# QUIZ REVIEW

| char | count / freq |
|---|---|
| M | 2 |
| A | 10 |
| V | 4 |
| E | 15 |
| S | 13 |
| T | 5 |
| | 49 bits |

ascii code is stable
so kept in fixed + var-sized codes

MM AAAAAAAAAA → 65 → 01000001 8 bit ascii

8 bits ascii for each char in message

without encoding: 49 chars * 8bit ascii = 392 bits

Fixed-size-code: n → bits → $2^n$ different pattern

3bits ← 6e
$2^3 = 8 > 6e$ characters

| char | count / freq |
|---|---|
| M | 2 |
| A | 10 |
| V | 4 |
| E | 15 |
| S | 13 |
| T | 5 |
| | 49 bits |

000
001
010
:

3 bits (own) each

Encoded Message + chart/table =

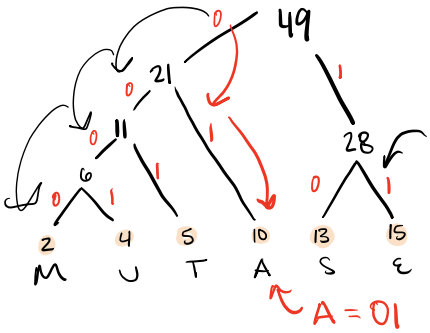
(49 char * 3bits) + ((6 * 3) + (6 * 8)) = 215 bits

(6 * 3): 6 ↑ letters, 3 ↑ own bits
(6 * 8): 6 ↑ letters, 8 ↑ ascii bits
stable

# var-size-code (HUFFMAN):

diff freq for chars → less bits if ↑ freq
so its not a buncha 8 bit stuff unnecessarily
↳ smaller message

optimal merge pattern:




13+15 < 13+21
then merge 28 w/ 21 to get 49

| char | count / freq | var-size-code |
|---|---|---|
| M | 2 | 0000 |
| A | 10 | 01 |
| U | 4 | 0001 |
| E | 15 | 11 |
| S | 13 | 10 |
| T | 5 | 001 |
| | 49 bits | 17 bits total |

| freq * code size |
|---|
| 2 · 4 bits = 8 |
| 10 · 2 bits = 20 |
| 4 · 4 bits = 16 |
| 15 · 2 bits = 30 |
| 13 · 2 bits = 26 |
| 5 · 3 bits = 15 |
| 115 bits total |

Encoded msg + Chart/table (stable)

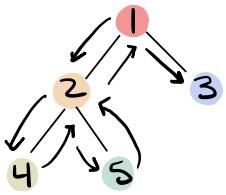
115 + (17 + (6 * 8)) = 180 bits

17: var-size-code; 6: letters; 8: ascii

BFS ← Queue
DFS ← stack

# Depth-First Search

DFS
stack




- Select any vertex as source
  - ↳ source = 1

1, 2, 4, 5, 3

## variables :

color [v] →
- white → vertex undiscovered
- gray → vertex discovered & is being processed
- black → vertex completely processed

$\pi$ [v] → predecessor of vertex v

d [v] → timestamp when vertex v is discovered (discovery time)

f [v] → timestamp when vertex v is completely processed (finish time)

Types of Edges

(B) back edge
- self-loop
- DFS tries to extend the visit from vertex 'u' to vertex 'v'

color [v] = gray

(T) tree edge
- edge that is included in the DFS tree

forward edge (F)
- DFS tries to extend the visit from vertex 'u' to vertex 'v' and finds

color [v] = black
d(u) < d(v)
- discovery time for start vertex #/#
- discovery time for other vertex #/#

cross edge (C)
- DFS tries to extend the visit from vertex 'u' to vertex 'v' finds

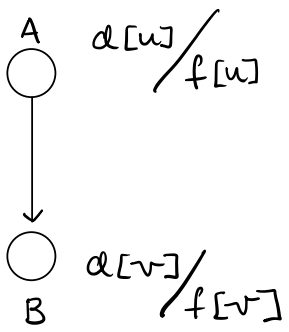
color [v] = black
d(u) > d(v)

