NI-KOP – úkol 1

Ondřej Kvapil

Kombinatorická optimalizace: problém batohu

První úkol předmětu NI-KOP jsem se rozhodl implementovat v jazyce Rust za pomoci nástrojů na *literate programming* – přístup k psaní zdrojového kódu, který upřednostňuje lidsky čitelný popis před seznamem příkazů pro počítač. Tento soubor obsahuje veškerý zdrojový kód nutný k reprodukci mojí práce.

Instrukce k sestavení programu

Program využívá standardních nástrojů jazyka Rust. O sestavení stačí požádat `cargo`.

```
cd solver
cargo build --release --color always
```

Benchmarking

Pro provedení měření výkonu programu jsem využil nástroje Hyperfine.

```
uname -a
cd solver
hyperfine --export-json ../docs/bench.json \
          --parameter-list n 4,10,15,20 \
          --parameter-list alg bf,bb,dp \
          --runs 2 \
          --style color \
          'cargo run --release -- {alg} < ds/NR{n}_inst.dat' 2>&1 \
    | fold -w 120 -s
```

Měření ze spuštění Hyperfine jsou uložena v souboru `docs/bench.json`, který následně zpracujeme do tabulky níže.

```
echo "algoritmus,\$n\$,průměr,\$\pm \sigma\$,minimum,medián,maximum" > docs/bench.csv
jq -r \
   '.[] | .[] | [.parameters.alg, .parameters.n
                , ([.mean, .stddev, .min, .median, .max]
                   | map("**" + (100000 * . + 0.5
                         | floor
                         | . / 100
                         | tostring
                         | if test("\\.") then sub("\\."; "**.") else . + "**" end
                         ) + " ms"
                        )
                  )
                ] | flatten | @csv' \
```

```
   docs/bench.json \
>> docs/bench.csv
echo ""
```

Table 1: Měření výkonu pro různé kombinace velikosti instancí problému (n) a zvoleného algoritmu.

algoritmus	n	průměr	$\pm\sigma$	minimum	medián	maximum
bf	4	**87**.32 ms	**2**.77 ms	**85**.36 ms	**87**.32 ms	**89**.28 ms
bf	10	**90**.19 ms	**0**.31 ms	**89**.98 ms	**90**.19 ms	**90**.41 ms
bf	15	**135**.75 ms	**2**.48 ms	**134** ms	**135**.75 ms	**137**.5 ms
bf	20	**1624**.38 ms	**24**.97 ms	**1606**.72 ms	**1624**.38 ms	**1642**.03 ms
bb	4	**85**.52 ms	**0**.35 ms	**85**.27 ms	**85**.52 ms	**85**.76 ms
bb	10	**93**.05 ms	**0**.76 ms	**92**.51 ms	**93**.05 ms	**93**.59 ms
bb	15	**142**.09 ms	**0**.37 ms	**141**.83 ms	**142**.09 ms	**142**.35 ms
bb	20	**1949**.43 ms	**0**.85 ms	**1948**.83 ms	**1949**.43 ms	**1950**.03 ms
dp	4	**90**.02 ms	**4**.96 ms	**86**.52 ms	**90**.02 ms	**93**.53 ms
dp	10	**97**.72 ms	**7**.33 ms	**92**.54 ms	**97**.72 ms	**102**.9 ms
dp	15	**99**.74 ms	**0**.25 ms	**99**.57 ms	**99**.74 ms	**99**.92 ms
dp	20	**113**.9 ms	**2**.8 ms	**111**.92 ms	**113**.9 ms	**115**.88 ms

