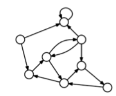
CS 340 - Analysis of Algorithms
Graph Theory Review

Dianna Xu

Graph

- A graph $G = (V, E)$ represents a set of vertices V and a set of edges E.
- E may consist of unordered or ordered pairs of vertices and the resulting graph is undirected/directed.
- Edges and vertices may have weights



2

Terminology

- Given an edge $e = (u,\ v)$, u and v are *endpoints* of e and e is *incident* on u and v. u and v are *adjacent*.
- The *degree* of a vertex $\deg(v)$ is the number of incident edges on v in an undirected graph, or the number of outgoing edges in a directed graph.
- $|V| = n$
- $|E| = m$

3

Basic Graph Size Estimates

- Undirected graph

$$|E| = 0 \le m \le \binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$$

$$\sum_{v \in V} \deg(v) = 2m$$

- Directed graph

$$|E| = 0 \le m \le n(n-1) = O(n^2)$$

$$\sum_{v \in V} \text{indeg}(v) = m$$

$$\sum_{v \in V} \text{outdeg}(v) = m$$

4

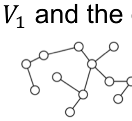
Terminology

- A *path* in a graph is a sequence of vertices $\langle v_0, \dots, v_k \rangle$ such that (v_{i-1}, v_i) is an edge for $i = 1, \dots, k$
- The *length* of a path is the number of edges, k.
- A *cycle* is a path containing at least one edge and for which $v_0 = v_k$
- A cycle is simple if its edges and vertices (except for v_0 and v_k) are distinct.

5

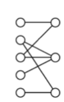
Terminology

- An *acyclic* graph contains no simple cycles
- An acyclic connected graph is a *tree*
- The vertices of a *bipartite* graph can be partitioned into two disjoint subsets, V_1 and V_2 such that all edges have one endpoint in V_1 and the other one in V_2



(Free) Tree

DAG

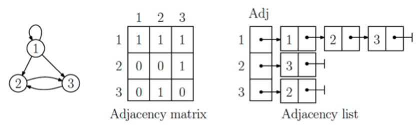


Bipartite graph

6

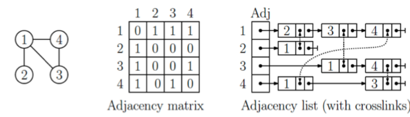
Representation

- Adjacency matrix: An $n \times n$ matrix defined for $1 \leq v, w \leq n$: $A[i,j] = 1$ if $(u,v) \in E$
- Adjacency list: An array of pointers where for $1 \leq v \leq n$, $Adj[v]$ points to a list containing the vertices that are adjacent to v

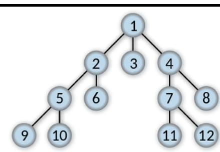


Undirected Graphs

- Adjacency matrix: store the edges twice
- Adjacency list: cross-links

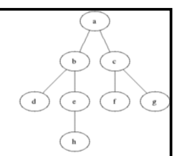


Graph Traversals: BFS



- Given a graph $G = (V, E)$, breadth-first search starts at some vertex s and visits vertices reachable from s in layers.
- Define the distance between a vertex v and s to be the minimum number of edges on a path from s to v.
- BFS visits the vertices in increasing order of distance.

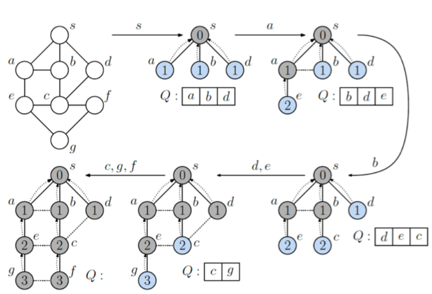
BFS



```
BFS(G, s) {
  set mark to all false
  mark[s] = true, Q = {s}
  while (Q is not empty) {
    u = dequeue of Q
    for each (v in Adj[u]) {
      if (!mark[v]){
        mark[v]=true
        append v to Q
      }
    }
  }
}
```

