

CSE 21—Mathematics for Algorithm and System Analysis

Summer, 2005

Day 9
Graph Theory
Asymptotic Analysis

Instructor: Neil Rhodes

Outline

Simple Graphs
Graphs
Directed Graphs
Trees
Minimum Spanning Trees
Asymptotic Complexity

2

Simple Graph

Definition

- Graph G consists of a pair (V, E)
 - V is a finite set called the vertices
 - E is a subset of the two-element subsets of V called the edges

Example

- Simple graph

- Non-simple graph

3

Graph

Motivation

- Sometimes, we need more than one edge between 2 vertices

Definition

- A graph G is a triple (V, E, Φ) where:
 - V is a finite set (vertices)
 - E is a finite set (edges)
 - Φ is a function with domain E and codomain 2-element subsets of V (incidence function)

Example

Relationship between graphs and simple graph

4

Definitions

Loop: an edge that connects a vertex to itself

- Graphs and simple graphs contain no loops
 - Can be allowed by adjusting codomain to be 2-element subsets of V unioned with 1-element subsets of V

Degree of vertices

- Given $v \in V$, the *degree*, $d(v)$ = the number of $e \in E$ such that $v \in \Phi(e)$
- The *degree sequence* of the vertices of G is the sequence of degrees of each vertex, sorted in increasing order

Isomorphic

- Two graphs G_1 , and G_2 are *isomorphic* iff
 - There is a bijection f_V from V_1 to V_2
 - There is a bijection f_E from E_1 to E_2
 - For all $e \in E_1$, if $\Phi_1(e) = \{v_i, v_j\}$, then $\Phi_2(f(e)) = \{\Phi_2(v_i), \Phi_2(v_j)\}$

5

Directed Graph (Digraph)

Motivation:

- Sometimes, we need to distinguish the direction $v_1 \rightarrow v_2$ from $v_2 \rightarrow v_1$

Definition

- A digraph $D = (V, E, \Phi)$ where
 - V is a finite set
 - E is a finite set
 - Φ is function with domain E and codomain $V \times V$

Converting a simple graph (with loops) to a simple digraph

6

Definitions

Walk

- Given G , a graph
- If
 - P is a sequence of elements of E , $e_1, e_2, \dots, e_{n-1}$.
 - There exists Q , a sequence of vertices of V , $v_1, v_2, \dots, v_n$ where $\Phi(e_i) = \{v_i, v_{i+1}\}$,
- then P is a walk of G

Trail

- If P is a walk of G and the elements of P are distinct, then P is a trail of G

Path

- If P is a walk of G and the elements of P are distinct, and the elements of Q are distinct, then P is a path of G .

7

Walk/Path/Trail Theorem

If u and v are distinct vertices in $G=(V, E, \Phi)$, then the following are equivalent:

- i) There is a walk from u to v
- ii) There is a trail from u to v
- iii) There is a path from u to v

Proof

8

Subgraph

Definition

- Let $G=(V, E, \Phi)$ be a graph. A graph $G'=(V', E', \Phi')$ is a subgraph of G if:
 - $V' \subseteq V$
 - $E' \subseteq E$
 - Φ' is a restriction of Φ to E' (Φ' has domain E' and $\Phi'(x) = \Phi(x) \forall x \in E'$)

Subgraph induced by V'

Subgraph induced by E'

9

Circuits and Cycles

Circuit

- Let $G = (V, E, \Phi)$ be a graph
- Let $e_1, \dots, e_n$ be a trail with vertex sequence $a_1, \dots, a_n, a_1$
- The subgraph G' of G induced by $\{e_1, \dots, e_n\}$ is a *circuit* of G (of length n)
- If the only repeated vertex in the trail is a_1 , then the circuit is called a *cycle*

Theorem: two distinct vertices, u, v are on a cycle of G iff there are ≥ 2 paths from u to v that have no vertices in common except the endpoints u and v

- Proof:*

10

Connected Graph

Definition

- Let $G=(V, E, \Phi)$ be a graph.
- If, for any two distinct vertices u and v , there is a path P from u to v , then G is a *connected graph*