DFS (V, E)

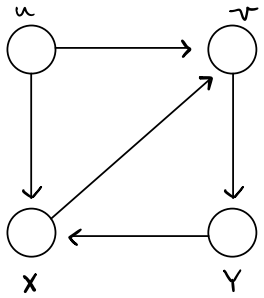
initialization

for each vertex u in V[G]
do color[v] ← white
π[v] ← NULL ← predecessor set to NULL
time ← 0

- for each vertex v in V[G]
  do if color[v] ← white ← if color white
  then Depth_First_Search (v)
  color[v] ← gray ← change to gray (1)
  time ← time + 1 ← time increment (2)
  d[v] ← time ← discovery time = prev time (3)
- for each vertex 'u' adjacent to 'v'
  do if color[u] ← white
  π[u] ← v ← predecessor set to v
- Depth_First_Search (u)
  color[v] ← black
  time ← time + 1
  f[v] ← time
  ↑timestamp when completes processed

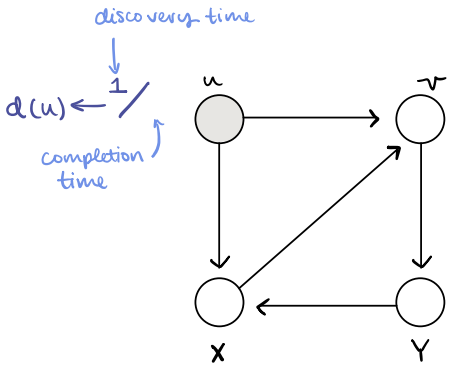


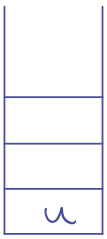
# DFS Example:

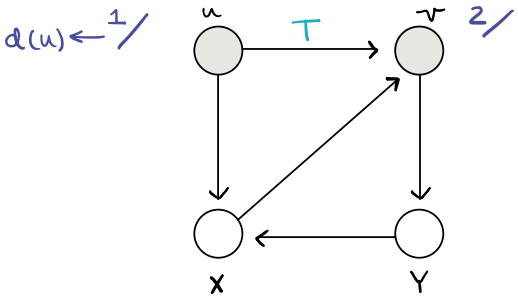


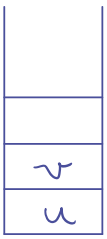

any starting v
* classify edges
* d(v)
* f(v)

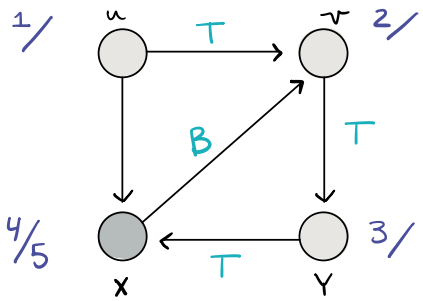
Source = u

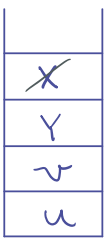


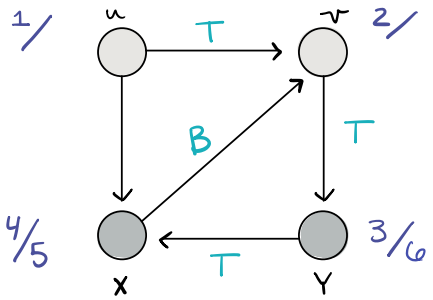



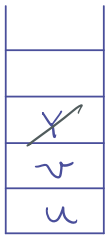



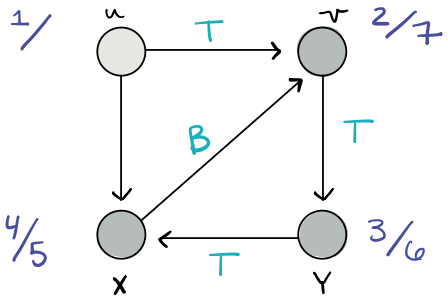



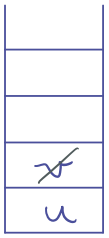

explore next vertex from X
but backtracking so B.
now complete!
↳ completion time /5
and node now black.



