CSE 421
Algorithms

Richard Anderson

Lecture 4

Announcements

- Homework 2, Due October 12, 1:30 pm.
- Reading
  - Chapter 3
  - Start on Chapter 4

Polynomial time efficiency

- An algorithm is efficient if it has a polynomial run time
- Run time as a function of problem size
  - Run time: count number of instructions executed on an underlying model of computation
  - T(n): maximum run time for all problems of size at most n

Polynomial Time

- Algorithms with polynomial run time have the property that increasing the problem size by a constant factor increases the run time by at most a constant factor (depending on the algorithm)

Why Polynomial Time?

- Generally, polynomial time seems to capture the algorithms which are efficient in practice
- The class of polynomial time algorithms has many good, mathematical properties

Constant factors and growth rates

- Express run time as O(f(n))
  - Ignore constant factors
- Prefer algorithms with slower growth rates
- Fundamental ideas in the study of algorithms
- Basis of Tarjan/Hopcroft Turing Award

Why ignore constant factors?

- Constant factors are arbitrary
  - Depend on the implementation
  - Depend on the details of the model
- Determining the constant factors is tedious and provides little insight

Why emphasize growth rates?

- The algorithm with the lower growth rate will be faster for all but a finite number of cases
- Performance is most important for larger problem size
- As memory prices continue to fall, bigger problem sizes become feasible
- Improving growth rate often requires new techniques

Formalizing growth rates

- T(n) is O(f(n)) [$T : Z^+ \rightarrow R^+$]
  - If sufficiently large n, T(n) is bounded by a constant multiple of f(n)
  - Exist c, n_0, such that for $n > n_0$, $T(n) < c\, f(n)$
- T(n) is O(f(n)) will be written as: T(n) = O(f(n))
  - Be careful with this notation

Prove $3n^2 + 5n + 20$ is $O(n^2)$

Choose $c = 6$, $n_0 = 5$

Lower bounds

- T(n) is $\Omega(f(n))$
  - T(n) is at least a constant multiple of f(n)
  - There exists an n_0, and $\varepsilon > 0$ such that $T(n) > \varepsilon f(n)$ for all $n > n_0$
- Warning: definitions of Ω vary
- T(n) is $\Theta(f(n))$ if T(n) is O(f(n)) and T(n) is $\Omega(f(n))$

Useful Theorems

- If $\lim (f(n) / g(n)) = c$ for $c > 0$ then $f(n) = \Theta(g(n))$
- If f(n) is O(g(n)) and g(n) is O(h(n)) then f(n) is O(h(n)))
- If f(n) is O(h(n)) and g(n) is O(h(n)) then f(n) + g(n) is O(h(n))

Ordering growth rates

- For b > 1 and x > 0
  - $\log_b n$ is $O(n^x)$
- For r > 1 and d > 0
  - n^d is $O(r^n)$

Graph Theory

Explain that there will be some review from 326

By default |V| = n and |E| = m

- G = (V, E)
  - V – vertices
  - E – edges
- Undirected graphs
  - Edges sets of two vertices {u, v}
- Directed graphs
  - Edges ordered pairs (u, v)
- Many other flavors
  - Edge / vertices weights
  - Parallel edges
  - Self loops

Definitions

- Path: $v_1, v_2, \ldots, v_k$, with (v_i, v_{i+1}) in E
  - Simple Path
  - Cycle
  - Simple Cycle
- Distance
- Connectivity
  - Undirected
  - Directed (strong connectivity)
- Trees
  - Rooted
  - Unrooted

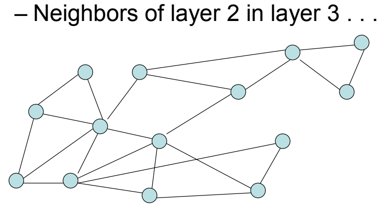
Graph search

- Find a path from s to t

```
S = {s}
While there exists (u, v) in E with u in S and v not in S
        Pred[v] = u
        Add v to S
        if (v = t) then path found
```

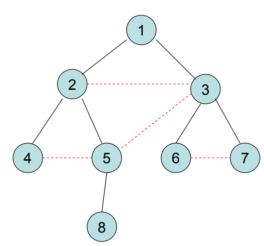
Breadth first search

- Explore vertices in layers
  - s in layer 1
  - Neighbors of s in layer 2
  - Neighbors of layer 2 in layer 3 . . .



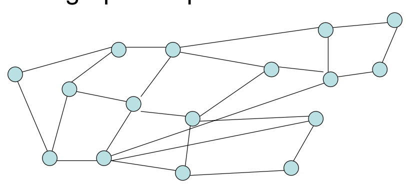
Key observation

- All edges go between vertices on the same layer or adjacent layers



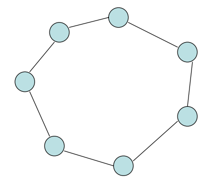
Bipartite

- A graph V is bipartite if V can be partitioned into V_1, V_2 such that all edges go between V_1 and V_2
- A graph is bipartite if it can be two colored



Testing Bipartiteness

- If a graph contains an odd cycle, it is not bipartite



Algorithm

- Run BFS
- Color odd layers red, even layers blue
- If no edges between the same layer, the graph is bipartite
- If edge between two vertices of the same layer, then there is an odd cycle, and the graph is not bipartite