Programování II pro matematiky

Teoretické cvičení 01

3. 3. 2021

Martin Mareš a Tomáš Karella

Pseudokód

Pravděpodobně budete znát z letního semestru, ale ukážeme si způsoby, jak zapsat nějaký algoritmus. Algoritmus si můžete zjednodušeně představit jako takovou kuchařku pro počítač.

Podmínka aka `if:`

Algoritmus: Maximum

Vstup: Číslo `a` , `b`

```
Pokud a > b:
    Vrátíme a
Jinak:
    Vrátíme b
```

Pseudokód

Pravděpodobně budete znát z letního semestru, ale ukážeme si způsoby, jak zapsat nějaký algoritmus. Algoritmus si můžete zjednodušeně představit jako takovou kuchařku pro počítač.

Podmínka aka `if`:

Algoritmus: Maximum

Vstup: Číslo `a` , `b`

```
Pokud a > b:
    Vrátíme a
Jinak:
    Vrátíme b
```

Příklad na minutku: Jak vypadá algoritmus pro minimum?

Cyklus aka `for`:

Algoritmus: Největší cifra v poli

Vstup: Pole cifer `x` (tj. každá cifra je přístupná jako `x[index]`)

```
Pro i od 1 do délka(x):
    maximum = max(x[i], maximum)
Vrátíme maximum
```

Cyklus aka `for`:

Algoritmus: Největší cifra v poli

Vstup: Pole cifer `x` (tj. každá cifra je přístupná jako `x[index]`)

```
Pro i od 1 do délka(x):
    maximum = max(x[i], maximum)
Vrátíme maximum
```

Příklad na pozornost: Co je na tomto přístup špatně?

Cyklus aka `for`:

Algoritmus: Největší cifra v poli

Vstup: Pole cifer `x` (tj. každá cifra je přístupná jako `x[index]`)

```
Pro i od 1 do délka(x):
    maximum = max(x[i], maximum)
Vrátíme maximum
```

Příklad na pozornost: Co je na tomto přístup špatně?

Co se stane pokud je pole prázdné? Takovýto algoritmus by v některých programovacích jazycích nešel přeložit (a to by Vás zachránilo). V jiných jazycích se nedefinované proměnné umí "tiše" přetypovat na `0`. Takovou chybu nechcete hledat.

Cyklus aka `for`:

Algoritmus: Největší cifra v poli

Vstup: Pole cifer `x` (tj. každá cifra je přístupná jako `x[index]`)

```
Pro i od 1 do délka(x):
    maximum = max(x[i], maximum)
Vrátíme maximum
```

Příklad na pozornost: Co je na tomto přístup špatně?

Co se stane pokud je pole prázdné? Takovýto algoritmus by v některých programovacích jazycích nešel přeložit (a to by Vás zachránilo). V jiných jazycích se nedefinované proměnné umí "tiše" přetypovat na `0`. Takovou chybu nechcete hledat.

```
maximum = -1
Pro i od 1 do délka(x):
    maximum = max(x[i], maximum)
Vrátíme maximum
```

Cyklus 2 aka. `foreach`:

Algoritmus: Největší cifra v poli

Vstup: Pole cifer `x` (tj. každá cifra je přístupná jako `x[index]`)

```
maximum = -1
Pro každý prvek v pole:
    maximum = max(prvek, maximum)
Vrátíme maximum
```

Cyklus 3 aka. `while`:

Algoritmus: Najdi `n` mocninu čísla `2`

Vstup: Číslo `n`

```
umocneni = 0
vysledek = 1
Dokud umoceni < n, opakuj:
    vysledek = vysledek * 2
    umocneni = umocneni + 1
Vrátíme vysledek
```

Příklad 1

Dostali jsme 32 různě těžkých, ale jinak stejných kuliček. K dispozici máme rovnoramenné váhy. Jakým způsobem máme použít váhy a kolikrát je musíme váhy použít abychom...

- ... nalezli nejlehčí kuličku?
- ... nalezli nejtěžší a nejlehčí kuličku?
- ... nalezli nejlehčí a druhou nejlehčí kuličku?

Příklad 1

Dostali jsme 32 různě těžkých, ale jinak stejných kuliček. K dispozici máme rovnoramenné váhy. Jakým způsobem máme použít váhy a kolikrát je musíme váhy použít abychom...

- ... nalezli nejlehčí kuličku?
- ... nalezli nejtěžší a nejlehčí kuličku?
- ... nalezli nejlehčí a druhou nejlehčí kuličku?
- Dostali jsme ještě i závaží. Zajímá nás, jestli jsme máme kuličku se stejnou váhou.
    - Kolik vážení budeme dělat v nejlepším případě?
    - Kolik vážení budeme dělat v nejhorším případě?
    - Jak se úloha změní, když budeme mít kuliček n ?

Příklad 2

Hledání největšího čísla ve skupině

Nechť máme skupinu n osob, každá z osob se vybere jedno číslo. Ve skupině každý může oslovit jiného člena, ale každý odpovídá pouze ano nebo ne. Zajímá nás, jaká je ideální strategie pokládání otázek (a jak vypadá otázka), abychom zjistili největší číslo ve skupině s minimem sociálního kontaktu.

