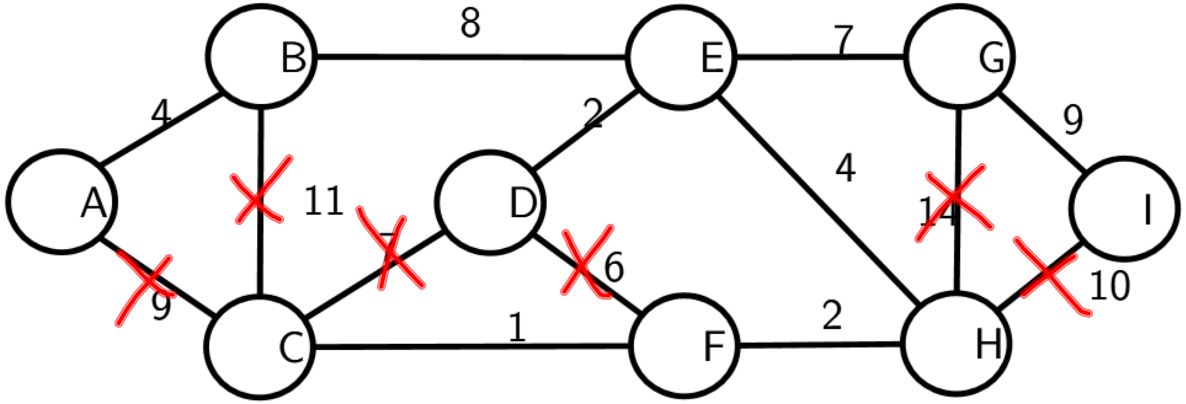
Minimum Spanning Trees

Read the problem description in your handout. Which edges would you keep for the following graph?



Could any valid solution contain a cycle? NO.

Suppose some "OPT" solution/output that contains a cycle

Let C be any cycle in sol'n

remove any $e \in C$ and remove it.

We have a strictly better sol'n now.

$\rightarrow\leftarrow$

Suppose C is a cycle within G. At least one edge in C won't be in our solution. Which edge and why?

Maximal / highest weight edge

Suppose "OPT" has max wt edge and omits a smaller one.

$OPT' = OPT + \text{omitted smaller} - \max$

OPT' strictly better

(might still not be optimal)

The Reverse-Delete Algorithm

runtime?

$O(m+n)$ to find cycle

while G contains a cycle **do**
  Let C be a cycle within G
  Let e be a maximal edge within C
  Remove e from G
return G

- Why is this correct?

while ((c = findCyc(G)) != nullptr)

$O(n)$

each iter: $O(m+n)$

how many iter?

$m-(n-1)$ is $O(m)$

Tree: connected, no cycles

+ every removed edge correct to do so

$O(m(m+n))$

The Cut Property

A is any $\subseteq V$

If vertices partitioned into A, $V-A$ (non-empty)

then smallest e : A — $(V-A)$
is in opt MST

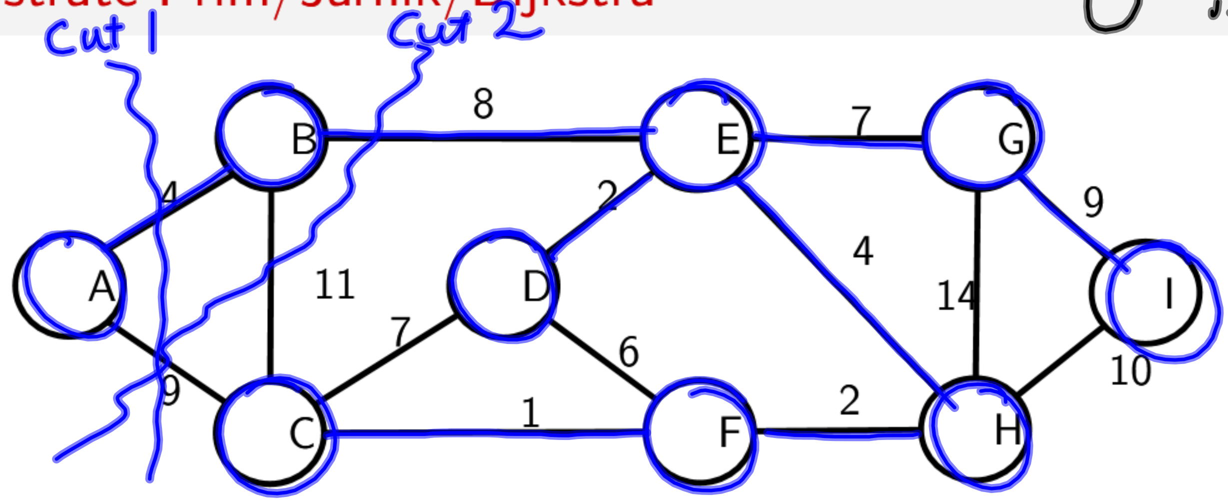
Modify Dijkstra's Algorithm?