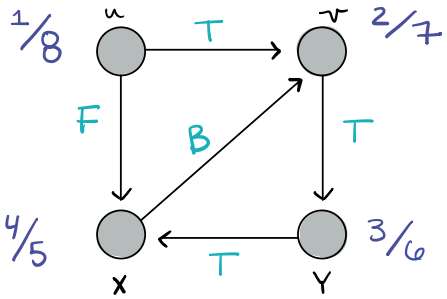




now nothing to explore from Y.
completion time /6
and node now black




now nothing to explore from v.
completion time /7
and node now black





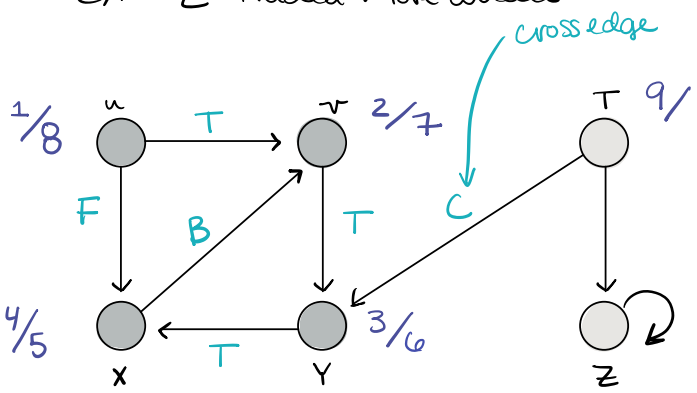
explore u v
color[v] = black
d(u) < d(v)
1 < 4
Forward edge
and /8 complete

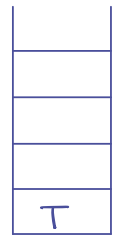
DONE!

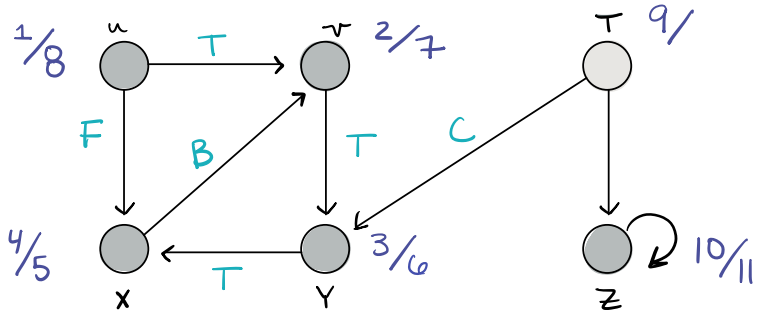
u v → tree edge
v Y → tree edge
Y x → tree edge
x v → back edge
u x → forward edge

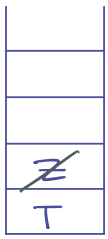
EX #2 Added More dudes

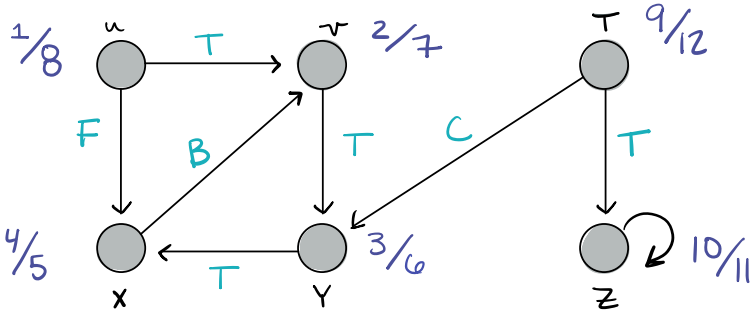
b/c color[v] = black
d(u) > d(v)