Příklad 2

Hledání největšího čísla ve skupině

Nechť máme skupinu `n` osob, každá z osob se vybere jedno číslo. Ve skupině každý může oslovit jiného člena, ale každý odpovídá pouze ano nebo ne. Zajímá nás, jaká je ideální strategie pokládání otázek (a jak vypadá otázka), abychom zjistili největší číslo ve skupině s minimem sociálního kontaktu.

Nechť si každý vybere nějaké datum ve tvaru den, měsíc a rok (třeba datum narození). Pro každé takové datum můžeme spočítat ciferný součet. Zajímá nás, jaký je můžeme určit, kdo z našeho tento součet největší.

Například, já si budu myslet 21. 3. 1994, proto je můj ciferný součet `2+1+3+1+9+9+4 = 29`

Příklad 3

Úsek s největším součtem

Máme zadanou nějakou posloupnost celých čísel $x_1,...,x_n$ a chceme v ní nalézt úsek (tím myslíme souvislou podposloupnost), jehož součet je největší možný.

1, −2 , 4, 5, −1, −5, 2, 7

- Co se stane pokud je posloupnost prázdná?
- Co se stane pokud je posloupnost nezáporná?

Příklad 3

Úsek s největším součtem

Máme zadanou nějakou posloupnost celých čísel $x_1,...,x_n$ a chceme v ní nalézt úsek (tím myslíme souvislou podposloupnost), jehož součet je největší možný.

1, −2 , 4, 5, −1, −5, 2, 7

- Jak nejjednodušeji se dá taková posloupnost najít?

Příklad 3

Úsek s největším součtem

Máme zadanou nějakou posloupnost celých čísel $x_1,...,x_n$ a chceme v ní nalézt úsek (tím myslíme souvislou podposloupnost), jehož součet je největší možný.

1, −2 , 4, 5, −1, −5, 2, 7

Vstup: Posloupnost `x`

```
max_soucet = 0
Pro zacatek od 0 do délka(x):
    Pro konec od zacatek do konec:
        soucet = 0
        Pro i od zacatek do konec:
            soucet = soucet + x[i]
        max_soucet = max(soucet, max_soucet)
Vratime max_soucet
```

Příklad 3

Úsek s největším součtem

Máme zadanou nějakou posloupnost celých čísel $x_1,...,x_n$ a chceme v ní nalézt úsek (tím myslíme souvislou podposloupnost), jehož součet je největší možný.

1, −2 , 4, 5, −1, −5, 2, 7

Vstup: Posloupnost `x`

```
max_soucet = 0
Pro zacatek od 0 do délka(x):
    Pro konec od zacatek do konec:
        soucet = 0
        Pro i od zacatek do konec:
            soucet = soucet + x[i]
        max_soucet = max(soucet, max_soucet)
Vratime max_soucet
```

- Kolik musíme udělat kroků?

Příklad 3

Úsek s největším součtem

Máme zadanou nějakou posloupnost celých čísel $x_1,...,x_n$ a chceme v ní nalézt úsek (tím myslíme souvislou podposloupnost), jehož součet je největší možný.

1, −2 , 4, 5, −1, −5, 2, 7

Vstup: Posloupnost `x`

```
max_soucet = 0
Pro zacatek od 0 do délka(x):
    Pro konec od zacatek do konec:
        soucet = 0
        Pro i od zacatek do konec:
            soucet = soucet + x[i]
        max_soucet = max(soucet, max_soucet)
Vratime max_soucet
```

- Kolik musíme udělat kroků?
- Řádově n^3

Příklad 3

Úsek s největším součtem

Máme zadanou nějakou posloupnost celých čísel $x_1,...,x_n$ a chceme v ní nalézt úsek (tím myslíme souvislou podposloupnost), jehož součet je největší možný.

1, −2 , 4, 5, −1, −5, 2, 7

- Co to udělat trochu lépe?

Příklad 3

Úsek s největším součtem

Máme zadanou nějakou posloupnost celých čísel $x_1,...,x_n$ a chceme v ní nalézt úsek (tím myslíme souvislou podposloupnost), jehož součet je největší možný.

1, −2 , 4, 5, −1, −5, 2, 7

- Co to udělat trochu lépe?

```
max_soucet = 0
Pro zacatek od 0 do délka(x):
    soucet = 0
    Pro konec od zacatek do konec:
        soucet = soucet + x[konec]
        max_soucet = max(soucet, max_soucet)
Vratime max_soucet
```

Příklad 3

Úsek s největším součtem

Máme zadanou nějakou posloupnost celých čísel $x_1,...,x_n$ a chceme v ní nalézt úsek (tím myslíme souvislou podposloupnost), jehož součet je největší možný.

