Optimalizace spotřeby paměti. ATD Řídké pole. ATD Graf

Programovací techniky

doc. Ing. Jiří Rybička, Dr.
Ústav informatiky
PEF MENDELU v Brně
rybicka@mendelu.cz

Princip řídkého pole

- Přesná definice neexistuje, jedna z používaných specifikací je: jednorozměrné pole, v němž je na většině indexů (např. 95 %) uložena tzv. **majoritní hodnota** (například nula) a na zbytku jsou významové hodnoty.

Princip řídkého pole

- Přesná definice neexistuje, jedna z používaných specifikací je: jednorozměrné pole, v němž je na většině indexů (např. 95 %) uložena tzv. **majoritní hodnota** (například nula) a na zbytku jsou významové hodnoty.
- Těch 95 % hodnot **není třeba** ukládat. Majoritní hodnota se uloží **nejvýše** jednou.

Princip řídkého pole

- Přesná definice neexistuje, jedna z používaných specifikací je: jednorozměrné pole, v němž je na většině indexů (např. 95 %) uložena tzv. **majoritní hodnota** (například nula) a na zbytku jsou významové hodnoty.
- Těch 95 % hodnot **není třeba** ukládat. Majoritní hodnota se uloží **nejvýše** jednou.
- Abstraktní typ dat Řídké pole má typicky následující operace:

Princip řídkého pole

- Přesná definice neexistuje, jedna z používaných specifikací je: jednorozměrné pole, v němž je na většině indexů (např. 95 %) uložena tzv. **majoritní hodnota** (například nula) a na zbytku jsou významové hodnoty.
- Těch 95 % hodnot **není třeba** ukládat. Majoritní hodnota se uloží **nejvýše** jednou.
- Abstraktní typ dat Řídké pole má typicky následující operace:
  - přístup k prvku na indexu *n* (čtení/zápis);

Princip řídkého pole

- Přesná definice neexistuje, jedna z používaných specifikací je: jednorozměrné pole, v němž je na většině indexů (např. 95 %) uložena tzv. **majoritní hodnota** (například nula) a na zbytku jsou významové hodnoty.
- Těch 95 % hodnot **není třeba** ukládat. Majoritní hodnota se uloží **nejvýše** jednou.
- Abstraktní typ dat Řídké pole má typicky následující operace:
  - přístup k prvku na indexu *n* (čtení/zápis);
  - vložení nového prvku na konec/jinam;

Princip řídkého pole

- Přesná definice neexistuje, jedna z používaných specifikací je: jednorozměrné pole, v němž je na většině indexů (např. 95 %) uložena tzv. **majoritní hodnota** (například nula) a na zbytku jsou významové hodnoty.
- Těch 95 % hodnot **není třeba** ukládat. Majoritní hodnota se uloží **nejvýše** jednou.
- Abstraktní typ dat Řídké pole má typicky následující operace:
  - přístup k prvku na indexu *n* (čtení/zápis);
  - vložení nového prvku na konec/jinam;
  - výpis celého obsahu pole;

Princip řídkého pole

- Přesná definice neexistuje, jedna z používaných specifikací je: jednorozměrné pole, v němž je na většině indexů (např. 95 %) uložena tzv. **majoritní hodnota** (například nula) a na zbytku jsou významové hodnoty.
- Těch 95 % hodnot **není třeba** ukládat. Majoritní hodnota se uloží **nejvýše** jednou.
- Abstraktní typ dat Řídké pole má typicky následující operace:
  - přístup k prvku na indexu *n* (čtení/zápis);
  - vložení nového prvku na konec/jinam;
  - výpis celého obsahu pole;
  - vyhledání zadané hodnoty v poli (vyhledání pozice zadané hodnoty).

Řídká matice

- Dvourozměrná varianta řídkého pole, někdy se předpokládá ještě nižší procento zaplnění významovými prvky, což akcentuje problém úspory paměti při ukládání.

Řídká matice

- Dvourozměrná varianta řídkého pole, někdy se předpokládá ještě nižší procento zaplnění významovými prvky, což akcentuje problém úspory paměti při ukládání.
- Různé aplikace (často jako incidenční matice – varianta implementace grafu).

