
MO-P 2001

Soutěž dětí a mládeže v programování
kategorie starší žáci
obvodní kolo — obvod Praha 1

Zadání a řešení úloh

Tento dokument se pokouší vyložit řešení úloh řešených v obvodním kole soutěže v programování kategorie starší žáci, pořádané v rámci Prahy 1. Nejde o vyčerpávající vysvětlení a od čtenářů předpokládá určitou znalost programátorských postupů (alespoň takovou, jaká je očekávatelná od účastníků soutěže v programování).

Oficiální stránka oblastních kol soutěží v programování na Praze 1 je na

<http://mo-p.czweb.org>

Na této adrese najdete kromě tohoto a loňského textu i vzorové programy, zdrojáky v dalších programovacích jazycích, výsledkovou listinu a případně podrobnější zamyšlení nad efektivitou prezentovaných řešení plus alternativní přístupy.

Z výše uvedených důvodů jsme se v tomto textu omezili pouze na vzorové programy v jazyce C.

V případě, že v textu nebo na webu najdete nějaké nedostatky či nesrovnalosti, případně budete mít nějaké dotazy, můžete se na nás obrátit prostřednictvím emailu.

Honza Vodička – Honza@printsoft.cz

Pavel Míka – Mika@Uhelnytrh.cz

Za spoluorganizaci tohoto kola děkujeme Jakubovi Fischerovi a Alexandře Svátové.

1 Chybějící čárky

1.1 zadání

Na světelné tabule se zobrazují rovnice ve tvaru $x+y=z$. Jednotlivé cifry jsou zobrazovány „digitálně“, tj. následujícím způsobem:

0 1 2 3 4 5 6 7 8 9

Tabule je bohužel poněkud starší, a tak některé čárky můžou v zobrazovaných cifrách chybět. Pokuste se najít původní verze rovnic. To znamená, napište program, který přečte ze vstupu tři čísla x, y a z ,

Nápověda: možné záměny, tj. seznamy cifer, které se mohly na tabuli mohly původně vyskytovat.

svítí	mohlo původně být	svítí	mohlo původně být
1	1347890	6	8
2	28	7	3890
3	389	8	8
4	489	9	89
5	5689	0	89

Příklad: pokud na tabuli svítí

1+2=3

mohlo to původně vypadat takto (nefungující segmenty jsou vyznačeny pouze obrysy):

1+2=3

1+8=9

7+2=9

Na zadání 1 2 3 (znaky $+$ a $=$ budeme v zadání vynechávat) by tedy měl program vypsat odpověď 1+2=3, 1+8=9, 7+2=9.

Příklad 2:

7+11=69

mohlo původně vypadat

7+81=69

8+81=69

$$\begin{array}{l} 8+80=88 \\ 9+79=88 \\ 9+80=89 \end{array}$$

Po zadání 7 11 69 by tedy odpověď měla vypadat: $7 + 81 = 88$, $8 + 81 = 89$,
 $8 + 80 = 88$, $9 + 79 = 88$, $9 + 80 = 89$

1.2 řešení

V dalším textu budeme označovat A , B a C členy rovnice $A + B = C$.

U této úlohy se nabízí nejjednodušší řešení — připravit si pro každou cifru každého ze tří čísel všechny možnosti, a potom postupně zkoušet všechny možné kombinace. Tato metoda jistě vede k cíli, ale není příliš šikovná. Zkouší všechny možné cifry C i v případě, že neexistuje žádné řešení. Příkladem může být například zadání

$$231$$

které nemá žádné řešení, ale tato metoda přesto musí projít dvě možnosti u prvního čísla (2, 8), tři u druhého (3, 8, 9) a dokonce sedm u třetího (1, 3, 4, 7, 8, 9, 0). To je dohromady 42 možností k projití — zdánlivě málo, ale když čísla příslušně „prodloužíme“ např. na

$$2222222\ 3333333\ 1111111$$

(rovněž žádné řešení), je potřeba projít již 5489031744 možností — a to jenom proto, abychom konstatovali, že řešení neexistuje.

Přitom jedno urychlení se nabízí téměř okamžitě. Spočívá v tom, že cifry čísla C nebudu zkoušet všechny, ale z A a B dopočítám, jak by mělo C vypadat a rovnou zjistím, zda je to vůbec možné. V praxi to vypadá tak, že si označím přijatelné cifry čísla C (v našem prvním případě již vyjmenované 1, 3, 4, 7, 8, 9, 0) a poté již pouze kombinuji cifry A a B — pro 2 a 3 jako počáteční hodnoty dostanu možnosti

$$\begin{array}{l} 2+3=5 \\ 2+8=10 \\ 2+9=11 \\ 8+3=11 \\ 8+8=16 \\ 8+9=17 \end{array}$$

tedy v každém řádu šest možností, pro které je pouze potřeba otestovat, zda

jsou pro C přípustné, či ne. Je tedy potřeba procházet podstatně méně variant, v druhém případě se nám tudíž počet zkoušených kombinací snižuje na 46656. Další možnost zrychlení hledání spočívá v tom, že A a B nemusím generovat vždy jako celek, ale zkoušet konstruovat od nejnižších cifer. Tím se vyhnou zkoušení možností, které nikdy nepovedou k cíli — jak je vidět z příkladu, pro $A = 2, B = 3, C = 1$ můžu kombinace cifer 2, 3 a 8, 8 můžu rovnou zahodit, neboť s nimi správné řešení nikdy nenajdu (neexistuje možnost, jak z jedničky udělat pětku nebo šestku). Tímto způsobem můžu snížit počet zkoušených kombinací v šesticiferném případě na 1457 (výpočet neprovádím, řešení viz vzorový program).