dist now "weight of Cheapest edge that crosses the cut" (and inc. this vertex)

for each vertex v **do**
  intree(v) = false
  parent(v) = N/A
  dist(v) = ∞
dist(s) = 0
while $\exists$ vertex u with intree(u) = false **do**
  $u \leftarrow$ vertex with intree(u) = false and *smallest* dist(u)
  intree(u) = true
  for each vertex $v \in$ adj[u] **do**
    if d(v) > $\cancel{\text{d}(u)}$ + w(u, v) **then**
      d(v) = $\cancel{\text{d}(u)}$ + w(u, v)
      parent(v) = u

Illustrate Prim/Jarník/Dijkstra

Key : O is set A
O is V−A

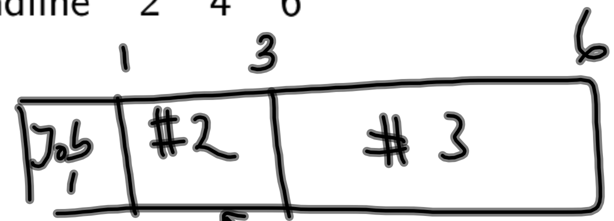


Understanding the Problem

Example 1: What is the optimal schedule for the following?

Time	1	2	3
Deadline	2	4	6

max. lateness Ø

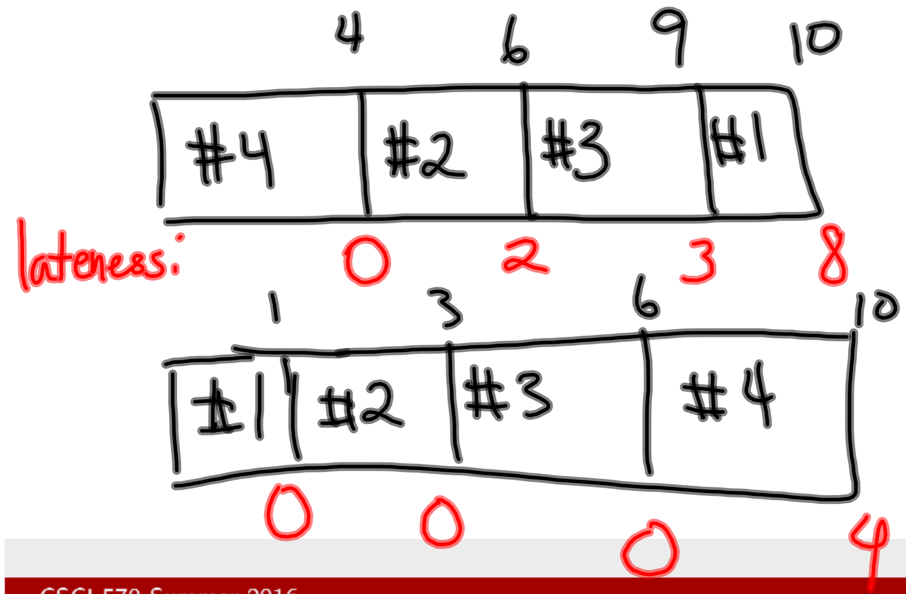


no "extra credit" for finishing early.

Understanding the Problem

Example 2: What is the optimal schedule for the following?

