Lecture 14

CSE 331

Sep 27, 2019

If you need it, ask for help

Mini Project group due Monday!

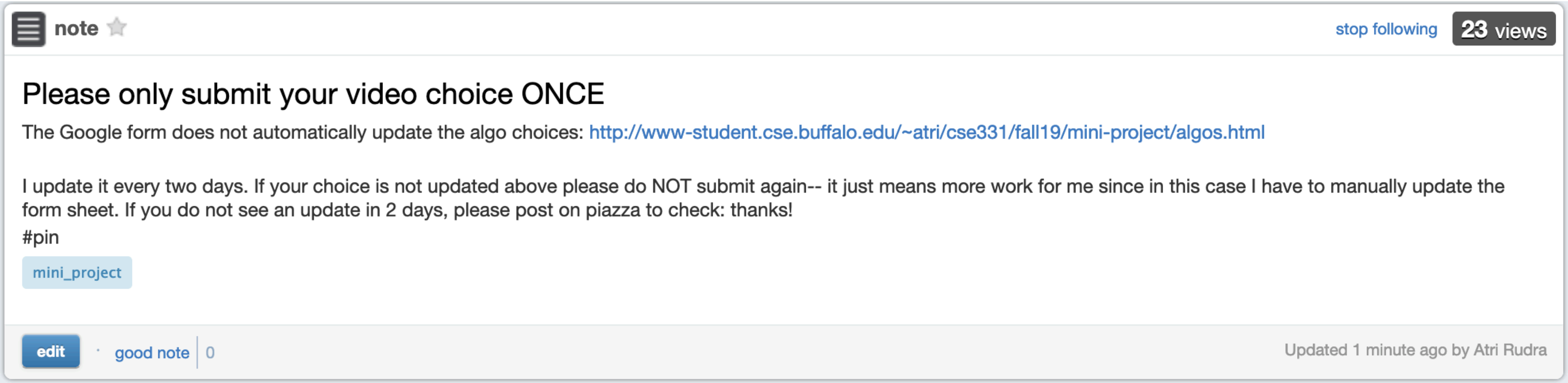
CSE 331 Mini project choices

Fall 2019

Please check the table below before submitting your mini project team composition to make sure your case study is not being used by another group. Case studies are assigned on a first come first serve basis.

Group	Chosen Algorithm	Case Study	Links
Daniel Shekhtman, William Nicholson, Andrew Quinonez (D's Get Degrees)	PageRank	Manipulation of PageRank for nefarious purposes	Link 1, Link 2, Link 3, Link 4
Jordan Clemons, Chris Burton, Christopher Perez (Group 1)	Pagerank	Google's use of Pagerank in sorting search results	Link 1, Link 2
Moulid Ahmed, Shrishty Shivani Jha, Shreya Lakhkar (ACE-MA)	Spotify Recommendation	Machine Learning Algorithm	Link 1, Link 2, Link 3
Justin Henderson, Hannah Wlasowicz, Judy Mei (PizzaTime)	Aes 256	ransomware	Link 1
Gillian Marcus, Jason Niu, Sharon Stack (2n^2 (//pls substitute caret for a superscript))	Deep Neural Networks for YT Recommendations	Social Media Targeted Advertising	Link 1, Link 2, Link 3, Link 4

The chosen list updates ~2 days


note

stop following 23 views

Please only submit your video choice ONCE

The Google form does not automatically update the algo choices: http://www-student.cse.buffalo.edu/~atri/cse331/fall19/mini-project/algos.html

I update it every two days. If your choice is not updated above please do NOT submit again-- it just means more work for me since in this case I have to manually update the form sheet. If you do not see an update in 2 days, please post on piazza to check: thanks!
#pin

mini_project

edit · good note | 0

Updated 1 minute ago by Atri Rudra

HW 4 out

Homework 4

Due by **11:00am, Friday, October 4, 2019**.

Make sure you follow all the homework policies.

All submissions should be done via Autolab.

Sample Problem

The Problem

Extend the topological ordering algorithm topological ordering care package so that, given an input directed graph G, it outputs one of two things: (a) a topological ordering, thus establishing that G is a DAG, or (b) a cycle in G, thus establishing that G is not a DAG.

The running time of your algorithm should be $O(m + n)$ for a directed graph with n nodes and m edges.

Click here for the Solution

Submission

You will **NOT** submit this question. This is for you to get into thinking more about designing algorithms on graphs.

HW 3 Solutions

At the end of the lecture

Graded HW 2

Hopefully by tonight

Questions?

Breadth First Search (BFS)

Build layers of vertices connected to s

$L_0 = \{s\}$

Assume $L_0,..,L_j$ have been constructed

L_{j+1} set of vertices not chosen yet but are connected to L_j

Stop when new layer is empty

Use linked lists

Use CC[v] array

Rest of Today’s agenda

Quick run time analysis for BFS

Quick run time analysis for DFS (and Queue version of BFS)

Helping you schedule your activities for the day

O(m+n) BFS Implementation

BFS(s)

Array

Input graph as Adjacency list

CC[s] = T and CC[w] = F for every $w \neq s$

Set $i = 0$

Set $L_0 = \{s\}$

While L_i is not empty

Linked List

$L_{i+1} = \emptyset$

For every u in L_i

For every edge (u,w)

If CC[w] = F then

CC[w] = T

Add w to L_{i+1}

i++

Version in KT also computes a BFS tree

All the layers as one

BFS(s)

$CC[s] = T$ and $CC[w] = F$ for every $w \neq s$

Set $i = 0$

Set $L_0 = \{s\}$

While L_i is not empty

$L_{i+1} = \emptyset$

For every u in L_i

For every edge (u,w)

If $CC[w] = F$ then

$CC[w] = T$

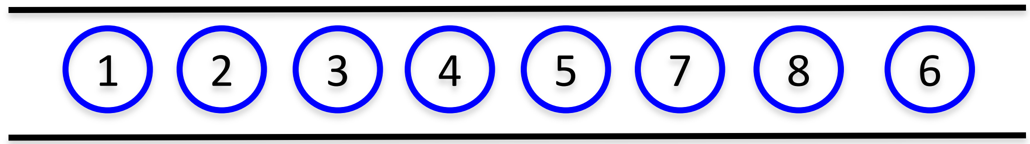
Add w to L_{i+1}

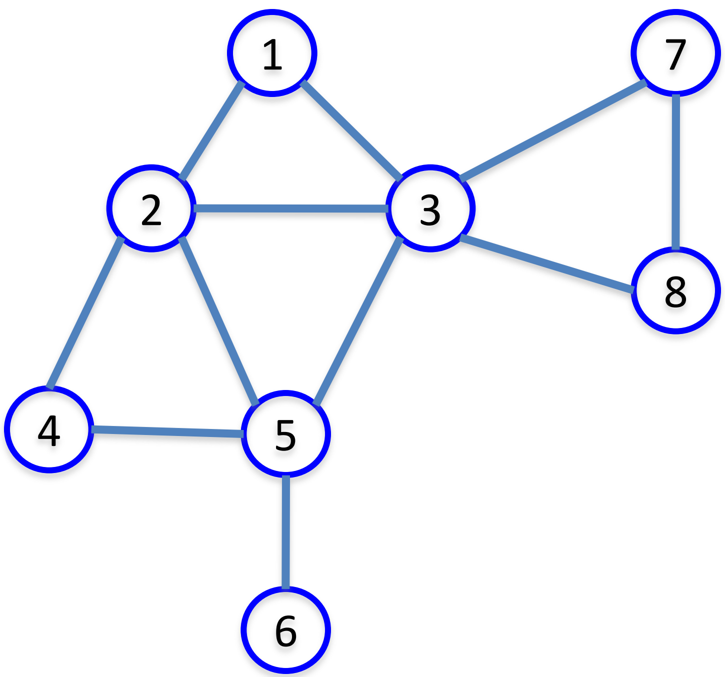
i++

All layers are considered in first-in-first-out order

Can combine all layers into one queue: all the children of a node are added to the end of the queue

An illustration





Queue O(m+n) implementation

BFS(s)

CC[s] = T and CC[w] = F for every w≠ s — O(n)

Intitialize Q= {s} — O(1)

While Q is not empty

    Delete the front element u in Q — O(1)

        For every edge (u,w) — O(1)

            If CC[w] = F then — $O(n_u)$

                CC[w] = T — O(1)

                Add w to the back of Q

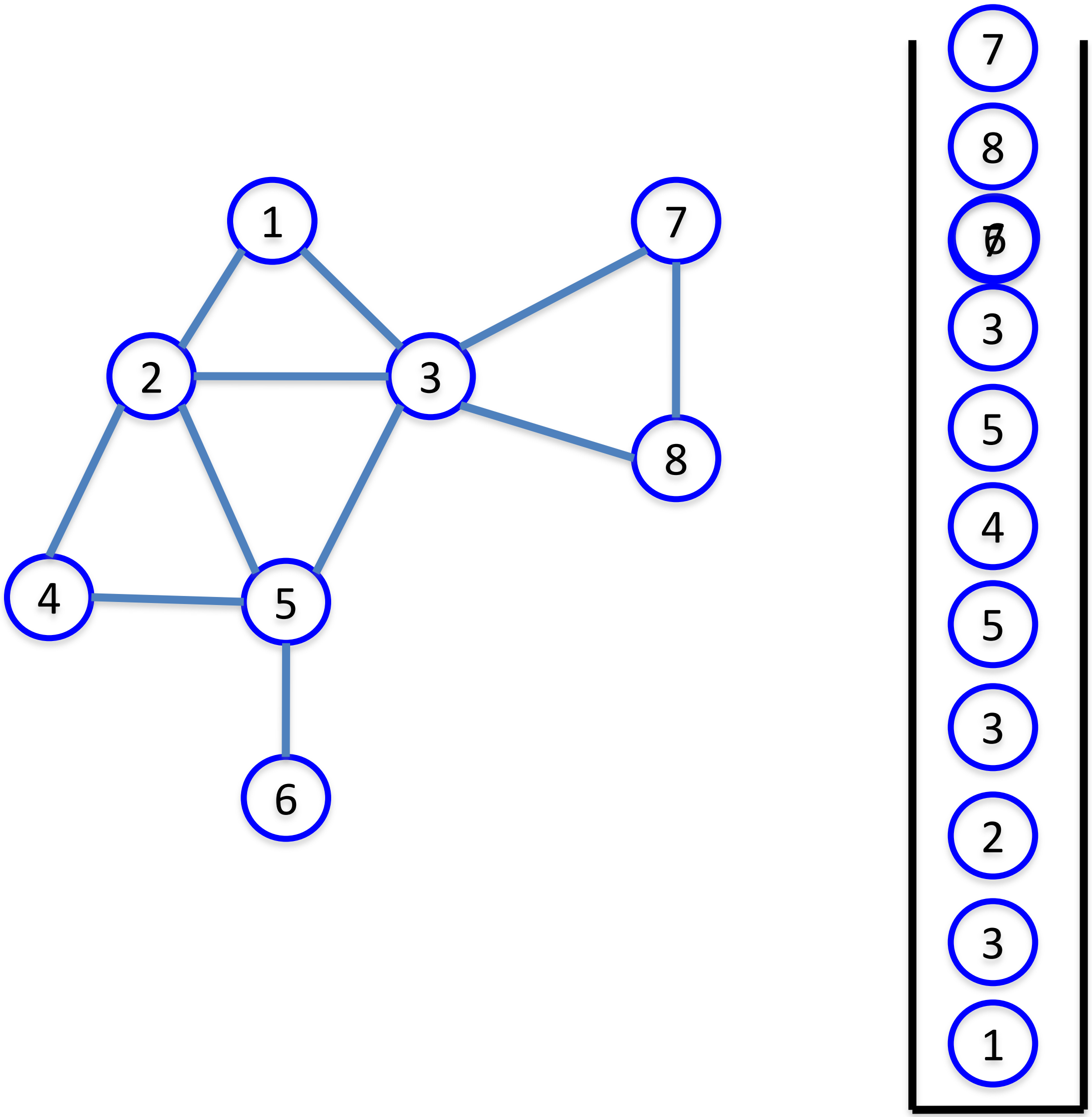
$\Sigma_u O(n_u) = O(\Sigma_u n_u) = O(m)$