Řídká matice

- Dvourozměrná varianta řídkého pole, někdy se předpokládá ještě nižší procento zaplnění významovými prvky, což akcentuje problém úspory paměti při ukládání.
- Různé aplikace (často jako incidenční matice – varianta implementace grafu).
- Více možností:

Řídká matice

- Dvourozměrná varianta řídkého pole, někdy se předpokládá ještě nižší procento zaplnění významovými prvky, což akcentuje problém úspory paměti při ukládání.
- Různé aplikace (často jako incidenční matice – varianta implementace grafu).
- Více možností:
  - obecná řídká matice – náhodně rozmístěné významové prvky;

Řídká matice

- Dvourozměrná varianta řídkého pole, někdy se předpokládá ještě nižší procento zaplnění významovými prvky, což akcentuje problém úspory paměti při ukládání.
- Různé aplikace (často jako incidenční matice – varianta implementace grafu).
- Více možností:
  - obecná řídká matice – náhodně rozmístěné významové prvky;
  - diagonální matice – jsou obsazeny vybrané diagonály;

Řídká matice

- Dvourozměrná varianta řídkého pole, někdy se předpokládá ještě nižší procento zaplnění významovými prvky, což akcentuje problém úspory paměti při ukládání.
- Různé aplikace (často jako incidenční matice – varianta implementace grafu).
- Více možností:
  - obecná řídká matice – náhodně rozmístěné významové prvky;
  - diagonální matice – jsou obsazeny vybrané diagonály;
  - trojúhelníková matice – je obsazena polovina (pod nebo nad hlavní diagonálou, hlavní diagonála je/není zahrnuta).

Řídká matice

- Dvourozměrná varianta řídkého pole, někdy se předpokládá ještě nižší procento zaplnění významovými prvky, což akcentuje problém úspory paměti při ukládání.
- Různé aplikace (často jako incidenční matice – varianta implementace grafu).
- Více možností:
  - obecná řídká matice – náhodně rozmístěné významové prvky;
  - diagonální matice – jsou obsazeny vybrané diagonály;
  - trojúhelníková matice – je obsazena polovina (pod nebo nad hlavní diagonálou, hlavní diagonála je/není zahrnuta).
- Operace podobné jako u řídkého pole, obtížnější.

Možnosti implementace jednorozměrného řídkého pole

- Řídké pole lze implementovat několika metodami:

Možnosti implementace jednorozměrného řídkého pole

- Řídké pole lze implementovat několika metodami:
  - souřadnicový formát – lineární seznam záznamů dvojic (původní index; hodnota);

Možnosti implementace jednorozměrného řídkého pole

- Řídké pole lze implementovat několika metodami:
  - souřadnicový formát – lineární seznam záznamů dvojic (původní index; hodnota);
  - rozptylovací tabulka, rozptylování podle původního indexu (je důkladněji řešeno v jiné přednášce);

Možnosti implementace jednorozměrného řídkého pole

- Řídké pole lze implementovat několika metodami:
  - souřadnicový formát – lineární seznam záznamů dvojic (původní index; hodnota);
  - rozptylovací tabulka, rozptylování podle původního indexu (je důkladněji řešeno v jiné přednášce);
  - seznam souvislých bloků, každý blok je realizován polem (vhodné pro blokové zaplnění);

Možnosti implementace jednorozměrného řídkého pole

- Řídké pole lze implementovat několika metodami:
  - souřadnicový formát – lineární seznam záznamů dvojic (původní index; hodnota);
  - rozptylovací tabulka, rozptylování podle původního indexu (je důkladněji řešeno v jiné přednášce);
  - seznam souvislých bloků, každý blok je realizován polem (vhodné pro blokové zaplnění);
  - kratším polem se změnou výpočtu indexu (pro řídká pole vyznačující se nějakou pravidelností v pozicích významových hodnot).

Implementace řídké matice

- Elementární způsob – souřadnicový formát: Jednorozměrná struktura záznamů trojic (původní *x*, původní *y*, hodnota);

Implementace řídké matice

- Elementární způsob – souřadnicový formát: Jednorozměrná struktura záznamů trojic (původní x, původní y, hodnota);
- rozptylovací tabulka, rozptylování podle složeného indexu, např. $i = (x - 1) \cdot M_x + y$, kde hodnota M_x je počet původních řádků matice nebo jiná vhodně zvolená konstanta; předpokládáme původní indexaci jako v matematice, tj. od 1;

Implementace řídké matice

- Elementární způsob – souřadnicový formát: Jednorozměrná struktura záznamů trojic (původní x, původní y, hodnota);
- rozptylovací tabulka, rozptylování podle složeného indexu, např. $i = (x - 1) \cdot M_x + y$, kde hodnota M_x je počet původních řádků matice nebo jiná vhodně zvolená konstanta; předpokládáme původní indexaci jako v matematice, tj. od 1;
- čtvercové matice n^2: formát CSR nebo CSC (compressed sparse rows, compressed sparse columns);

