Computational Complexity and Graphs

Amin Farjudian[1]

Hamlstad University

HÖGSKOLAN I HALMSTAD

DT8014 Algorithms Course, Autumn 2016

[1]Originally by Masoumeh Taromirad.

In this Session ...

Computational Complexity

- Refreshment
- Asymptotic Complexity
- "Big-Oh" Notation

Presentations

Graphs

- Terminology and Properties
- Graph Representations

Exercise

- *Complexity* roughly refers to the number of steps the algorithm will take for input of size n.
  - An algorithm that compares every pair of values in a list of n items will have to make n^2 comparisons, and so it takes about n^2 steps.
- An algorithm's running time on inputs of size n is proportional to the number of steps the algorithm will take (expressed by a function $f(n)$).
  - $f(n)$: the number of 'atomic' steps taken for inputs of size n;

Asymptotic Complexity

- An exact formulation of $f(n)$:
  - Tedious/complicated activity
  - Differentiates (classes) of algorithms that have *similar* behaviour
  - Many of the details are of little significance
- Compared with exact derivation of f, deriving its *growth rate*:
  - Simpler due to the richer structure of growth rates
  - Not sensitive to irrelevant additive or multiplicative factors
  - Informally: **the fastest-growing term**
  - e.g.
    - $1.62n^2 + 3.5n + 8$ grows like n^2
    - $1.62 \times 10^{-1000} n^2 + 3.5 \times 10^{1000} n + 8 \times 10^{1000}$ also grows like n^2, not like n.

- Asymptotic Upper Bounds (O) (**"Big-Oh" Notation**)
- Asymptotic Lower Bounds (Ω)
- Asymptotically Tight Bounds (Θ)

"Big-Oh" Notation

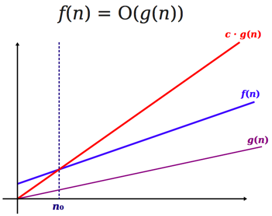
- Asymptotic Upper Bounds
  - Focus on the worst-case running time
- "Big-O" notation is a mathematical notation for upper-bounding a function's growth rate.
- Examples:
  - $n + 137 = O(n)$
  - $n^2 + 3n - 2 = O(n^2)$
  - $n^3 + 10n^2 \log n - 15n = O(n^3)$
  - $2^n + n^2 = O(2^n)$
  - $n! + 2^n = O(n!)$
  - $2^{2^n} + n^n + n! = O(2^{2^n})$
  - $\ldots$

"Big-Oh" Notation
Formal Definition

- $f, g : \mathbb{N} \to \mathbb{N}$
- We say $f(n) = O(g(n))$—equivalently, $f(n) \in O(g(n))$— iff

$$\exists n_0 \in \mathbb{N}, c \in \mathbb{R} : \forall n \in \mathbb{N} : (n \geq n_0 \to f(n) \leq c\, g(n))$$

- When n gets "sufficiently large" (i.e. greater than n_0), $f(n)$ is bounded from above by some constant multiple (specifically, c) of $g(n)$.
- Asymptotic growth of f is not more than that of g.



$$3n^2 + 2n + 1 = O(n^2)$$

Proof

- Take $n_0 = 1$ and $c = 6$
- Then for any $n \geq n_0$, we have

$$\begin{aligned} 3n^2 + 2n + 1 &\leq 3n^2 + 2n.n + 1.n^2 \\ &= 3n^2 + 2n^2 + n^2 \\ &= 6n^2 \\ &\leq 6(n^2) \end{aligned}$$

"Big-Oh" Notation

Algorithm Runtime

- For any algorithm A that works on inputs of size n, the function $T(n)$ gives the execution time of A working on an input of size n.
- Algorithm A has runtime of $O(f(n))$ means

$$T(n) \in O(f(n))$$

- $T(n)$: Execution time T is a function of the problem size n.
- $O(f(n))$: Complexity class (e.g. $O(\log n)$)

```
for i in 1 .. N do
    for j in 1 .. M do
        sequence of statements;
        // without any breaks or jumps to outside
    end for
end for
```

- The outer loop executes N times.
- Every time the outer loop executes, the inner loop executes M times.
- The statements in the inner loop execute a total of $N \times M$ times.
- The complexity is $O(NM)$.

