Kostkové vzorce

Dokumentace k zápočtovému programu
z předmětu Programování 1

Jakub Mestek
1. ročník Bc. studia, obor IOI

zimní semestr 2017/2018

Zadání

V mnoha hrách na hrdiny se objevují vzorce typu |XdY|. To znamená „Vezmi kostku o |Y| stěnách, |X|-krát s ní hoď a výsledek sečti.“ Jednoduchý vzorec stylu |2d4|, |3d6+1|, |2d10+1d20-5| se ještě dá pochopit a odhadnout, ale jak se chová vzorec |(2d3-2)d(3d2)|, to už se určuje opravdu těžko. Napište program, který na vstupu přečte takovýto vzorec (s operátory |+-,*,d| v tomto pořadí priorit vzestupně) a vypíše pro každý možný výsledek pravděpodobnost, s jakou nastane.

Program Kostkové vzorce vyhodnocuje výrazy s operacemi $+$, $-$, $*$ a d a závorkami. Výraz XdY znamená vezmi Y-stěnnou kostku (tzn. se stěnami 1,...,Y), hoď s ní X-krát a výsledky hodů sečti. Výsledkem takového výrazu tedy může být několik různých hodnot v závislosti na padlých číslech. Program vypíše všechny možné výsledky spolu s jejich pravděpodobnostmi (přičemž předpokládá spravedlivou kostku).

Formální definice operace d: Výsledkem operace d není jedno číslo, ale množina možných hodnot a jejich pravděpodobnosti, tedy množina dvojic $V = \{(h_1, p_1), (h_2, p_2), \ldots, (h_n, p_n)\}$, kde h_i je hodnota jednoho možného výsledku a p_i jeho pravděpodobnost.

Výraz XdY, kde X,Y jsou celá čísla, $X \geq 0$, $Y \geq 1$, definujeme takto:

- Pro $X = 0$ je jediný možný výsledek, a to $(0, 1)$ – „prázdný součet“
- Pro $X = 1$ jsou výsledkem všechny možnosti, co mohou padnout při jednom hodu kostkou, tedy $1dY = \{(1, 1/Y), (2, 1/Y), \ldots, (Y, 1/Y)\}$.
- Pro $X > 1$ definujeme XdY „induktivně“: $XdY = 1dY + (X-1)dY$.

  Poznámka: $1dY + (X-1)dY$ znamená „sečtení“ možných výsledků výrazů $1dY$ a $(X-1)dY$ (viz 3.2).

Uživatelská část

1 Formát a omezení vstupu

Vstup se čte ze standartního vstupu. Vstup je výraz obsahující číslice, závorky `(,)` a operátory `+,-,*,d`. Vstup musí být zapsán na jednom řádku a bez mezer. Maximální délka vstupu je 255 znaků. Čísla mohou být v rozsahu -2 147 483 648 až 2 147 483 647.

2 Ukázkové vstupy a výstupy

Jednoduchá ukázka operace `d`

Vstup:

```
2d3
```

Výstup:

```
Mozne vysledky tohoto vyrazu jsou:
    2 s pravdepodobnosti 11.11 %
    3 s pravdepodobnosti 22.22 %
    4 s pravdepodobnosti 33.33 %
    5 s pravdepodobnosti 22.22 %
    6 s pravdepodobnosti 11.11 %
```

Použití jako jednoduchá výrazová kalkulačka

Vstup:

```
(-3*2-4)*(2-1)+15
```

Výstup:

```
Mozne vysledky tohoto vyrazu jsou:
    5 s pravdepodobnosti 100.00 %
```

Poznámka: Při nepoužití operátoru *d* nabývá výraz vždy pouze jedné hodnoty.

Složitější výraz

Vstup:

```
-5*(3d(2d2+1)-4d3)
```

Výstup:

```
Mozne vysledky tohoto vyrazu jsou:
  -55 s pravdepodobnosti 0.00 %
  -50 s pravdepodobnosti 0.02 %
  -45 s pravdepodobnosti 0.07 %
  -40 s pravdepodobnosti 0.21 %
  -35 s pravdepodobnosti 0.52 %
  -30 s pravdepodobnosti 1.13 %
  -25 s pravdepodobnosti 2.20 %
  -20 s pravdepodobnosti 3.87 %
```

```
-15 s pravdepodobnosti 6.17 %
-10 s pravdepodobnosti 8.92 %
 -5 s pravdepodobnosti 11.66 %
  0 s pravdepodobnosti 13.68 %
  5 s pravdepodobnosti 14.28 %
 10 s pravdepodobnosti 13.09 %
 15 s pravdepodobnosti 10.41 %
 20 s pravdepodobnosti 7.07 %
 25 s pravdepodobnosti 4.02 %
 30 s pravdepodobnosti 1.85 %
 35 s pravdepodobnosti 0.66 %
 40 s pravdepodobnosti 0.16 %
 45 s pravdepodobnosti 0.02 %
```

Neplatný vstup

Vstup:

```
1d(2d2-3)
```

Výstup:

```
Neplatny vstup:
  Neplatne argumenty operace d!
Program bude ukoncen.
```

Poznámka: Výraz je neplatný, protože možné hodnoty výrazu v závorce jsou -1, 0 nebo 1, ale 2. argument operace d musí být ≥ 1.

