CSE 417 Algorithms

Richard Anderson

Autumn 2024

Lecture 5

Announcements

- HW 1 Due tonight on Gradescope, turn in open until Sunday, 11:59 pm
  - If you do not have access to Gradescope, let me know.
- HW 2 Available

Worst Case Runtime Function

- Problem P: Given instance I compute a solution S
- A is an algorithm to solve P
- T(I) is the number of steps executed by A on instance I
- T(n) is the maximum of T(I) for all instances of size n

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
- Express run time as $T(n) = O(f(n))$

Formalizing growth rates

- $T(n)$ is $O(f(n))$ $[T : Z^+ \rightarrow R^+]$
  - If n is sufficiently large, $T(n)$ 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 $\Omega(f(n))$
  - $T(n)$ 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 $n > n_0$
- $T(n)$ is $\Theta(f(n))$ if $T(n)$ is $O(f(n))$ and $T(n)$ is $\Omega(f(n))$

Efficient Algorithms

- Polynomial Time (P): Class of all problems that can be solved with algorithms that have polynomial runtime functions
- Polynomial Time has been a very successful tool for theoretical computer science
- Problems in Polynomial Time often have practical solutions

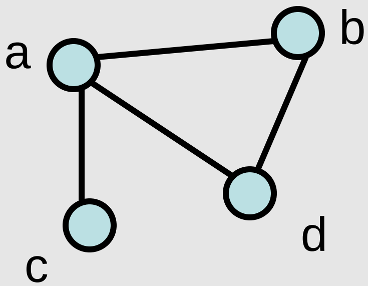
Graph Theory

- 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

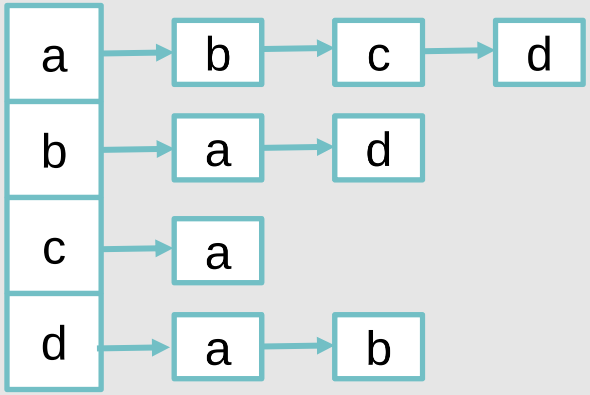
- Path: $v_1, v_2, \ldots, v_k$, with (v_i, v_{i+1}) in E
  - Simple Path
  - Cycle
  - Simple Cycle
- Neighborhood
  - N(v)
- Distance
- Connectivity
  - Undirected
  - Directed (strong connectivity)
- Trees
  - Rooted
  - Unrooted

Graph Representation



V = { a, b, c, d}

E = { {a, b}, {a, c}, {a, d}, {b, d} }



Adjacency List

	1	1	1
1		0	1
1	0		0
1	1	0	

Incidence Matrix

Implementation Issues

- Graph with n vertices, m edges
- Operations
  - Lookup edge
  - Add edge
  - Enumeration edges
  - Initialize graph
- Space requirements