- `mark` array (booleans, indexed by vertices) tracks which vertices have been visited
- `Q` is a FIFO queue
- `Q` contains the frontier (discovered but unvisited) vertices

BFS on a Graph



BFS Time Analysis

```
BFS(G, s) {
  set mark to all false
  mark[s] = true, Q = {s}
  while (Q is not empty) {
    u = dequeue of Q
    for each (v in Adj[u]) {
      if (!mark[v]){
        mark[v]=true
        append v to Q
      }
    }
  }
}
```

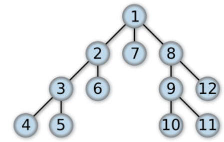
- $|V| = n, |E| = m$
- Initialization requires $O(n)$
- Traversal loop
  - while: we never visit a vertex twice
  - for each: depends on the degree of vertex
- $T(n,m) = n + \sum_{u \in V} (\deg(u) + 1)$
- $= n + \sum_{u \in V} \deg(u) + \sum_{u \in V} 1$
- $= n + \sum_{u \in V} \deg(u) + n$
- $= 2n + \sum_{u \in V} \deg(u)$
- $= 2n + 2m$
- $O(n+m)$

BFS with More Record Keeping

```
BFS(G, s) {
  for each (u in V) {
    mark[u] = false, d[u] = infinity, pred[u] = null
  }
  mark[s] = true, d[s] = 0, Q = {s}
  while (Q is not empty) {
    u = dequeue of Q
    for each (v in Adj[u]) {
      if (!mark[v]){
        mark[v] = true
        d[v] = d[u]+1
        pred[v] = u
        append v to Q
    }
  }
}
```

13

Graph Traversals: DFS

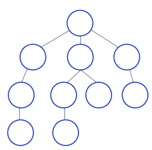


- Given a graph $G = (V, E)$, depth-first traversal strives for maximal depth and backtracks only when necessary.
- Recursive algorithm

14

DFS

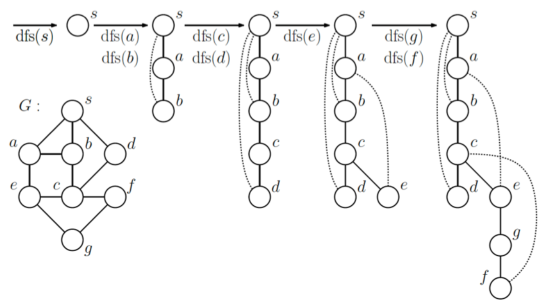
```
DFSG(G) {
  set mark to all false
  for each (v in V) {
    if (!mark(v))
      DFS(v)
  }
}
DFS(u) {
  mark[u] = true
  for each (v in Adj[u]) {
    if (!mark[v]){
      DFS(v)
    }
  }
}
```



- `mark` array used to track seen vertices
- The wrapper is only needed if not all vertices are reachable from source

15

DFS on Graph



16

Running Time

```
DFSG(G) {
  set mark to all false
  for each (v in V) {
    if (!mark(v))
      DFS(v)
  }
}
DFS(u) {
  mark[u] = true
  for each (v in Adj[u]) {
    if (!mark[v]){
      DFS(v)
    }
  }
}
```

- $|V| = n, |E| = m$
- Initialization $O(n)$
- DFS is called once per vertex (in wrapper or recursively)
- $T(n, m) = n + \sum_{u \in V}(\deg(u) + 1)$
- $= n + \sum_{u \in V} \deg(u) + \sum_{u \in V} 1$
- $= 2n + \sum_{u \in V} \deg(u)$
- $= 2n + 2m$
- $O(n + m)$