Let's do some exercise!

Big-Oh Relevance to Practice and P vs NP

when the problem size N grows to N'
the execution time T grows to T'
according to the term “inside” the Big-Oh

Big-Oh	$N' = 2N$	$N' = 10N$
c	$T' = T$	$T' = T$
$\log N$	$T' = T + c$	$T' = T + 3.32c$
$\log^2 N$	$T' = T + (1 + 2\log N)c$	$T' = T + 3.32(1 + 2\log N)c$
N	$T' = 2T$	$T' = 10T$
$N \log N$	$T' = 2(Nc + T)$	$T' = 10(3.32Nc + t)$
N^2	$T' = 4T$	$T' = 100T$
N^3	$T' = 8T$	$T' = 1000T$
2^N	$T' = \sqrt{c}T^2$	$T' = \sqrt[10]{c}T^{10}$

Ω and Θ

- Asymptotic Lower Bounds: a complementary notation for lower bounds
  - for large input sizes n, the function $T(n)$ is at least a constant multiple of some specific function $f(n)$

$$\exists n_0 \in \mathbb{N}, c > 0 : \forall n \in \mathbb{N} : (n \geq n_0 \rightarrow f(n) \geq c\, g(n))$$

  - $T(n) \in \Omega(f(n))$ or $T(n) = \Omega(f(n))$

- Asymptotically Tight Bounds: $T(n)$ grows exactly like $f(n)$
  - $T(n)$ is both $O(f(n))$ and also $\Omega(f(n))$
  - $T(n)$ is $\Theta(f(n))$ or $T(n) = \Theta(f(n))$

In this Session ...

Computational Complexity
- Refreshment
- Asymptotic Complexity
- “Big-Oh” Notation

Presentations

Graphs
- Terminology and Properties
- Graph Representations

Exercise

Presentation Topics

- Topics
  1. Divide and conquer technique (e.g. Mergesort)
  2. Longest Path Problem
  3. Minimum Steiner Tree
  4. Huffman Codes
- 20-minute presentations
  - What the problem is
  - Overview of the existing solutions or a well-know solution
- Two presentations in each lecture session in week 38 and 39
- Assignment ....

In this Session ...

Computational Complexity

- Refreshment
- Asymptotic Complexity
- "Big-Oh" Notation

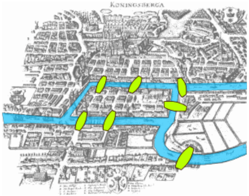
Presentations

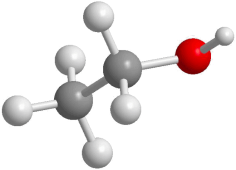
Graphs

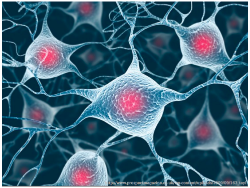
- Terminology and Properties
- Graph Representations

Exercise

Graphs







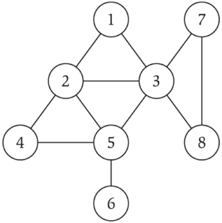
A mathematical structure for representing relationships

- E.g. Chemical Bonds, Transportation Maps, ...

Terminology and Properties

Formalism

- $G = (V, E)$: graph
- V = nodes
- E = edges between pairs of nodes.
- Captures pairwise relationship between objects
- Graph size parameters: $n = |V|$, $m = |E|$.



$$
\begin{aligned}
V =& \{1, 2, 3, 4, 5, 6, 7, 8\} \\
E =& \{1-2, 1-3, 2-3, 2-4, \\
& 2-5, 3-5, 3-7, 3-8, \\
& 4-5, 5-6, 7-8\} \\
m =& 11, n = 8
\end{aligned}
$$

Terminology and Properties

Simple Graph

- undirected
- no loops (edges that start and end at the same node)
- at most one edge between any two vertices

Regular Graph

- each vertex has the same number of neighbours

Complete Graph

- every pair of vertices has an edge connecting them

Planar Graph

- can be drawn on the plane such that no edges intersect.