Srovnání algoritmů

```
import matplotlib.pyplot as plt
import pandas as pd
from pandas.core.tools.numeric import to_numeric

df = pd.read_csv("docs/bench.csv", dtype = "string")
df.rename({
        "algoritmus": "alg",
        "$n$": "n",
        "průměr": "avg",
        "$\pm \sigma$": "sigma",
        "medián": "median",
        "minimum": "min",
        "maximum": "max",
    },
    inplace = True,
    errors  = "raise",
    axis    = 1,
)

numeric_columns = ["n", "avg", "sigma", "min", "median", "max"]
df[numeric_columns] = df[numeric_columns].apply(lambda c:
    c.apply(lambda x:
        to_numeric(x.replace("**", "").replace(" ms", ""))
    )
)

# Create a figure and a set of subplots.
fig, ax = plt.subplots(figsize = (11, 6))
```

```
labels = { "bf": "Hrubá síla"
         , "bb": "Branch & bound"
         , "dp": "Dynamické programování"
         }

# Group the dataframe by alg and create a line for each group.
for name, group in df.groupby("alg"):
    (x, y, sigma) = (group["n"], group["avg"], group["sigma"])
    ax.plot(x, y, label = labels[name])
    ax.fill_between(x, y + sigma, y - sigma, alpha = 0.3)

# Axis metadata: ticks, scaling, margins, and the legend
plt.xticks(df["n"])
ax.set_yscale("log", base = 10)
ax.set_yticks(list(plt.yticks()[0]) + list(df["avg"]), minor = True)
ax.margins(0.05, 0.1)
ax.legend(loc="upper left")

plt.savefig("docs/graph.svg")
```

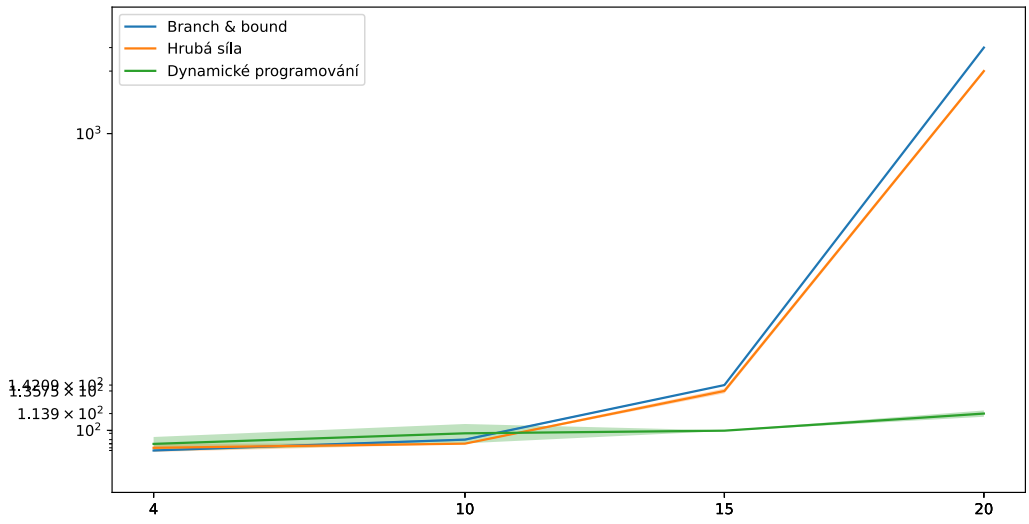

Figure 1: Závislost doby běhu na počtu předmětů. Částečně průhledná oblast značí směrodatnou odchylku (σ).

Implementace

Program začíná definicí datové struktury reprezentující instanci problému batohu.

```
struct Instance {
    id: i32, m: u32, b: u32, items: Vec<(u32, u32)>
}
```

```rust
use std::{io::stdin, str::FromStr};
use anyhow::{Context, Result, anyhow};

<<problem-instance-definition>>

fn main() -> Result<()> {
    let alg = {
        <<select-algorithm>>
    };

    loop {
        match parse_line()? {
            Some(inst) => println!("{}", alg(&inst)),
            None => return Ok(())
        }
    }
}

<<parser>>

impl Instance {
    <<solver-dp>>

    <<solver-bb>>

    <<solver-bf>>
}
```

Solvers

Brute force

```rust
fn solve_stupider(&self) -> u32 {
    let (m, b, items) = (self.m, self.b, &self.items);
    fn go(items: &Vec<(u32, u32)>, cap: u32, i: usize) -> u32 {
        use std::cmp::max;
        if i >= items.len() { return 0; }

        let (w, c) = items[i];
        let next = |cap| go(items, cap, i + 1);
        let include = || next(cap - w);
        let exclude = || next(cap);
        let current = if w <= cap {
            max(c + include(), exclude())
        } else {
            exclude()
        };
        current
    }

    go(items, m, 0)
}
```