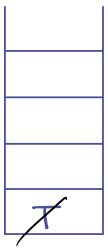


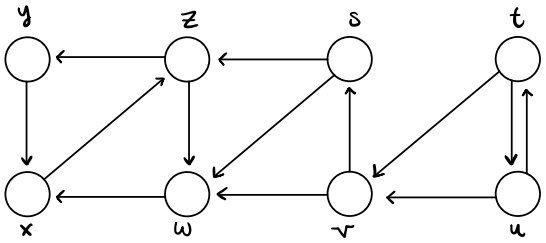

("T to Y") TY → cross edge
TZ → tree edge

week 14

# THURSDAY

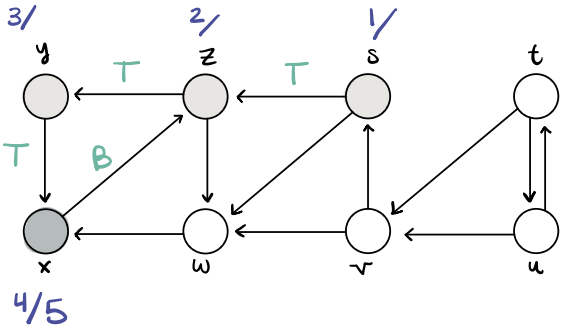
Start vertex : S

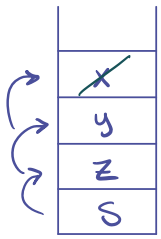
**DEPTH-FIRST SEARCH**
- visit node
- turn gray
- Increment time
- Suspend in stack
- explore node
  ↳ where pointing?



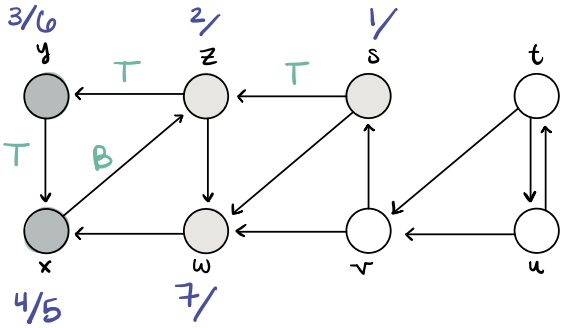

d[v] → time to discover
f[v] → completion time
- Forward edge sw,
- back edge xz, ut
- Cross edge wx, vw, vs, uv
- tree edge sz, zy, yx, zw, tv, tu

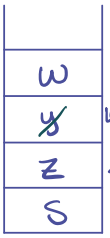
Change colors



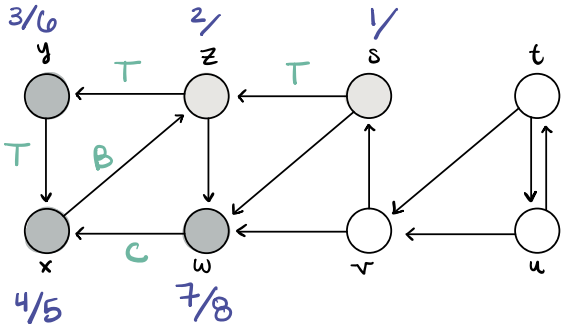



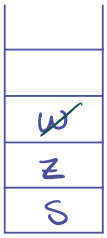

adjacent node so BACK EDGE.
now fully explored, change to black and remove from stack.
(Completion time = 5)