Questions?

Implementing DFS in $O(m+n)$ time

Same as BFS except stack instead of a queue

A DFS run using an explicit stack



DFS stack implementation

DFS(s)

CC[s] = T and CC[w] = F for every w≠ s

Intitialize Ŝ = {s}

While Ŝ is not empty

Pop the top element u in Ŝ

For every edge (u,w)

If CC[w] = F then

CC[w] = T

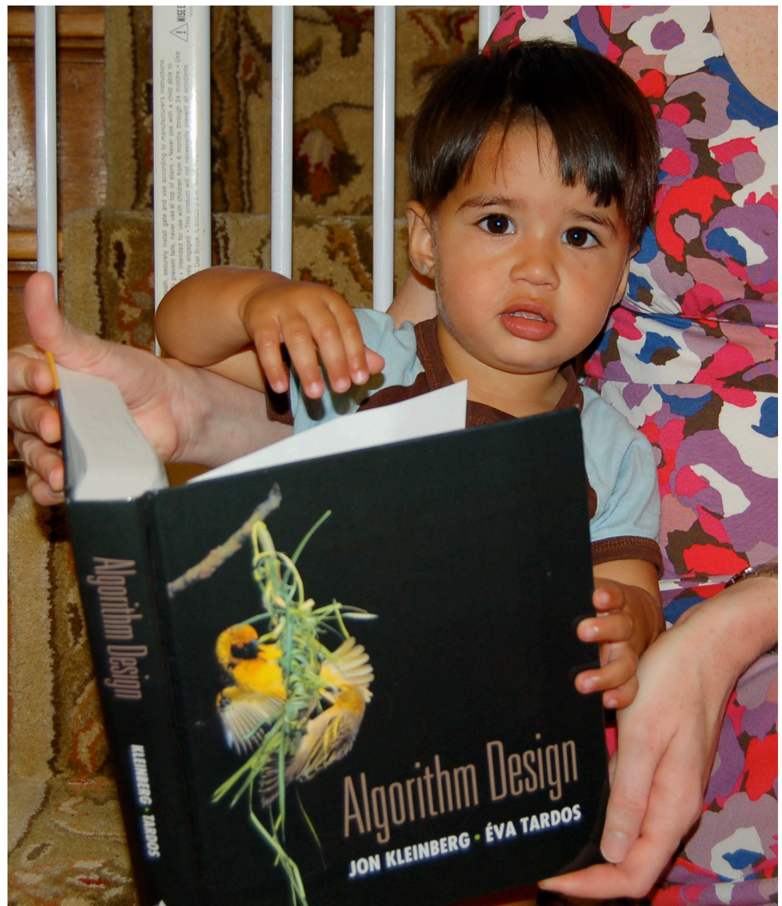
Push w to the top of Ŝ

Same O(m+n) run time analysis as for BFS

Questions?

Reading Assignment

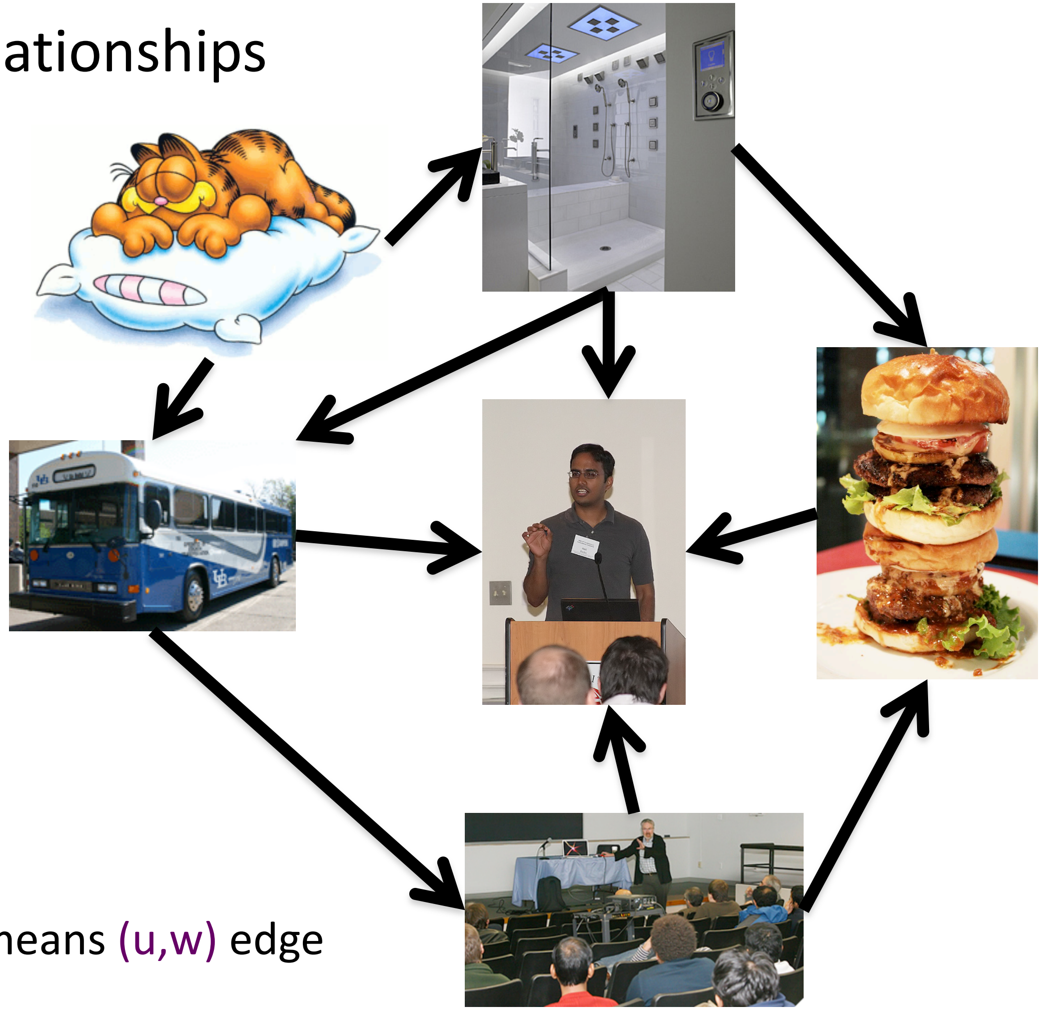
Sec 3.3, 3.4, 3.5 and 3.6 of [KT]