Existují další možnosti urychlení výpočtu, ne příliš složitým postupem lze snížit počet zkoušených kombinací u šesticiferného případu na 20 (což je mimochodem méně, než u nejhlupejšího přístupu k jednocifernému řešení) Případně zájemce odkazují opět na vzorový program a na Web.

1.3 výpis

```
/* Program digits.c urcuje vsechna mozna reseni rovnic napsanych na digitalni
 * svetelne tabuli (viz zadani uloh) */
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
/* Maximalni delka cisel */
```

```
#define MAX_DELKA 200
```

```
/* Pole pro ulozeni zadani */
```

```
char a[MAX_DELKA],b[MAX_DELKA],c[MAX_DELKA];
```

10

```
#ifndef PRINT_RESULTS
```

```
/* Pole, do kterych se bude zapisovat zkoumane reseni (pro pripadny vypis) */
```

```
char Spravne_a[MAX_DELKA], Spravne_b[MAX_DELKA], Spravne_c[MAX_DELKA];
```

```
#endif
```

```
/* Do teto promenne se ulozi pocet reseni */
```

```
int PocetReseni=0;
```

20

```
/* Povolen prepisy */
```

```
char *(Nahrady[])= { "08", "1347890", "28", "389", "489", "5689", "68",
                    "73890", "8", "98", "0" };
```



```

#ifdef QUICKER
/* Pole, kam se budou ukladat nefungujici varianty prenosu do jednotlivych
 * radu */
int NesmiByt[MAX_DELKA][2];
#endif

/* Pomocna funkce. Vraci nenulovou hodnotu, pokud je dany znak mezera,
 * novy radek, apod. Jinak vraci nulu. */
int JePrazdny (int Znak) {
    return (Znak==' ' || Znak=='\t' || Znak=='\n' || Znak=='\r');
}

/* Nacte ze vstupu cislo, tj. retezec obsahujici ciselne znaky. Odstranjuje
 * mezery apod., ignoruje neciselne znaky. Delka retezce je omezena argumentem
 * MaxDelka */
void PreciCislo(char *Cil, int MaxDelka) {
    int c;

    /* Ignoruj prazdne znaky na zacatku */
    while (JePrazdny(c=getchar()))
        if (c==EOF)
        {
            *Cil=0;
            return;
        }

    do {
        if (c==EOF)
        {
            *Cil='\0';
            return;
        }
        /* Pokud se jedna o ciselny znak, poznamenej */
        if (c>='0' && c<='9')
        {
            *Cil=c;
            /* Posun zapisovaci pozici */
            Cil++;
            /* Zmensi pocet volnych znaku. Pokud jsme na nule, je prusvih */

```

30

JePrazdny

40 PreciCislo

50

60


```

    if (--MaxDelka == 0)
    {
        fprintf(stderr,"Chyba: zadano prilis dlouhe cislo.");
        exit(1);
    }
}
/* Opakuj, dokud nenajdes prazdny znak */
} while (!JePrazdny(c=getchar()));
/* Na konec retezce dej '\0' */
*Cil='\0';
}

/* Funkce, která převrací poradi znaku v retezci */
void Prevrat(char *Text) {
    int start, konec;
    char Prenos;

    for(start=0, konec=strlen(Text)-1; start<konec; start++, konec--) {
        Prenos=Text[start];
        Text[start]=Text[konec];
        Text[konec]=Prenos;
    }
}

/* Funkce vypisuje nalezene reseni, tj. obsah poli Spravne_a, Spravne_b a
 * Spravne_c v "převrácenem" poradi */
void PisReseni() {
    Prevrat(Spravne_a);
    Prevrat(Spravne_b);
    Prevrat(Spravne_c);

    printf("%s + %s = %s\n",Spravne_a, Spravne_b, Spravne_c);

    Prevrat(Spravne_a);
    Prevrat(Spravne_b);
    Prevrat(Spravne_c);

    #endif
    PocetReseni++;
}

```

70

Prevrat

80

90 PisReseni

100


```

/* Tato funkce vraci ukazatel na polozku pole Nahrady prislusejici znaku ch.
 * Tzn. na polozky Nahrady[0] az Nahrady[9] pro znaky '0' az '9'. Nahrady[10]
 * pro '\0'. */
char *Nahrad(char ch) {                                Nahrad
    if (ch=='\0')
        return Nahrady[10];                            110
    else
        return Nahrady[ch-'0'];
}

/* Hlavni "testovací" funkce. Jako parametr bere pozici, kterou ma zkouset
 * a prenos ze scitani v nizsim radu */
int ZkusRad(int Rad, int Prenos) {                    ZkusRad
    char *Prvni;
    char *Druha;
    char *Treti;                                       120
    char Vysledek;
    char PrenosDal;
    int Uspech;
    int Navrat;

    /* Mimo mozny rozsah cisla? */
    if (Rad>MAX_DELKA)
    {
        if (Prenos==1)
            return 0;                                    130
        else
        {
            PisReseni();
            return 1;
        }
    }

#ifdef QUICKER
    /* Pokud uz jsme si vyzkouseli, ze tohle nejde, tak to zkratka nejde */
    if (NesmiByt[Rad-1][Prenos])                        140
        return 0;
#endif
#endif

