i} + \frac{P_s(n)}{n} (\log_2(n) + 1).$$

Остава да анализираме $\sum_{i=0}^k \frac{i+1}{2^i}$, което е стандартен анализ на сума:

$$\sum_{i=0}^k \frac{1}{2^i} = 1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^k} + \frac{1}{2^k} - \frac{1}{2^k} = 1 + 1 - \frac{1}{2^k} = 2 - \frac{1}{2^k}$$

За сумата $\sum_{i=0}^k \frac{i}{2^i}$ може да подходим по няколко начина.

I н/н (анализ): $\sum_{i=0}^k \frac{i}{2^i} \leq \sum_{i=0}^{\infty} \frac{i}{2^i} = \frac{x}{(1-x)^2} \bigg|_{x=\frac{1}{2}} = \frac{\frac{1}{2}}{\left(\frac{1}{2}\right)^2} = 2$, тук използвахме факта,

че ако $f(x) = \sum_{i=0}^{\infty} x^i \stackrel{|x| \leq 1}{=} \frac{1}{1-x}$, то

$$\frac{\partial}{\partial x} f(x) = \frac{\partial}{\partial x} \sum_{k=0}^{\infty} k x^{k-1} \stackrel{|x| \leq 1}{=} \frac{\partial}{\partial x} \left(\frac{1}{1-x} \right) = \frac{0 - \frac{\partial}{\partial x}(1-x)}{(1-x)^2} = \frac{1}{(1-x)^2}.$$

II н/н (комбинаторен):

$$\begin{aligned} \sum_{i=0}^k \frac{i}{2^i} &= \sum_{i=0}^k \sum_{j=1}^i \frac{1}{2^i} = \sum_{j=1}^k \sum_{i=j}^k \frac{1}{2^i} = \sum_{j=1}^k \frac{1}{2^j} \sum_{i=0}^{k-j} \frac{1}{2^i} = \sum_{j=1}^k \frac{1}{2^j} \left(2 - \frac{1}{2^{k-j}} \right) = \\ &= \sum_{j=1}^k \left(\frac{1}{2^{j-1}} - \frac{1}{2^k} \right) = \sum_{j=0}^{k-1} \frac{1}{2^j} - \frac{k}{2^k} = 2 - \frac{1}{2^{k-1}} - \frac{k}{2^k} \leq 2. \end{aligned}$$

$$\Rightarrow \overline{\mathcal{F}}_{\mathcal{D}}(n) \in O\left(\log_2(n) \frac{P_s(n)}{n}\right).$$

$$3) \text{ Сложност по памет: } O\left(\sum_{i=0}^k a_i S_s(2^i) + \underbrace{n}_{\text{списъци}} \log_2 n\right) =$$

(отново използваме, че $\frac{S_s(n)}{n}$ е не намаляваща)

$$= O\left(\sum_{i=0}^k a_i 2^i \frac{S_s(2^i)}{2^i} + n\right) \stackrel{2^i \leq n}{\leq} O\left(\sum_{i=0}^k a_i 2^i \frac{S_s(n)}{n} + n\right) = O\left(\frac{S_s(n)}{n} \underbrace{\sum_{i=0}^k a_i 2^i}_{\stackrel{\text{def.}}{=} n} + n\right) =$$

$$= O\left(S_s(n) + m\right) \stackrel{\frac{S_s(n)}{n} \text{ е не нам.}}{=} O\left(S_s(n)\right), \text{ т.е. паметта се запазва.}$$

Твърдение: Ако Q е разложим проблем за търсене с характеристики същите както в предишното твърдение:

$\mathcal{Q}_s(n)$ - не намаляваща

Колкото повече елемента има в нашето множество - заявката не може да стане по-лесна.

$\mathcal{P}_s(n) : \frac{\mathcal{P}_s(n)}{n}$ - не намаляваща

Поне трябва да погледнем елементите в това множество, т.е. средно на елемент трябва да отделяме повече време, когато имаме повече елементи.

$\mathcal{S}(n) : \frac{\mathcal{S}_s(n)}{n}$ - не намаляваща

За повече елементи най-вероятно ще ни е необходима повече памет за запазването на информация, която свързва текущ елемент с останалите.

то тогава има полудинамична структура \mathcal{D} за проблема Q със следните характеристики:

1) $\mathcal{Q}_{\mathcal{D}} \in O\left(\mathcal{Q}_s(n) \times \log(n)\right)$ - т.е. заявката в новата структура от данни се влошава с фактор от логаритъм.

