Упражнение №0 по САА

Защо и как?

За да си отговорим на въпроса защо учим САА, може би първо трябва да си отговорим на въпроси като „Що е то алгоритъм и има ли той почва у нас?“, „Защо изобщо пишем програми?“ или „Що за животно е леймъра?“.

Да почнем отзад напред...

Какво е леймър? [*Наков*-2003]

На първо място, най-важната характеристика за един леймър е, че той не подозира, че е такъв.
По-нататък, едно от най-разпространените определения за леймър, е програмист, който пише програми в колонка и който, като чуе за рекурсия, получава леко разстройство.
Леймър е програмист, който знае няколко езика за програмиране, но няма никакво понятие от алгоритми.
Леймър е програмист, който ако изобщо слага коментари в програмите с и, те са главно от следния вид:

```
i++; /* увеличаваме i с единица */
```

Леймър е програмист, който не знае, че зад простичката игра *Minesweeper* за *Windows* се крие една от най-сериозните задачи в информатиката, за разрешаването на която университетът Станфорд е определил награда от 1 милион долара [*Prize-2000*].
Леймър е програмист, който може да напише *бързо сортиране* единствено, ако го научи наизуст. Нещо повече, истинските програмисти рядко могат да го запомнят и напишат бързо, но за сметка на това за 10 минути могат сами да си го “измислят”. Нашата цел е читателят да достигне това ниво, при което решаването на сложни задачи се свежда до просто упражнение на интуицията му.
Леймър е програмист, който не знае, че “събирачът на боклук” (от англ. *garbage collector*) не е измислен, специално за да събира “труповете” на “починалите” обекти на *Java*, а зад него стои мощна и красива теория, представена още в средата на *XX* век.
Леймър е програмист, който не знае, че най-ефективната търсеща машина в Интернет *Google* дава толкова релевантни резултати, защото се основава на новаторски методи от теорията на графите.
Леймър е програмист, който не знае, че формулата за ефективната реализация на про изво- лен компютърен проект е малко спецификации + много и ефективни алгоритми.
Леймър е програмист, който не може да проумее защо една програма, за която "някой бил казал, че има сложност $\Theta(n^2)$", е по-бърза от друга със сложност $\Theta(n^3)$, след като $n^3 > n^2$.
Леймър е програмист, който може да програмира на Си и който може да намери най-малко общо кратно на две числа на лист, но не може да си напише програма за това.
Леймър е програмист, който не може да си обясни защо като се компресира един файл, и после се компресира още веднъж, той не намалява още повече.

Леймър е програмист, който знае, че това  е дърво, но не подозира, че това

 също е дърво. И, че всъщност той самият, като програмист, също е едно младо и зелено дърво.

Леймър е програмист, който за 10 минути може да направи дадена програма 10 пъти по - бавна, без да се притесни, че някой преди него е загубил 10 часа в оптимизиране, за да я направи с 10% по-бърза.

Леймър е програмист, който *brute-force*-ва *ftp* (от англ. *File Transfer Protocol* — протокол за пренос на файлове в компютърна мрежа) пароли, без да съобрази, че при сегашната скорост на Интернет трудно би могъл да *hack*-не парола, по-дълга от 5 символа (а за това е необходимо едно съвсем просто комбинаторно изчисление), освен ако не извади луд късмет.

Леймър е програмист, който трудно може да проумее асоциация на хубава вечеря с хубава програма (а тяхното качество зависи предимно от рецептата/алгоритъма, който се използва).

Леймър е програмист, който не знае, че най-краткото разстояние между две точки не винаги е правата линия (понякога, то се намира по алгоритъма на Дейкстра).

Леймър е програмист, за когото *двоично търсене* означава *търсене* в *двоични* файлове.

Леймър е програмист, който се възхищава на програмни фрагменти като този:

```
0x000000FF & (i >> 24)
```

често, просто защото си няма понятие какво точно означават и панически преминава на следващата страница, когато види “йероглиф” от вида:

$$\chi^2 = \frac{n}{m}\sum_{i=1}^{n}\left(f_i - \frac{m}{n}\right)^2$$

Леймър е програмист, който не е чувал за динамично оптимиране. Всъщност, един от на- истина добрите начини да разбиете в спор някой леймър, е да му заявите: “Хей, ами тази задача е добре известна и се решава лесно, елегантно и ефективно с динамично оптимиране”.

В началото, като всички начинаещи програмисти, и ние не бяхме нищо повече от едни млади и зелени леймъри. С течение на времето с хубави книги и статии, и най-вече с упорит труд по решаване на алгоритмични проблеми, започнахме да надрастваме това ниво. Днес сме готови да помогнем и на теб, драги читателю, да избягаш от стерилитета на занаятчийското програмира- не.

Ще завършим настоящия параграф с една от прекрасните мисли на друг знаменит автор (*Марк Твен*):

“Когато искам да прочета нещо хубаво, сядам и си го написвам”.

Целта ни ще е малко по малко да се отдръпнете от образа на леймъри и да стигнете до положение, при което да имате самочувствието да заявите:

“Когато искам да видя хубава програма, сядам и си я написвам”.

Теми

1. Въведение в алгоритмите. Сложност. Сортиране и търсене.
2. Прости и съвършени числа. Редица на Фибоначи. Най-голям общ делител, най-малко общо кратно. Рекурсия. Комбинаторни алгоритми.
3. Разделяй и владей
4. Структури от данни.
5. Динамично оптимиране.
6. Графи

Оценяване и точки * чернова

Всяко упражнение ще разглежда нов материал придружен от примери и задачи. На всяко упражнение ще се провежда „състезание“ със решаване на определен брой задачи за време не по-малко от половината упражнение. За всяка успешно решена задача студентите ще получават точки, които в края на семестъра ще бъдат използвани при оценяването на студента.

	Точки	Брой	Общо
Присъствие на упражнение	2	6	12
Решена състезателна задача	3	12	36

Задачи за упражнение:

Задача 1. Дадена е матрица unsigned a[MAX][MAX]. Да се напише функция

```
void fillMatrix(unsigned a[][MAX], unsigned n),
```

която запълва с числа елементите на a[][] по следния начин:

0	20	19	17	14
1	0	18	16	13
2	5	0	15	12
3	6	8	0	11
4	7	9	10	0

Задача 2: Дадена е редица от цели числа a_1, a_2, ..., a_n. Търсим броя на нулите, на които завършва произведението $P = a_1.a_2. \ldots .a_n$.

Тук извършването на умножението е нежелателно и не винаги ще доведе до получаване на търсения резултат. За да решим задачата, ще обърнем внимание на следния факт: Единствените числа, чието произведение завършва на нула, са 2 и 5, или произведение на число, кратно на 2, с число, кратно на 5.

Литература

1. Програмиране = ++Алгоритмни, Наков, Добринков – Можете да свалите безплатно книгата на адрес: http://www.programirane.org