```



```

if (a[Rad-1]=='\0' && b[Rad-1]=='\0' && c[Rad-1]=='\0')
{
    if (Prenos==0)
    {
        PisReseni();
        return 1;
    }
    else
    {
#ifdef QUICKER
        NesmiByt[Rad-1][Prenos]=1;
#endif
        return 0;
    }
}

/* Nastavime si ukazatel na pole moznych vysledku */
Treti=Nahrad(c[Rad-1]);

/* Zde budeme sledovat uspesnost pokusu */
Uspech=0;

/* Ted zbyva jenom projit vsechny mozne zameny a hledat, jestli pro ne
* existuje vysledek */
for (Prvni=Nahrad(a[Rad-1]); *Prvni!='\0'; Prvni++)
for (Druha=Nahrad(b[Rad-1]); *Druha!='\0'; Druha++)
{
    /* Je jeste potreba otestovat, jestli nejvyssi pozici nektereho ze scitancu
    * nenahrazujeme nulou. */
    if (Rad==MAX_DELKA)
    {
        if (*Prvni=='0' || *Druha=='0')
            continue;
    }
    else
    {
        if ((*Prvni=='0' && a[Rad]=='\0' && a[Rad-1]!='\0') ||
            (*Druha=='0' && b[Rad]=='\0' && b[Rad-1]!='\0'))
            continue;
    }
}

```



```

    }

    /* Secti cisla reprezentovana znaky *Prvni a *Druha, preved na znak
    * a zkus najit v poli Treti */
    Vysledek=*Prvni - '0' + *Druha - '0' + Prenos;
    PrenosDal=Vysledek/10;
    Vysledek=Vysledek%10 + '0';
    /* Otestuji, jestli nemam v nejvyssim miste vysledku nulu */
    if (Vysledek=='0' && c[Rad]=='\0')
        continue;
    /* Funkce index hleda znak v retezci; pokud ho nenajde, vraci NULL,
    * jinak adresu, na niz byl znak nalezen */
    if (index(Treti,Vysledek) != NULL)
    {
#ifdef PRINT_RESULTS
        if (a[Rad-1]!='\0')
            Spravne_a[Rad-1]=*Prvni;
        if (b[Rad-1]!='\0')
            Spravne_b[Rad-1]=*Druha;
        Spravne_c[Rad-1]=Vysledek;
#endif
        if ((Navrat=ZkusRad(Rad+1,PrenosDal))== -1)
            return -1;
        else
            Uspech|=Navrat;
    }
}
if (!Uspech)
{
#ifdef QUICKER
    NesmiByt[Rad-1][Prenos]=1;
    /* Pokud uz pres tento rad nelze projit, vrat -1 */
    if (NesmiByt[Rad-1][1-Prenos])
        return -1;
#endif
}
return 0;
else
return 1;
}

```



```

int main(void) {
    #ifdef QUICKER
        memset(NesmiByt,0,sizeof(NesmiByt));
    #endif
    /* Vycisti si pole pro cisla */
    memset(a,0,sizeof(a));
    memset(b,0,sizeof(b));
    memset(c,0,sizeof(c));
    #ifdef PRINT_RESULTS
        memset(Spravne_a,0,sizeof(Spravne_a));
        memset(Spravne_b,0,sizeof(Spravne_b));
        memset(Spravne_c,0,sizeof(Spravne_c));
    #endif

    /* Nacti cisla */
    PreciCislo(a,MAX_DELKA);
    PreciCislo(b,MAX_DELKA);
    PreciCislo(c,MAX_DELKA);

    if (*a=='\0' || *b=='\0' || *c=='\0')
    {
        fprintf(stderr,"Chyba ve vstupu\n");
        exit(1);
    }

    /* Prevrat vstup ... bude se s tim lepe pracovat */
    Prevrat(a);
    Prevrat(b);
    Prevrat(c);

    /* Cisla jsou nactena, muzeme to spustit */
    ZkusRad(1,0);
    printf("%i %i\n",PocetReseni,clock());
    return 0;
}

```

main

230

240

250

260

2 Ukecaní řidiči

2.1 zadání

Láďa a Pepa jezdí s autobusem. Ráno před osmou se sejdou na konečné, kde každý z nich dostane přidělenou linku, na které bude celý den jezdit a kterou urazí za určitý počet minut (předpokládáme, že se dopravní situace přes den moc nemění, takže doba projetí linky ráno, v poledne i večer bude stejná). Jakmile dojde na konečnou, vyrazí hned na další kolečko.

Aby si Pepa s Láďou mohli trochu odfrknout, mají k dispozici třicetiminutovou přestávku, kterou si každý z nich začne čerpat po prvním odpoledním příjezdu na konečnou (to znamená po 12:00 včetně). Po této přestávce zase jezdí dál až do 17:00 (pokud jsou v 17:00 zrovna na trase, dojedou tuto — poslední — jízdu, zamknou autobus a jdou domů).

Urči, kolikrát denně si mohou Láďa a Pepa na konečné zamávat, mezi osmou ranní a pátou odpolední. (Zamávat si mohou, pokud se tam oba vyskytují ve stejném čase, tj. např. v 10:34; pokud oba současně tráví přestávku, mávají si jenom jednou)

Nepočítej jejich setkání na začátku (tj. v 8:00) a po konci směny (tj. v 17:00 a později). To znamená, vytvoř program, který přečte ze vstupu dvě celá kladná čísla vyjadřující periody linek v minutách a vypíše, kolikrát si Pepa s Láďou zamávají.

