Domácí úkol k cvičení číslo 4

25. března 2024

1 Příklady ke složitosti

1.1 Rozhodnětě a zdůvodněte, jestli platí následující tvrzení:

1. Pro funkci $f(n) = 8n^2 + 7n + 6$ platí:
$$f(n) \in \Omega(n^2), \tag{1}$$
$$f(n) \in \Theta(n^2). \tag{2}$$
2. Pro funkci $g(n) = 4n \log(8n) + n + 10$ platí:
$$g(n) \in O(n^2), \tag{3}$$
$$g(n) \in \Theta(n \log n). \tag{4}$$
3. Pro funkci $h(n) = 3n^2 + 10^4 n + \pi$ platí:
$$h(n) \in O(n^3), \tag{5}$$
$$h(n) \in \Theta(n^3). \tag{6}$$

Za dostatečné zdůvodnění se považuje výpočet odpovídající limity nebo ukázka toho, že lze nalézt c_1, c_2 a n_0 v definicích Ω, Θ, O, viz přednáška a doporučená literatura. Dostatečné zdůvodnění by mělo obsahovat komentář v přirozeném jazyce, alespoň na úrovni: „Tvrzení platí/neplatí protože:. . . “

1.2 Určete a zdůvodněte jaká je složitost daného algoritmu

Algorithm 1: What does this do?

```
//Input: n × (n + 1) matrix A[0 . . . n − 1; 0 . . . n] of real numbers
for i = 0, . . . , n − 2 do
    for j = i + 1, . . . , n − 1 do
        for k = i, . . . , n do
            A[j, k] = A[j, k] − A[i, k] ∗ A[j, i]/A[i, i]
        end
    end
end
return A
```

Předpokládejme, že cena všech operací násobení, sčítání, odečítání a dělení je stejná, a že je nezávislá na velikosti čísel. Dále se nebudeme trápit dělením nulou. Zápis `for` cyklu v algoritmu je myšlený tak, že se začíná v dolní mezi a jde se do horní meze včetně.

Jak by šlo algoritmus snadno zefektivnit?

1.3 Vyřešte rekurentní rovnici

Najděte funkci T takovou, že $T(0) = 1$ pro všechna $n \in \mathbb{N}$ (všechna kladná celá čísla) platí

$$T(n) = 2 + T(n-1). \tag{7}$$

2 Úloha k naprogramování

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 incideční matici vrcholů a hran.

2.1 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: 6
```

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

Pokud bude v textovém souboru nějaká hrana vícekrát, do grafu ji přidáme jen jednou. Tj. následující soubor popisuje stejný graf

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

Povolíme si i hrany, které začínají a končí ve stejném vrcholu, tj. `1:1` je ok. Stejně tak akceptujeme i vrcholy, které nejsou v žádné hraně, tj. `1:` je ok.

2.2 Implementace grafu

Pro implementaci grafu použijte `std::unordered_map`. Je to lepší přístup než skrze pole/vektor, protože je rychlejší a umožňuje přístup a zápis typu `value␣=␣mapa[key]`. Pro graf se hodí `unordered_map<int,␣vector<int>>`, kde v prvním argumentu budou vrcholy a ve druhém bude seznam/vektor jeho sousedů.

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

$$M_{ij} = \begin{cases} 1, & \text{pokud existuje hrana mezi vrcholy } v_i, v_j, \\ 0, & \text{jinak.} \end{cases} \tag{8}$$

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

Minimalistická kostra programu je na následující stránce, nenechte se omezovat a rozšiřte podle potřeby jak struktury tak funkce.

```cpp
struct Graph {
  unordered_map<int, vector<int>> adjacency_list;
  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;
  // 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 printMatrix(const vector<vector<int>>& matrix) {
  for (auto& line: matrix) {
    for(int val: line) {
      std:: cout << val << " ";
    }
    std::cout << "\n";
  }
}
```