Directed graphs

Model asymmetric relationships

Precedence relationships

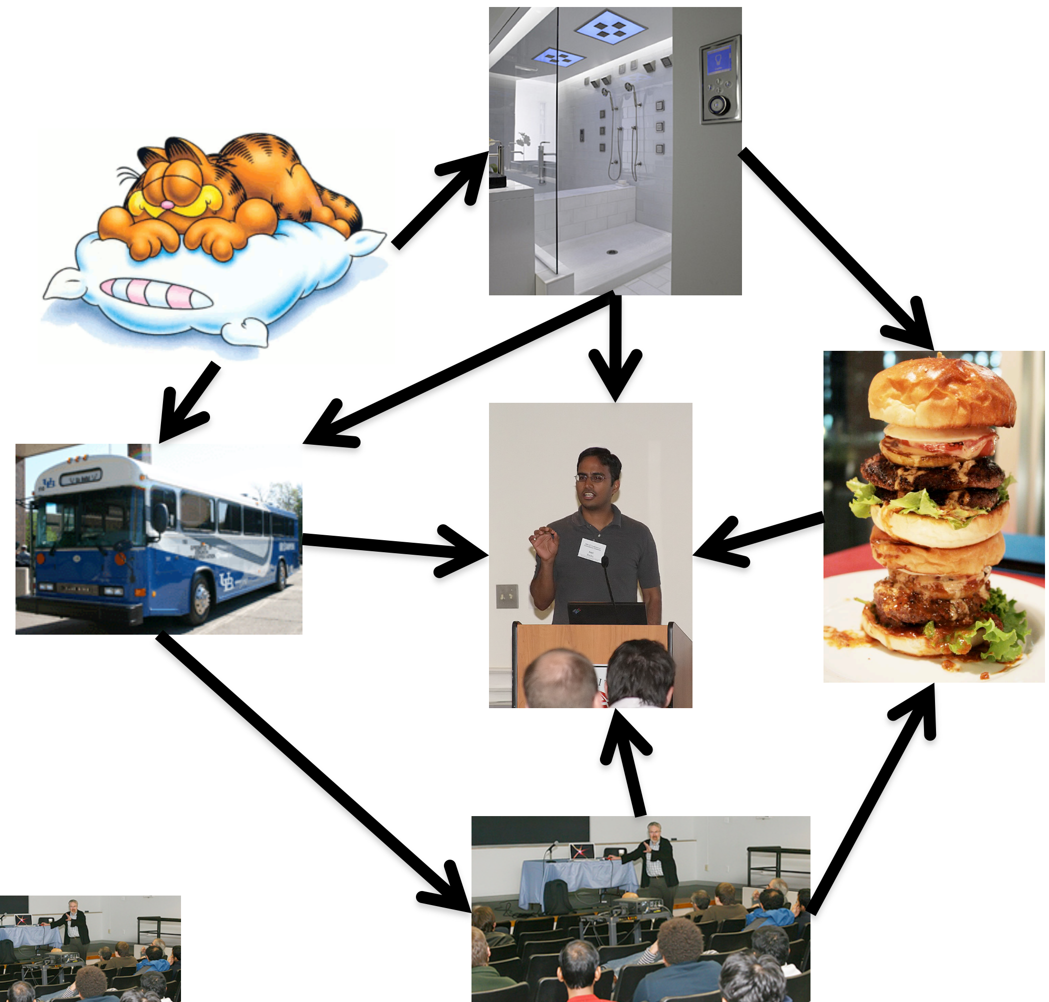


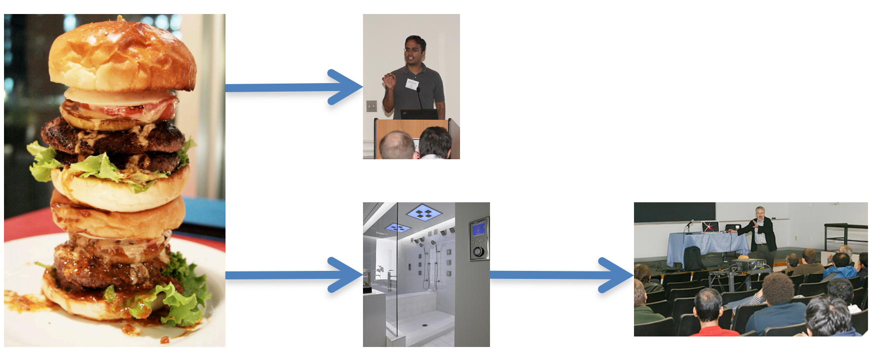
u needs to be done before w means (u,w) edge

Directed graphs

Adjacency matrix is not symmetric

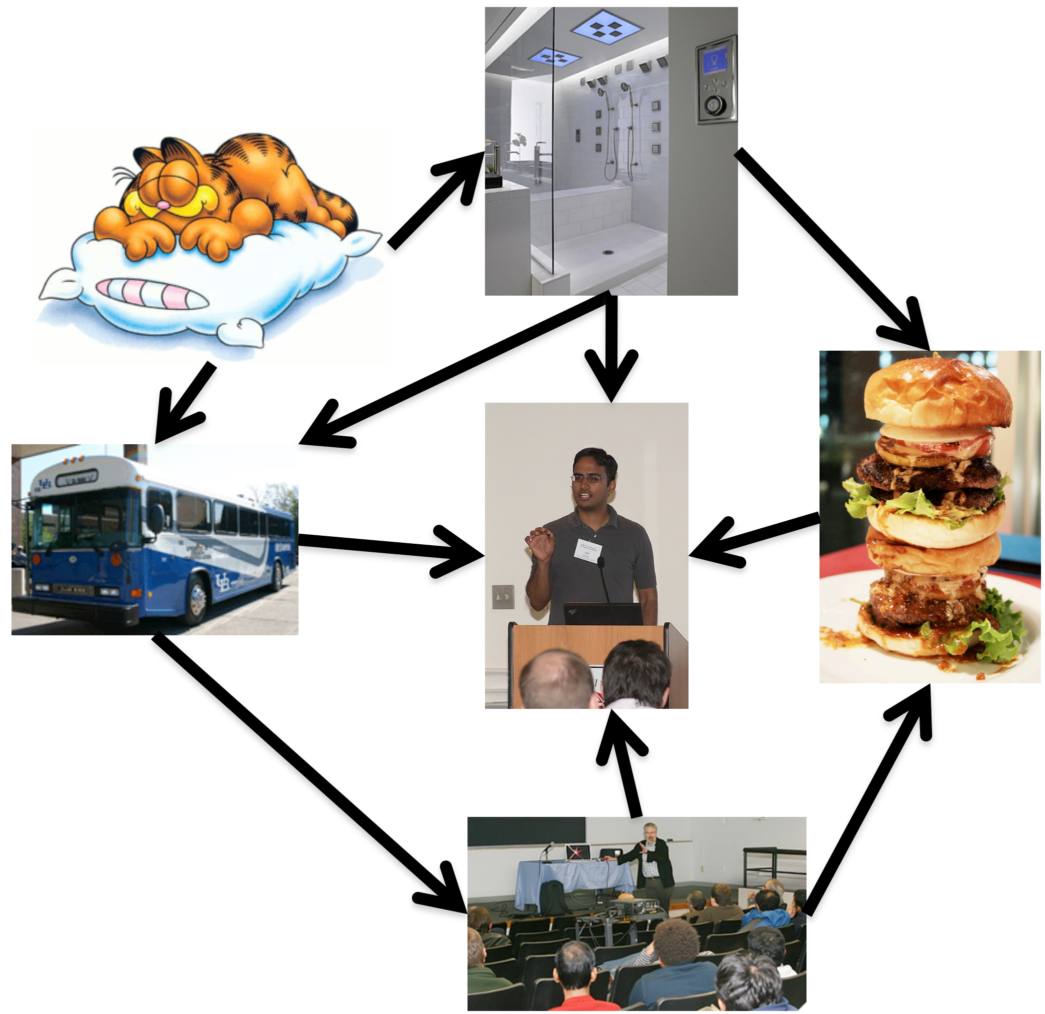
Each vertex has two lists in Adj. list rep.





Directed Acyclic Graph (DAG)