Příklad:

Láďovi trvá cesta 45 min	Pepovi trvá cesta 60 min
8:00	8:00
8:45	9:00
9:30	10:00
10:15	11:00
11:00	12:00 (přestávka do 12:30)
11:45	13:30
12:30 (přestávka do 13:00)	14:30
13:45	15:30
14:30	16:30
15:15	17:30 (konec)
16:00	
16:45	
17:30 (konec)	

Odpověď: Celkem si zamávají třikrát (v 11:00, o přestávce a ve 14:30)

2.2 řešení

Poznámka na úvod: Pokud by řidiči neměli povinnou přestávku, tak nepůjde o programovací úlohu, ale o úlohu matematickou. Stačilo by pouze spočítat nejmenší společný násobek časů jízdy, a tímto číslem vydělit celkovou dobu směny.

Protože však přestávku mají, tak matematický výraz použít nelze.

Začneme tím, že nejprve si označme některé hodnoty:

Lčas – čas jedné jízdy pro Láďu (v minutách) **Pčas** – čas jedné jízdy pro Pepu (v minutách) **D** – dopoledne, přesněji počet minut před časem přestávky tj. 12:30 (spočítáme jako $(12-8)*60 + 30$) **C** - celý den, přesněji počet minut před celé směny (spočítáme jako $(17-8)*60$)

Řešení této úlohy je docela jednoduché, není potřeba vymýšlet žádný složitý postup. Budeme postupně emulovat jízdu Pepy a Ládi, tedy budeme průběžně zaznamenávat v jakou dobu budou na společné konečné. K tomu nám budou stačit 2 číselné proměnné - jejich význam bude aktuální počet minut jízdy na trati. Nebude nás ale zajímat přesná poloha na každé z tratí, ale pouze čas, kdy je Láďa nebo Pepa ve společné konečné stanici. Označme si tato čísla jako **L** resp. **P** (pro Láďu resp. Pepu). Hodnoty těchto čísel se budou vždy zvětšovat o **Lčas** resp. **Pčas** - krom toho, ale také vždy jednou o délku přestávky. Musím postupovat tak, abych zachytil všechny okamžiky, kdy jsou hodnoty **L** a **P** stejné. Celý princip postupu je takový, že vždy "nechám" jet na další okruh toho řidiče, který má menší čas (menší hodnotu **L** resp. **P**). Pokud mají stejnou hodnotu, tak si libovolně jednoho vyberu. Tento jeden postup budu nazývat iterace. Když se po takové iteraci budou hodnoty rovnat, tak jsou ve stejnou dobu na stejném místě, a mohou si tedy zamávat. Takto budeme iterovat, dokud jeden z řidičů nedosáhne konce směny. Tedy dokud hodnota **L** resp. **P** dosáhne hodnoty alespoň **C**.

Problém ještě může nastat s přestávkami. Proto si u obou řidičů ještě navíc budeme pomatovat jejich aktuální stav - buď je každý z řidičů před přestávkou, nebo je na přestávce, nebo je po přestávce. Podmínku, kdy si mohou zamávat proto rozšíříme takto (musí být splněna alespoň jedna z podmínek): 1) **L** a **P** jsou shodné 2) Pepa je na přestávce, **P** je větší rovno **L** a **P** je nejvýše o 30 vyšší (tj. délka přestávky) než **L** 3) Láďa je na přestávce, **L** je větší rovno **P** a **L** je nejvýše o 30 vyšší (tj. délka přestávky) než **P** První podmínka znamená, že jsou na konečné ve stejnou dobu, druhá resp. třetí podmínka znamená, že jeden z řidičů je na přestávce a druhý v době přestávky přijel na konečnou.

Zamávat po konci směny už si nemohou, protože vždy iterujeme pouze s jedním z řidičů, a proto také právě jeden řidič dosáhne času 17:00 nebo později (druhý zůstane před 17:00). Nebo-li pro nás dosáhne konce směny pouze jeden řidič.

2.3 výpis

```
#include <stdio.h>

#define NASTUP      8      /* kdy nastupuji do prace */
#define KONEC      17     /* kdy prace konci - minimalni cas */
#define PRESTAVKA  12.5   /* kdy maji prestavku - minimalni cas */
#define PRESTAVKA_MIN 30 /* jak dlouha je prestavka */

int
main (void)                                     main
{
    int cas_lada, cas_pepa;      /* casy jedne jizdy v minutach (tam i zpet) */
    int aktualne_lada, aktualne_pepa; /* kolik minut jsou zrovna v jizde */
    int pocet_zamavani
        = 0;                      /* celkovy pocet zamavani */
    int
        prestavka_lada, prestavka_pepa; /* nastaveni, zda jeste musi na prestavku */
    /* hodnota 0 znamena - je pred prestavkou */
    /* hodnota 1 znamena - je na prestavce */
    /* hodnota 2 znamena - je po prestavce */
    20

    int celkova_doba =
        (KONEC -
         NASTUP) * 60;      /* celkovy pocet minut v praci */
    int dopoledne =
        (PRESTAVKA -
         NASTUP) * 60;      /* celkovy pocet minut v praci */

    /* nacteme vstupni data */
    30
    printf
        ("Zadej cas Lady (dohromady tam a zpet): ");
    scanf ("%i",
           &cas_lada);
    printf
        ("Zadej cas Pepy (dohromady tam a zpet): ");
    scanf ("%i",
           &cas_pepa);
```