17

DFS Additional Record Keeping

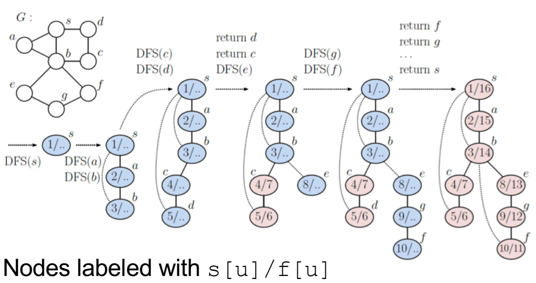
- start times array: `s`
- finish times array: `f`
- predecessors array: `pred`

```
DFSG(G) {
  time = 0
  for each (u in V) {
    mark[u] = unseen
  }
  for each (u in V) {
    if (mark[u] == unseen)
      DFS(u)
  }
}
```

```
DFS(u) {
  mark[u] = seen
  s[u] = time++
  for each (v in Adj[u]) {
    if (mark[v] == unseen){
      pred[v] = u
      DFS(v)
    }
  }
  mark[u] = finished
  f[u] = time++
}
```

18

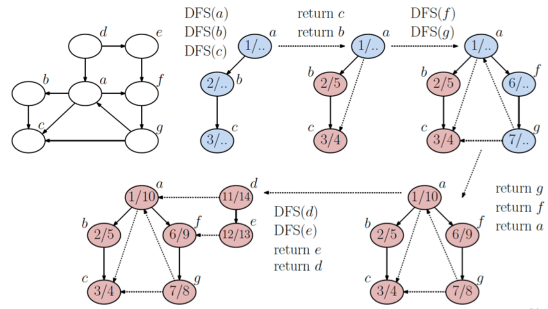
DFS on Undirected Graph



Nodes labeled with `s[u]/f[u]`

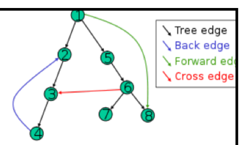
19

DFS on Directed Graph



20

DFS Edge Classification



- If v is visited for the first time as we traverse (u,v), then (u,v) is a tree edge
- else, v has already been visited
  - if v is an ancestor of u, (u,v) is a back edge
  - if v is a descendent of u, then (u,v) is a forward edge
  - if v is neither, then (u,v) is a cross edge

21

CS 340 - Analysis of Algorithms Greedy Algorithms – Interval Scheduling

Dianna Xu

Optimization Problems

- Arise naturally in many applications of science and engineering
- Problem is subject to various constraints
- Want to minimize cost or maximize objective
- Efficient solutions are not a given
- Optimality also a concern

23

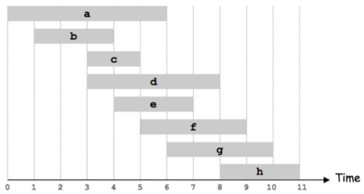
Greedy Algorithms

- An algorithm that builds up a solution by "myopically" selecting the best choice at the moment
- Greedy algorithms don't always produce optimal solutions
- Even when they don't, they provide fast heuristics that gives us good approximations

24

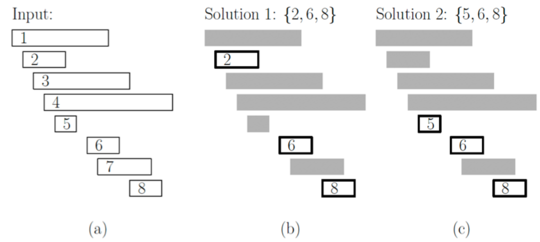
Interval Scheduling

- Given a set R of n activities with start-finish times $[s_i, f_i], 1 \leq i \leq n$, determine a maximum subset of R consisting of compatible requests



25

Example



26

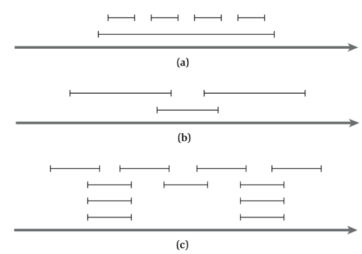
Greedy Design Basics

- For each request, use a simple rule to decide if it should be accepted.
- Once accepted, it can not be rescinded (greedy does not backtrack).
- What criteria should we use here?