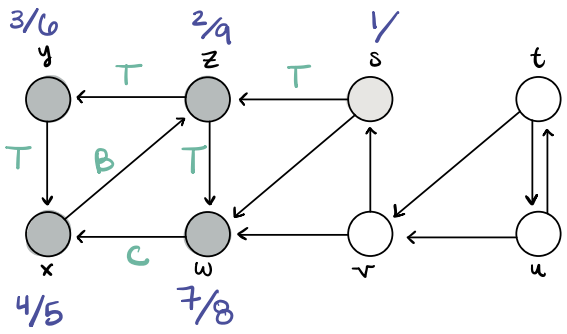



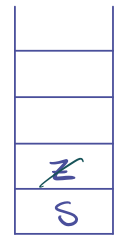

① (Completion time = 6) fully explored
② explore to w, add to stack



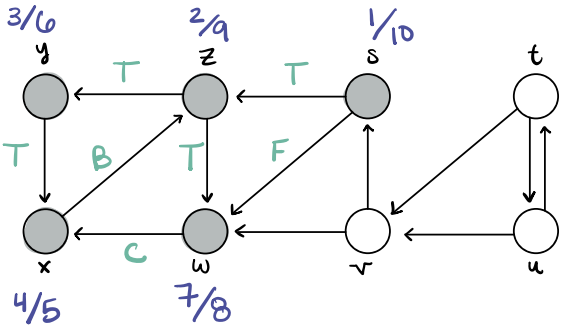



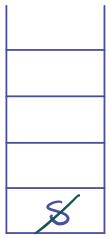

explore w to x, already black so CROSS EDGE
now fully explored



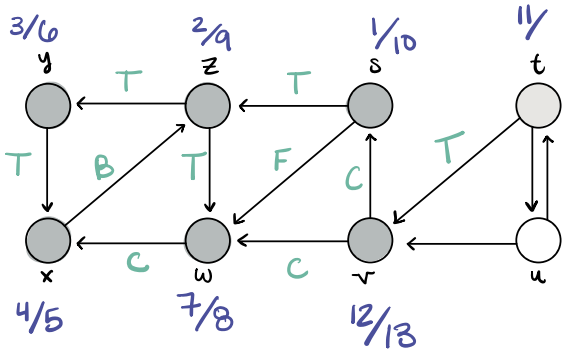

nothing left to explore, now black
TREE EDGE

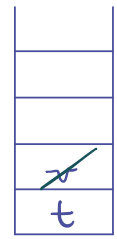


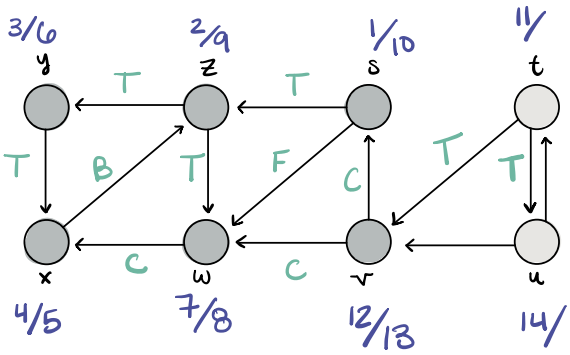

sw is FORWARD EDGE
b/c w is black
and d[w] < d[v].
Completely explored

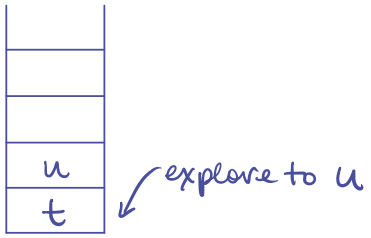
WE STARTED @ S and ended @ S
- now we move to t, keep time increment