Programátorská část

3 Algoritmické řešení

Úkolem programu je v podstatě vyhodnocování aritmetického výrazu s jednou nestandartní operací. Bylo tedy nutné vyřešit:

- datovou reprezentaci výsledků
- provedení daných operací nad touto reprezentací
- algoritmus samotného vyhodnocení výrazu
- ošetření chybného vstupu

3.1 Reprezentace výsledků

Výsledkem operace d (a tudíž obecně i všech ostatních) není jedno číslo, ale množina možných hodnot a jejich pravděpodobnosti, tedy množina dvojic $V = \{(h_1, p_1), (h_2, p_2), \ldots, (h_n, p_n)\}$, kde h_i je hodnota jednoho možného výsledku a p_i jeho pravděpodobnost. Počet těchto dvojic není předem znám. Z toho důvodu byla zvolena reprezentace pomocí jednosměrného spojového seznamu:

```
PVysledek = ^TVysledek;
TVysledek = record
    h: THodnota;
    p: TPravdp;
    dalsi: PVysledek;
  end;
```

Během výpočtu je tento spojový seznam udržován setříděný vzestupně podle položky h.

Pro uložení hodnoty a pravděpodobnosti jsou definovány (a v celém programu používány) vlastní datové typy

```
TPravdp = real;
THodnota = longint;
```

Důvodem je zjednodušení případné změny datového typu – např. v případě implementace dělení by mohly být hodnoty i necelá čísla.

Poznámka: Vzhledem k častému vyhledávání prvku s určitou (nebo největší menší) hodnotou h jsem uvažoval i o efektivnější reprezentaci binárním vyhledávácím stromem. Pro očekávané vstupy s malou množinou možných výsledků je ale reprezentace spojovým seznamem dostatečně rychlá a reprezentace pomocí BVS by byla výrazně náročnější na implementaci.

3.2 Provedení operací

Vzhledem k více možným výsledkům i jednotlivých částí výrazu je nutno každou operaci $\diamond$ provádět stylem „každý s každým“, tedy všechny možné výsledky získáme provedením operace $\diamond$ na každou dvojici (h_i, p_i) a (h_j, p_j) , kde (h_i, p_i) je možný výsledek 1. argumentu a (h_j, p_j) je možný výsledek 2. argumentu.

Samotnou operaci $\diamond$ pak provedeme na hodnotách možných výsledků. Pravděpodobnost výsledku s takto vypočítanou hodnotou pak bude součin p_i a p_j, protože výsledek bude mít hodnotu $h_i \diamond h_j$ právě tehdy, když argument 1 bude mít hodnotu h_i (pravděpodobnost toho je p_i) a zároveň argument 2 bude mít hodnotu h_j (pravděpodobnost je p_j).

Tedy $Arg_1 \diamond Arg_2 = \{(h_i \diamond h_j, p_i \cdot p_j); (h_i, p_i) \in Arg1,\ (h_j, p_j) \in Arg2\}$.

Operace d

Vyhodnocení výrazu XdY nebude program (z důvodu optimalizace) provádět přesně podle definice, ale pomocí modifikace algoritmu pro rychlé mocnění: [1]

1. Převeď X do dvojkové soustavy.
2. $výsledky \leftarrow 0dY$ *. . . = nula – neutrální prvek sčítání*
3. **for** *číslice* X zleva doprava **do**

   case *číslice* **of**

   0: $výsledky \leftarrow výsledky + výsledky$

   1: $výsledky \leftarrow výsledky + výsledky + 1dY$
4. **return** $výsledky$

3.3 Vyhodnocení výrazu

Pro samotné vyhodnocení výrazu byla použita modifikace rekurzivního algoritmu pro vyhodnocení aritmetického výrazu (ukázán na přednášce M. Pergela):

function $Vyhodnot(odkud, kam)$:

1. Najdi operaci s nejnižší prioritou

   Priority:

   1: poslední `+`,`-` mimo závorku

   2: poslední `*` mimo závorku

   3: poslední `d` mimo závorku

   4: odstranění závorek

   5: načtení čísla
2. Proveď ji:

`+,-,*,d`: $Vyhodnot \leftarrow Vyhodnot(odkud, pozice\_operátoru - 1) \diamond Vyhodnot(pozice\_operátoru + 1, kam)$, kde $\diamond$ je příslušný operátor

závorky: $Vyhodnot \leftarrow Vyhodnot(odkud + 1, kam - 1)$

číslo: $Vyhodnot \leftarrow$ číslo na pozicích $odkud \ldots kam$, načte se pomocí Hornerova schématu

[1] `https://en.wikipedia.org/wiki/Exponentiation_by_squaring#Basic_method`

3.4 Kontrola vstupu

Hned po načtení se kontroluje, zda je výraz neprázdný a zda ve výrazu na vstupu nejsou nepovolené kombinace znaků:

- více operátorů vedle sebe
- operátor bezprostředně za otevírací závorkou nebo před uzavírací závorkou
- výraz začíná nebo končí operátorem
- chybějící operátor mezi závorkami nebo závorkou a číslem – např. `(2d3)(2+1)`
- prázdné závorky – tj. `()`

Navíc se kontroluje ještě správnost uzávorkování. Uzávorkování je správné, pokud při průchodu zleva doprava nikdy nenarazíme na více uzavíracích než otevíracích závorek a celkový počet uzavíracích je roven počtu otevíracích.

Kontrola se provádí průchodem výrazu zleva doprava, přičemž se vždy zjišťuje, zda aktuálnímu znaku nepředchází nějaký nepovolený.

Kromě tohoto je nutné kontrolovat argumenty operace `d` – pro XdY musí platit, že $X \geq 0$ a $Y \geq 1$. Celočíselnost argumentů se kontrolovat nemusí, neboť na vstupu přijímáme jen celá čísla a výsledkem podporovaných operací jsou proto také jen celá čísla. Kontrola argumentů se provádí až přímo při výpočtu ve funkci `Proved`.

4 Procedury a funkce

Hlavní část výpočtu obstarává funkce `Vyhodnot(vyraz: string; odkud, kam: byte): PVysledek` provádějící rekurzivní vyhodnocení výrazu pomocí algoritmu popsaného v kap. 3.3.

Načítání čísel zajišťuje funkce `V_Cislo(vyraz: string; odkud, kam: byte): PVysledek` pomocí Hornerova schématu.

4.1 Provedení operací

Výpočet výrazu typu *argument*1 $\diamond$ *argument*2 provádí funkce `Proved(operace: char; arg1, arg2: PVysledek; dis_arg1: boolean):PVysledek`, kde `operace` je jeden z operátorů `+,-,*,d`. Argumenty v paměti funkce nijak nemění, výstup funkce je nový spojový seznam. Při nastavení `dis_arg1=true` se během výpočtu provede `dispose` spojového seznamu `arg1`.

Výpočet se provádí stylem každý prvek `arg1` s každým prvkem `arg2` (viz 3.2), pro přidání možného výsledku do výstupního spoj. seznamu používá funkci `PridejVysledek`. Při operaci `d` otestuje správnost argumentů a následně volá pomocnou funkci `Proved_d`.

Funkce `Proved_d` provede výpočet algoritmem zmíněným v 3.2, pro mazání mezivýsledků využívá pomocnou proměnnou a volbu `dis_arg1=true` funkce `Proved`.

4.2 Práce se spojovým seznamem

Pro práci se spojovým seznamem slouží především procedura `PridejVysledek(var spojak: PVysledek; h: THodnota; p: TPravdp)`, která přidá dvojici (h, p) a udržuje přitom seznam setříděný. K tomu používá pomocné procedury a funkce `PV_VytvorVysledek`, `PV_NajdiPoslMensi` a `PV_PridejZa`.

Dále pak program využívá proceduru `VypisVysledky(spojak: PVysledek)`, která vypíše ve správném formátu všechny možné výsledky výrazu uložené ve spoj. seznamu, a proceduru `SmazVysledky(var spojak: PVysledek)`, která smaže daný spojový seznam z paměti (tj. zavolá `dispose` na všechny prvky spoj. seznamu).

4.3 Kontrola vstupu

Při nalezení problému se volá procedura `Chyba(hlaska: string)` s argumentem popisujícím nalezený problém. Procedura vypíše na výstup chybovou hlášku a ukončí program (volá `halt`).

Kontrolu vstupu podle kritérii uvedených v 3.4 provádí procedura `ZkontrolujVstup(vyraz: string)`. V případě nalezení nepovolené kombinace znaků volá proceduru `Chyba`.