27

Approaches

- Earliest Activity First
- Shortest Activity First
- Lowest Conflict Activity First



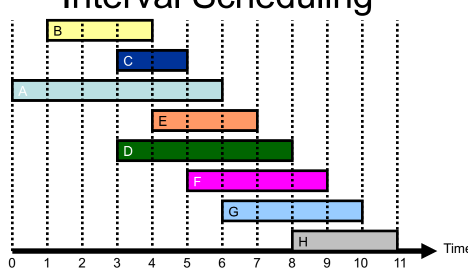
28

Earliest Finish First

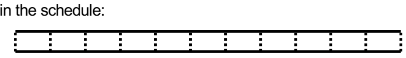
```
greedySchedule(R) { // R the set of requests
  A = empty // A the set of scheduled activities
  while (R is nonempty) {
    r = request in R with the smallest finish time
    append r to A
    delete from R all requests that overlap r
  }
  return A
}
```

29

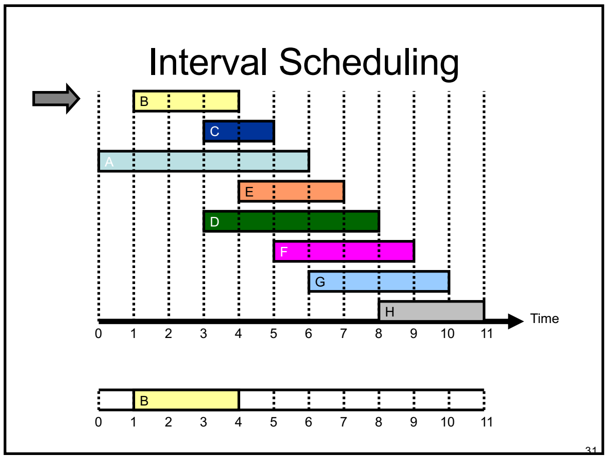
Interval Scheduling

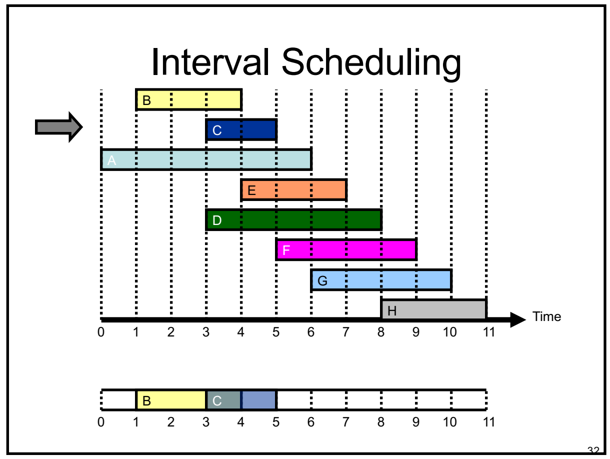


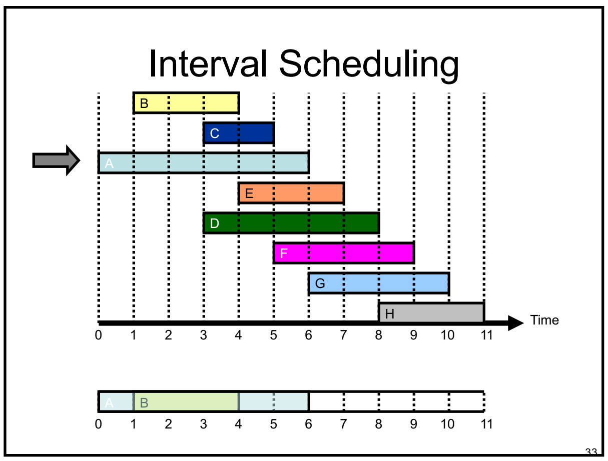
Fill in the schedule:

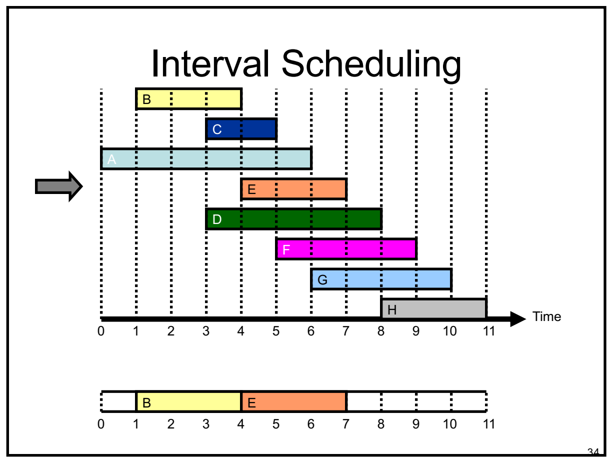


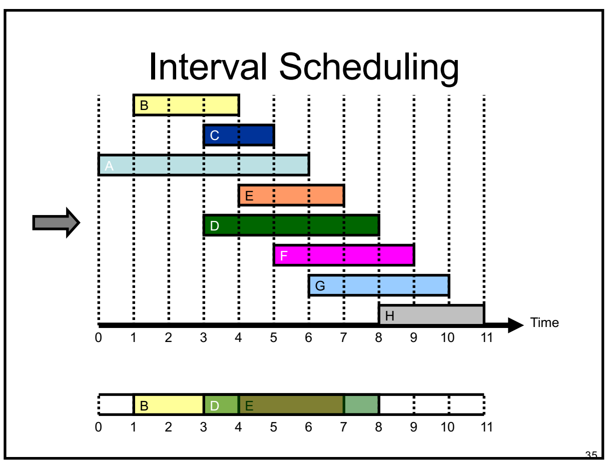
30

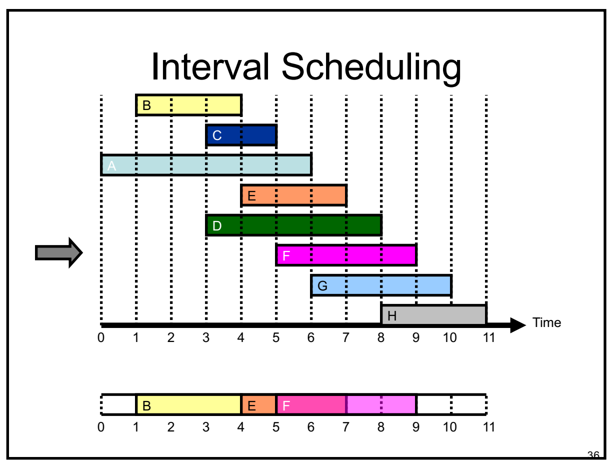

Interval Scheduling
B
C
A
E
D
F
G
H
Time
0 1 2 3 4 5 6 7 8 9 10 11
B
0 1 2 3 4 5 6 7 8 9 10 11


Interval Scheduling
B
C
A
E
D
F
G
H
Time
0 1 2 3 4 5 6 7 8 9 10 11
B C
0 1 2 3 4 5 6 7 8 9 10 11

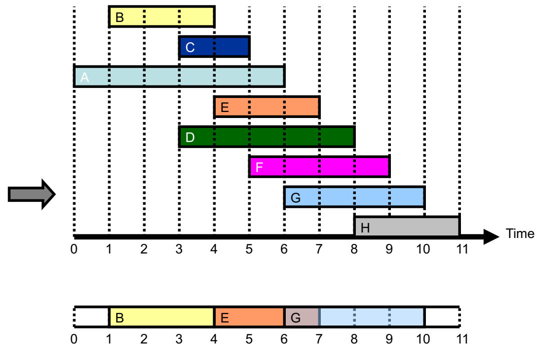

Interval Scheduling
B
C
A
E
D
F
G
H
Time
0 1 2 3 4 5 6 7 8 9 10 11
A B
0 1 2 3 4 5 6 7 8 9 10 11


Interval Scheduling
B
C
A
E
D
F
G
H
Time
0 1 2 3 4 5 6 7 8 9 10 11
B E
0 1 2 3 4 5 6 7 8 9 10 11