Implementace řídké matice

- Elementární způsob – souřadnicový formát: Jednorozměrná struktura záznamů trojic (původní x, původní y, hodnota);
- rozptylovací tabulka, rozptylování podle složeného indexu, např. $i = (x - 1) \cdot M_x + y$, kde hodnota M_x je počet původních řádků matice nebo jiná vhodně zvolená konstanta; předpokládáme původní indexaci jako v matematice, tj. od 1;
- čtvercové matice n^2: formát CSR nebo CSC (compressed sparse rows, compressed sparse columns);
- reindexace do jednorozměrného pole (dobře použitelné pro trojúhelníkové matice);

Implementace řídké matice

- lineární struktura záznamů ($\lambda(i,j)$, hodnota), kde $\lambda(i,j) = i + (j-1) \cdot M_y$, kde M_y je počet sloupců původní matice nebo jiná vhodně zvolená konstanta, indexace od 1;

Implementace řídké matice

- lineární struktura záznamů ($\lambda(i,j)$, hodnota), kde $\lambda(i,j) = i + (j-1) \cdot M_y$, kde M_y je počet sloupců původní matice nebo jiná vhodně zvolená konstanta, indexace od 1;
- další (komplikovanější) možnosti pro speciální případy.

Implementace řídké matice

- lineární struktura záznamů ($\lambda(i,j)$, hodnota), kde $\lambda(i,j) = i + (j-1) \cdot M_y$, kde M_y je počet sloupců původní matice nebo jiná vhodně zvolená konstanta, indexace od 1;
- další (komplikovanější) možnosti pro speciální případy.
- Obecná struktura (platí jak pro řídké pole, tak i pro řídké matice): hodnotami jsou obecné ukazatele na libovolná data.

CSR

- Formát CSR – metoda tří jednorozměrných polí:

CSR

- Formát CSR – metoda tří jednorozměrných polí:
  - pole hodnot (všechny významové hodnoty, celkem *m* prvků);

CSR

- Formát CSR – metoda tří jednorozměrných polí:
  - pole hodnot (všechny významové hodnoty, celkem *m* prvků);
  - pole sloupcových indexů významových hodnot, celkem *m* prvků;

CSR

- Formát CSR – metoda tří jednorozměrných polí:
  - pole hodnot (všechny významové hodnoty, celkem *m* prvků);
  - pole sloupcových indexů významových hodnot, celkem *m* prvků;
  - pole odkazů na začátky řádků v poli sloupcových indexů, nejvýše *n* prvků

CSR

- Formát CSR – metoda tří jednorozměrných polí:
  - pole hodnot (všechny významové hodnoty, celkem m prvků);
  - pole sloupcových indexů významových hodnot, celkem m prvků;
  - pole odkazů na začátky řádků v poli sloupcových indexů, nejvýše n prvků
- Implementace zabere $2m + n$ paměťových míst; aby byla implementace efektivní, musí zjevně platit

$$2m + n \ll n^2$$

CSR – příklad

- Je dána matice s majoritní hodnotou 0:

$$\mathbf{A} = \begin{pmatrix} 10 & 0 & 3 & 0 \\ 0 & 8 & 0 & 4 \\ 2 & 2 & 6 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

CSR – příklad

- Je dána matice s majoritní hodnotou 0:

$$\mathbf{A} = \begin{pmatrix} 10 & 0 & 3 & 0 \\ 0 & 8 & 0 & 4 \\ 2 & 2 & 6 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

- Datové pole obsahuje všechny hodnoty zapsané po řádcích: $D = [10, 3, 8, 4, 2, 2, 6, 5]$

CSR – příklad

- Je dána matice s majoritní hodnotou 0:

$$\mathbf{A} = \begin{pmatrix} 10 & 0 & 3 & 0 \\ 0 & 8 & 0 & 4 \\ 2 & 2 & 6 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

- Datové pole obsahuje všechny hodnoty zapsané po řádcích: $D = [10, 3, 8, 4, 2, 2, 6, 5]$
- Sloupcové pole obsahuje ke každé hodnotě její sloupcový index: $C = [1, 3, 2, 4, 1, 2, 3, 4]$

CSR – příklad

- Je dána matice s majoritní hodnotou 0:

$$\mathbf{A} = \begin{pmatrix} 10 & 0 & 3 & 0 \\ 0 & 8 & 0 & 4 \\ 2 & 2 & 6 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

- Datové pole obsahuje všechny hodnoty zapsané po řádcích: $D = [10, 3, 8, 4, 2, 2, 6, 5]$
- Sloupcové pole obsahuje ke každé hodnotě její sloupcový index: $C = [1, 3, 2, 4, 1, 2, 3, 4]$
- Pole indexů řádků obsahuje indexy v poli D, kde začínají jednotlivé řádky: $R = [1, 3, 5, 8]$