2) Паметта не се влошава: $\mathcal{S}_{\mathcal{D}}(n) \in O\left(\mathcal{S}_{\mathcal{D}}(n)\right)$
space

3) $\underbrace{\mathcal{F}_{\mathcal{D}}(n)}_{\text{(НЕ амортизирано)}} \in O\left(\frac{\mathcal{P}_s(n)}{n} \times \log(n)\right)$ - амортизираното време за n операции по добавяне.

Динамизацията идва на цената на допълнителен фактор от $\log(n)$ за n операции.
Разликата тук е, че тази оценка е в лошия случай (а не амортизирана).

Доказателство: Отново идеята е да поддържахме множества, които да бъдат с размер равен на степените на двойката. Представяме множеството M като обединение на няколко множества от вида $M_i[0]$, $M_i[1]$ и $M_i[2]$ за $i = \overline{0, k}$.

Индекса ще показва колко елемента има това множество - то или ще има 0 елемента, или ще има 2^i елемента, т.е.
$$\begin{cases} 1. |M_i[j]| = \{0, 2^i\} \\ 2. M_i[j] \cap M_i[j'] = \emptyset, \text{ за } (i, j) \neq (i', j') \end{cases}$$

Неформално описание на идеята/интуицията зад множествата $M_i[0]$, $M_i[1]$ и $M_i[2]$: Първите две множества $M_i[0]$ и $M_i[1]$ в първия момент, в който и двете са **непразни**, т.е. имат по 2^i елемента, те общо ще имат $2 \times 2^i = 2^{i+1}$ елемента и ще започнат да строят едно ново множество с 2^{i+1} елемента. Те ще построят това множество в следващите 2^{i+1} стъпки (2^{i+1} добавяния на елементи). По време на това строене обаче, може да се случи някой потребител да добави нови 2^i елемента. Тук се намешва ролята на $M_i[2]$, която е именно да погълне/задържи този прилив на елементи към $M_i[0]$ и $M_i[1]$ така, че те да имат достатъчно време да изградят своето множество с 2^{i+1} елемента и едва когато те са готови - едва тогава да има потенциална възможност за добавянето на още множества с 2^i елемента.

Ролята на третото (последно) множество $M_i[2]$ е да задържа наплива от елементи към i -тия етаж на тази йерархия от 2^i елемента. За да се контролира това, може да си ги представяме тези множества като някакви диги, които се строят на i -тия етаж и в момента, в който има достатъчно вода в $M_i[0]$ и $M_i[1]$ те започват да се подготвят да прелеят към $i + 1$ -вия етаж, но някои може да ги залее тях през това време и затова $M_i[2]$ служи като предпазна дига. В случая се оказва, че само една дига стига.

Идея (малко по-формално): В момента, в който $M_i[0]$ и $M_i[1]$ са непразни, започваме да строим нова *статична* структура за $M_i[0] \cup M_i[1]$. Разпределяме тази работа през следващите 2^{i+1} добавяния на елементи.

Представяне:

за $i = 0, 1, \dots, k$ ($M_k[0] \cup M_k[1] \cup M_k[2] \neq \emptyset$)

- $B_i[j]$, за $j = \overline{0, 2}$ статични структури за $M_i[j]$;
block
- $L_i[j]$, за $j = \overline{0, 2}$ - списъци от елементите на $M_i[j]$;
list
- U_i - статична структура, която е в процес на изграждане.
under construction

(Правенето на заявки към U_i е безсмислено, тъй като няма никаква гаранция, че там заявката ще се обработи по някакъв начин)

$2^k \leq n$ и следователно $k \leq \log_2 n$

1) **Търсене:**

Вход: $x \in T_1$

$r \leftarrow Q_s(x, \emptyset)$

for $i = 0$ **to** k **do**

for $j = 0$ **to** 2 **do**

$r \leftarrow r \sqcup Q_s(x, B_i[j])$

done

done

return r

Времева сложност: $|B_i[j]| \leq 2^j \leq n$; $3(k+1)Q_s(n) \leq 3(\log_2 n + 1)Q_s(n)$

2) **Сложност по памет:**

$$O\left(\sum_{B_i[j] \neq \emptyset} S_s(B_i[j])\right) \leq O\left(n + \sum_{B_i[j] \neq \emptyset} \frac{S_s(B_i[j])}{2^i} \times 2^i\right) \leq O\left(\frac{S_s(n)}{n} \sum_{B_i[j] \neq \emptyset} 2^i + n\right) =$$

$$O\left(\frac{S_s(n)}{n} \underbrace{\sum |B_i[j]|}_{\stackrel{\text{def.}}{=} n} + n\right) = O(S_s(n)).$$