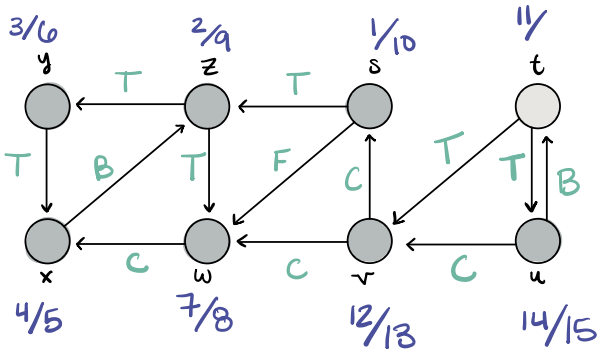


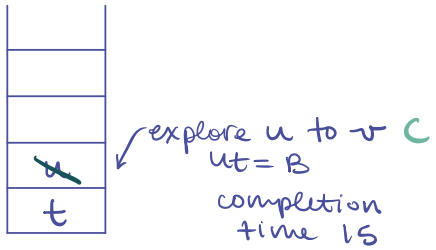



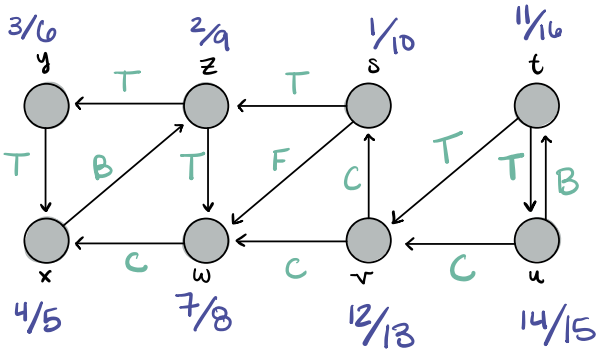

② explore v to vw + vs } both CROSS EDGES
① explore t to v

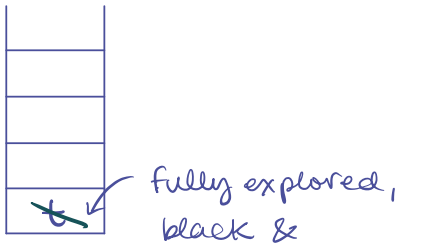

3/6
y
2/9
z
1/10
s
11/
t
T
T
T
B
T
F
C
T
T
x
w
v
u
C
C
4/5
7/8
12/13
14/


u
t
explore to u


3/6
y
2/9
z
1/10
s
11/
t
T
T
T
B
T
F
C
T
T
B
x
w
v
u
C
C
C
4/5
7/8
12/13
14/15


u
t
explore u to v C
ut = B
completion
time 15


3/6
y
2/9
z
1/10
s
11/16
t
T
T
T
B
T
F
C
T
T
B
x
w
v
u
C
C
C
4/5
7/8
12/13
14/15


t
fully explored,
black &
completion time 16

Minimization not max problems
but convert max to min and can use algo
Ex: Min Penalty instead of Max Profit

# Backtracking vs. Branch & Bound

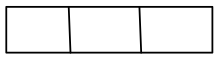
- both brute force strategy → try out all possible solutions and we want all of them as long as don't violate conditions (unlike dynamic programming which FINDS OPTIMAL)
(DESIRED SOL VS. OPTIMAL SOL)

| Depth First Search | Breadth First Search |
|---|---|
| ↑ State_Space_tree | ↑ State_Space_tree |

- both have all solutions satisfy bounding function
  - Kill node if violate this condition

---

**PROBLEM**

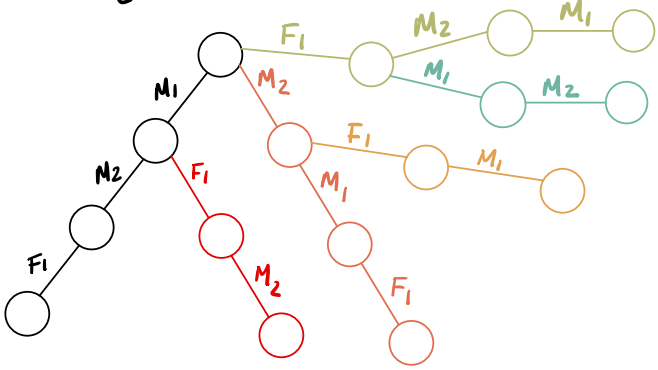


$M_1$ $F_1$
$M_2$

3! ways to assign 3 seats for 3 students

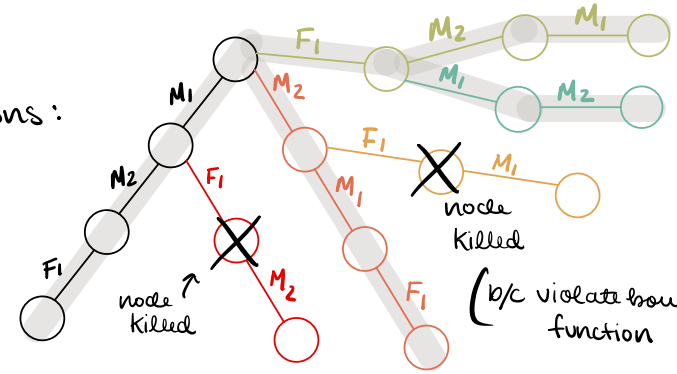
## SOLVE W/ BACKTRACKING:

① State_Space_tree based on DFS
all possible solutions




bounding function:
$F_1$ should not sit in middle

② desired solutions:

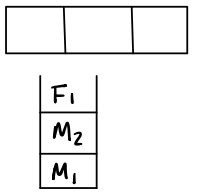


## SOLVE W/ BRANCH & BOUND:

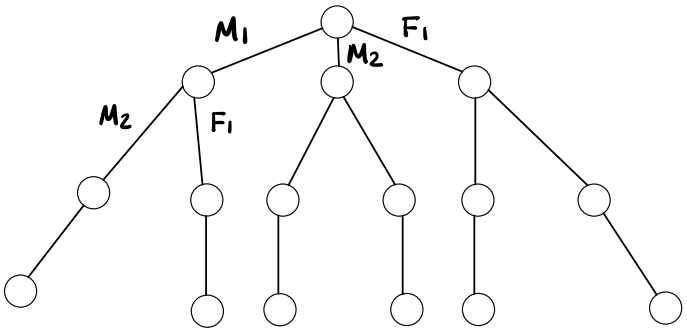
① state_space_tree based on BFS

Branch & Bound:

- BB_FIFO queue
- BB_LIFO stack
- BB_LC




pick whatever, doesn't change anything!




② desired solutions

Backtracking:
queen problem
graph coloring
sum of subsets
Hamiltonian cycle

# PRACTICE BACKTRACKING

## Sum of the subsets

$w\{1:6\} = \{5, 10, 12, 13, 15, 18\}$

Bounding function:

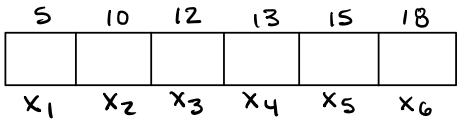
$m = 30$ ← max capacity = 30

RULE 1 $\sum_{i=1}^{k} w_i x_i + w_{k+1} \leq m$ (weight of added objs ≤ 30)


RULE 2 $\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i > m$ (added obj; remaining objects)

