Pracovní list 9: Řídká pole, grafy

Co už máme znát

- princip řídkého pole;
- princip řídké matice;
- typické operace ATD Řídké pole;
- metoda ukládání řídkého pole a řídké matice souřadnicovým formátem;
- metoda ukládání řídké matice linearizací lambda;
- metoda ukládání řídké matice CSR;
- informace z teorie grafů;
- způsoby implementace ATD Graf.

Kontrolní otázky

9.1 V čem spočívá princip řídké struktury (pole, matice)?

9.2 Jaké typické operace je potřebné implementovat u řídkého pole?

9.3 Co znamená souřadnicový formát pro ukládání řídké struktury?

9.4 Jaký je princip linearizace matice přepočtem indexů?

9.5 Jaký je princip ukládání řídké matice metodou CSR?

9.6 Jakou technikou lze úsporněji uložit trojúhelníkovou matici?

9.7 Jakým algoritmem lze zjistit, zda je graf souvislý?

9.8 Jak lze určit minimální délku cesty v ohodnoceném grafu mezi dvěma uzly?

9.9 Jakými dvěma metodami lze implementovat hodnoty ATD Graf?

Příprava na cvičení

Ve cvičení budeme potřebovat překladač jazyka C++, editor pro přípravu zdrojových textů a vybavení příkazového řádku. Pro jednotlivé úlohy jsou k dispozici soubory s daty, případně výsledné soubory v adresáři `/home/rybicka/vyuka/progt/cecko/cviceni/cv09` na serveru akela. Konkrétní jména těchto souborů jsou uvedena u jednotlivých úloh.

Řešené příklady

Příklad 9.1 Vytvořte modul implementující řídkou čtvercovou matici řádu 10 000 metodou „lambda“ indexů (detailní informace na přednášce). Operace: inicializace, získání prvku, uložení prvku.

Řešení: Datovým typem implementujícím hodnoty řídké matice bude pole hodnotami reprezentovanými obecným ukazatelem a pole hodnot $\lambda(i,j) = i + (j - 1) \cdot 10\,000$. Zároveň bude potřebné evidovat počet uložených prvků v obou polích.

Inicializace vyžaduje nastavení počtu obsazených hodnot na nulu a rovněž nastavuje majoritní hodnotu.

Získání hodnoty prvku vyžaduje zadání původních souřadnic, stejně jako vkládání nové hodnoty.

Hlavičkový soubor s názvem například `rmat.h` bude mít tvar:

```
#ifndef RMAT_H
#define RMAT_H
const unsigned int MaxN = 10000; //řád matice

typedef unsigned long long int TypIndex;
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;

void Start(TypRMatice &M, void *E);
void *Ziskej(TypRMatice M, TypIndex i, TypIndex j);
void Uloz(TypRMatice &M, TypIndex i, TypIndex j, void * E);

#endif
```

Implementační soubor `rmat.cpp` bude obsahovat těla operací:

```
#include <iostream>
#include ”rmat.h”

void Start(TypRMatice &M, void * E){
  M.obsaz = 0;
  M.majorit = E;
}
```

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

void Uloz(TypRMatice &M, TypIndex i, TypIndex j, void * E){
  if (i<=MaxN and j<=MaxN){
     M.lambda[M.obsaz] = i + (j-1)*MaxN;
     M.hodnoty[M.obsaz] = E;
     M.obsaz++;
  }
}
```

Příklad 9.2 Implementujte neorientovaný graf ohodnocený desetinnými čísly představujícími vzdálenost. Uvažujte, že graf může mít až 30 uzlů. Ze standardního vstupu načtěte potřebné hodnoty a nalezněte délku nejkratší cesty mezi dvěma zvolenými uzly.

Řešení: Pro úschovu potřebných dat využijeme matici sousednosti, kde na souřadnicích (i, j) bude uloženo desetinné číslo představující ohodnocení hrany. Pro souřadnice, jejichž hodnota nebude vložena, uvažujeme nekonečnou vzdálenost – vhodná konstanta `Infty`. Pro velikost matice 30×30 nebudeme v tomto řešení uvažovat úsporná uložení.

Algoritmus nalezení délky nejkratší cesty mezi dvěma uzly vypočítá vzdálenosti mezi všemi uzly. Projde všechny dvojice uzlů a pokusí se nalézt kratší vzdálenost přes nějaký třetí uzel. Pokud ji najde, uloží ji místo původní vzdálenosti do matice sousednosti. Stačí pracovat pouze v trojúhelníkové matici, pokud předpokládáme neorientovaný graf (délka cesty z i do j je stejná jako z j do i). Pro účel řešení vzdáleností nastavíme konstantu `Infty` na hodnotu 1000.