40

```

/* osetřime jednoduse vstup */
if (cas_lada <= 0)
{
    printf
        (" Neplatny cas pro Ladu\n");
    return -1;
}

```

```

if (cas_pepa <= 0)
{
    printf
        (" Neplatny cas pro Pepu\n");
    return -1;
}

```

50

```

/* pocatecni hodnoty */
aktualne_lada =
    aktualne_pepa = 0;      /* zaciname stejne */
prestavka_lada =
    prestavka_pepa = 0;    /* jsme bez prestavky */

```

60

```

/* dokud jezdi oba, tak bude pracovat program */
while (
    (aktualne_lada
        <
        celkova_doba)
    &&
    (aktualne_pepa
        <
        celkova_doba))
{
    if
        (aktualne_lada
            <
            aktualne_pepa)
        /* kdyz jede dele Pepa, tak "posuneme" Ladu, jinak "Pepa" */
        {
            aktualne_lada

```

70


```

+= cas_lada;      /* pricteme k aktualnimu casu jednu jizdu */
80
if
(prestavka_lada
== 1)           /* kdyz byl Lada na prestavce, tak uz tam neni */
{
    prestavka_lada
    = 2;        /* Lada uz je po prestavce */
}

if (
    (prestavka_lada
    == 0)
    &&
    (aktualne_lada
    >=
    dopoledne))
/* Lada nebyl na prestavce, ale uz neni dopoledne */
{
    aktualne_lada
    += PRESTAVKA_MIN; /* doba prestavky */
    prestavka_lada
    = 1;             /* jsem na prestavce */
100
}
}
else
{
    aktualne_pepa
    += cas_pepa;     /* pricteme k aktualnimu casu jednu jizdu */

    if
    (prestavka_pepa
    == 1)           /* kdyz byl Pepa na prestavce, tak uz tam neni */
    {
        prestavka_pepa
        = 2;        /* Pepa uz je po prestavce */
    }

    if (
        (prestavka_pepa

```



```

        == 0)
        &&
        (aktualne_pepa
        >=
        dopoledne))
        /* Pepa nebyl na prestace, ale uz neni dopoledne */
        {
            aktualne_pepa
            += PRESTAVKA_MIN; /* doba prestavky */
            prestavka_pepa
            = 1;          /* jsem na prestavce */
        }
    }

    /* ted nasleduje test, zda si mohou zamavat */
    if
    (aktualne_lada
    == aktualne_pepa) /* ve stejny cas na stejnem miste */
    {
        printf
        ("Zamavaji si v %.2d:%.2d\n",
        NASTUP
        +
        (aktualne_lada
        / 60),
        aktualne_lada
        % 60);

        pocet_zamavani++; /* zvetsim aktualni pocet zamavani */
    }
    else
    if (
        (prestavka_pepa
        == 1)
        &&
        (
        (aktualne_pepa
        -
        aktualne_lada)
        <=

```

120

130

140

150


```

    PRESTAVKA_MIN)
    &&
    (aktualne_pepa
    >=
    aktualne_lada))
{
    printf
    ("Potkaji se na prestavce (Pepa ji konci v %.2d:%.2d) - %d %d\n",
    NASTUP
    +
    (aktualne_pepa
    / 60),
    aktualne_pepa
    % 60,
    aktualne_pepa,
    aktualne_lada);

    pocet_zamavani++; /* zvetsim aktualni pocet zamavani */
}
else
    if (
        (prestavka_lada
        == 1)
        &&
        (
            (aktualne_lada
            -
            aktualne_pepa)
            <=
            PRESTAVKA_MIN)
        &&
        (aktualne_lada
        >=
        aktualne_pepa))
    {
        printf
        ("Potkaji se na prestavce (Lada ji konci v %.2d:%.2d)\n",
        NASTUP
        +
        (aktualne_lada

```



```

        / 60),
        aktualne_lada
        % 60);

        pocet_zamavani++; /* zvetsim aktualni pocet zamavani */
    }

    /* tim jsem dokoncil jednu iteraci */
}

printf
("Celkovy pocet zamavani: %d\n",
 pocet_zamavani);

return 0;
}

```

3 Morseovka

3.1 zadání

Petr se učí morseovku a umí zatím několik písmen (konkrétně b,c,h,j,l,o,p,u,v,y,z), ke všemu neumí vysílat mezery mezi jednotlivými písmeny. Zkus dekodovat Petrovu zprávu. To znamená, napiš program, který přečte ze vstupu posloupnost teček (·) a čárek (-) a vypíše původní zprávu.

