remember. The 1st lemma says that the optimal solution cannot be smaller than the depth.
OPT ≥ DEPTH

Algorithms

DATE: 17-03-2022

ALG(I): (still a greedy algorithm).

bad idea ~~# Bubble sort n^2~~

Because of the two loops that have to scan 2 things. (n^2)

- Let d be the depth(I)
- Sort the $|I| = n$ intervals increasingly by their starting time. (fast version $O(n \log n)$)
- Let $I = \{(s_1, f_1), (s_2, f_2), \dots, (s_n, f_n)\}$ with $s_1 \le s_2 \le s_3 \le \dots \le s_n$
- For $j = 1$ to n # J going from 1 to n.
  - $L \leftarrow \{1, 2, \dots, d\}$ # we initialize a set of labels for the resources.
  - For $i = 1$ to $j-1$
    - if (s_i, f_i) is incompatible with (s_j, f_j): # if overlap
      - $L \leftarrow L - \{e(i)\}$ # cannot use the label of i to schedul J.
  - IF $|L| \ge 1$: # if there's at least 1 label available we pick it and we assign it to interval J.
    - Let $a \in L$
    - Set $e(j) = a$
  - ELSE # we will prove that we will never run the FAIL instruction because we'll alway find a "free" label.
    - FAIL
- RETURN THE LABELING $e(1), e(2), \dots, e(n)$ # we return the labels assigned to the intervals.

if (s_i, f_i) is scheduled on pc n.3, then (s_j, f_j) cannot.

we are checking which of the (s_i, f_i) intervals are causing trouble with (s_j, f_j)

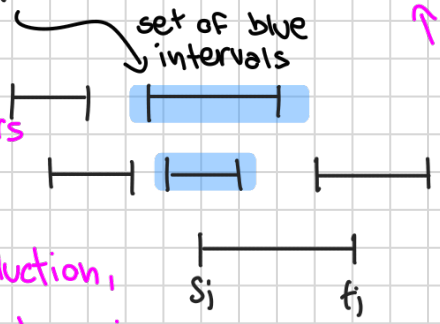
L2: The algorithm never fails.

P: Consider the generic iteration of the outer loop.
Let j be the value of that loop's index j in this iteration.
Let S_j be the set of intervals that the algorithm considered before the jth interval (s_j, f_j) and that (11) ends after f_j.

$$S_j = \{(s_i, f_i) \mid i \le j-1 \text{ and } f_i \ge s_j\}$$

↑ The set of intervals incompatible with (s_j, f_j) and that start before (s_j, f_j). We consider them increasingly before (s_j, f_j).

If we can prove that this algo doesn't fail, then we can prove that the number of computers needed to schedule is d.



The proofs are by induction, by contradiction, or just a mix.

S_j is the set of intervals that we have already labelled and that interfere with (s_j, f_j).

Each interval in S_j will have a label that the inner loop will remove from L. No other label will be removed from L.

Thus, after the inner loop,

$$|L| \geq d - |S_j|,$$

since L started with d labels.

affermazione

CLAIM: $|S_j| \leq d - 1$

if this claim holds, at the end of the inner loop, the set L will contain at least 1 label.

P: Each interval in S_j (set of incomp. intervals.) passes through time s_j, the starting time of the interval that we're labeling at this iteration. and also comes before (s_j, f_j) in the ordering. Then $(s_j, f_j) \notin S_j$.

This is the opposite of the claim.

Suppose, by contradiction, that $|S_j| \geq d$. Then, since $(s_j, f_j) \notin S_j$, the set

$$T = S_j \cup \{(s_j, f_j)\}$$ has cardinality $|T| \geq d + 1$

But, each interval in T passes through time s_j.

Then, DEPTH$(I) \geq |T| \geq d+1$. This contradicts $d =$ DEPTH(I) ■ # If the DEPTH(i)=d, then how the fuck can we say that DEPTH(i) is less than T, which is less than d+1.

But, then, after the loop, $|L| > d - |S_j| \geq 1$. Thus, $L \neq \emptyset$ and the algorithm doesn't fail ■

Greedy doesn't schedule 2 conflicting intervals on 1 resource.