Time	1	2	3	4
Deadline	2	4	6	6



Possible Algorithm 1

Sort the jobs by increasing time t_i; schedule them in that order.

t_i	1	2	3	4
d_i	4	3	2	1
l_i	0	0	4	9

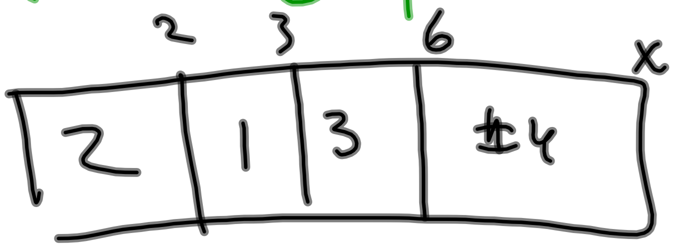
alt order: 4 3 2 1

3 5 6 6

Possible Algorithm 2

Sort the jobs by $d_i - t_i$; schedule them in that order.

t_i	1	2	3	4
d_i		2	4	1000
$d_i - t_i$	1	0	1	996



t_i	2	4	6	8
d_i	1	2	3	4
$d_i - t_i$	-1	-2	-3	-4
	19	16	11	4
l_i		8	14	18 20

8	6	4	2

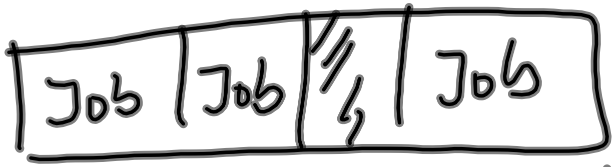
l_i :	1	4	9	16

2 4 6 8

Possible Algorithm 3

Sort the jobs by deadline d_i; schedule them in that order.

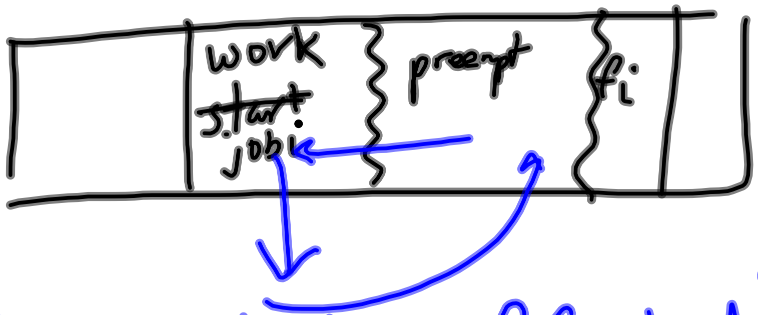
When deciding start times, don't leave any gaps; $s_{i+1} = s_i + t_i$.



eliminate gap, "slide down" everything after.

take any sol'n w/ ≥ 1 gap. Elim one gap,
any done before gap, unaffected.
any after, affected lateness either
none (already on time)
or made better (less late)

Claim: no need for preemption



finish job i affected? no

finish after f_i? no

finish before f_i affected?

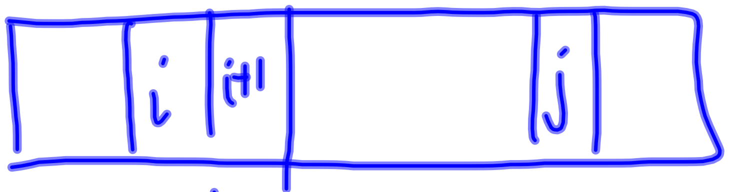
Yes, none for the worse.

Proof

Any schedule that doesn't agree with our algorithm has at least one pair of *consecutive* intervals $i, i+1$ that are *inverted* relative to our order.

Note: You may take this fact as a given for the related homework problem. You do not need to prove it again.

ALT:



in mine, j before i (not nec. immediately)

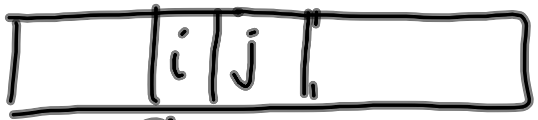
if $j = i+1$, done

else if $i, i+1$ inverted? done

else $(i+1, j)$ are inverted. Repeat.

Proof

Any schedule with an inversion can be modified to be more like our algorithm's output without making it worse.