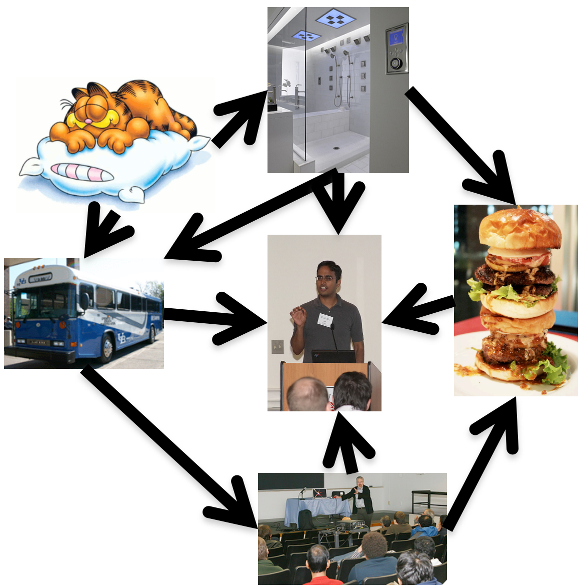
No directed cycles

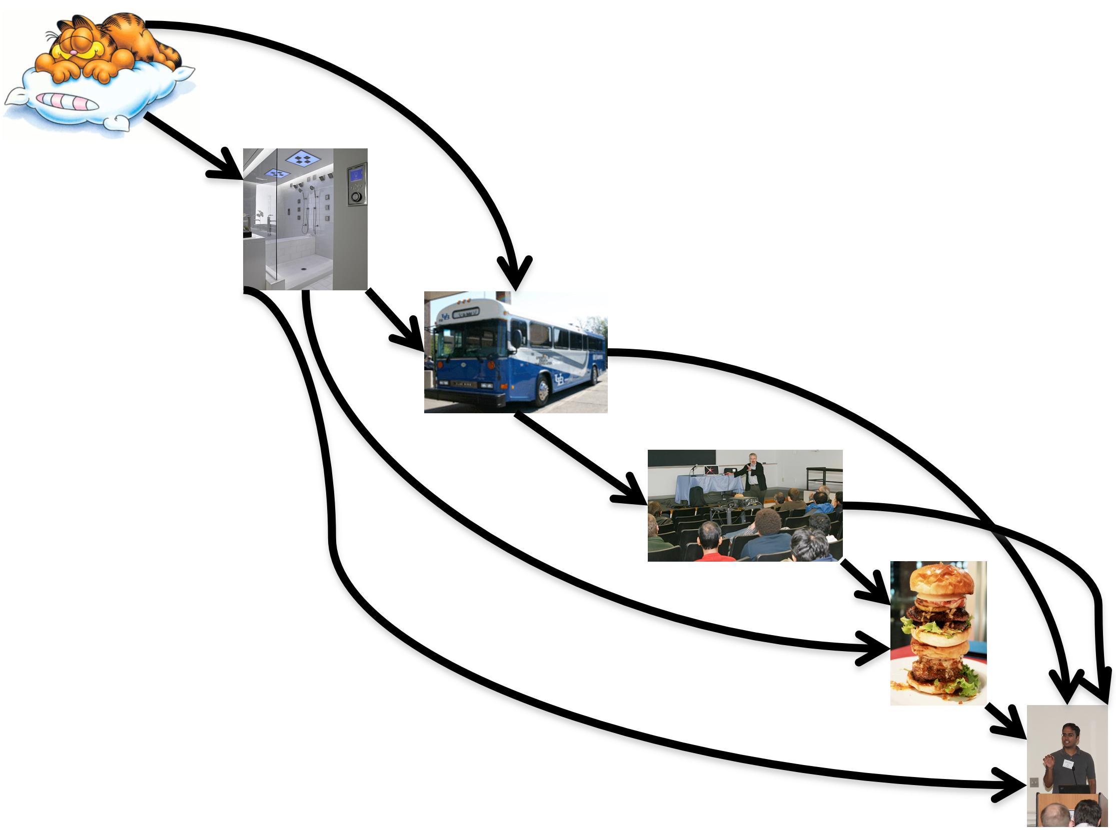


Precedence relationships are consistent

Topological Sorting of a DAG

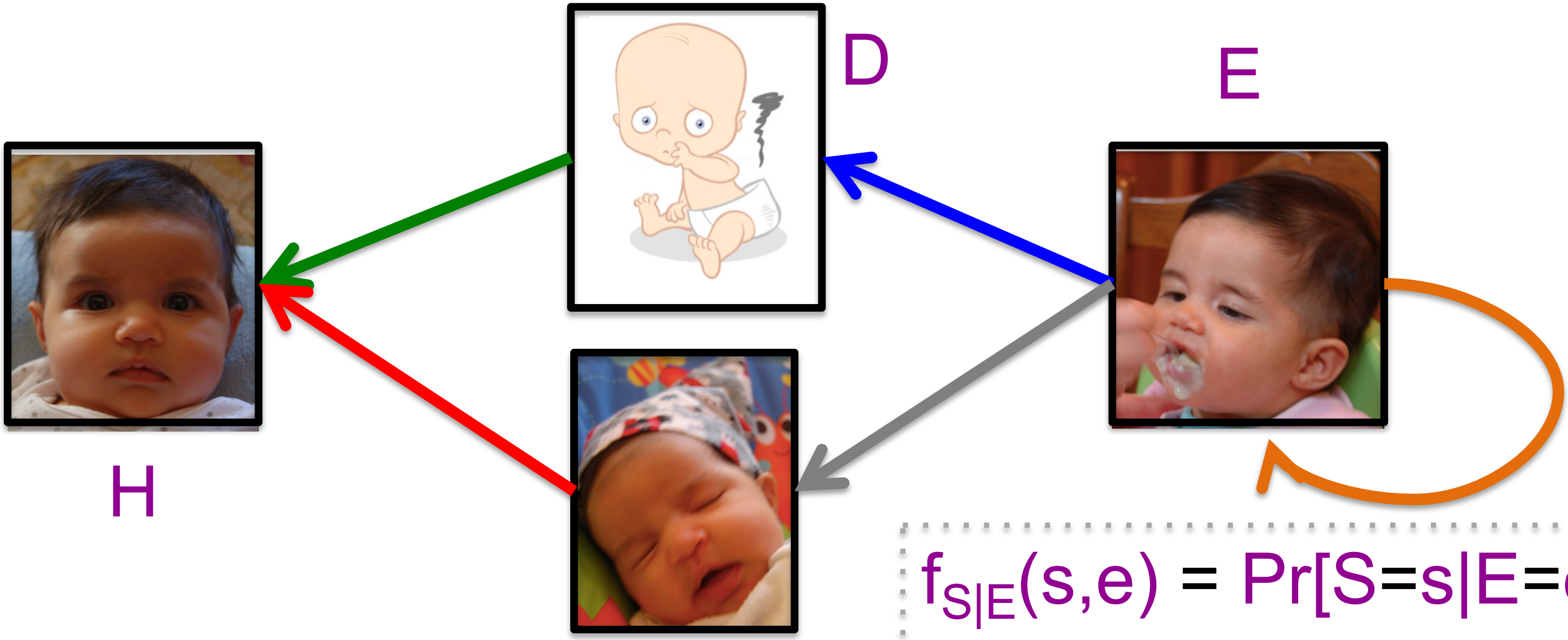
Order the vertices so that all edges go "forward"





Probabilistic Graphical Models (PGMs)

http://ginaskokopelli.com/wp-content/uploads/2013/01/DiaperDealsLogo.jpg



$f_{S|E}(s,e) = \Pr[S{=}s|E{=}e]$

S	E	$f_{S\|E}$
1	1	0.8
1	0	0.3
0	1	0.2
0	0	0.7

$$\varphi(h) = \sum_{d,s,e} f_{H|D,S}(h,d,s) \times f_{S|E}(s,e) \times f_{D|E}(d,e) \times f_E(e)$$

More details on Topological sort

Topological Ordering

This page collects material from previous incarnations of CSE 331 on topological ordering.

Where does the textbook talk about this?

`Section 3.6` in the textbook has the lowdown on topological ordering.

Fall 2018 material

First lecture

Here is the lecture video:

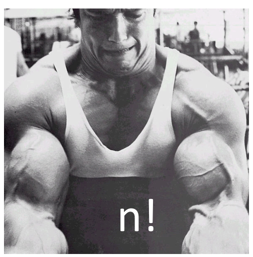
Questions?

Main Steps in Algorithm Design

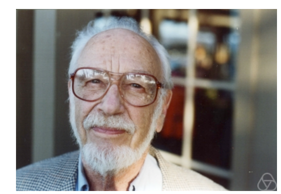
Problem Statement

NRMP
National Resident Matching Program

Problem Definition



Algorithm



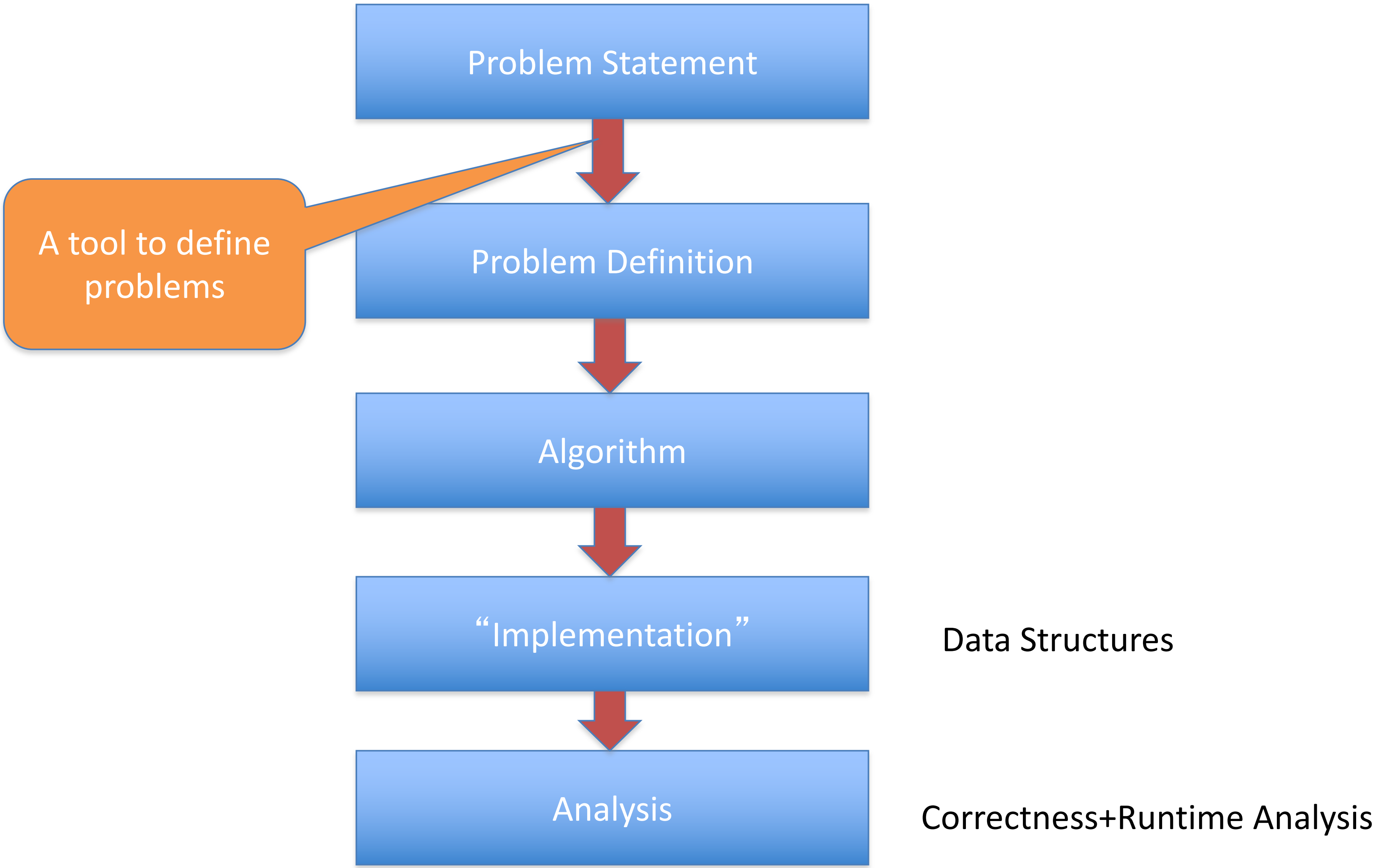
"Implementation"

