Динамизация на структури от данни част 1
15.01.2021

Проблема, който ще разгледаме тук е абстрактен.
<u>Дефиниция</u>: Имаме три множества T_1, T_2 и T_3 и имаме проблем за търсене: т.е. функция $\underset{query}{Q} : T_1 \times 2^{T_2} \to T_3$. В този проблем няма никакви конкретни типове или обекти, за които да си мислим или някаква конкретна заявка.

<u>Примери</u>:
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) $T_1 = T_2 = \mathbb{N};\ T_3 \in \mathbb{N}$
   $Q(x, A) = \text{succ}_A(x) = \min\{y \in A \,|\, y \geq x\}$ или
   $Q(x, A) = \text{pred}_A(x) = \max\{y \in A \,|\, y \leq x\}$

<u>Постановка</u>:
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 $\mathscr{P}_s(n)$ $\quad \mathscr{P} \equiv Procedure$
2) с памет $\mathscr{S}_s(n)$ $\quad \mathscr{S} \equiv Space$
3) отговор на заявка $Q(x, A)$ за време $\mathscr{Q}_s(n)$

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

<u>Дефиниция</u>: (Полу)динамична структура от данни за Q е структура от данни $\underset{dynamic}{\mathscr{D}}$, за която може да:
1) отговаряме на заявки от вида $Q(x, A)$ за време $\mathscr{Q}_{\mathscr{D}}\big(|A|\big)$ $\quad Q \equiv Query$
2) изисква памет $\mathscr{S}_{\mathscr{D}}\big(|A|\big)$ $\quad S \equiv Space$
3) да добавяме елемент $a \notin A$ към A $\big(I(a, \mathscr{D})\big)$ за време $\mathscr{I}_{\mathscr{D}}\big(|A|\big)$ $\quad I \equiv Insert$
4) да изтриваме елемент $a \in A$ $\big(D(a, \mathscr{D})\big)$ за време $\mathscr{E}_{\mathscr{D}}\big(|A|\big)$ $\quad D \equiv Delete$

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

<u>Дефиниция</u>: Наричаме проблема за търсене $Q : T_1 \times 2^{T_2} \times T_3$ *разложим*, ако има функция $\sqcup : T_3 \times T_3 \to T_3$ 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
   $Q(x,\ A \cup B) = \sqcup\big(Q(x, A), Q(x, B)\big)$
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 $T_{3,x} = \{Q(x, A) \,|\, A \subseteq T_2\}$ 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:

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

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

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

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

Примери:

1) $T_1 = T_2;\; T_3 = \{\text{true}, \text{false}\}$
   $Q(a, A) \Leftrightarrow x \in A$
   $A \cap B = \emptyset$
   $x \in A \cup B \Leftrightarrow x \in A$ или $x \in B$
   $Q(x, A \cup B) = Q(x, A) \underset{\equiv \sqcup}{\vee} Q(x, B)$

2) $T_1 = T_2 = \mathbb{N};\; T_3 = \mathbb{N}.$
   $Q(x, A) = \text{succ}_A(x)$
   Нека $A \cup B = \emptyset$
   $\text{succ}_{A \cup B}(x) = \underbrace{\min}_{\equiv \sqcup} \left(\text{succ}_A(x), \text{succ}_B(x) \right)$

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

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

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

$\mathcal{Q}_s(n)$ - ненамаляваща

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

$\mathcal{P}_s(n) : \dfrac{\mathcal{P}_s(n)}{n}$ - ненамаляваща

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

$\mathcal{S}(n) : \dfrac{\mathcal{S}_s(n)}{n}$ - ненамаляваща

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 $\mathscr{D}$ за проблема Q със следните характеристики:

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

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

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

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

<u>Доказателство</u>:

Във всеки момент от време, ако M е множеството което педставяме 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 $n + 1$ множества със следното свойство:

$M = \bigcup_{i=0}^{k} M_i$, 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 двойчното представяне на n е 0, в противен случай ще бъде 2^i)

Това което поддържаме е следното:

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

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

ОПЕРАЦИИ:

1) <u>**Търсене**</u>:

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

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

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

2) <u>**Добавяне**</u> 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):

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

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 сформитащи новото множество.

Ще намерим първия индекс $\left(\xleftarrow{\text{посока}}\right)$, 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:

$L \leftarrow \{a\}$
$i \leftarrow 0$
while $L_i \neq \emptyset$ **do**
    премахваме S_i
    $L \leftarrow L \cup L_i$
    $L_i \leftarrow \emptyset$ (въпрос на имплементация)
    $i = i + 1$
done
$S_i \leftarrow P_s(L)$
$L_i \leftarrow L$

Времева сложност: Ако сме създали ново множество M_i, то времето което сме похарчили е $1 + i + P_s(2^i)$. За да направим <u>**амортизиран анализ**</u> 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-ти, $m \leq n$ елемент когато $m \equiv 0 \mod 2^i$ и $m \equiv 1 \mod 2^{i+1}$ $\left(2^i | m \text{ и } 2^{i+1} \nmid m\right)$.