Tabulka znaků a jim odpovídajících kódů:

b	····	c	··-·	h	····
j	··---	l	····	o	----
p	··---	u	··--	v	····-
y	··---	z	··---		

Příklad:

----- lze dekodovat jako zbylychlupy

3.2 řešení

Patrně nejprímější metoda pro řešení tohoto problému spočívá v postupném zkoušení jednotlivých písmen. První písmeno, kterým by zpráva mohla začínat, potom vypíšeme a odebereme příslušné čárky a tečky, pokud ještě nějaké zbyly, postup opakujeme.


```
char *Kody[]={".-",
             "...",
             ". . . ."};
```

```
char Pismena[]=('a',
               "...",
               'h');
```

```
void Dekoduj(char *Zprava) {
    int i;
    while (*Zprava!='\0')
        for (i=0;i<POCET_PISMEN;i++)
            if (strncmp(Zprava,Kody[i],strlen(Kody[i]))==0)
                {
                    printf("%c",Pismena[i]);
                    Zprava+=strlen(Kody[i]);
                    continue;
                }
}
```

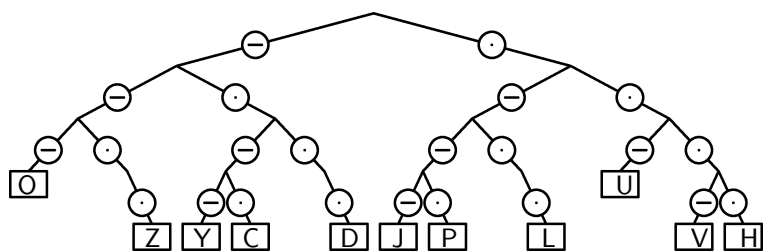
10 Dekoduj

20

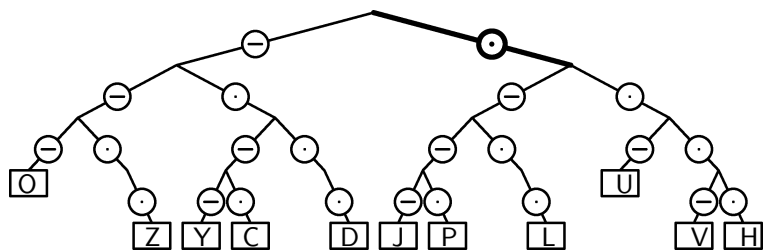
Když se na postup dekódování zprávy podíváme, zjistíme, že některá písmena zkusíme naprosto zbytečně. Pokud například začíná zpráva čárkou, je zbytečné zkoušet písmeno **a** či **h**. Pokud je *i* na druhém místě čárka, vypadává z možných kandidátů například **k**. Další aplikací tohoto postupu budeme postupně vyřazovat další z možných písmen.

Pokud se nám podaří nějak využít myšlenky naznačené v předchozím odstavci, můžeme běh programu výrazně urychlit. Zatímco v prvotním řešení bylo třeba porovnávat postupně všechna možná písmena, nyní může být potřeba podstatně méně porovnání — velké množství písmen během načítání rovnou vypadne.

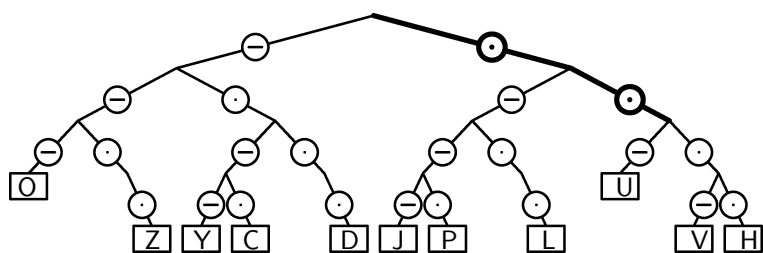
Jak však zařídit, aby zjišťování toho, která písmena mají být vyřazena, nebylo nakonec složitější, než prosté porovnávání? Představme si následující strukturu (pro podobnost s obráceným stromem ji budeme dále nazývat stromem):



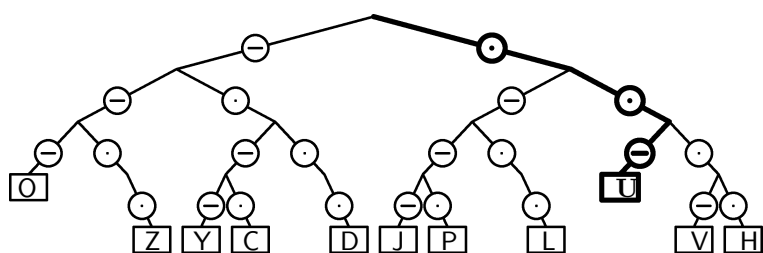
Jak je vidět, pokud si seřadíme znaky u každé spojnice (větve) na cestě od nejvyššího vrcholu do bodu odpovídajícího každému písmenu, dostaneme kód tohoto písmene v morseovce. Této vlastnosti se dá využít i obráceně: pokud si napíšeme zprávu v morseovce a budeme podle jednotlivých znaků postupovat ve stromě dolů po větvích, které odpovídají znakům, které jsou právě na řadě, dostaneme se po chvíli do koncového bodu, který odpovídá některému písmenu (pokud je v zakódované zprávě použito pouze znaků, které se vyskytují ve stromě). Pokud zpráva začíná znaky $···$, budeme při dekódování postupovat následovně: podle první tečky vyrazíme z kořene doprava:



podle další tečky postupujeme dále doprava:



a nakonec se podle třetího znaku, tj. čárky, vydáme levou větví.



Vidíme, že tímto jednoduchým postupem jsme zjistili první písmeno zprávy: U. Nyní bychom pro další znaky zprávy opět začali v kořeni stromu a postupovali stejným způsobem.

Dále si můžeme všimnout, že se nám podařilo použít dříve naznačené myšlenky: pokud je prvním znakem čárka, již neexistuje způsob, jak se po některé větvi dostat do libovolného znaku, který začíná tečkou (zdůrazňuji, že stromem se pohybuje výlučně shora dolů).

Zbývá vyřešit problém, jak takový strom vyrobit a jak ho uložit do paměti, aby se nám dobře procházel.