Data Structures

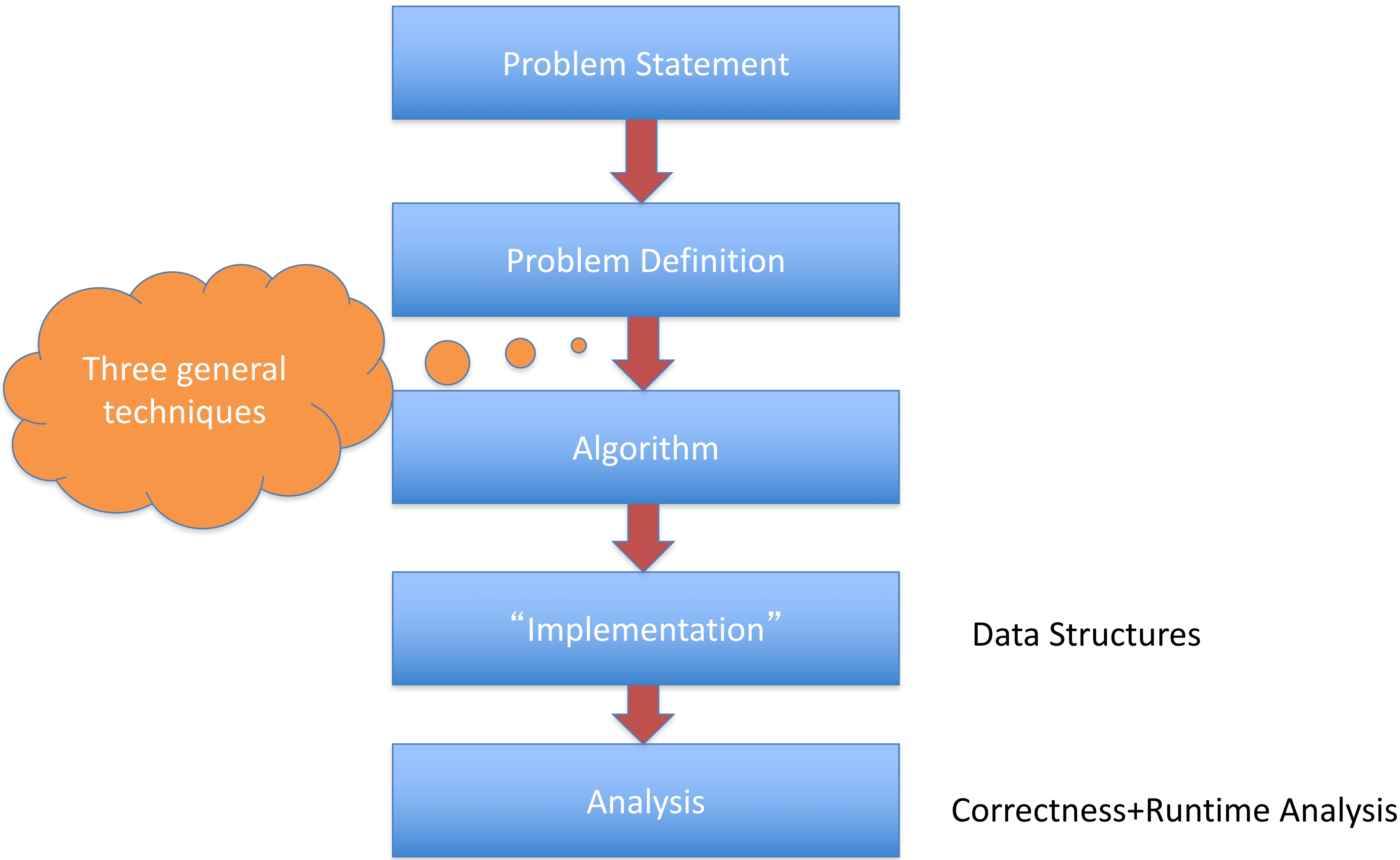
Analysis

Correctness+Runtime Analysis

Where do graphs fit in?

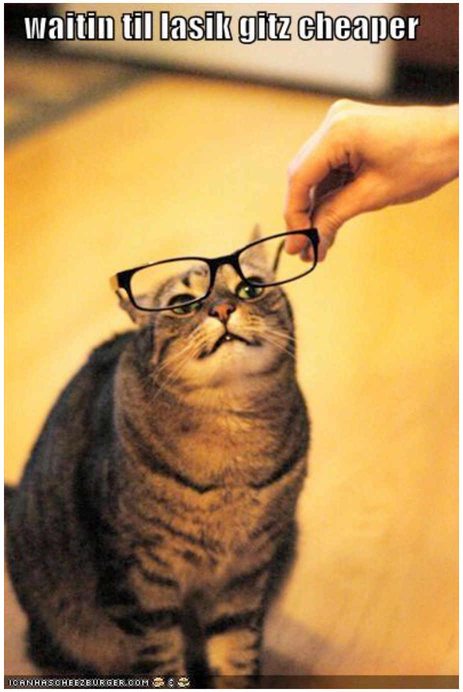


Rest of the course*



Greedy algorithms

Build the final solution piece by piece



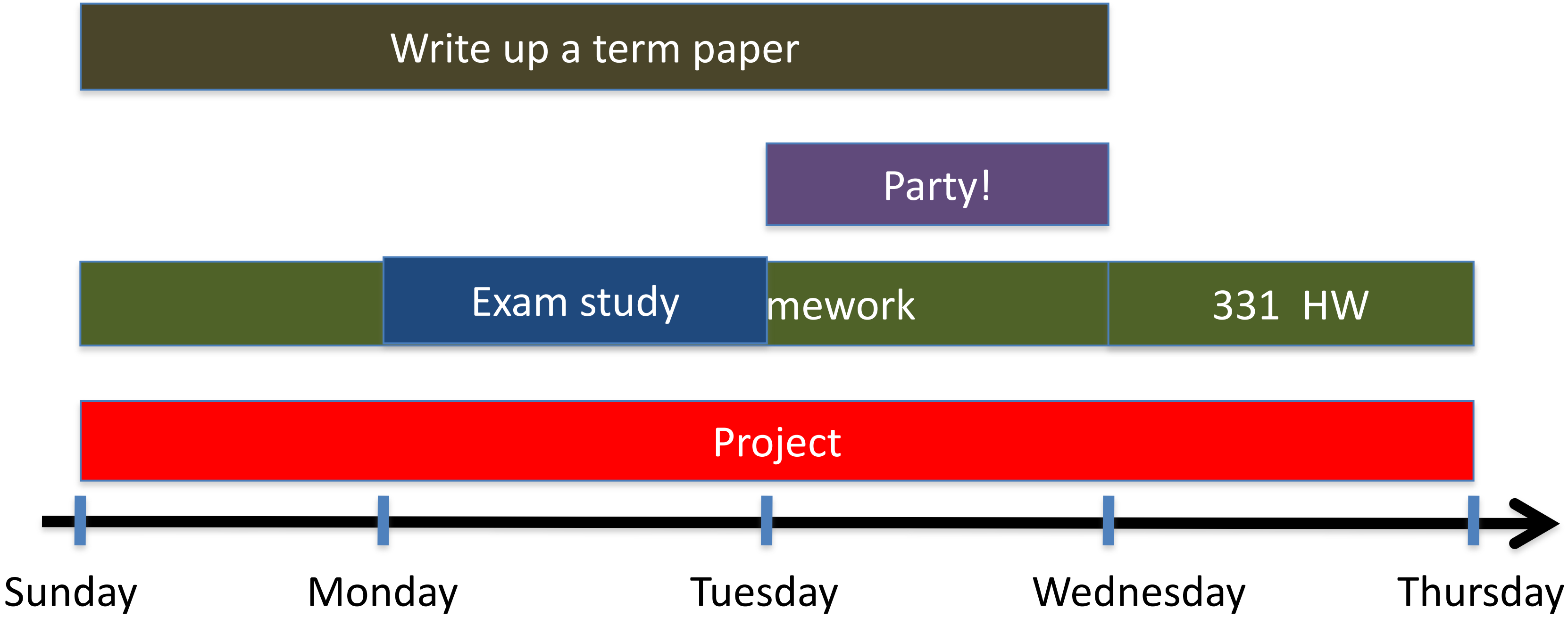
Being short sighted on each piece

Never undo a decision

Know when you see it

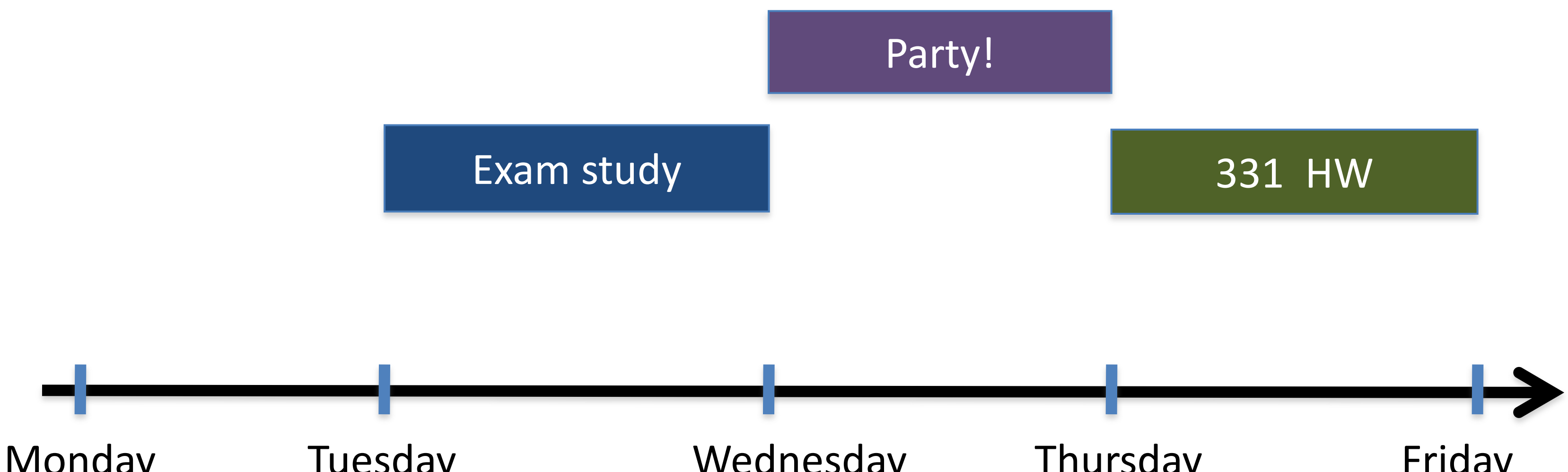
End of Semester blues

Can only do one thing at any day: what is the maximum number of tasks that you can do?



The optimal solution

Can only do one thing at any day: what is the maximum number of tasks that you can do?



Interval Scheduling Problem

Input: n intervals $[s(i), f(i))$ for $1 \leq i \leq n$

$\{ s(i), \ldots, f(i)-1 \}$

Output: A *schedule* S of the n intervals

No two intervals in S conflict

$|S|$ is maximized

Algorithm with examples

Interval Scheduling via examples

In which we derive an algorithm that solves the Interval Scheduling problem via a sequence of examples.

The problem

In these notes we will solve the following problem:

Interval Scheduling Problem

`Input:` An input of n intervals $[s(i), f(i))$, or in other words, $\{s(i), \ldots, f(i) - 1\}$ for $1 \leq i \leq n$ where i represents the intervals, $s(i)$ represents the start time, and $f(i)$ represents the finish time.