L3: The algorithm returns a valid labeling.

P: Suppose that (s_i, f_i) and (s_j, f_j) are overlapping intervals, with $i < j$. Then, the label of (s_i, f_i) is chosen before the label of (s_j, f_j). Since (s_i, f_i) and (s_j, f_j) are overlapping, the label we assign to (s_j, f_j) cannot be the label of (s_i, f_i) – indeed we removed that label from L, before picking a label for (s_j, f_j) from L.

T: The algorithm returns an optimal solution.

P: No feasible (or valid) solution can have fewer than DEPTH(I) resources (Lemma 1).
The solution returned by the algorithm uses DEPTH(I) resources (L2); and it is FEASIBLE/VALID (L3) ■

It means that there are no-overlapping intervals in the solution.

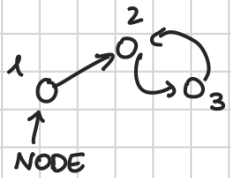
ALG(I): # This algorithm can run N Log N fast by using the priority queue data structure, but actually it takes polinomyal time to execute (n^2).

- Let d be the depth(I)
- Sort the $|I| = n$ intervals increasingly by their starting time.
- Let $I = \{(s_1, f_1), (s_2, t_2), \dots, (s_n, f_n)$ with $s_1 \leq s_2 \leq s_3 \leq \dots \leq s_n$
- For $j = 1$ to n
  - $L \leftarrow \{1, 2, \dots, d\}$
  - For $i = 1$ to $j-1$
    IF (s_i, f_i) is incompatible with (s_j, t_j): // if they overlap
    $L \leftarrow L - \{e(i)\}$
  - // INVARIANT: $|L| \geq 1$ → # A property/comment that we write to remember that HAS to hold at any point during the execution of the algorithm.
  - Let $a \in L$
  - Set $e(j) = a$
- Return the labeling $e(1), e(2), \dots, e(n)$

---

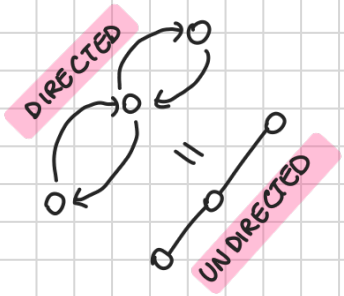
v* starting point
w* finish point

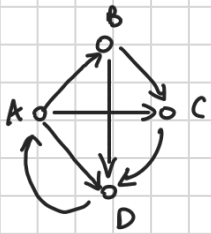
A (directed) graph $G(V, E)$ is composed of a set of vertices V and of a set of arcs E, with $E \subseteq \{(v, w) \mid v \neq w \text{ and } v, w \in V\}$



$V = \{1, 2, 3\}$

$E = \{(1,2), (2,3), (3,2)\}$

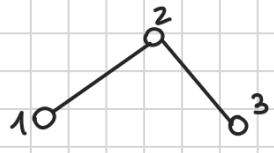




$V = \{A, B, C, D\}$

$E = \{(A,B), (B,C), (C,D), (D,A), (A,D), (A,C), (B,D)\}$

An undirected graph $G(V, E)$ is composed of a set of vertices V and of a set of edges E, with $E \subseteq \{(v, w) \mid v \neq w \text{ and } v, w \in V\}$



$V = \{1, 2, 3\}$

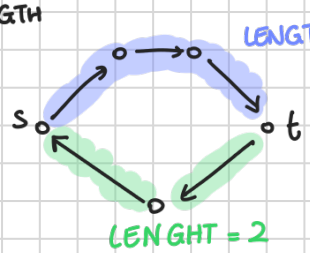
$E = \{(1,2), (2,3)\}$

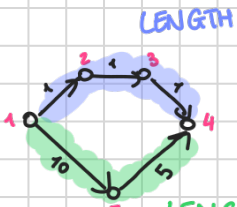
DIGRAPH

A weighted directed graph is a directed graph $G(V,E)$ with a function $\ell: E \to \mathbb{R}_{\geq 0}$

LENGTH

But this function could represent anything, not only a length





(10+5 TOO MUCH PATH
1+1+1 QUICKER)

LENGTH = 15 (Holy shit.....)