Branch & bound

```rust
// branch & bound
fn solve_stupid(&self) -> u32 {
    let (m, b, items) = (self.m, self.b, &self.items);
    let prices: Vec<u32> = items.iter().rev()
        .scan(0, |sum, (_w, c)| {
            *sum = *sum + c;
            Some(*sum)
        })
        .collect();
    fn go(items: &Vec<(u32, u32)>, best: u32, cap: u32, i: usize) -> u32 {
        use std::cmp::max;
        if i >= items.len() { return 0; }

        let (w, c) = items[i];
        let next = |best, cap| go(items, best, cap, i + 1);
        let include = || next(best, cap - w);
        let exclude = || next(best, cap);
        let current = if w <= cap {
            max(c + include(), exclude())
        } else {
            exclude()
        };
        max(current, best)
    }

    go(items, 0, m, 0)
}
```

Dynamic programming

```rust
fn solve(&self) -> u32 {
    let (m, b, items) = (self.m, self.b, &self.items);
    let mut next = Vec::with_capacity(m as usize + 1);
    next.resize(m as usize + 1, 0);
    let mut last = Vec::new();

    for i in 1..=items.len() {
        let (weight, cost) = items[i - 1];
        last.clone_from(&next);

        for cap in 0..=m as usize {
            next[cap] = if (cap as u32) < weight {
                last[cap]
            } else {
                use std::cmp::max;
                let rem_weight = max(0, cap as isize - weight as isize) as usize;
                max(last[cap], last[rem_weight] + cost)
            };
        }
    }
```

```
    *next.last().unwrap() //>= b
}
```

Appendix

Zpracování vstupu neošetřuje chyby ve vstupním formátu,

```
<<boilerplate>>

fn parse_line() -> Result<Option<Instance>> {
    let mut input = String::new();
    match stdin().read_line(&mut input)? {
        0 => return Ok(None),
        _ => ()
    };

    let mut numbers = input.split_whitespace();
    let id: i32   = numbers.parse_next()?;
    let  n: usize = numbers.parse_next()?;
    let  m: u32   = numbers.parse_next()?;
    let  b: u32   = numbers.parse_next()?;

    let mut items: Vec<(u32, u32)> = Vec::with_capacity(n);
    for _ in 0..n {
        let w = numbers.parse_next()?;
        let c = numbers.parse_next()?;
        items.push((w, c));
    }

    Ok(Some(Instance {id, m, b, items}))
}
```

Výběr algoritmu je řízen argumentem předaným na příkazové řádce. Příslušnou funkci vrátíme jako hodnotu tohoto bloku:

```
use std::env;
let args: Vec<String> = env::args().collect();
if args.len() != 2 {
    println!(
        "Usage: {} <algorithm>, where <algorithm> is one of bf, bb, dp",
        args[0]
    );
    return Err(anyhow!("Expected 1 argument, got {}", args.len() - 1));
}
match &args[1][..] {
    "bf"    => Instance::solve_stupider,
    "bb"    => Instance::solve_stupid,
    "dp"    => Instance::solve,
    invalid => panic!("\"{}\" is not a known algorithm", invalid),
}
```

Trait `Boilerplate` definuje funkci `parse_next` pro zkrácení zápisu zpracování vstupu.

```
trait Boilerplate {
```

```
    fn parse_next<T: FromStr>(&mut self) -> Result<T>
      where <T as FromStr>::Err: std::error::Error + Send + Sync + 'static;
}

impl Boilerplate for std::str::SplitWhitespace<'_> {
    fn parse_next<T: FromStr>(&mut self) -> Result<T>
      where <T as FromStr>::Err: std::error::Error + Send + Sync + 'static {
        let str = self.next().ok_or(anyhow!("unexpected end of input"))?;
        str.parse::<T>()
           .with_context(|| format!("cannot parse {}", str))
    }
}
```