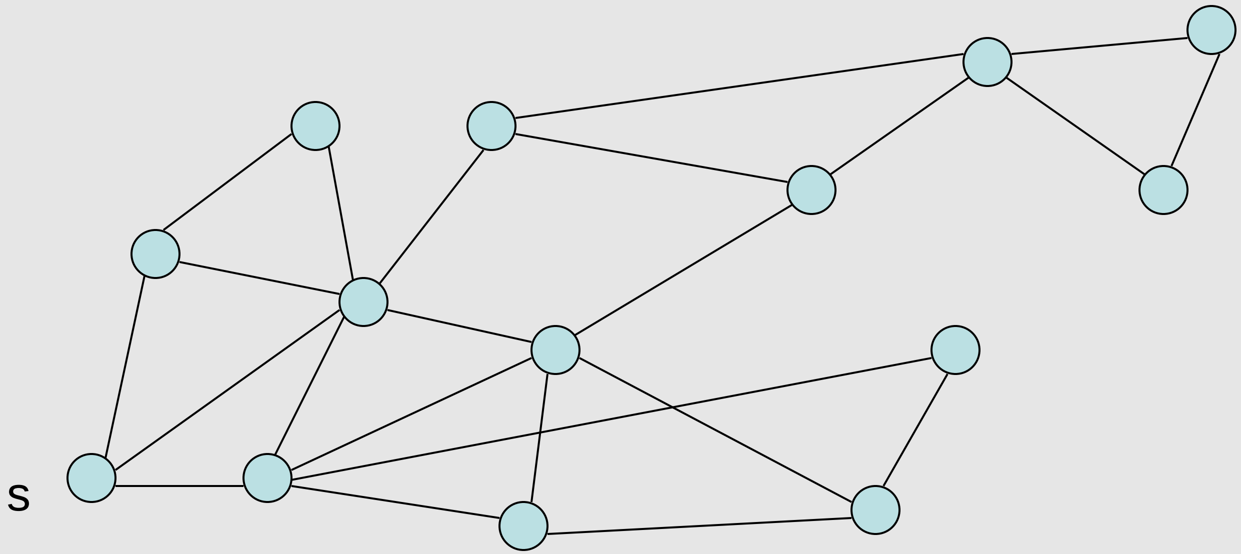
Graph search

- Find a path from s to t

```
S = {s}
while S is not empty
        u = Select(S)
        visit u
        foreach v in N(u)
                if v is unvisited
                        Add(S, v)
                        Pred[v] = u
                if (v = t) then path found
```