```
#include <iostream>
using namespace std;

const unsigned int MaxUzlu = 30;
const float Infty = 1000;
typedef float TypRadek[MaxUzlu];
typedef TypRadek TypMat[MaxUzlu];

float min(float X, float Y){
  //výběr menší ze dvou hodnot
  if (X<Y) return X; else return Y;
```

```
}

float pristup(TypMat X, int a, int b){
  //převod souřadnic do dolního trojúhelníku
  if (a>b) return X[a][b];
  else return X[b][a];
}

int main(){
  TypMat V;
  float d;
  int i, j, x, y;
  cin >> x >> y; //čísla uzlů s hledanou vzdáleností
  for (int i=0; i<MaxUzlu; i++)
    for (int j=0; j<i; j++) //jen dolní trojúhelník
       if (i==j) V[i][i]=0; //hlavní diagonála nulová
       else V[i][j]=Infty;

  while (cin>>i>>j>>d)
    if (i>j) V[i-1][j-1]=d;
    else V[j-1][i-1]=d;

  for (int i=0; i<MaxUzlu; i++)
    for (int j=0; j<i; j++)
       for (int k=0; k<i; k++)
          V[i][j]=min(V[i][j], pristup(V,i,k)+pristup(V,k,j));

  cout << ”Vzdálenost ”<< x << ” a ” << y << ” je ”
       <<pristup(V,x,y)<<endl;
  return 0;
}
```

Příklady

Příklad 9.3 Ve skladu jablek se provádí měření teploty každých 5 minut. Je-li změřená teplota odlišná od nastavené požadované teploty větší než 0,5 °C, ukládá se tento rozdíl do záznamového vektoru, v opačném případě se ukládá nula. Implementujte uložení těchto hodnot v řídkém poli souřadnicovou metodou s majoritní hodnotou nula. Nastavená požadovaná teplota je atributem implementovaného vektoru.

Příklad 9.4 Využijte implementovaného řídkého pole z úlohy 9.3 pro zpracování těchto dat ze vstupu: jako první hodnota je zadána standardní teplota a za ní posloupnost naměřených teplot počínaje půlnocí určitého dne. Předpokládejte, že měřené hodnoty pokrývají alespoň týden. Zjistěte, zda byla teplota v pořádku vždy ve 14 a v 21 hodin v prvních třech dnech vložených ze vstupu. Pro ověření použijte datový soubor `teploty.txt`.

Řešení: Při použití uvedeného souboru by se na výstupu měla objevit následující informace:

```
Den 1:
Čas: 840; nestandardní teplota t = 7.04
Čas: 1260; teplota v normě
Den 2:
Čas: 2280; nestandardní teplota t = 7.09
Čas: 2700; nestandardní teplota t = 7.27
Den 3:
Čas: 3720; teplota v normě
Čas: 4140; teplota v normě
```

Příklad 9.5 Implementujte operaci vkládání hodnoty do řídké matice uložené ve formátu CSR.

Příklad 9.6 Standardní vstup je tvořen řadou celočíselných hodnot představujících odstíny šedi v rastrovém obrazu o rozměrech 1 000 × 1 000 pixelů. Předpokládejte, že se jedná o jednoduchý motiv na černém pozadí (odstín 0) a barva pozadí silně převládá. Implementujte uložení tohoto obrazu pomocí řídké matice v souřadnicovém formátu, načtěte do ní vstupní hodnoty a spočtěte úsporu paměťového prostoru proti plné matici. Pro odladění lze použít připravený soubor `obraz.txt`.

Řešení: V zadaném souboru je 1 000 000 pixelů, nenulových pixelů je 99 966 a zaberou v paměti 499 830 bytů. V tomto případě je jasně vidět, že se už nejedná o typický případ řídkého pole, protože nemajoritních hodnot je cca 10 %. Proto také vychází celková úspora na pouhých cca 50 %.

Příklad 9.7 Předpokládejte, že ze standardního vstupu budou přečteny údaje o vzájemných korelacích mezi skupinou 10 hodnot očíslovaných čísly 0 až 9. Uložte tyto informace do trojúhelníkové matice linearizované do vektoru a vypište tuto matici na výstup v tiskové podobě. Tam, kde nebyla korelace zadána ze vstupu, vypisujte pomlčku. Činnost programu můžete vyzkoušet se vstupním souborem `korel.txt`.

Řešení: Po načtení připraveného souboru by se na výstupu měla objevit vypsaná matice ve tvaru podobném následujícímu výpisu:

```
       1
  ---        1
  ---    -0.254       1
-0.785    ---     0.753       1
-0.762    ---    -0.747    ---        1
  ---     ---      ---     ---     0.763       1
  ---     ---      ---     ---      ---     ---        1
 0.805    ---      ---     ---      ---     ---     ---        1
  ---    0.756     ---     ---      ---     ---     ---     ---        1
-0.014    ---      ---     ---      ---     ---     ---     ---     ---        1
```

Příklad 9.8 Na standardním vstupu se nachází řada trojic čísel $u_1 u_2 d$ představující zadání ohodnocených hran neorientovaného grafu (hrana mezi uzly u_1 a u_2 má hodnotu d). Implementujte tento graf a aplikujte na něm algoritmus nalezení nejkratší cesty mezi uzly, jejichž čísla jsou zadána jako první dva údaje standardního vstupu.

Příklad 9.9 Na standardním vstupu se nachází řada dvojic čísel představujících pořadová čísla incidujících vrcholů neorientovaného grafu. Předpokládejte, že vrcholů není celkem více než 20. Zjistěte, zda je takto zadaný graf souvislý. Pro ověření činnosti algoritmu použijte dva soubory: `souvis.txt` (zadání nesouvislého grafu) a `souvis2.txt` (zadání souvislého grafu).

Příklad 9.10 V úloze 9.9 nyní předpokládejte, že vrcholů může být až 1 000 a opět zjistěte, zda je zadaný graf souvislý. Vypočtěte a vypište na standardní výstup informaci o procentu zaplnění vstupní řídké matice a o paměťové úspoře, kterou vhodným uložením docílíte oproti vytvoření a naplnění matice řádu 1 000. Tutéž informaci vypište po provedeném výpočtu souvislosti grafu.

Co máme po cvičení umět

- Souřadnicovou metodu implementace řídkých struktur.
- Metodu CSR při ukládání řídké matice.
- Metodu linearizace metodou výpočtu souhrnného indexu nebo reindexací pro trojúhelníkovou matici.
- Metodu implementace ATD Graf pomocí matice sousednosti.
- Jednoduché grafové algoritmy.

Kontrolní otázky

9.10 Za jaké podmínky je ukládání řídké struktury souřadnicovou metodou paměťově úspornější?

9.11 Co je potřebné při ukládání do řídké matice modifikovat, jde-li navíc o symetrickou nebo trojúhelníkovou matici?

9.12 Jak pracuje algoritmus pro určení souvislosti grafu?

9.13 Jak pracuje algoritmus pro určení nejkratší cesty mezi zadanými uzly v grafu?