

Domácí úkol k cvičení číslo 6

5. 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 incidenci 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 předpokládat, 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

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

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

Testy Svůj program vyzkoušejte na několika různých 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 dvěma vrcholy je hrana) na čtyřech vrcholech a jeho matice sousednosti by tedy měla mít jedničku na každé pozici. 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 následující stránce je návrh struktury a některé užitečné funkce.

```

1 struct Graph {
2     unordered_map<int, vector<int>> adjacency_list;
3     unordered_map<int, int> vertex_to_mat_index;
4     int no_of_vertices;
5     int no_of_edges;
6 };
7
8
9 void printGraph(const Graph& graph) {
10     std::cout << "Graph:\n";
11     for (const auto& [vertex, neighbours] : graph.adjacency_list) {
12         std::cout << vertex << ": ";
13         for (int neighbour : neighbours) {
14             std::cout << neighbour << " ";
15         }
16         std::cout << "\n";
17     }
18     std::cout << std::endl;
19 }
20
21
22 Graph readGraphFromFile(const string& filename) {
23     Graph graph;
24     // TODO parsing goes here
25     return graph;
26 }
27
28
29 vector<vector<int>> createAdjacencyMatrix(const Graph& graph) {
30     int n = graph.no_of_vertices;
31     vector<vector<int>> matrix(n, vector<int>(n, 0));
32     // matrix construction goes here
33     return matrix;
34 }
35
36
37 void constructIndexingMap(Graph& graph) {
38     int index = 0;
39     unordered_map<int, int> res;
40     for (const auto& kv: graph.adjacency_list) {
41         res[kv.first] = index;
42         index += 1;
43     }
44     graph.vertex_to_mat_index = res;
45 }
46
47
48 void printGraphIndexingMap(const Graph& graph) {
49     std::cout << "Graph vertex to index map:\n";
50     for (const auto& [vertex, index] : graph.vertex_to_mat_index) {
51         std::cout << vertex << ": " << index << ", ";
52     }
53     std::cout << std::endl;
54 }
55
56
57 void printMatrix(const vector<vector<int>>& matrix) {
58     for (auto& line: matrix) {
59         for (int val: line) {
60             std::cout << val << " ";
61         }
62         std::cout << "\n";
63     }
64 }

```

61		}
63	}	