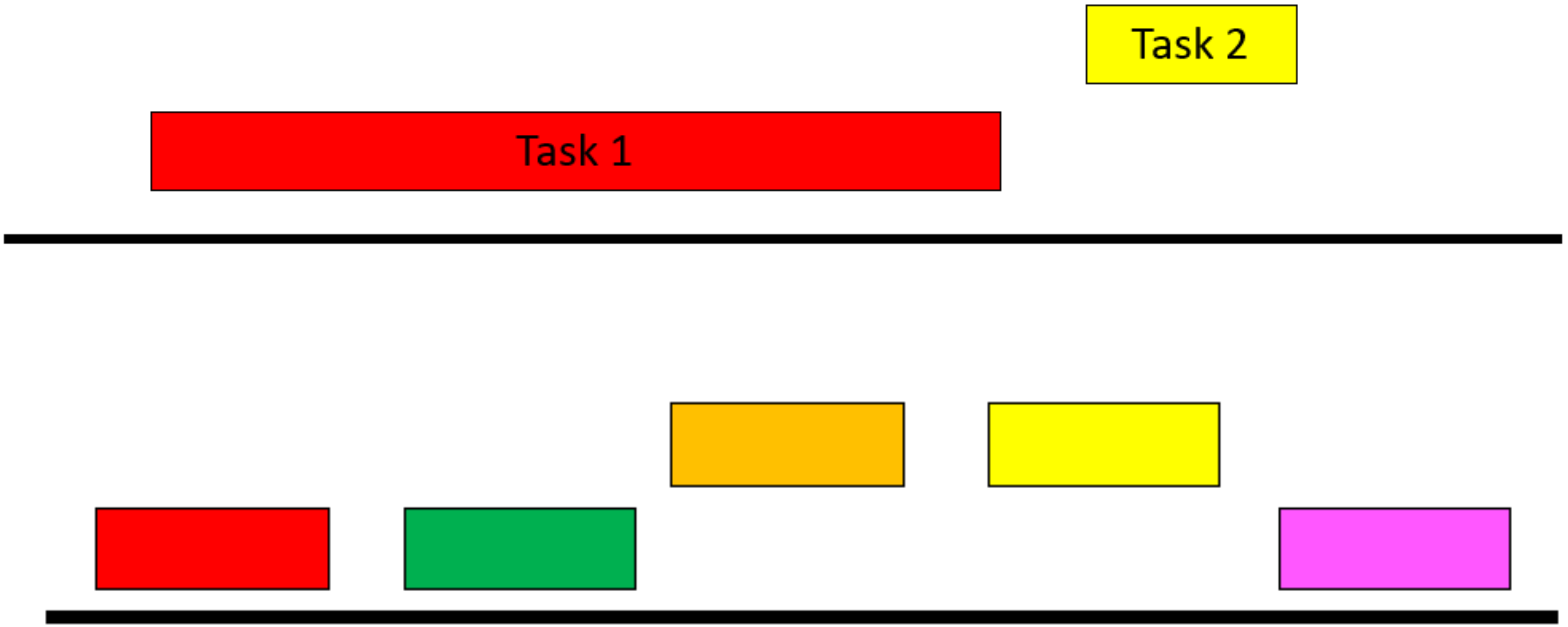
`Output:` A schedule S of n intervals where no two intervals in S conflict, and the total number of intervals in S is maximized.

Sample Input and Output

`Input:`

Example 1

No intervals overlap



Algorithm?

No intervals overlap

R: set of requests

Set S to be the empty set

While R is not empty

Choose i in R

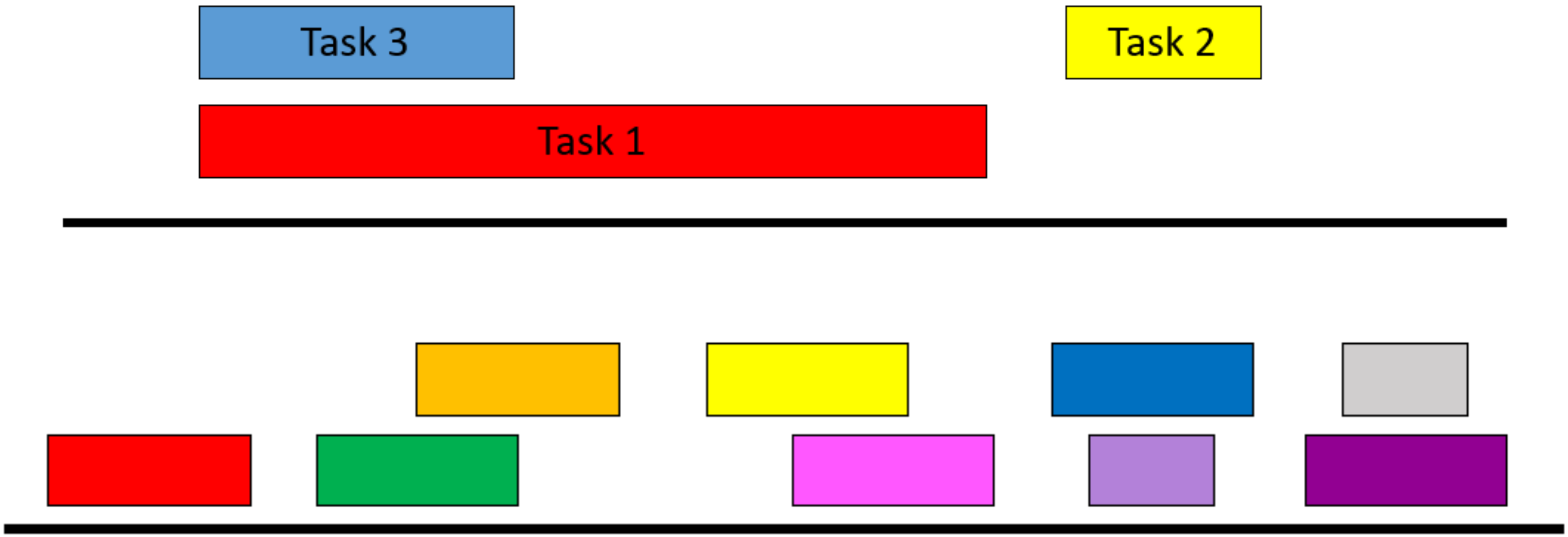
Add i to S

Remove i from R

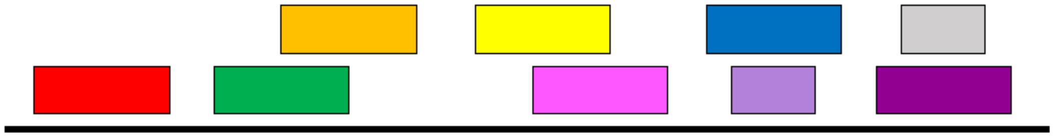
Return $S^{*}= S$

Example 2

At most one overlap



Algorithm?



At most one overlap

R: set of requests

Set S to be the empty set

While R is not empty

Choose i in R

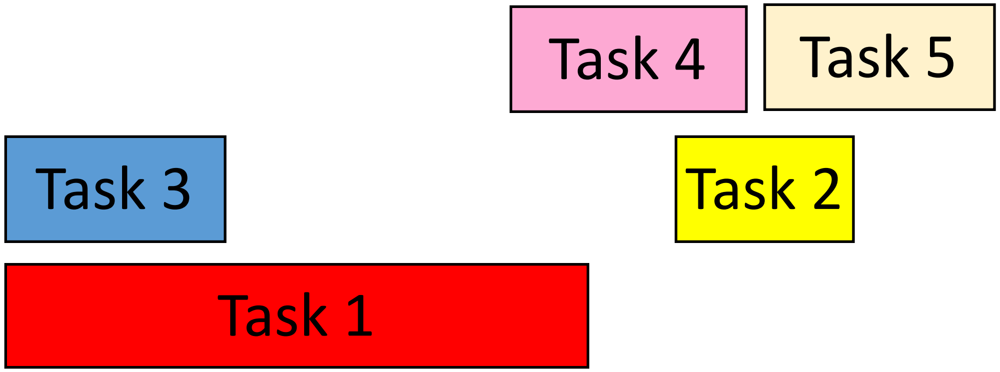
Add i to S

Remove all tasks that conflict with i from R

Return $S^*= S$

Example 3

More than one conflict



Set S to be the empty set

While R is not empty

Choose i in R

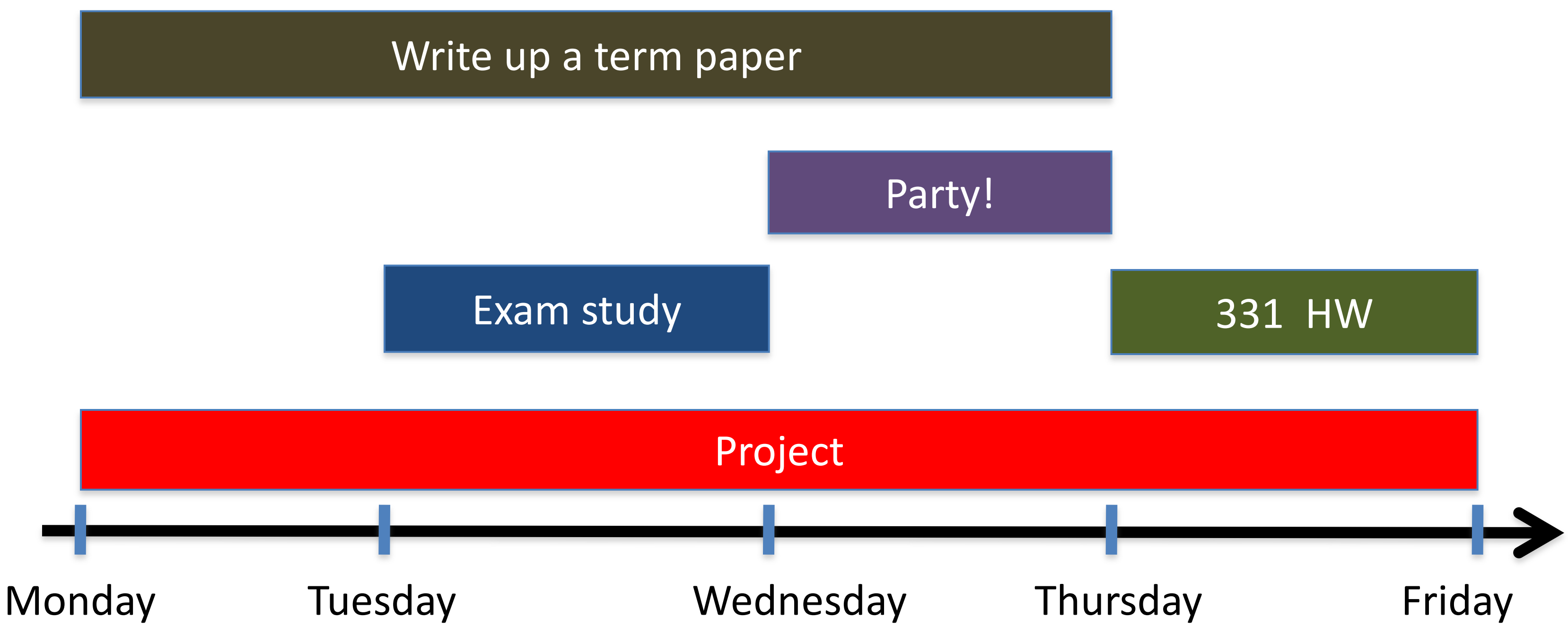
Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

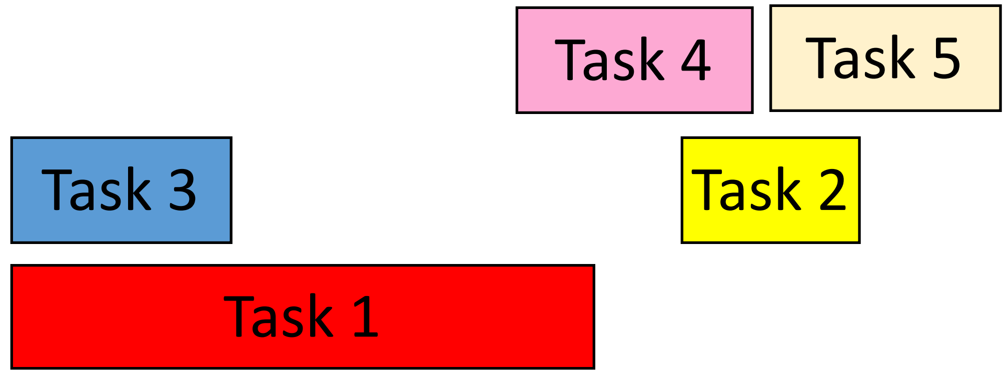
Greedily solve your blues!