Informally

- Any vertex is reachable from any other vertex

Connected components

- Given a graph G , the *connected components* are the minimal set of subgraphs, where each subgraph is connected

11

Eulerian/Hamiltonian Circuit

Let $C=(V', E', \Phi')$ be a circuit of $G=(V, E, \Phi)$

- If $E=E'$, then C is an *Eulerian circuit* of G
- If $V=V'$, then C is a *Hamiltonian circuit* of G

If a graph G has a Hamiltonian circuit, then G is a Hamiltonian graph

12

Trees

Definition

- If G is a connected graph without any cycles, then G is a *tree*

Equivalent definitions

- If G is a connected graph then the following are equivalent:
 - G is a tree
 - G has no cycles
 - For every pair of distinct vertices u, v in G , there is exactly one path from u to v
 - Removing any edge from G gives a graph which is not connected
 - $|V| = |E| + 1$

Forest: a graph whose connected components are all trees

13

Rooted Tree

Definition

- A rooted tree, T , is a pair (G, r) where G is a tree and r is a vertex of V . r is the *root* of the tree
- For every vertex, w , other than r , given a unique path from r to w : $\langle r, \dots, v_k, w \rangle$ v_k is the *parent* of w , and w is a *child* of v_k .
- Vertices that have no children are called *leaves*
- Vertices with the same parent are called *siblings*

Rooted Plane Tree (RP-tree)

- Definition: rooted tree where the children of each vertex are ordered.

Binary tree

- Each vertex has at most 2 children

14

Breadth/Depth-First Search

Breadth-First Search (BFS)

- Add root to queue
while queue is not empty
 retrieve vertex from head of queue
 print vertex
 add children (in order) to tail of queue

Depth-First Search (DFS)

- Visit(root)
- procedure Visit(vertex) *preorder traversal*
 - print vertex
 - foreach child of vertex
 - visit(child)

15

Spanning Tree

Definition

- A *spanning tree* of a simple graph $G=(V,E)$ is a subgraph $T=(V,E')$ which is a tree

Minimum Spanning Tree

- Given a connected weighted graph $G=(V,E,W)$ (where W is a function with domain E and codomain $\mathbb{R}$)
- A minimum spanning tree $T=(V,E',W')$ of G is a spanning tree whose sum of weights is no more than that of any other spanning tree of G

16

Minimum spanning Tree

Generating a minimum spanning tree for a simple graph
 $G=(V,E,W)$ (Prim's)

- Start with $E'=\{\}$
- Start with $V'=\{v\}$ for some v in V .
- While $|V'| < |V|$
 - Find the edge e from E with exactly one edge in V' of minimum weight
 - Add e to E'
 - Add the other vertex of e to V'

Alternative (Kruskal's)

- Start with $E'=\{\}$
- While $T=(V,E')$ is not connected
 - Find the cheapest edge from E that doesn't create a cycle in (V,E')
 - Add e to E'

17

Computational Tractability

Worst case running time. Obtain bound on largest possible running time of algorithm on input of a given size N , and see how this scales with N .

- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

Desirable scaling property. When the input size increases by a factor of 2, the algorithm should only slow down by some constant factor C .

There exists constants $c > 0$ and $d > 0$ such that on every input of size N , its running time is bounded by $c N^d$ steps.

Def. An algorithm is **efficient** if it has **polynomial** running time.

Justification. It really works in practice!

18

Asymptotic Order of Growth

Upper bounds. $T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $0 \leq T(n) \leq c \cdot f(n)$.

Lower bounds. $T(n)$ is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \geq c \cdot f(n)$.

Tight bounds. $T(n)$ is $\Theta(f(n))$ if $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$.

Ex: $T(n) = 32n^2 + 17n + 32$.

- $T(n)$ is $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, and $\Theta(n^2)$.
- $T(n)$ is not $O(n)$, $\Omega(n^3)$, $\Theta(n)$, or $\Theta(n^3)$.

Slight abuse of notation. $T(n) = O(f(n))$.

Vacuous statement. Any comparison-based sorting algorithm requires at least $O(n \log n)$ comparisons.