1, −2 , 4, 5, −1, −5, 2, 7

- Co to udělat trochu lépe?

```
max_soucet = 0
Pro zacatek od 0 do délka(x):
    soucet = 0
    Pro konec od zacatek do konec:
        soucet = soucet + x[konec]
        max_soucet = max(soucet, max_soucet)
Vratime max_soucet
```

- Více věcí si pamatujeme a mám díky tomu podstatné zrychlení - n^2 kroků

Příklad 3

Úsek s největším součtem

Máme zadanou nějakou posloupnost celých čísel $x_1,...,x_n$ a chceme v ní nalézt úsek (tím myslíme souvislou podposloupnost), jehož součet je největší možný.

1, −2 , 4, 5, −1, −5, 2, 7

- Není ještě něco lepšího?

Příklad 3

Úsek s největším součtem

Máme zadanou nějakou posloupnost celých čísel $x_1,...,x_n$ a chceme v ní nalézt úsek (tím myslíme souvislou podposloupnost), jehož součet je největší možný.

1, −2 , 4, 5, −1, −5, 2, 7

- Není ještě něco lepšího?

```
globalni_max = 0
konecne_max = 0
Pro i od 0 do délka(x):
    konecne_max = max(konecne_max + x[i], x[i])
    globalni_max = max(globalni_max, konecne_max)
Vratime globalni_max
```

Příklad 3

Úsek s největším součtem

Máme zadanou nějakou posloupnost celých čísel $x_1,...,x_n$ a chceme v ní nalézt úsek (tím myslíme souvislou podposloupnost), jehož součet je největší možný.

1, −2 , 4, 5, −1, −5, 2, 7

- Není ještě něco lepšího?

```
globalni_max = 0
konecne_max = 0
Pro i od 0 do délka(x):
    konecne_max = max(konecne_max + x[i], x[i])
    globalni_max = max(globalni_max, konecne_max)
Vratime globalni_max
```

- Teď už nám stačí jen řádově n kroků
- Podrobnější popis naleznete v knize Průvodce labyrintem algoritmů, Kapitola 1.1

Příklad 4

Nepřítel nám dal 9 opticky stejných kovových kuliček a rovnoramenné váhy a prozradil nám, že jedna z kuliček je nepatrně těžší, než ostatní.

- Kolikrát musíme použít váhy, abychom tuto kuličku identifikovali?

Příklad 4

Nepřítel nám dal 9 opticky stejných kovových kuliček a rovnoramenné váhy a prozradil nám, že jedna z kuliček je nepatrně těžší, než ostatní.

- Kolikrát musíme použít váhy, abychom tuto kuličku identifikovali?
- Jaká je odpověď, je-li kuliček 8? Jaká je odpověď, je-li kuliček 10?

Příklad 4

Nepřítel nám dal 9 opticky stejných kovových kuliček a rovnoramenné váhy a prozradil nám, že jedna z kuliček je nepatrně těžší, než ostatní.

- Kolikrát musíme použít váhy, abychom tuto kuličku identifikovali?
- Jaká je odpověď, je-li kuliček 8? Jaká je odpověď, je-li kuliček 10?
- Jaká je odpověď, může-li být odlišná kulička i lehčí?

Příklad 5

V továrně máme na stole 5 krabic s 20 malými ocelovými kuličkami. V každé krabici je jeden druh kuliček - buď tam najdeme 10 g, nebo 11 g kuličky. Kuličky od sebe neumíme opticky rozlišit a z krabic je můžeme libovolně vyndávat.

Úlohou je zjistit pomocí jediného vážení na digitální váze (která udává přímo váhu v gramech), ve které krabici jsou 10 gramové a ve které 11 gramové kuličky.

- Jak budete postupovat, pokud víte, že jsou 11g kuličky maximálně v jedné krabici?
- Jak budete postupovat pokud 11g kuličky mohou být v libovolném množství krabic?

Příklad 6

Dostali jsme 12 kuliček. Všechny vypadají stejně, ale 1 kulička má jinou hmotnost (11 kuliček má hmotnost stejnou). K dispozici máme rovnoramenné váhy.

- Popište strategii, jak pomocí 3 vážení na rovnoramenných vahách zjistit, která kulička má jinou hmotnost a zda je lehčí či těžší.

Domácí úkol

Od nepřítele jsme dostali `n` kuliček, které opticky neumíme rozeznat. Víme, že `n - 1` kuliček je stejných a jedna kulička je trochu těžší. K dispozici máme rovnoramenné váhy a zajímá nás, která kulička je nejtěžší.

- Kolik vážení budete muset minimálně udělat (v libovolné strategii)?
  - Nápověda: Při každém vážení si kuličky rozdělíme do skupin. Zamyslete se, jak velké skupiny kuliček jsou při vážení užitečné. V jaké skupině pak bude nejtěžší kulička v nejhorším případě?
- Najděte strategii, jak nalézt nejtěžší kuličku. Vaše strategie by měla použít, co nejméně vážení (i v nejhorším případě).
- Dokažte, že Vaše strategie je optimální (tj. v nejhorším případě algoritmus provede stejně vážení jako je minimální počet vážení).

Vysvětlete, proč Vaše řešení funguje (a proč funguje ve všech případech).