Arrange tasks in some order and iteratively pick non-overlapping tasks



Making it more formal

More than one conflict



Set S to be the empty set

While R is not empty

Choose i in R that minimizes v(i)

Add i to S

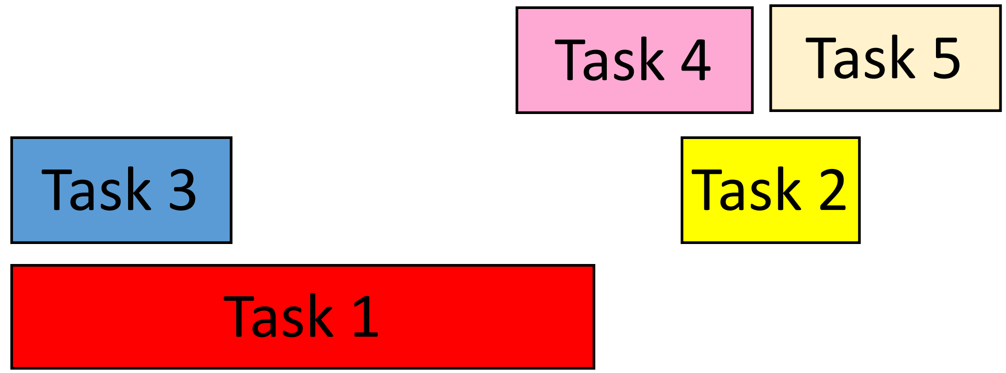
Remove all tasks that conflict with i from R

Return $S^* = S$

Associate a value v(i) with task i

What is a good choice for v(i)?

More than one conflict



Set S to be the empty set

While R is not empty

  Choose i in R that minimizes $v(i)$

  Add i to S

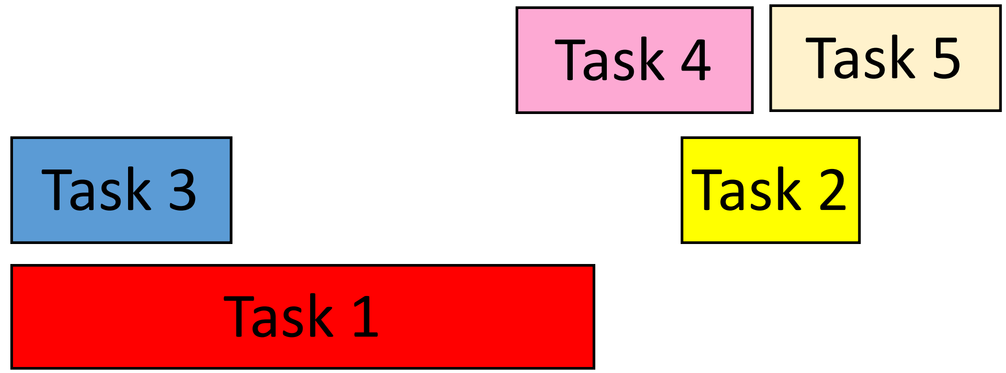
  Remove all tasks that conflict with i from R

Return $S^* = S$

Associate a value $v(i)$ with task i

$v(i) = f(i) - s(i)$

Smallest duration first



Set S to be the empty set

While R is not empty

Choose i in R that minimizes $f(i) - s(i)$

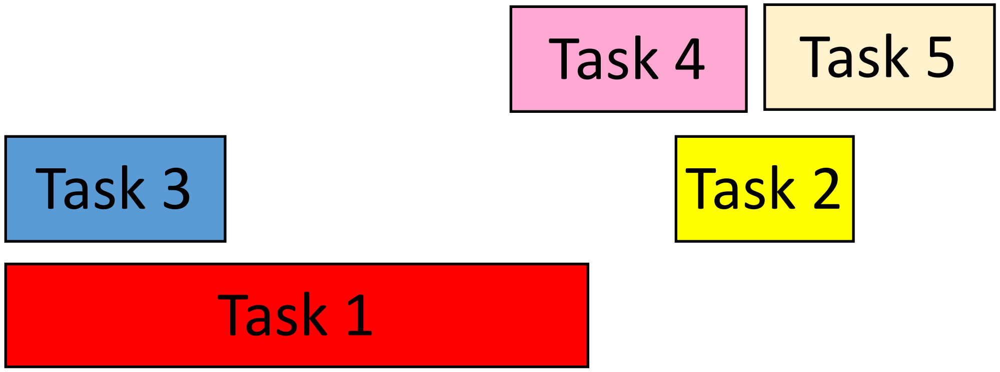
Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

$$v(i) = s(i)$$

Earliest time first?



Set S to be the empty set

While R is not empty

    Choose i in R that minimizes $s(i)$

    Add i to S

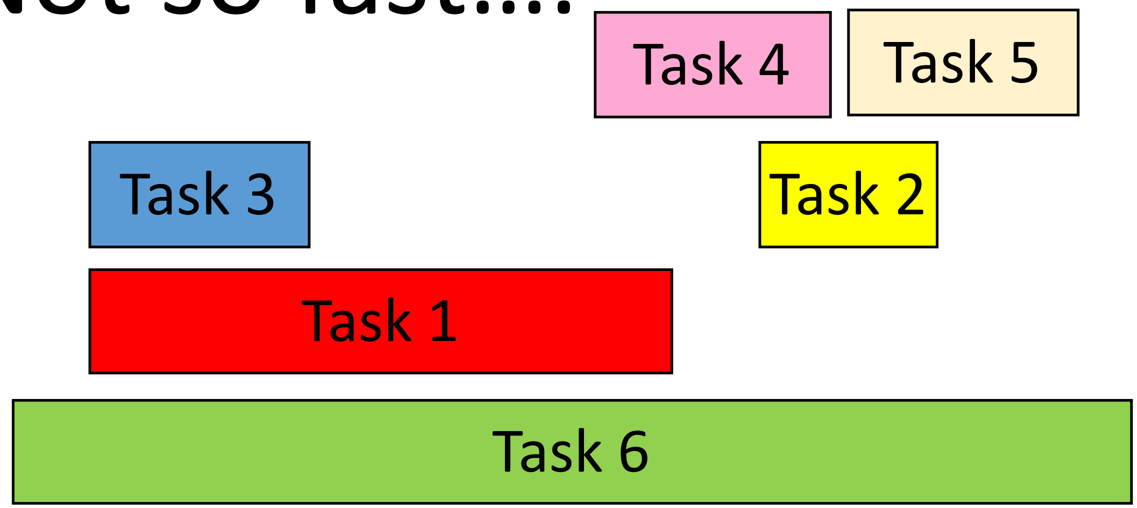
    Remove all tasks that conflict with i from R

Return $S^* = S$

So are we done?

Not so fast….

Earliest time first?



Set S to be the empty set

While R is not empty

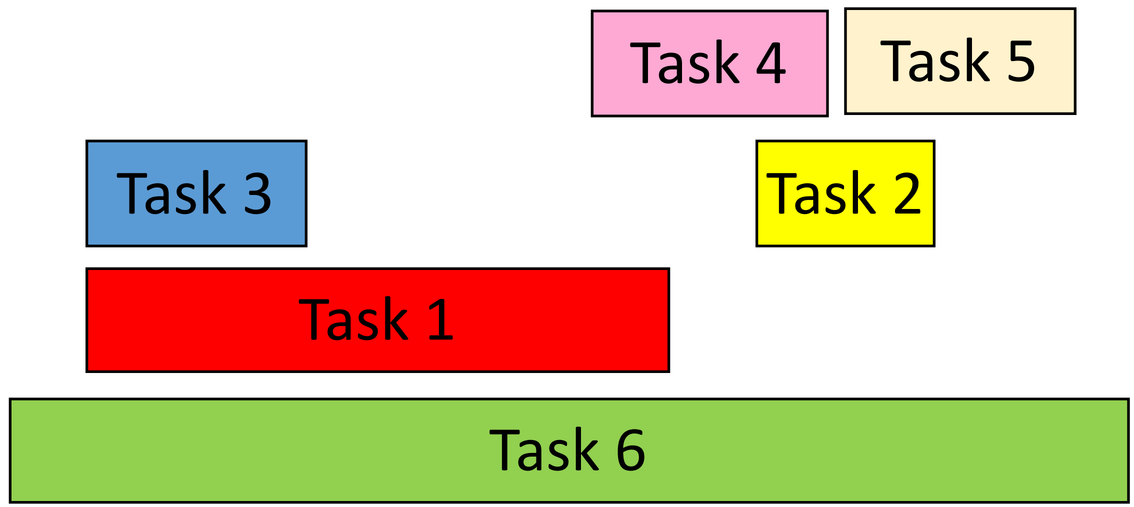
Choose i in R that minimizes s(i)

Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

Pick job with minimum conflicts



Set S to be the empty set

While R is not empty

Choose i in R that has smallest number of conflicts

Add i to S

Remove all tasks that conflict with i from R

Return $S^*= S$

So are we done?