Interval Scheduling
B
C
A
E
D
F
G
H
Time
0 1 2 3 4 5 6 7 8 9 10 11
B D E
0 1 2 3 4 5 6 7 8 9 10 11

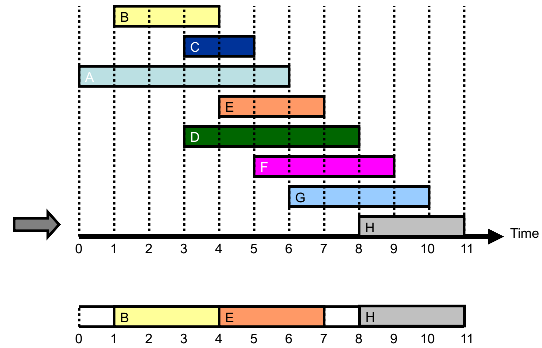

Interval Scheduling
B
C
A
E
D
F
G
H
Time
0 1 2 3 4 5 6 7 8 9 10 11
B E F
0 1 2 3 4 5 6 7 8 9 10 11

Interval Scheduling



37

Interval Scheduling



38

Earliest Finish First

```
greedySchedule(R) { // R the set of requests
  A = empty     // A the set of scheduled activities
  sort R by finish times
  prevA = null  // last picked activity
  for (each r in R) {
    if (r doesn't conflict with prevA) {
      append r to A
      prevA = r
    }
  }
  return A
}
```

39

Time Analysis

- $|R| = n$
- Naïve implementation
  - `while` runs $O(n)$
  - `findmin, delete overlap` each runs $O(n)$
  - $O(n^2)$
- Sort R by finish time first – $O(nlogn)$
  - `while` changes to `for` and runs $O(n)$
  - `findmin/delete` not needed any more
  - $O(nlogn + n) = O(nlogn)$

40

Correctness

- Valid schedule?
- Optimality?
  - maximizes the solution cardinality, i.e. does it schedule the max number of activities?

41

Proof

- Consider any optimal schedule O and let G be the schedule produced by greedy
- If $O = G$ then we are done
- Otherwise, show we can construct a schedule O' that is more similar to G than it is to O, and keep going so that O converges to G.

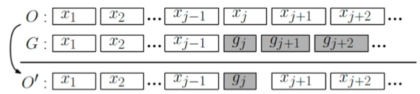
42

Proof

- Order the activities in the schedules in increasing finishing times.
- Let $O = \langle x_1, x_2, \dots, x_k \rangle$
- Since O and G differ, we have:
  - $O = \langle x_1, \dots, x_{j-1}, x_j, \dots \rangle$
  - $G = \langle x_1, \dots, x_{j-1}, g_j, \dots \rangle$, where $g_j \neq x_j$
  - Note $k \geq j$ (why?)
  - g_j has earlier finish time than x_j
  - replace x_j with g_j in O, resulting in O'

43

Proof



- O' is valid
  - g_j does not conflict with earlier activities
  - or later activities
- and optimal (same cardinality)
- Keep doing it until O' becomes G
- What if $|O| > |G|$?

44

Lemma: Greedy Has Earlier Finish Times

- Given $O = \langle [o_{s_1}, o_{f_1}], \dots, [o_{s_i}, o_{f_i}], \dots \rangle$ and $G = \langle [g_{s_1}, g_{f_1}], \dots, [o_{s_i}, o_{f_i}], \dots \rangle$,
- Claim: $g_{f_i} \leq O_{f_i}\ \forall i$
- Proof by induction
  - base case $i = 1$: by greedy construction
  - inductive hypothesis: $g_{f_{i-1}} \leq O_{f_{i-1}}$
  - inductive step: $o_{s_i} > g_{f_{i-1}}$, thus job o_i has no conflict with $g_{i-1} \rightarrow o_i$ was in the pool Greedy considered but didn't pick $\rightarrow g_{f_i} \leq o_{f_i}$
- $|O| = |G|$

45