When we give a weighted graph, apart from specify the nodes and the edges, we should also specify the length of the edges.

$V = \{1,2,3,4,5\}$

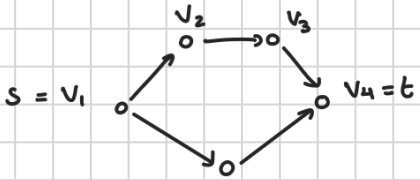
$E = \{(1,2), (2,3), (3,4), (1,5), (5,4)\}$

$\ell(1,2) = 1$
$\ell(2,3) = 1$
$\ell(3,4) = 1$
$\ell(1,5) = 10$
$\ell(5,4) = 5$

} LENGTH OF PATHS

HOW TO FIND THE SHORTEST PATH FROM A NODE "s" TO A NODE "t" IN THE GRAPH?

DEF: Given $G(V,E)$ a path from $s \in V$ to $t \in V$ is a sequence of nodes $s = v_1, v_2, v_3, \dots, v_K = t$ such that $(v_1, v_2) \in E$, $(v_2, v_3) \in E$, $\dots$, $(v_{K-1}, v_K) \in E$.

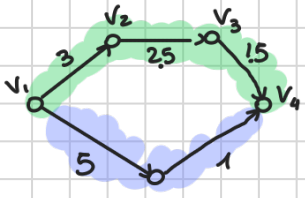


$\pi = v_1, v_2, v_3, v_4$ is a path from $v_1 = s$ to $v_4 = t$

$v_1 v_3 v_4$ is not a path
↳ NEIN

DEF: If $G(V,E)$, ℓ is a weighted graph and if $\pi = v_1, v_2, \dots, v_K$ is a path in $G(V,E)$, then the length of π is:

$$\ell(\pi) = \sum_{i=1}^{K-1} \ell(v_i, v_{i+1})$$



$\pi = v_1, v_2, v_3, v_4$
$\ell(\pi) = 7$

$\pi = v_1, v_5, v_4$
$\ell(\pi) = 5+1 = 6$

PROBLEM: Given a weighted graph $G(V,S), \ell$, and given $s, t \in V$. What is the length of the shortest path from s to t?

Edgar Daekstra

E. W. DIJKSTRA'S ALGO (G(V,E), ℓ, s) # figures out the shortest path

// the algorithm will use the set S to denote the
// set of nodes it visited so far.
// Moreover, ∀u∈V, d(u) will be set to the length of a
// shortest path from "s" to "v".

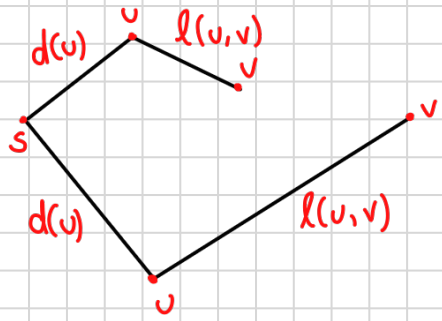
S ← {s} # Set of visited nodes. At the beginning will contain only the starting, so the actual distance is set at 0.

distance ↖ d(s) = 0

WHILE S ≠ V: (S: set of visited nodes. V: The vertices that we haven't visited yet nodes)

- SELECT A NODE v∈V−S THAT CAN BE REACHED DIRECTLY FROM SOME NODE IN S, AND FOR WHICH $d'(v) = \min_{\substack{(u,v)\in E \\ u\in S}} (d(u) + \ell(u,v))$ is MINIMUM (the shortest path from the actual node to another one.)

In this while loop we select a node v that we haven't visited yet, so that doesn't belong to the set S.



- $S \leftarrow S \cup \{v\}$

- $d(u) = d'(v)$

RETURN (d) — Vector of shortest paths starting from S.

We're adding to the set of visited nodes, the actual one.

We're claiming that the length of that the path that we've crossed until now, is for sure the shortest one.

Dijkstra's algorithm is almost linear time.