Nope (but harder to show)

Set S to be the empty set

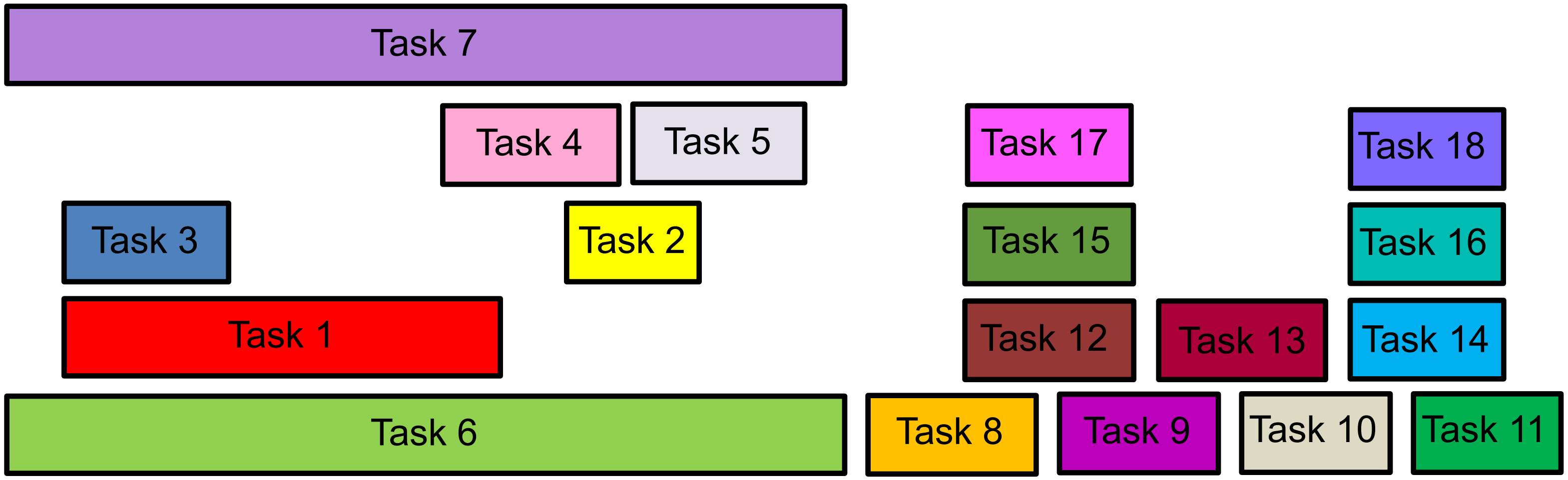
While R is not empty

Choose i in R that has smallest number of conflicts

Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$



Set S to be the empty set

While R is not empty

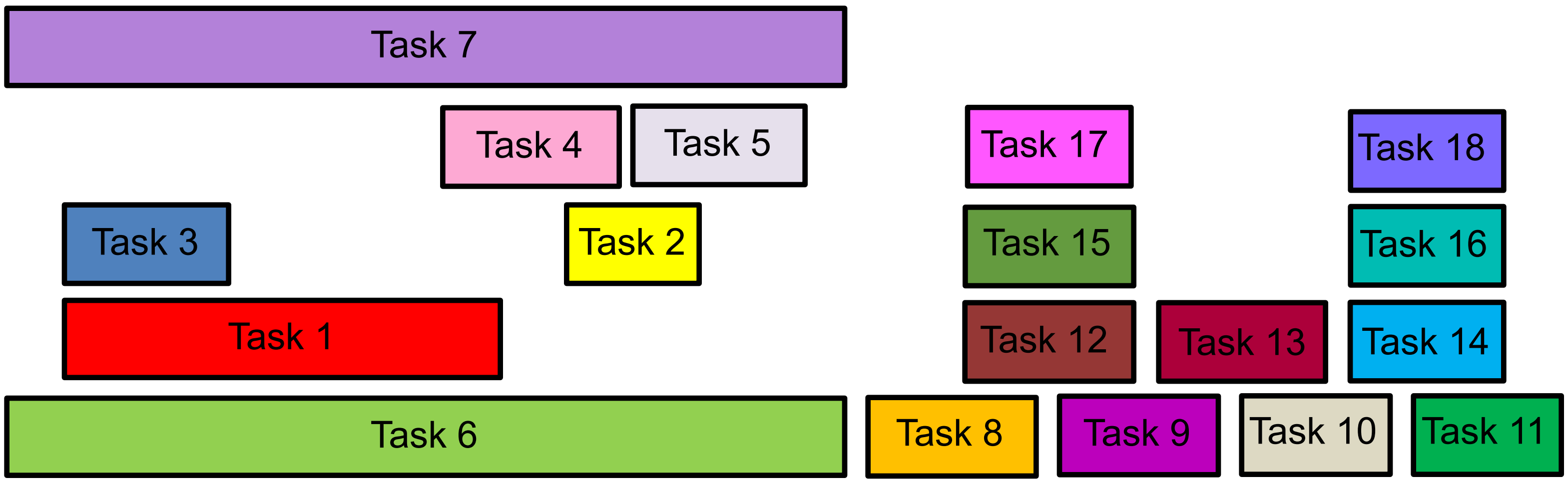
Choose i in R that has smallest number of conflicts

Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

Algorithm?



Set S to be the empty set

While R is not empty

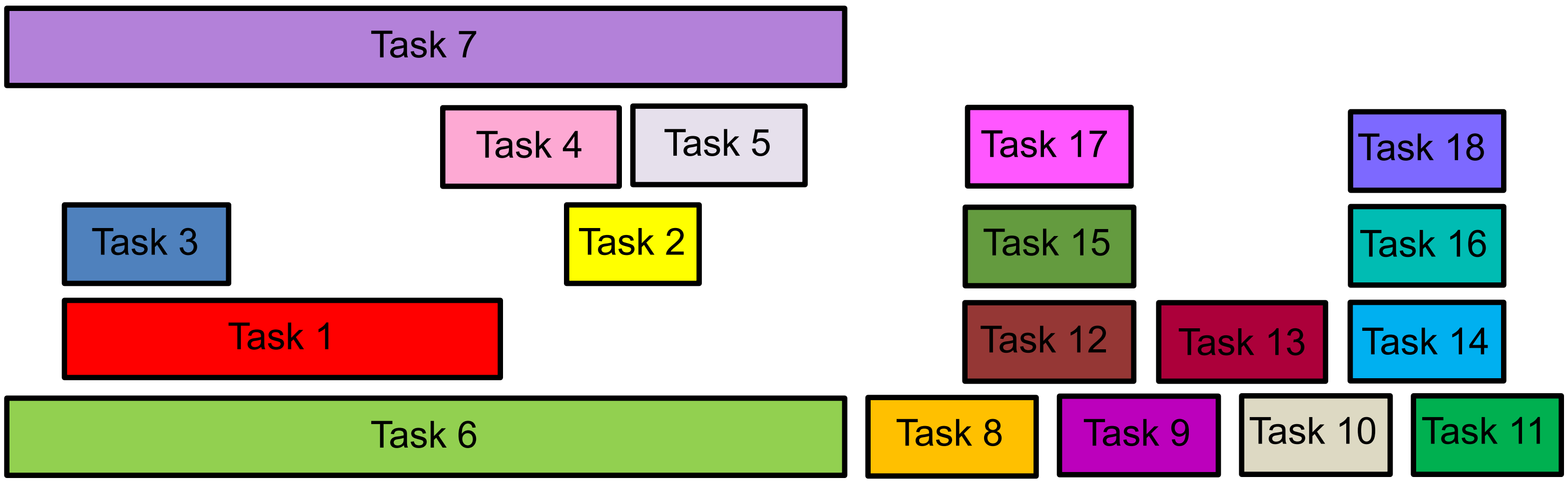
Choose i in R that minimizes v(i)

Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

Earliest finish time first



Set S to be the empty set

While R is not empty

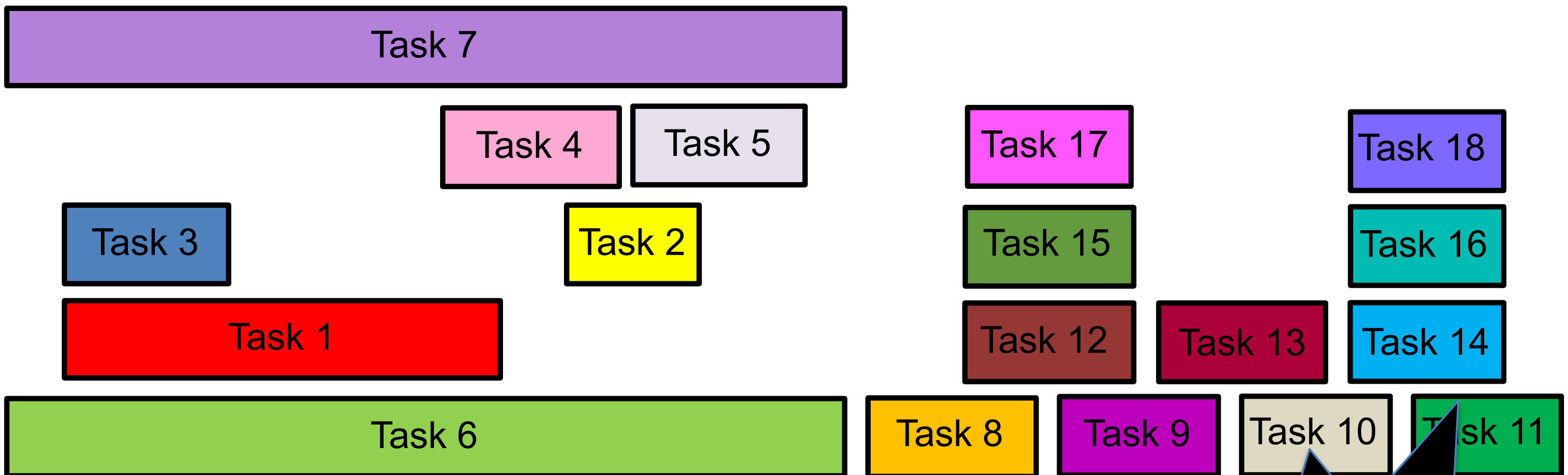
    Choose i in R that minimizes f(i)

    Add i to S

    Remove all tasks that conflict with i from R

Return $S^* = S$

Find a counter-example?



Set S to be the empty set

While R is not empty

Choose i in R that minimizes f(i)

Add i to S

Remove all tasks that conflict with i from

Return $S^* = S$

It works!

Questions?

Today’s agenda

Prove the correctness of the algorithm

Final Algorithm

R: set of requests

Set S to be the empty set

While R is not empty

Choose i in R with the earliest finish time

Add i to S

Remove all requests that conflict with i from R

Return $S^* = S$