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

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

$$n\overline{\mathcal{I}}_{\mathcal{D}}(n) \leq \sum_{i=0}^{k} \frac{n}{2^i}\left(1 + i + P_s(2^i)\right) = \sum_{i=0}^{k} \frac{n(1+i)}{2^i} + \underbrace{\sum_{i=0}^{k} \frac{nP_s(2^i)}{2^i}}_{B} = A$$

За B: Припомняме, че $\frac{P_s(n)}{n}$ е намаляваща. Това означава, че т.к. $2^i \leq n$, то дробта

$$\frac{P_s(2^i)}{2^i} \leq \frac{P_s(n)}{n} \Rightarrow A \leq \sum_{i=0}^{k} \frac{n(1+i)}{2^i} + \sum_{i=0}^{k} \frac{\cancel{n}P_s(n)}{\cancel{n}} = \sum_{i=0}^{k} \frac{n(1+i)}{2^i} + \left(\log_2(n) + 1\right)P_s(n)$$

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

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

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

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

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

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

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

II н/н (комбинаторен):

$$\sum_{i=0}^{k} \frac{i}{2_i} = \sum_{i=0}^{k}\sum_{j=1}^{i} \frac{1}{2_i} = \sum_{j=1}^{k}\sum_{i=j}^{k} \frac{1}{2_i} = \sum_{j=1}^{k} \frac{1}{2^j}\sum_{i=0}^{k-j} \frac{1}{2^i} = \sum_{j=1}^{k} \frac{1}{2^j}\left(2 - \frac{1}{2^{k-j}}\right) =$$

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

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

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

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

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

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

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

$\mathcal{Q}_s(n)$ - ненамаляваща

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

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

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

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

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

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

3) $\underbrace{\mathcal{I}_{\mathcal{D}}(n)}_{\text{(НЕ амортизирано)}} \in O\left(\frac{\mathcal{P}_s(n)}{n} \times \log(n)\right)$ - 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).**

<u>Доказателство</u>: Отново идеята е да поддържаме 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. $\begin{cases} 1.\ |M_i[j]| = \{0, 2^i\} \\ 2.\ M_i[j] \cap M_{i'}[j'] = \emptyset, \text{ за } (i,j) \neq (i',j') \end{cases}$

<u>Неформално описание на идеята/интуицията</u> 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 **не**празни, 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 $\left(2^{i+1}\right.$ добавяния на елементи$\left.\right)$. По време на това строене обаче, може да се случи някой потребител да добави нови 2^i елемента. Тук се намесва 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.

<u>Идея</u> (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.

<u>Представяне</u>:
за $i = 0,1,\ldots,k$ $(M_k[0] \cup M_k[1] \cup M_k[2] \neq \emptyset)$

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

(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: $x \in T_1$

$$
\begin{array}{l}
r \leftarrow Q_s(x, \varnothing) \\
\textbf{for } i = 0 \textbf{ to } k \textbf{ do} \\
\quad \textbf{for } j = 0 \textbf{ to } 2 \textbf{ do} \\
\quad\quad r \leftarrow r \sqcup Q_s(x, B_i[j]) \\
\quad \textbf{done} \\
\textbf{done} \\
\textbf{return } r
\end{array}
$$

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**:

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

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

(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:

$$
\begin{array}{l}
\textbf{for } i = k \textbf{ down to } 0 \textbf{ do} \\
\quad \textbf{if } U_i \textit{ is under construction } \textbf{then} \\
\quad\quad \textbf{do } \dfrac{P_s(2^{i+1})}{2^{i+1}} \textit{ work on } U_i \ (\textit{out of the elements } L_i[0] \cup L_i[1]) \\
\quad \textbf{if } P_s(2^{i+1}) \textit{ work has been done on } U_i \textbf{ then} \\
\quad\quad j_0 \leftarrow \min\{ \underset{\{0,1,2\}}{j} \mid L_{i+1}[j] \neq \varnothing\} \\
\quad\quad L_{i+1}[j_0] \leftarrow L_i[0] \underset{\substack{concat \\ 2\ lists \\ = const.}}{\cup} L_i[1]
\end{array}
$$

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

няма реално действие (инициализираме)

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

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
$j_0 \leftarrow \min\{j \mid L_0[j] = \emptyset\}$
$L_0[j_0] \leftarrow \{x\}$
$B_0[j_0] \leftarrow P_s\left(L_0[j_0]\right)$
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:
$L_i[0] \cup L_i[1] \subseteq M \Rightarrow |M| \geq |L_0[0]| + |L_i[1]| = 2 \times 2^i = 2^{i+1} \Rightarrow n \geq 2^{i+1}$

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

<u>Коректност</u>: 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 $t + 2^{i+1}$, $\Bigg($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 $\dfrac{P_s(2^{i+1})}{2^{i+1}}\Bigg)$ се освобождават $B_i[0], B_i[1]$ (тъй като $j_0 \in \{0,1\}$)