Domácí úkol k cvičení číslo 6

10. dubna 2024

1 Graf ze souboru

Finálním cílem je vyrobit si šikovnou reprezentaci grafu. Budeme používat obvyklou definici grafu jako dvojice množiny vrcholů a hran a budeme chtít mít možnost sestavit jeho matici sousednosti a později i incidencí matici vrcholů a hran. Výsledné struktury později použijeme pro procházení do hloubky a šířky.

1.1 Vstup

Graf dostaneme zadaný textovým souborem, kde každý řádek bude ve formátu

```
vrchol: seznam sousedních vrcholů oddělený čárkou a mezerou
```

Vrcholy jsou označené čísly, ne nutně po sobě jdoucími. Například:

```
1: 5, 6
5: 1, 6
6: 1, 5
```

je graf o třech vrcholech $\{1, 5, 6\}$ a třech hranách $\{(1, 5), (5, 6), (1, 6)\}$ a lze nakreslit jako trojúhelník.

Budou nás zajímat pouze neorientované grafy, ve kterých je hrana (v_i, v_j) totéž jako (v_j, v_i). Grafy by pak bylo možné zapisovat i úsporněji. Tj. následující soubor popisuje stejný graf

```
1: 5, 6
5: 6
```

a měl by mít i stejnou reprezentaci pomocí seznamu sousedů. Všimněte si, že do grafu je pak třeba přidávat i vrcholy, které jsou napravo od dvojtečky, v tomto případě vrchol `6`.

Povolíme si i hrany, které začínají a končí ve stejném vrcholu, tj. `1:1` je ok. Vrcholy, které nejsou v žádné hraně, akceptovat nebudeme a budeme předpkládat, že je nedostaneme na vstup, tj. `1:` není ok vstup.

1.2 Implementace grafu

Pro implementaci grafu použijte `std::unordered_map`. Je to lepší přístup než skrze pole/vektor. Pro graf se hodí `unordered_map<int, vector<int>>`, kde v prvním argumentu budou vrcholy a ve druhém bude vektor jeho sousedů, stejně jako na cvičení.

Při procházení souboru a zapisování grafu se bude hodit funkce `std::find` pro určení, jestli hrana už v grafu je anebo jestli je třeba ji přidat.

Pro přidání hrany, pokud už v grafu není, stačí použít operátor `[]`:

```
graph.adjacency_list[vertex_1].emplace_back(vertex_2);
```

Matice sousednosti Až budete mít hotové parsování souboru a uložení grafu, tak implementujte funkci která vytvoří *matici sousednosti* grafu. Tohle by mělo být zadarmo ze cvičení. *Matice sousednosti* M je čtvercová matice o velikosti $n_v \times n_v$, kde n_v je počet vrcholů, taková, že

$$M_{ij} = \begin{cases} 1, & \text{pokud existuje hrana mezi vrcholy } v_i, v_j, \\ 0, & \text{jinak.} \end{cases} \tag{1}$$

Pro matici sousednosti tedy bude nutné vybrat si nejaké očíslování množiny vrcholů, implementovatelné třeba zase jako `std::unordered_map`.

Testy Svůj program vyzkoušejte na několika ruzných rozumně malých grafech. Například:

```
7: 10, 111, 3
10: 7, 111, 3
111: 7, 10, 3
3: 7, 10, 111
```

Odpovídá kompletnímu grafu(=mezi každými dvěmi vrcholy je hrana) na čtyrech vrcholech a jeho matice sousednosti by tedy měla mít jedničku na každé pozici kromě diagonály. Jeho úsporná varianta

```
7: 10, 111, 3
10: 111, 3
111: 3
```

by měla vést ke stejným výsledkům.

Graf zadaný jako

```
1: 22, 55555
22: 1, 333
333: 22, 4444
4444: 333, 55555
55555: 1, 4444
```

Odpovídá kružnici/pětiúhelníku a jeho matice sousednosti bude mít jedničky jen nad a pod diagonálou a v levém dolním a pravém horním rohu. Úspornější zápis téhož grafu je:

```
1: 22
22: 333
333: 4444
4444: 55555
55555: 1
```

Na náledující stránce je návrh struktury a některé užitečné funkce.

```
struct Graph {
  unordered_map<int, vector<int>> adjacency_list;
  unordered_map<int, int> vertex_to_mat_index;
  int no_of_vertices;
  int no_of_edges;
  };


  void printGraph(const Graph& graph) {
    std::cout << "Graph:\n";
    for (const auto& [vertex, neighbours] : graph.adjacency_list) {
      std::cout << vertex << ": ";
      for( int neighbour : neighbours) {
        std::cout << neighbour << " ";
      }
      std::cout << "\n";
    }
    std::cout << std::endl;
  }


  Graph readGraphFromFile(const string& filename) {
    Graph graph;
    // TODO parsing goes here
    return graph;
  }


  vector<vector<int>> createAdjacencyMatrix(const Graph& graph) {
    int n = graph.no_of_vertices;
    vector<vector<int>> matrix(n, vector<int>(n, 0));
    // matrix construction goes here
    return matrix;
  }

  void constructIndexingMap(Graph& graph) {
    int index = 0;
    unordered_map<int, int> res;
    for (const auto& kv: graph.adjacency_list) {
      res[kv.first] = index;
      index += 1;
    }
    graph.vertex_to_mat_index = res;
  }


  void printGraphIndexingMap(const Graph& graph) {
    std::cout << "Graph vertex to index map:\n";
    for (const auto& [vertex, index] : graph.vertex_to_mat_index) {
      std::cout << vertex << ": " << index << ", ";
    }
    std::cout << std::endl;
  }

  void printMatrix(const vector<vector<int>>& matrix) {
    for (auto& line: matrix) {
      for(int val: line) {
        std:: cout << val << " ";
      }
      std::cout << "\n";
```

```
61          }
        }
63
```