Řídké pole kratším polem

- Předpokládejme obecně zaplněné řídké pole. Potřebujeme záznam hodnoty:

```
typedef long long int TypIndex;
typedef struct {
   void *data;
   TypIndex puvi; // původní index
} TypPrvek;
```

Řídké pole kratším polem

- Předpokládejme obecně zaplněné řídké pole. Potřebujeme záznam hodnoty:

```
typedef long long int TypIndex;
typedef struct {
   void *data;
   TypIndex puvi; // původní index
} TypPrvek;
```

- Z těchto prvků vytvoříme kratší strukturu pro *m* záznamů, důležitý je odhad maximální hodnoty *m* (ilustrace – 1000):

```
const unsigned int MaxHodnot = 1000;
typedef TypPrvek TypPoleDat[MaxHodnot];
```

Řídké pole kratším polem

- Abstraktní datový typ Řídké pole pak můžeme definovat:

```
typedef struct {
   TypPoleDat pole;
   unsigned int zaplneno;
   TypIndex max; // původní rozměr
   void *majorit;
} TypRPole;
```

Řídké pole kratším polem

- Abstraktní datový typ Řídké pole pak můžeme definovat:

```
typedef struct {
   TypPoleDat pole;
   unsigned int zaplneno;
   TypIndex max; // původní rozměr
   void *majorit;
} TypRPole;
```

- Datová struktura `pole` je doplněna celočíselnou hodnotou `zaplneno` uchovávající aktuální počet uložených hodnot.

Řídké pole kratším polem

- Abstraktní datový typ Řídké pole pak můžeme definovat:

```
typedef struct {
   TypPoleDat pole;
   unsigned int zaplneno;
   TypIndex max; // původní rozměr
   void *majorit;
} TypRPole;
```

- Datová struktura `pole` je doplněna celočíselnou hodnotou `zaplneno` uchovávající aktuální počet uložených hodnot.
- Složka `majorit` je potřebná pouze tehdy, není-li majoritní hodnota dopředu (například z principu dat) známa.

Řídké pole lineárním seznamem

- Definice záznamu hodnoty zůstává, místo pole pak definujeme:

```
typedef struct Clen{
   TypPrvek hodnota;
   Clen *dalsi;
} Clen;
typedef Clen *UkClen;
```

Řídké pole lineárním seznamem

- Definice záznamu hodnoty zůstává, místo pole pak definujeme:

```
typedef struct Clen{
   TypPrvek hodnota;
   Clen *dalsi;
} Clen;
typedef Clen *UkClen;
```

- Abstraktní datový typ Řídké pole pak bude mít následující definici:

```
typedef struct {
   UkClen zac, kon; // ukazatele na seznam
   TypIndex max; // původní rozměr
   void *majorit; // je-li nezbytně třeba
} TypRPole;
```

Přístup k prvkům

- Princip: zadáváme původní index *i* (rozsah $\langle 1; M \rangle$) a očekáváme získání hodnoty, která se na tomto indexu má nacházet.

Přístup k prvkům

- Princip: zadáváme původní index *i* (rozsah $\langle 1; M \rangle$) a očekáváme získání hodnoty, která se na tomto indexu má nacházet.
- Obecný algoritmus:

Přístup k prvkům

- Princip: zadáváme původní index *i* (rozsah $\langle 1; M \rangle$) a očekáváme získání hodnoty, která se na tomto indexu má nacházet.
- Obecný algoritmus:
  - test, zda $i \in \{1, \ldots, M\}$; neúspěch: ohlášení chyby

Přístup k prvkům

- Princip: zadáváme původní index *i* (rozsah $\langle 1; M \rangle$) a očekáváme získání hodnoty, která se na tomto indexu má nacházet.
- Obecný algoritmus:
  - test, zda $i \in \{1, \ldots, M\}$; neúspěch: ohlášení chyby
  - vyhledání *i* v uložených záznamech

Přístup k prvkům

- Princip: zadáváme původní index i (rozsah $\langle 1; M \rangle$) a očekáváme získání hodnoty, která se na tomto indexu má nacházet.
- Obecný algoritmus:
  - test, zda $i \in \{1, \ldots, M\}$; neúspěch: ohlášení chyby
  - vyhledání i v uložených záznamech
  - úspěch: získání hodnoty z nalezeného záznamu