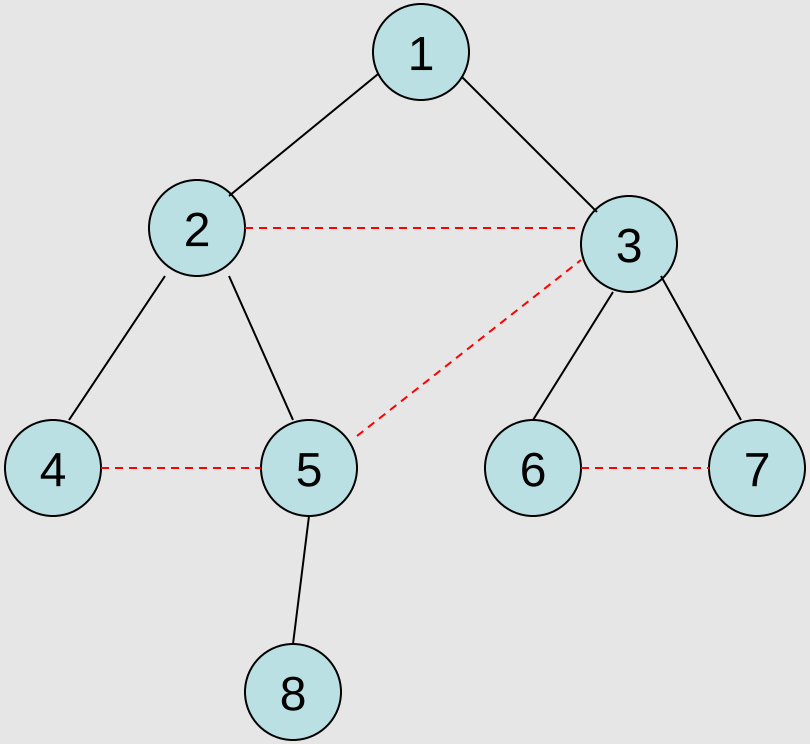
Breadth first search

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



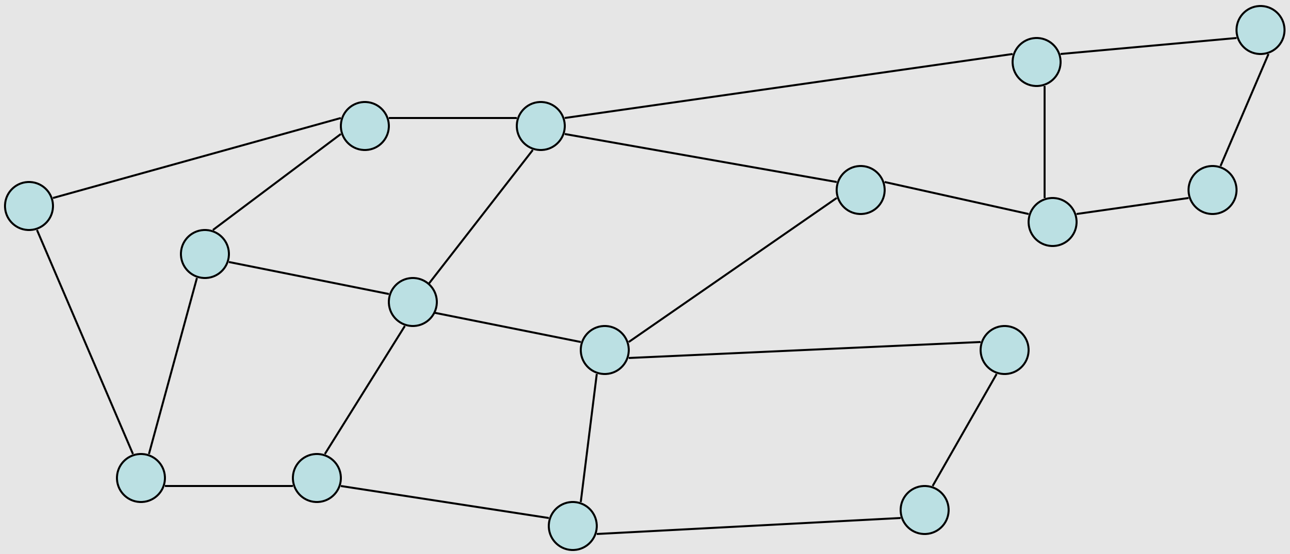
Key observation

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

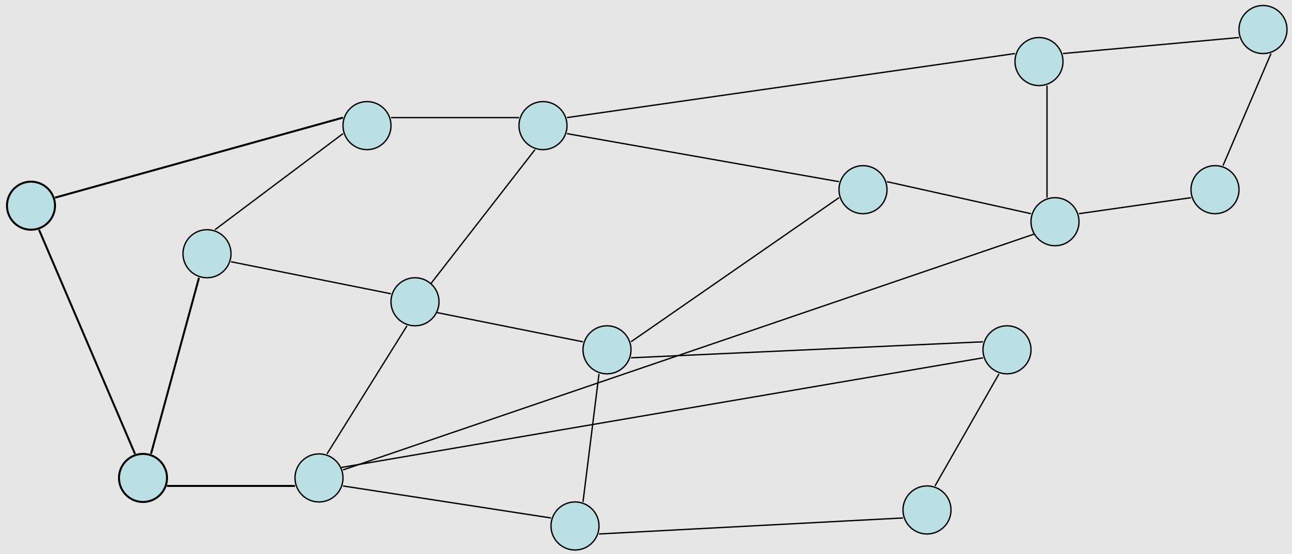


Bipartite Graphs

- 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



Can this graph be two colored?



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