Bipartite Graph

- vertices can be split in two sets so that, in both sets, no two vertices are adjacent.

Terminology and Properties

Subgraph of $G = (V, E)$

- its vertices form a subset of V
- its edges form a subset of E

Clique

- a set of pairwise adjacent vertices
- a **k-clique** has k vertices in this set

Path

- a sequence of nodes $v_1, v_2, ..., v_k$ with the property that each consecutive pair v_{i-1}, v_i is joined by an edge in E.

Simple Path

- a path in which all nodes are distinct.

Two vertices u and v are

- **connected** if there is a path from u to v.
- **adjacent** if there is an edge between them

A graph is

- **connected** if every pair of vertices is connected.
- **k-connected** if no set of $k - 1$ vertices exist that, if removed, would disconnect the graph.

Cycle

- a path $v_1, v_2, ..., v_k$ in which $v_1 = v_k, k > 2$, and the first $k - 1$ nodes are all distinct.

Tree

- a connected acyclic graph.
- for directed graphs, each vertex has at most one incoming edge.

Theorem. Let G be an undirected graph on n nodes. Any two of the following statements imply the third.

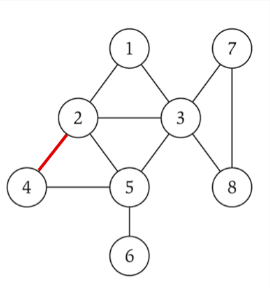
- G is connected.
- G does not contain a cycle.
- G has $n - 1$ edges.

Graph Representations

Adjacency Matrix

Adjacency matrix: n-by-n matrix with $A_{uv} = 1$ if (u, v) is an edge.

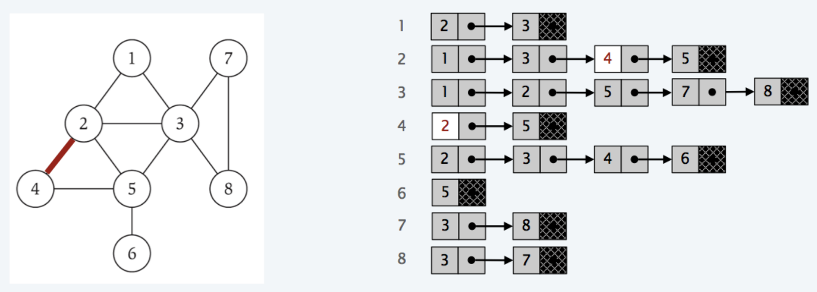
- Two representations of each edge.
- Space proportional to n^2
- Checking if (u, v) is an edge takes:
  - $\Theta(1)$ time with arrays.
  - $O(n^2)$ time with lists.
- Identifying all edges takes $\Theta(n^2)$ time.



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Adjacency lists: Node indexed array of lists.

- Two representations of each edge.
- Space is $\Theta(m+n)$.
- Checking if (u, v) is an edge takes:
  - $O(degree(u))$ time time with arrays.
  - $O(n^2)$ time with lists.
- Identifying all edges takes $\Theta(m+n)$ time.



In this Session ...

Minimum Spanning Tree

Exercise 2[2]

- A **spanning tree** of a connected, undirected graph is a subgraph of that graph which is a tree and connects all the vertices together. In a weighted graph, the sum of the weights of the edges in a spanning tree computes the weight of that spanning tree.
  A **minimum spanning tree (MST)** is then a spanning tree with weight less than or equal to the weight of every other spanning tree.
- The aim is to get familiar with MSTs:
  - Applications of MST in real-world problems
  - naive algorithm to find MST
  - Kruskal's algorithm
- Complete description is available on the course web page (Blackboard)
  - My homepage for now

[2] Taken from DT8014, 2014

In this session ...

- Asymptotic Complexity
- "Big-Oh" notation
- How O is used to demonstrate algorithm complexity
- Fundamentals of Graphs
- Exercise

Any Question?