Co se týká uložení, jako vhodná volba se jeví složitější struktury nabízené vyššími programovacími jazyky (**struct** u C, **record** u Pascalu). U každého bodu stromu, ve kterém dochází k větvení, si musíme pamatovat, do jakého uzlu vede spojnice s tečkou a do jakého spojnice s čárkou (případně že z bodu žádná další spojnice nevede a že tento bod představuje písmeno). Body přitom můžeme uložit do pole pevné délky (jejich počet můžeme zjistit přímo ze stromu, případně jej jinak odhadnout). Tedy zhruba takto:

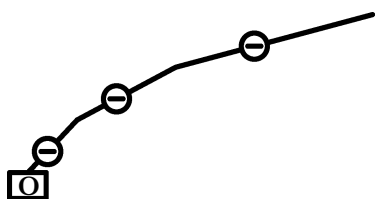
```
type Bod=record Tecka:integer; Carka:integer; Pismo:char; end;
type Strom=array [1..32] of Bod;
```


Kořen takto uloženého stromu bude ležet například v bodě 1.

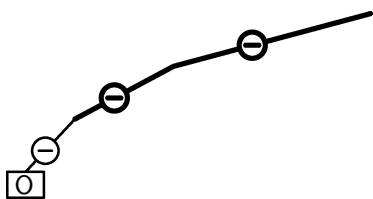
Jak bude vypadat konstrukce takového stromu? Velmi podobně jako jeho procházení, s tím rozdílem, že pokud se dostaneme do situace, kdy potřebujeme projít větví, která neexistuje, jednoduše ji vytvoříme. Na začátku máme jediný bod, kořen stromu. Řekněme, že chceme do tohoto (prázdného) stromu přidat znak **O** (kód ---). Nastavíme se tedy (stejně jako později při dekódování) do kořene a zkusíme jít po větvi odpovídající čárce (první znak kódu písmene **O**). Zjistíme, že taková větev ve stromě neexistuje, takže ji do kořene přidáme:



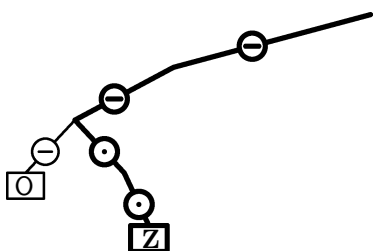
Chceme-li nyní jít po větvi odpovídající druhému znaku kódu, tj. opět čárce, zjistíme, že tato větev rovněž neexistuje. Vložíme ji, stejně tak jako větev odpovídající třetí čárce písmene **O**. Poslední bod, do kterého jsme se tímto způsobem dostali, označíme písmenem **O**.



Pokud jako další znak budeme přidávat **Z** (kód ---), větve reprezentující první dvě čárky již existují,



další dvě větve pro tečky vytvoříme, bod, do kterého se dostaneme, označíme písmenem **Z**.



Další aplikací tohoto postupu vytvoříme postupně celý strom, který je připraven na dekódování.

Na závěr je třeba ještě uvést na pravou míru nedostatek, který si všímavější z vás uvědomili: v předchozím textu uváděný postup nelze použít, pokud pracujeme s celou morseovkou. Pokud se podíváte do stromu uvedeného na předchozí stránce a zkusíte do něj zařadit písmeno **A**, sice se vám to povede, ale při dekódování textu si náhle nebudete vědět rady: pokud dekódovaný text bude začínat na ---, není jasné, jestli na začátek dát písmeno **A**, či **J** ... z bodu označeného písmenem **A** vycházejí další větve, což se nám doposud nevyskytlo. Pokud přijmeme větší abecedu, skutečně nám pro jeden zakódovaný text může vyjít víc možností dekódování. V takových případech nemusíme na ukázanou metodu úplně zanevřít, je pouze potřeba přidat další postupy, které nám umožní sledovat více možností zároveň.

3.3 výpis

/ Program dekoduje zprávu v morseovce s chybejícími mezerami mezi písmeny, předpokládá, že se ve zprávě nevyskytují žádné chyby. */*


```

struct Uzel {
    char Pismeno;
    int Tecka;
    int Carka;
};

struct Uzel Strom[31];

int Pouzito=0;

/* Zkontroluje, jestli je dany uzel v poradku, tj. pokud je v nem pismeno,
   uz nesmi nic pokracovat */
int KontrolujUzel (int Cislo) {
    if (Strom[Cislo].Pismeno!='\0' &&
        (Strom[Cislo].Tecka!=-1 || Strom[Cislo].Carka!=-1))
        /* Chyba — vrat nulu */
        return 0;
    else
        /* OK — vrat jednicku */
        return 1;
}

/* Vezme dalsi volny uzel (pokud existuje), pripravi jeho polozky a vrati
   jeho cislo. */
int PripravUzel() {
    /* Uz nemas volne uzly, vrat se */
    if (Pouzito>=31)
        return -1;
    /* Nastav pole na prazdne hodnoty */
    Strom[Pouzito].Pismeno='\0';
    Strom[Pouzito].Tecka=-1;
    Strom[Pouzito].Carka=-1;
    /* Vrat cislo uzlu a zvys pocet pouzitych uzlu */
    return Pouzito++;
}

/* Vlozi do stromu nove pismeno, pripadne vytvori potrebne uzly */
int VlozPismeno (char Pismeno, char *Kod) {
    /* Soucasny uzel (tj. ten, do ktereho jsem se dostal) */

