ЗАБЕЛЕЖКИ ПО ДОМАШНО № 2

В решенията на задачите от второто домашно по “Дизайн и анализ на алгоритми” са допуснати някои масови грешки.

Задача 1. Определението на функцията f задава алгоритъм за нейното пресмятане, но той не е достатъчно ефективен — нито по време, нито по памет.

Сложността по време се задава с рекурентното уравнение

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(1).$$

Множителят 2 в дясната страна се дължи на двете рекурсивни извиквания при нечетно n. Решението на уравнението се намира с помощта на мастър-теоремата: $T(n) = \Theta(n)$. Това надвишава максималното допустимо време $\Theta(\log n)$.

В много решения се твърди погрешно, че алгоритъмът има сложност по памет $\Theta(1)$. Това е сложността само на едно равнище от рекурсията. Обаче броят на равнищата, тоест дълбочината на рекурсията, е $\Theta(\log n)$. Следователно $M(n) = \Theta(\log n)$, поради което количеството памет също не отговаря на ограничението $M(n) = \Theta(1)$.

Накратко, наивният алгоритъм, получен от определението на f, не е приемлив.

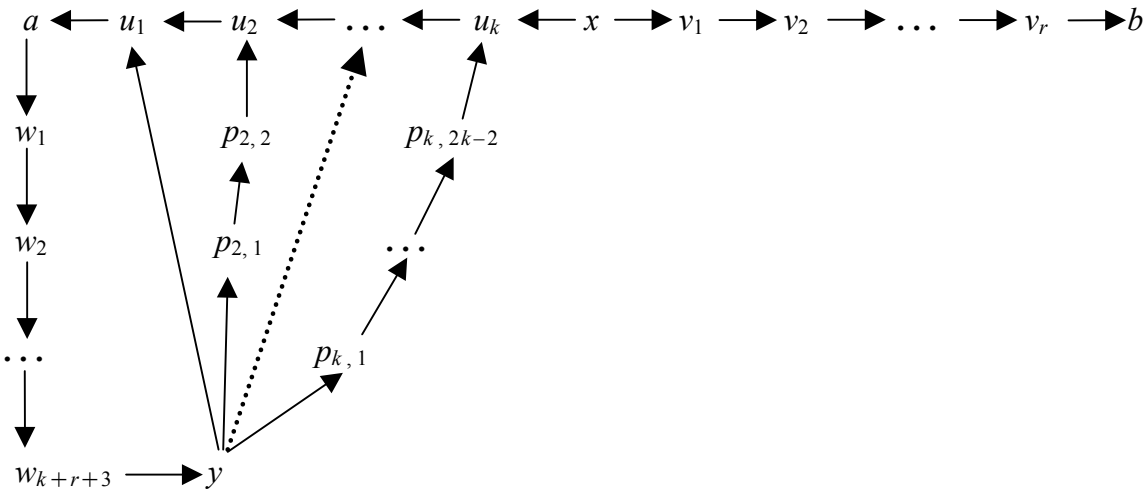
Задача 2. В много решения се правят опити за съчетаване на търсенето в ширина с техниката, наречена релаксация. Такива решения са неправилни. Търсенето в ширина не съдържа релаксация. Релаксация има в алгоритъма на Дейкстра (и в други алгоритми).

Къде е проблемът? Нека за всеки връх на графа пазим по едно число — най-малкия брой нарушения, с които можем да достигнем до този връх от посочения начален връх. Щом намерим път с по-малък брой нарушения, намаляваме числото на текущия връх. Релаксацията представлява именно това намаляване. Обаче числата на други върхове може да са били пресметнати междувременно с помощта на грешната стойност. Затова, когато поправим стойността на даден връх, трябва да поправим и стойностите на върховете, които са били пресметнати междувременно с негова помощ. Това поправяне е бавно.

Алгоритъмът на Дейкстра избягва многократните поправки, като пресмята числата на върховете в определен ред: на всяка стъпка избира върха с най-малкото число. Но това изисква използването на приоритетна опашка и не може да работи в линейно време (а целта на решаващия е да .получи линейно време, иначе не би използвал търсене в ширина).

В някои решения се твърди, че е невъзможно да се получат твърде много релаксации, тоест въпросното забавяне на алгоритъма е привиден проблем. За съжаление, не е така.

К о н т р а п р и м е р :



Върхът y е свързан с u_2 чрез път с два междинни върха; с u_3 — чрез път с четири междинни върха; и т.н. (с всеки следващ връх u — чрез път с два междинни върха повече).

Търсим път от a до b с най-малък брой нарушения на посоките на ребрата. Обхождаме графа в ширина, започвайки от a и пренебрегвайки посоката на ребрата, тоест от всеки връх излизаме и по влизащите, и по излизащите ребра. Отначало на върха a приписваме стойност 0 (няма нарушения), а на всеки друг връх приписваме стойност $+\infty$. По време на обхождането на всеки връх, различен от a, приписваме стойността на предходния връх, ако последното ребро, по което сме минали, е било в правилната посока; в противен случай числото на новия връх е с единица повече от числото на стария връх (защото сме извършили едно допълнително нарушение). Ако някой връх бъде посетен многократно, извършва се релаксация: ако новата стойност на върха е по-малка от старата, то присвояваме на върха новата стойност на броя на нарушенията.

След $k + r + 3$ стъпки върхът u_1 ще има стойност 1, върхът u_2 ще има стойност 2, . . . , върхът u_k ще има стойност k, върхът x ще има стойност $k + 1$, същата стойност $k + 1$ ще имат върховете v и върхът b; върхът y и върховете w ще имат нулеви стойности, а върховете p ще имат положителни стойности, включително $+\infty$.

На следващата стъпка ще настъпи релаксация при върховете u_1 , $p_{2,\,1}$, . . . , $p_{\mathrm{k},\,1}$: те ще получат нулеви стойности. Всеки от тях, в това число u_1 , ще добави наследниците си в опашката на обхождането повторно, за да се актуализират и техните стойности. Тъй като пътят $y u_1 u_2$ има едно ребро по-малко от пътя $y p_{2,\,1}\, p_{2,\,2}\, u_2$, то върхът u_2 ще претърпи две релаксации: стойността му ще спадне най-напред от 2 на 1, а после — от 1 на 0.

Аналогично, върхът u_3 ще претърпи три релаксации, u_4 — четири релаксации и т.н.

Върхът u_k ще претърпи k релаксации. Затова също толкова релаксации ще настъпят при върха x, върховете v и при върха b: техните числа ще спаднат от k до 0, преминавайки през всички междинни стойности. Тъй като тези върхове са поне r на брой, то алгоритъмът ще извърши поне kr операции, затова времевата сложност е $\Omega(kr)$.

Не е трудно да се види, че при подходящи k и r тази сложност е по-голяма от линейна. Действително, броят n на върховете и броят m на ребрата на графа се изразяват чрез k и r с помощта на следните формули:

$$n = k^2 + k + 2r + 7,$$
$$m = k^2 + 2k + 2r + 6.$$

Да вземем например $k = \Theta\left(\sqrt[3]{n}\right)$. Тогава от първото уравнение намираме $r = \Theta(n)$, а от второто уравнение следва, че $m = \Theta(r) = \Theta(n)$. Дължината на входа на алгоритъма е равна на $\Theta(m+n) = \Theta(n)$, а времето за изпълнението му е $\Omega\left(n.\sqrt[3]{n}\right)$. Тоест времето е от степен поне $4/3$, следователно сложността на алгоритъма е по-голяма от линейна.