Adam Jablonski
CS-260

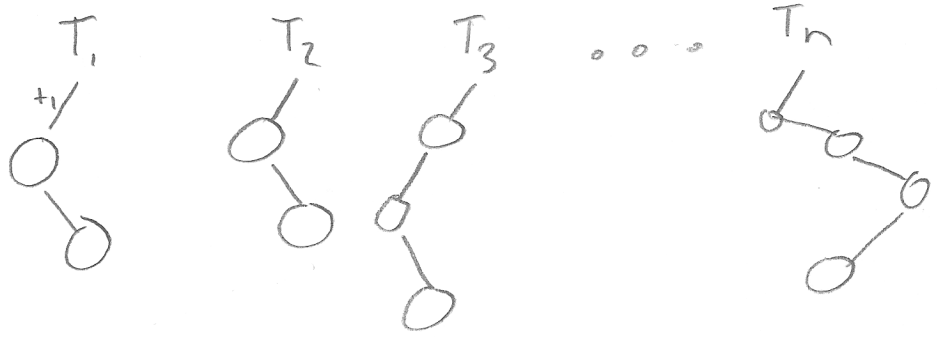
Section 6.2

$T_1, T_2, \ldots, T_n$ for times $t_1, t_2, t_3, \ldots, t_n$

Assuming they run parallel:

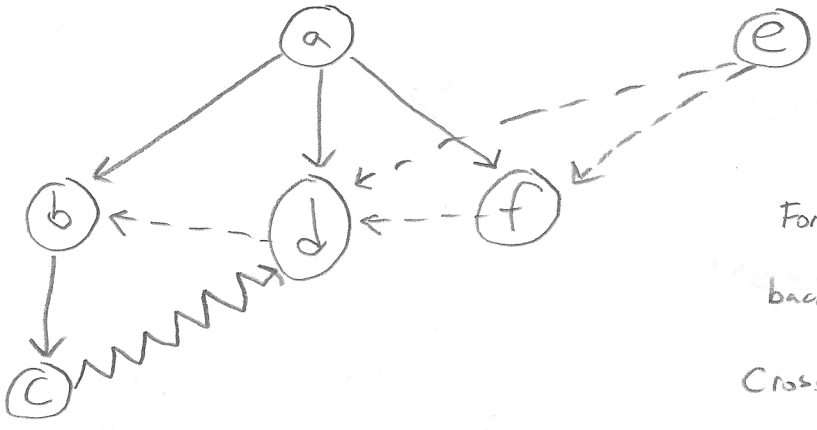
$$= Min(t_1, t_2, \ldots, t_n)$$

Build a tree



The tree with min t_n, have a storage node for the total time

Section 6.8



Forward arc = none

backward arc = $c \to d$

Cross arc = $f \to d$, $d \to b$, $e \to f$, $e \to d$, $b \to f$

Section 6.10

assuming I have access to all verticies

Procedure

```
List_of_Verticies = [list of vertices]

for x in List_of_verticies:
    # modified dfs to return height of tree at that point
    Count = mod_dfs(x).len
    if Count == len(List_of_vertices):
        return True, "vertex is", x

return false.
```

Section 6.14

```
Procedure dfs(vertex v):
    # same as normal dfs but
      returns an additional field
      called count which is the height.


Procedure longest path(List_of_vertices L)
    vertex = 0
    max = 0
    for x in L:
        Count = dfs(x).height
        if count > max:
            Max = count
            vertex = x

    return vertex
```

Time complexity is

$O(n \cdot 2^n)$

Since a for loop is n and dfs is recursive

Section 6.15

{f, d}, {f}, {d}, {b}, {c}

{f, d, b} {f, d, b, c} {b, c} {b, c, d}

{c, d}

{f, d, b, c}, {d, b, c}

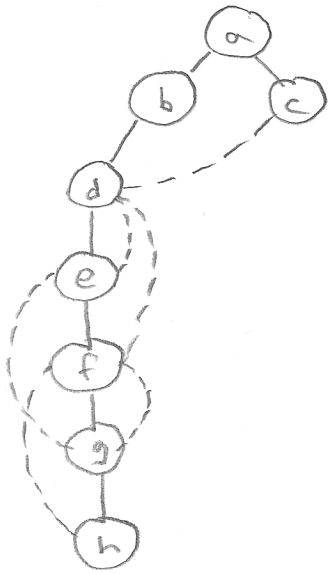
{d, b, f}

Section 6.20

```
Procedure simple(diagraph d, vertici: v1, v2)
    d.First = v1
    d.Last = v2
    visited_List = []
    Stack S
    while len(visited_List) != total_Number_of_vertices:
        While d.Next != d.Last
            if visited_List !contain d.value
                visited_List.add(d.value)
                d.Next()
    while len(visited_List) != total number of vertices
            Print d.value
            d.Next()
        else:
            d.previous() # try again
    d.First()
```

Section 7.3 Part b

Starting at a



Starting at d