(при предположение, че паметта необходима за процеса на построяване на U_i не се освобождава и се включва в паметта, която е вече за готовата структура, която ще получим, когато U_i бъде завършено)

3) **Добавяне** на $x \notin M$ в M :

for $i = k$ **down to** 0 **do**

if U_i *is under construction* **then**

do $\frac{P_s(2^{i+1})}{2^{i+1}}$ *work on* U_i (out of the elements $L_i[0] \cup L_i[1]$)

if $P_s(2^{i+1})$ *work has been done on* U_i **then**

$j_0 \leftarrow \min_{\{0,1,2\}} \{j \mid L_{i+1}[j] \neq \emptyset\}$

$L_{i+1}[j_0] \leftarrow L_i[0] \underset{\substack{\text{concat} \\ 2 \text{ lists} \\ = \text{const.}}}{\cup} L_i[1]$

$B_{i+1}[j_0] \leftarrow U_i$ (*pointer redirection = const.*)

if $j_0 = 1$ **then**

няма реално действие
(инициализираме)

$\left\{ \begin{array}{l} \text{start working on the consruction of } U_{i+1} \\ \text{out of } L_{i+1}[0] \text{ and } L_{i+1}[1] \text{ (това означава, че от следващата} \\ \text{стъпка нататък (от следващото добавяне на елемент) ще} \\ \text{работим по изграждането на това множество. Т.е. в този момент} \\ \text{не се извършва реална работа.)} \end{array} \right.$

discard $B_i[0], B_i[1], L_i[0] \leftarrow \emptyset, L_i[1] \leftarrow \emptyset$

$L_i[0] \leftarrow L_i[2]$

$B_i[0] \leftarrow B_i[2]$

$L_i[2] \leftarrow \emptyset$

discard $B_i[2]$

(на ниво оказатели: след като сме нулирали указателите на $B_i[0]$ и $B_i[1]$ и съответно на $L_i[0]$ и $L_i[1]$, това което правим после е смяна на указателите на $L_i[0]$ с $L_i[2]$ и $B_i[0]$ с $B_i[2]$)

done

$j_0 \leftarrow \min\{j \mid L_0[j] = \emptyset\}$

$L_0[j_0] \leftarrow \{x\}$

$B_0[j_0] \leftarrow P_s(L_0[j_0])$

if $j_0 = 1$ **then**

start working on the construction of U_0 out of $L_0[0] \cup L_0[1]$

Оценка на времевата сложност: за добавяне на един елемент ще имаме време

$$\leq O\left(\sum_{i=0}^k \frac{P_s(2^i + 1)}{2^{i+1}} + P_s(1) + (k + 1)\right) = Y$$

Когато изграждаме U_i :

$$L_i[0] \cup L_i[1] \subseteq M \Rightarrow |M| \geq |L_0[0]| + |L_i[1]| = 2 \times 2^i = 2^{i+1} \Rightarrow n \geq 2^{i+1}$$

$$Y \leq O\left(\sum_{i=0}^k \frac{P_s(n)}{n} + k\right) = O\left(\frac{P_s(n)}{n} \times \log n\right).$$

Коректност: Защо j_0 винаги са добре дефинирани? Да фиксираме i . Целта е да видим, че когато прехвърляме елементи към i винаги ще има място. Нека t е момент от време, в който започва изграждането на U_i :

1) В момента t , $L_i[2] = \emptyset$

2) В момента t , $B_i[1] = U_{i-1}$ и поради тази причина следващото завършено U_{i-1} ще бъде след още 2^i добавяния, а по-следващото след още 2^{i+1} добавяния. Т.е.

в моментите от време $\begin{cases} t + 2^i \\ t + 2^i + 2^i = 2 + 2^{i+1} \end{cases}$

- 3) В момента $t + 2^i$ нашата структура все още има свободен блок $B_i[2]$, $L_i[2]$ е празно и съответно $j_0 = 2$
- 4) В момента $t + 2^{i+1}$, (конструкцията на U_i ще завърши, защото времето за нейното изграждане е разделено на 2^{i+1} фрагмента, всеки от които заема максималното време от $\frac{P_s(2^{i+1})}{2^{i+1}}$) се освобождават $B_i[0], B_i[1]$ (тъй като $j_0 \in \{0,1\}$)