19

Properties

Transitivity. If $f = O(g)$ and $g = O(h)$ then $f = O(h)$.

Additivity. If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$.

20

Example

Prove that $f(n) = 3n^3 - 10n^2 + n - 10 = O(n^3)$

21

Example

Prove that $f(n) = 3n^3 - 10n^2 + n - 10 \neq O(n^2)$

22

Example

Prove that $f(n) = 3n^3 - 10n^2 + n - 10 = \Omega(n^2)$

23

Asymptotic Bounds for Some Common Functions

Polynomials. $a_0 + a_1n + \dots + a_dn^d$ is $\Theta(n^d)$ if $a_d > 0$.

Polynomial time. Running time is $O(n^d)$ for some constant d independent of the input size n .

Logarithms. $O(\log_a n) = O(\log_b n)$ for any constants $a, b > 0$.
can avoid specifying the base, assuming it is a constant

Logarithms. For every $x > 0$, $\log n = O(n^x)$.
log grows slower than every polynomial

Exponentials. For every $r > 1$ and every $d > 0$, $n^d = O(r^n)$.
every exponential grows faster than every polynomial

24

Linear Time: $O(n)$

Linear time. Running time is at most a constant factor times the size of the input.

Computing the maximum. Compute maximum of n numbers $a_1, \dots, a_n$.

```
max ← a1
for i = 2 to n {
  if (ai > max)
    max ← ai
}
```

25

Linearithmic Time: $O(n \log n)$

Linearithmic time. Arises in divide-and-conquer algorithms.

Sorting. Mergesort and heapsort are sorting algorithms that perform $O(n \log n)$ comparisons.

Largest empty interval. Given n time-stamps $x_1, \dots, x_n$ on which copies of a file arrive at a server, what is largest interval of time when no copies of the file arrive?

$O(n \log n)$ solution. Sort the time-stamps. Scan the sorted list in order, identifying the maximum gap between successive time-stamps.

26

Quadratic Time: $O(n^2)$

Quadratic time. Enumerate all pairs of elements.

Closest pair of points. Given a list of n points in the plane $(x_1, y_1), \dots, (x_n, y_n)$, find the pair that is closest.

$O(n^2)$ solution. Try all pairs of points.

```
min ← (x1 - x2)2 + (y1 - y2)2
for i = 1 to n {
  for j = i+1 to n {
    d ← (xi - xj)2 + (yi - yj)2
    if (d < min)
      min ← d
  }
}
```

— don't need to take square roots

Remark. $\Omega(n^2)$ seems inevitable, but this is just an illusion.

27

Cubic Time: $O(n^3)$

Cubic time. Enumerate all triples of elements.

Set disjointness. Given n sets $S_1, \dots, S_n$ each of which is a subset of $1, 2, \dots, n$, is there some pair of these which are disjoint?

$O(n^3)$ solution. For each pairs of sets, determine if they are disjoint.

```
foreach set Si {
  foreach other set Sj {
    foreach element p of Si {
      determine whether p also belongs to Sj
    }
    if (no element of Si belongs to Sj)
      report that Si and Sj are disjoint
  }
}
```

28

Polynomial Time: $O(n^k)$ Time

Independent set of size k. Given a graph, are there k nodes such that no two are joined by an edge?

$O(n^k)$ solution. Enumerate all subsets of k nodes.

```
foreach subset S of k nodes {
  check whether S is an independent set
  if (S is an independent set)
    report S is an independent set
}
```

- Check whether S is an independent set = $O(k^2)$.
- Number of k element subsets = $\binom{n}{k} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{k(k-1)(k-2)\cdots(2)(1)} \leq \frac{n^k}{k!}$
- $O(k^2 n^k / k!) = O(n^k)$.

assuming k is a constant

Exponential Time

Independent set. Given a graph, what is maximum size of an independent set?

$O(n^2 2^n)$ solution. Enumerate all subsets.

```
S* ← ∅
foreach subset S of nodes {
  check whether S is an independent set
  if (S is largest independent set seen so far)
    update S* ← S
}
```