ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

KIV/PRO

Maximalizujte svůj zisk

Autor: Antonín NEUMANN

Akademický rok: 2012/2013

Zadání

Se soutěžemi se v televizi roztrhl pytel. Je na čase, abyste si vydělali také vy. Otázky jsou pro vás naprosto triviální a přichází čas vybrat si odměnu. Přivedou před vás n asistentek. Asistentky se drží za ruce a každá má na krku pověšenou ceduli s celočíselnou sumou, kterou získáte (kladné číslo) nebo naopak ztratíte (záporné číslo), když si tuto asistentku vyberete. Můžete si vybrat jakýkoliv počet asistentek, ale jejich spojené ruce smíte přerušit max. na dvou místech, tedy posloupnost odměn, kterou si vyberete, musí být spojitá, ale může být libovolně dlouhá. Asistentky nezískáváte, pouze sumy výher, takže jiná než finanční kritéria není třeba brát v úvahu.

Řešení

Obecné vylepšení

Použití všech těchto vylepšení závisí na konkrétní situaci. Někdy se nám nevyplatí předzpracování a jindy zase ano. Ale tyto možnosti zde jsou.

Odstranění záporných čísel – víme, že záporné číslo na začátku nám vždy zhorší výsledný součet posloupnosti, takže je můžeme vyřadit.

Sloučení kladných resp. záporných vedle sebe ležících (stojících) čísel – víme, že pokud máme vedle sebe dvě kladná resp. záporná čísla a chceme-li získat součet spojité posloupnosti můžeme tyto čísla sloučit. Protože dvě kladná čísla stojící vedle sebe a tvořící největší spojitý součet pokaždé zahrneme obě, pokud bychom zahrnuli jen jedno, dalo by nám zahrnutí druhého větší součet a tedy posloupnost pouze s jedním z nich by neměla maximální součet. Nejsou-li čísla zahrnuta v největší posloupnosti, tak nám na nich vlastně nezáleží. Obdobně pro záporná čísla.

Nuly – číslice 0 nám výsledný součet nikterak neovlivní, tedy se nic nestane pokud 0 z posloupnosti zcela vynecháme.

Jednoduchý O(n^2) algoritmus – Brutal force

Pro každé číslo z posloupnosti (označíme **k**) projdeme následující prvky (označíme **m**). Zároveň si udržujeme proměnou **max** ve které je dosud nejvyšší hodnota posloupnosti a proměnou **sum** ve které je aktuální suma tedy $k+m_1+m_2+m_3+...$

Po každém přičtení dalšího prvku **m** zkontrolujeme jestli aktuální součet není větší než maximum, pokud ano uložíme si do proměnné **max** novou hodnotu a pokračujeme dále. Pokud bychom potřebovali znát i celou posloupnost tvořící nejvyšší součet, není nic jednoduššího než si do dvou proměnných uložit **k** a **m** případně indexy těchto prvků. Jelikož **k** nám označuje začátek posloupnosti a **m** poslední prvek posloupnosti která tvoří maximální možný součet.

Implementace v javě

```
public static void brutalForce(int[] pole) {
	int max = 0;
	int k = 0, m = 0;
	for(int i=0; i<pole.length; i++) {
		int sum = 0;
		for(int j=i; j<pole.length; j++) {
			sum = sum + pole[j];
			if(sum > max) {
				max = sum;
				k = i;
				m = j;
			}
		}
	}
	System.out.println("Maximum je " + max + " (od " + k + " do " + m);
}
```

O(n) algoritmus na principu Dynamického programování

Algoritmus bude procházet pole posloupnosti zleva doprava přičemž si bude udržovat hodnotu maximálního součtu (označíme **max**), aktuálního součtu (označíme **sum**). Chceme-li navíc znát prvky tvořící posloupnost s nejvyšším součtem budeme si udržovat následující proměnné. Index prvku který nám tvoří počáteční číslo aktuálního součtu (označíme **actualStart**), počáteční index prvku maximálního součtu (například **maxStart**) a koncový index (například **maxEnd**).

Pokud je aktuální součet vetší než maximum, tak přiřadíme do proměnné **max** hodnotu z proměnné **sum**, dále do proměnné **maxStart** vložíme hodnotu proměnné **actualStart** a do proměnné **maxEnd** index aktuálního prvku (toho který jsme přičetli naposled).

Pokud je aktuální součet menší než maximální neděje se nic.

Pokud je ovšem hodnota aktuálního součtu záporná nemá smysl se touto posloupností dále zabývat. Proto si vynulujeme proměnnou s aktuálním součtem **sum=0** a také index **actualStart** nastavíme na hodnotu která bude i+1 (což je číslo které bude v poli sčítáno jako další).

Implementace v javě

```
public static void dp(int[] pole) {
	int max = 0;
	int sum = 0;
	int actualStart = 0, maxStart = 0, maxEnd = 0;

	for(int i = 0; i < pole.length; i++) {
		sum += pole[i];
		if (max < sum) {
			max = sum;
			maxEnd = i;
			maxStart = actualStart;
		}else if (sum < 0) {
			sum = 0;
			actualStart = i+1;
		}
	}
	System.out.println("Maximum je " + max + " (od " + maxStart + "
do " + maxEnd);
}
```

Závěr

Jako nejlepší na tuto úlohu se hodí algoritmus založený na dynamickém programování a to díky své jednoduché implementaci a malé náročnosti na paměť.

Algoritmus založený na D&C bych na tuto úlohu nepoužil z důvodu větší paměťové náročnosti (díky rekurzivnímu volání) a také má horší složitost než algoritmus DP. Navíc algoritmy založené na metodě D&C jsou obecně složitější na implementaci, pochopení i představitelnost.