Přístup k prvkům

- Princip: zadáváme původní index i (rozsah $\langle 1; M \rangle$) a očekáváme získání hodnoty, která se na tomto indexu má nacházet.
- Obecný algoritmus:
  - test, zda $i \in \{1, \ldots, M\}$; neúspěch: ohlášení chyby
  - vyhledání i v uložených záznamech
  - úspěch: získání hodnoty z nalezeného záznamu
  - neúspěch: dodání majoritní hodnoty

Přístup k prvkům

```
void *Ziskej(TypRPole X, TypIndex i){
  unsigned int kde=0;
  if (i>0 and i<=X.max) {
     while (kde<X.zaplneno and
            X.pole[kde].puvi!=i) kde++;
     if (kde<X.zaplneno)
        return X.pole[kde].hodnota;
     else return X.majorit;
  }
  else error();
}
```

Přístup k prvkům

```
void Vloz(TypRPole &X, TypIndex i, void *d){
  unsigned int kde=0;
  if (i>0 and i<=X.max) {
     while (kde<X.zaplneno and
            X.pole[kde].puvi!=i) kde++;
     if (kde<X.zaplneno)
         X.pole[kde].hodnota = d;
     else {
         X.pole[X.zaplneno].data = d;
         X.pole[X.zaplneno].puvi = i;
         X.zaplneno++;
     }
  } else error();
}
```

Řídká matice metodou CSR


```
typedef unsigned int TypVyznam;
const TypVyznam Max = 1000; //max. významových
typedef void * TypHodnoty[Max]; //vektor hodnot
typedef TypIndex TypCols[Max]; //vektor sloupců
typedef TypVyznam TypRows[Max]; //začátky řádků
typedef struct {
   TypHodnoty hodnoty; //datový vektor
   TypCols sloupce;    //sloupcové indexy
   TypRows indradky;   //začátky řádků
   TypVyznam obsaz, radku; //počet obsazených
   void * majorit;
} TypRMatice;
TypRMatice RM;
RM.obsaz = RM.radku = 0; //inicializace
```

Řídká matice metodou CSR


```
void *Ziskej(TypRMatice M,TypIndex i,TypIndex j){
  TypVyznam kde, konecr;
  if (i<=radku and M.indradky[i-1]!=0) {
      kde=M.indradky[i-1];
      konecr=M.indradky[i];
      if (konecr==0) konecr=M.obsaz;
      while (kde<konecr and sloupce[kde]!=j)
         kde++;
      if (kde<konecr) return M.hodnoty[kde];
          else return M.majorit;
  } else return M.majorit;
}
```

Řídká matice metodou „lambda“


```
const TypIndex MaxN = 1000000; //rozměr matice
typedef unsigned int TypVyznam;
const TypVyznam Max = 1000; //max. významových
typedef void * TypHodnoty[Max]; //vektor hodnot
typedef TypIndex TypLambda[Max]; //vektor lambda
typedef struct {
   TypHodnoty hodnoty; //datový vektor
   TypLambda lambda;   //hodnoty lambda
   TypVyznam obsaz;    //počet obsazených
   void * majorit;
} TypRMatice;
TypRMatice RM;
RM.obsaz = 0; //inicializace
```

Řídká matice metodou „lambda“

```
void *Ziskej(TypRMatice M,TypIndex i,TypIndex j){
  if (i<=MaxN and j<=MaxN){
    TypIndex lambda = i + (j-1)*MaxN;
    TypVyznam kde = 0;
    while (kde<M.obsaz and M.lambda[kde]!=lambda)
       kde++;
    if (kde<M.obsaz) return M.hodnoty[kde];
       else return M.majorit;
  } else return NULL;
}
```

Graf

- Abstraktní typ reprezentující graf (teorie grafů)

Graf

- Abstraktní typ reprezentující graf (teorie grafů)
- Graf: (U, H), kde $H \subseteq U \times U$

Graf

- Abstraktní typ reprezentující graf (teorie grafů)
- Graf: (U, H), kde $H \subseteq U \times U$
- Implementace:

Graf

- Abstraktní typ reprezentující graf (teorie grafů)
- Graf: (U, H), kde $H \subseteq U \times U$
- Implementace:
  - Matice incidence (řídká matice); čtvercová matice řádu $|U|$, hodnoty: logické (hrana existuje/neexistuje), reálné (vzdálenosti) apod.

Graf

- Abstraktní typ reprezentující graf (teorie grafů)
- Graf: (U, H), kde $H \subseteq U \times U$
- Implementace:
  - Matice incidence (řídká matice); čtvercová matice řádu $|U|$, hodnoty: logické (hrana existuje/neexistuje), reálné (vzdálenosti) apod.
  - Dynamická struktura uzlů s libovolným počtem následníků (podobně jako obecný strom)