ALT 

i, j are inverted. $d_j \le d_i$

Claim: $ALT' = ALT$ w/ i,j swap (s_i)

ALT' at least as good as ALT

Pf. are any $k \neq i,j$ affected? NO

Let $f_i = s_i + t_i$ $\quad f_j = s_i + t_i + t_j$

in ALT': $f_i' = s_i + t_j + t_i = f_j$ $\quad f_j' < f_j$ so $l_j' < l_j$

$l_i' \le l_j$

Algorithmic Pizza 2

- Ordering: - shortest prep time first
  - largest baking time first

try to prove

Consider any ALT. $\exists\, i,k$ adj. inv. pair $b_k > b_i$

Claim: swap i,k in ALT (to get ALT') makes it no worse.

ALT | | i | k | |

$f_i = S_i + p_i + b_i$ $\quad$ $f_k = S_i + P_i + P_k + b_k$

$f_i < f_k$

$f_i' = S_i + P_i + P_k + b_i < f_k$

$f_k' = S_i + P_k + b_k < f_k$

Problems with some other encodings...

prefix

- a = 0, b = 1, c = 00, d = 01, e = 10, etc

000 aaa? ac? ca?

unused!

- a = 00000, b = 00001, c = 00010, ..., z = 11001
- a = 00000, b = 00001, ..., v = 10101, w = 1100, x = 1101, y = 1110, z = 1111

00000 1101

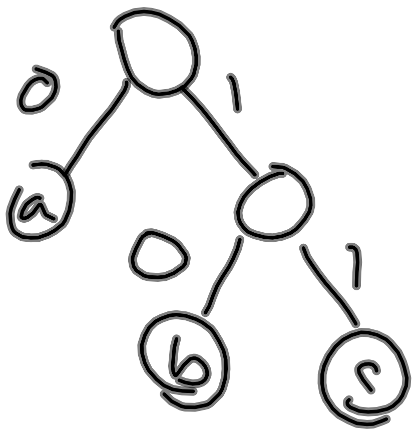
0000 11 111

How can we use a binary tree to represent an encoding?

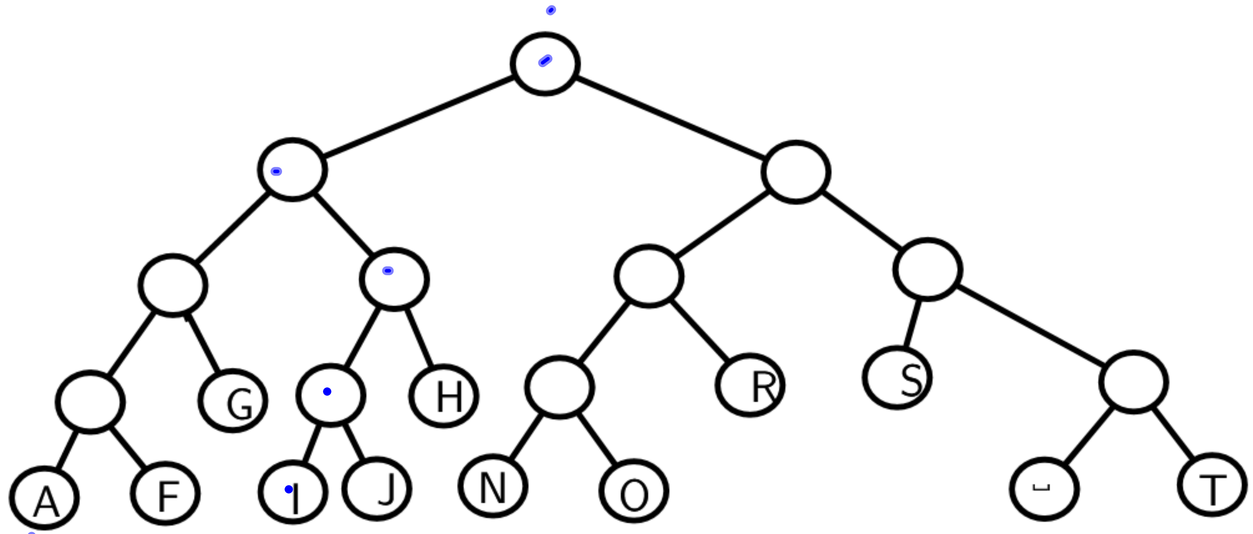
leaf: characters

path: encoding

ex:

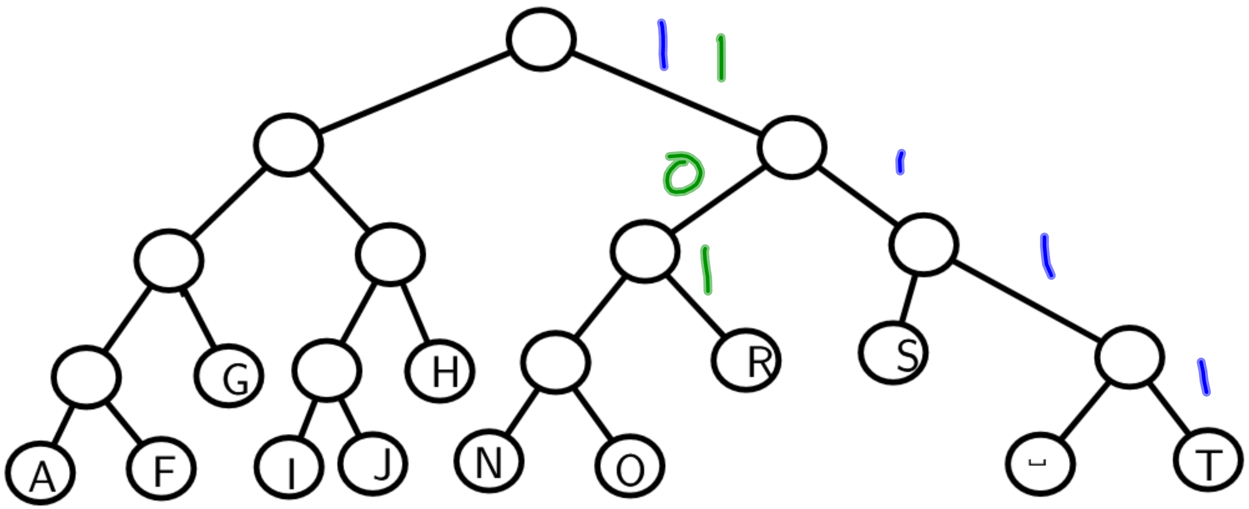


Binary Tree Based Codes



000101000010111111111010011000 = ???

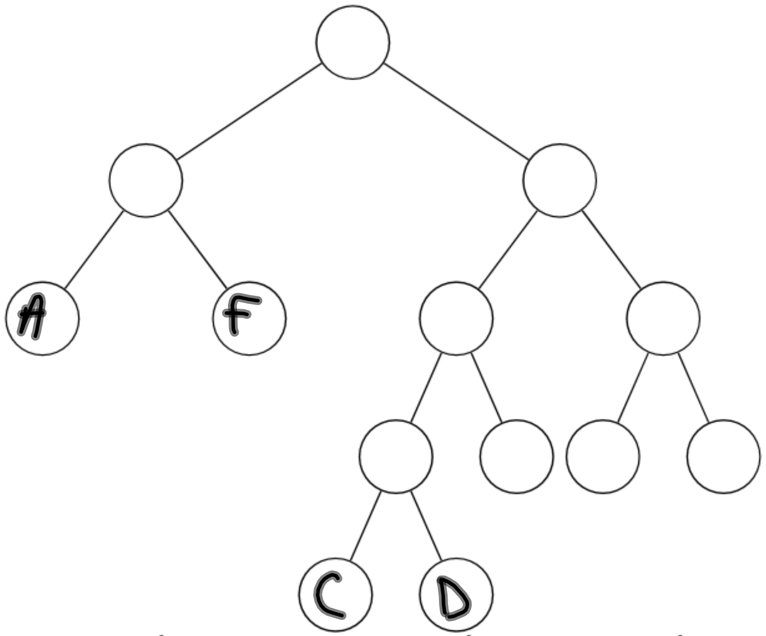
F I



TROJANS = ???

1111101100001 etc

letter x	frequency f_x
~~A~~	~~21%~~
B	18%
C	6%
D	5%
E	12%
~~F~~	~~23%~~
G	15%



Where should the letters go in order to minimize the average bit length of a compressed message?

Optimal tree for these letters?

letter x	frequency f_x
A	21%
~~B~~	~~18%~~
~~C~~	~~6%~~
~~D~~	~~5%~~
~~E~~	~~12%~~
F	23%
~~G~~	~~15%~~