```

10

KontrolujUzel

20

PripravUzel

30

40

VlozPismeno


```

int Soucasny=0;
/* Prochazim strom, dokud jsem se nedostal na spravne misto */
while (*Kod != '\0')
{
    /* Tecka */
    if (*Kod == '.')
    {
        /* Takhle zatim nikdo nesel — vytvor novy uzel */
        if (Strom[Soucasny].Tecka == -1)
        {
            /* Pokud by se to nahodou nepovedlo ... */
            if ((Strom[Soucasny].Tecka = PripravUzel()) == -1)
                /* ... tak se vrat s chybou */
                return -1;
            else
                /* Pokud se povedlo, zkontroluj soucasny uzel */
                if (KontrolujUzel(Soucasny)==0)
                    /* pokud je v nem problem, vrat chybu */
                    return -4;
        }
        /* Presunu se na dalsi uzel */
        Soucasny=Strom[Soucasny].Tecka;
    }
    /* Carka */
    else if (*Kod == '-')
    {
        /* Takhle zatim nikdo nesel — vytvor novy uzel */
        if (Strom[Soucasny].Carka == -1)
        {
            /* Pokud by se to nahodou nepovedlo ... */
            if ((Strom[Soucasny].Carka = PripravUzel()) == -1)
                /* ... tak se vrat s chybou */
                return -1;
            else
                /* Pokud se povedlo, zkontroluj soucasny uzel */
                if (KontrolujUzel(Soucasny)==0)
                    /* pokud je v nem problem, vrat chybu */
                    return -4;
        }
        /* Presunu se na dalsi uzel */

```

50

60

70

80


```

    Soucasny=Strom[Soucasny].Carka;
}
/* Nejakej nesmysl */
else
{
    /* V kodu se vyskytl spatny znak, vrat se s chybou */
    return -2;
}
/* Posun se na dalsi znak kodu */
Kod++;
}
/* Jses ve spravnem uzlu, nastav pismeno */
if (Strom[Soucasny].Pismeno!='\0')
    /* Kdyz uz tam je jine pismeno, vrat chybu */
    if (Strom[Soucasny].Pismeno!=Pismeno)
        return -3;
Strom[Soucasny].Pismeno=Pismeno;
/* Zkontroluj soucasny uzel */
if (KontrolujUzel(Soucasny)==0)
    /* pokud je v nem problem, vrat chybu */
    return -4;
else
    return Soucasny;
}

/* Zkusi dekodovat zpravu, pokud se mu to podari, vrati její delku a
   zapise ji do pole Original. Pokud se vyskytne chyba, vrati zapornou
   hodnotu */
int Dekoduj(char *Zprava, char *Original) {
    /* Kde se zrovna nachazis ve strome */
    int Soucasny=0;
    /* Delka dekodovane zpravy */
    int Delka=0;

    /* Dokud je co dekodovat */
    while (*Zprava!='\0')
    {
        /* Tecka */
        if (*Zprava == '.')
            /* Pokud neznas pokračování pro tečku, vrat chybu */

```

90

100

110

Dekoduj

120


```

    if (Strom[Soucasny].Tecka == -1)
        return -1;
    else
        Soucasny=Strom[Soucasny].Tecka;
    /* Carka */
    else if (*Zprava == '-')
        /* Pokud neznas pokracovani pro carku, vrat chybu */
        if (Strom[Soucasny].Carka == -1)
            return -1;
        else
            Soucasny=Strom[Soucasny].Carka;
    /* Nesmysl — vrat chybu */
    else
        return -2;
    /* Koukni se, jestli jsi nenasel pismeno, pokud ano, vypis a nastav
       se zase na vrchol stromu */
    if (Strom[Soucasny].Pismeno!='\0')
    {
        *Original=Strom[Soucasny].Pismeno;
        Original++;
        Delka++;
        Soucasny=0;
    }
    Zprava++;
}

/* Pokud je konec zpravy, ale jsme uprostred pismena, tak chyba */
if (Soucasny!=0)
    return -1;

/* Jinak dej na konec '\0' a vrat delku zpravy */
*Original='\0';
return Delka;
}

int main(void) {
    char Zprava[100];
    char Original[100];

    /* Na zacatku vytvorim hlavni uzel budouciho stromu. */

```

130

140

150

main

160


```

PripravUzel();

if (VlozPismeno('b',"-. .")<0)
    return -1;
if (VlozPismeno('c',"-. .")<0)
    return -1;
if (VlozPismeno('h'," . . .")<0)
    return -1;
if (VlozPismeno('j'," .---")<0)
    return -1;
if (VlozPismeno('l'," .- .")<0)
    return -1;
if (VlozPismeno('o',"---")<0)
    return -1;
if (VlozPismeno('p'," .--")<0)
    return -1;
if (VlozPismeno('u'," . .-")<0)
    return -1;
if (VlozPismeno('v'," . . -")<0)
    return -1;
if (VlozPismeno('y',"-. --")<0)
    return -1;
if (VlozPismeno('z',"-- .")<0)
    return -1;

/* Vsechno je pripraveno, precti si zpravu */
scanf("%s",Zprava);

/* Zkus ji dekodovat */
if (Dekoduj(Zprava, Original)<0)
    return -1;

/* Kdyz se ti to nahodou povedlo, tak ji vypis */
printf("Puvodni zprava:%s\n",Original);
}

```

170

180

190