# АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
# ПРОЦЕСОРНО ВРЕМЕ

Можем да приемем, че програма в сектор с номер 1 започва да се предпроцесва в момент t=0. Нека с $tabeg_i$ означим началото на предпроцесване на програма в сектор с номер *i*. За начало имаме $tabeg_1=0$. Нека сме изчислили $tabeg_i$. Тогава $tabeg_{i+1}$ ще се изчисли, изхождайки от следните съображения:
$tabeg_{i+1} \geq tabeg_i+a_i$ – процесор *A* трябва да е завършил предпроцесването на програма в сектор с номер *i,* за да може да започне предпроцесването на програма в сектор *i+1*;
$tabeg_{i+1}+a_{i+1} \geq tabeg_i+a_i+b_i$ - процесор *A* трябва да завърши предпроцесването на програма в сектор с номер *i+1* не по-рано от момента, в който процесор *B* ще се освободи от същинското изпълнение на програма в сектор с номер *i*. Това е необходимо, тъй като веднага след предпроцесването на програма в сектор с номер *i+1* процесор *B* трябва да започне същинското и изпълнение. Тези две неравенства ни позволяват да напишем следната формула:
$tabeg_{i+1}= tabeg_i+max(a_i, a_i+b_i- a_{i+1})$.

Тогава можем да напишем следната верига от равенства:
$tabeg_n= tabeg_{n-1}+max(a_{n-1}, a_{n-1}+b_{n-1}- a_n)= tabeg_{n-2}+ max(a_{n-2}, a_{n-2}+b_{n-2}- a_{n-1})+max(a_{n-1}, a_{n-1}+b_{n-1}- a_n)=\ldots.= tabeg_1+ max(a_1, a_1+b_1- a_2)+\ldots\ldots+ max(a_{n-2}, a_{n-2}+b_{n-2}- a_{n-1})+max(a_{n-1}, a_{n-1}+b_{n-1}- a_n)= max(a_1, a_1+b_1- a_2)+\ldots\ldots+ max(a_{n-2}, a_{n-2}+b_{n-2}- a_{n-1})+max(a_{n-1}, a_{n-1}+b_{n-1}- a_n)$.
Това е моментът, в който започва предпроцесването на програмата в последния сектор с номер *n*. Общото време на изпълнение ще бъде равно на

$max(a_1, a_1+b_1- a_2)+\ldots\ldots+ max(a_{n-2}, a_{n-2}+b_{n-2}- a_{n-1})+max(a_{n-1}, a_{n-1}+b_{n-1}- a_n)+a_n+b_n$.

При първото изпълнение на програмите по тази формула може да се сметне и запомни общото време за изпълнение. При следващите размени и изпълнения трябва да се забележи, че размяната на програми в секторите с номера *i* и *j* влияе максимум на четири от събираемите и, за да пресметнем новото време на изпълнение след размяната, не трябва да смятаме отново всичко, а трябва само да извадим от изчислената сума старите стойности на членовете на сумата, които се влияят от размяната, да направим размяната и да добавим новите стойности на същите членове.
Сложността на алгоритъма в тази му част е $O(N+K)$.
Времената на бездействие на процесорите при първото изпълнение на програмите могат да се пресметнет по следния начин.
Времето, което бездейства процесор *A* след като завърши предпроцесването на програма в сектор *i* и преди да започне предпроцесването на програма в сектор *i+1* е равно на $tabeg_{i+1}- tabeg_i-a_i= max(a_i, a_i+b_i- a_{i+1}) -a_i=max(0, b_i- a_{i+1})$. Тук трябва да се има предвид, че след предпроцесването на програмата в последния сектор с номер *n* процесор *A* бездейства време $b_n$, необходимо за същинското изпълнение на програмата в този сектор от процесор *B*.
Времето, което бездейства процесор *B* след като завърши същинското изпълнение на програма в сектор *i* и преди да започне същинското изпълнение на програма в сектор

$i+1$ е равно на $(tabeg_{i+1}+a_{i+1})-(tabeg_i+a_i+b_i)=(tabeg_{i+1}-tabeg_i)+a_{i+1}-a_i-b_i=max(a_i, a_i+b_i-a_{i+1})+a_{i+1}-a_i-b_i=max(0, b_i-a_{i+1})+a_{i+1}-b_i$. Тук трябва да се има предвид, че преди същинското изпълнение на програмата в сектор с номер 1 процесор $B$ бездейства време $a_1$, необходимо за предпроцесването на програмата в този сектор от процесор $A$.

Ако, при първото изпълнение на програмите, изградим структури, в които държим времената на непрекъснато бездействие на двата процесора, такива че намирането на максимален елемент в тях да е с константна сложност, а обновяването да е със сложност $logN$, то, тъй като при размяна на програмите в два сектора се променят най-много 8 времена на бездействие на процесорите, ще получим алгоритъм със сложност $O(N+KlogN)$. Такава структура може да е пирамида или индексно дърво. В реализацията в **proctime.cpp** е използвано индексно дърво.

*Автор: Руско Шиков*