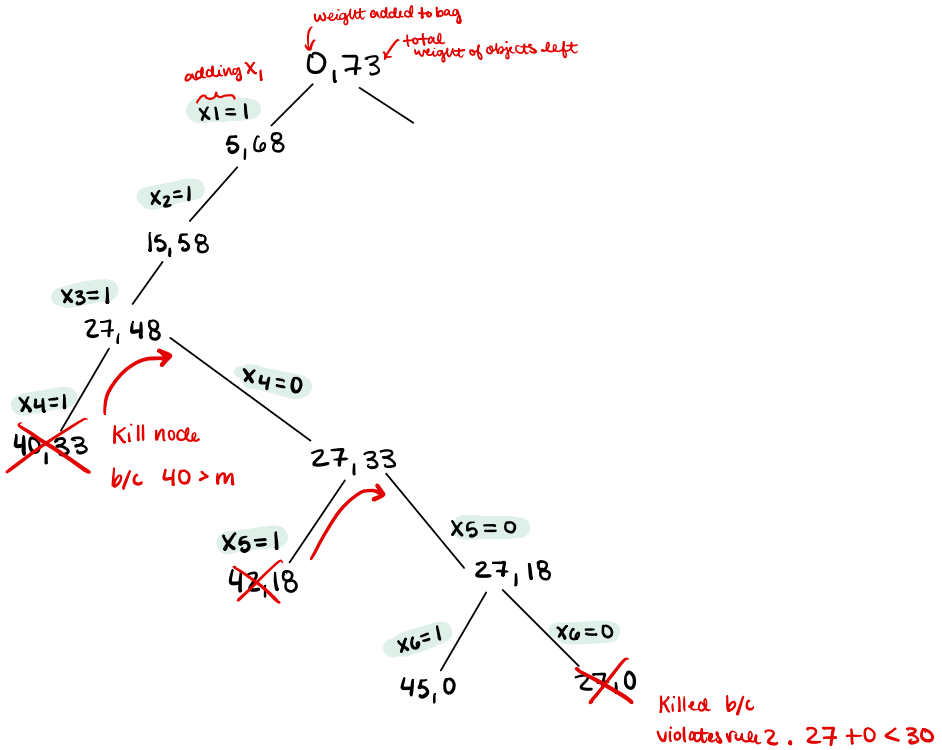
must be greater than 30

---

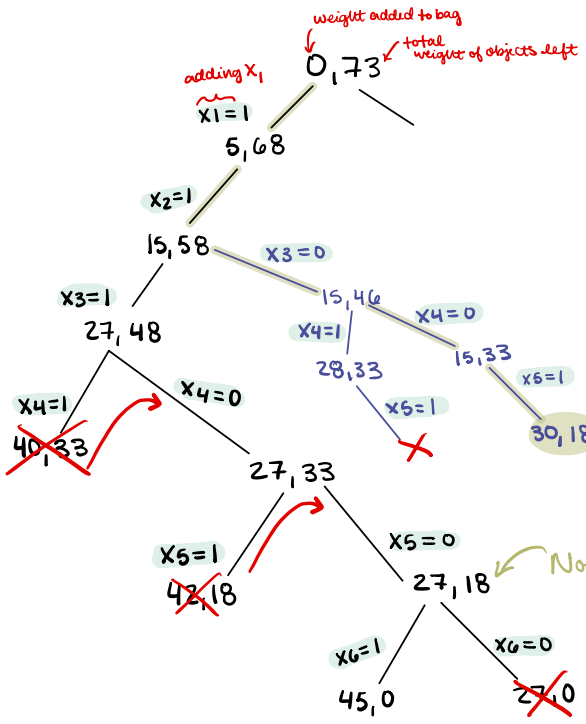



① state_space_tree based on DFS

Total weight = 5+10+12+13+15+18 = 73



# CONTINUED!




- Satisfies bounding function exactly

Not a solution even though does it violate bounding function b/c weight added (27) not exactly 30 & that's ideal

## (2) ONE OF THE SOLUTIONS:

| 5 | 10 | 12 | 13 | 15 | 18 |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 0 |
| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ |

$O(2^n)$

## branch & bound and state-space tree

- $2^n$ exponential time

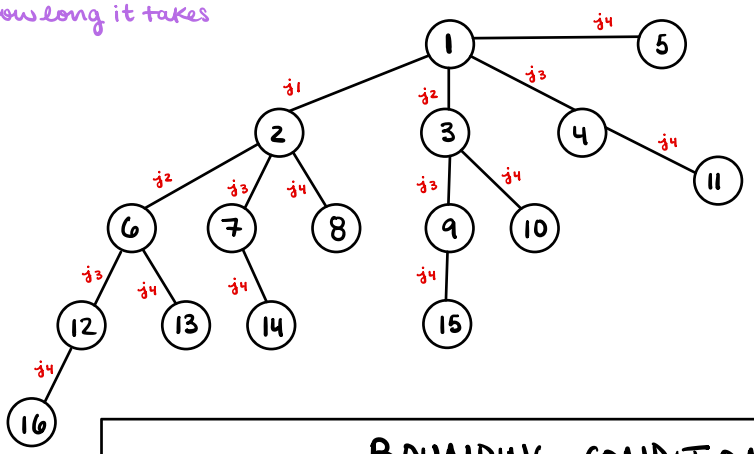
  So we use bounding condition to kill nodes

- branch & bound = minimization problem

---

| how long customer can wait → | jobs | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| | penalty | 5 | 6 | 10 | 3 |
| | deadline | 1 | 2 | 3 | 1 |
| ↑ how long it takes | time | 1 | 1 | 2 | 1 |

penalty ↑
consequence if job not included
(GOAL IS TO MINIMIZE THIS)




getting to $j_4$ from all nodes

### BOUNDING CONDITIONS

upper bound $(u) = \infty \rightarrow$ update

$u = \sum_{i \notin S} P_i$ = "sum of all penalties except the jobs included in the solution"

$\text{Cost} = C = \sum_{i \in S_k} P_i$ = "sum of all penalties till the last job we have considered in the solution"

$C < u$

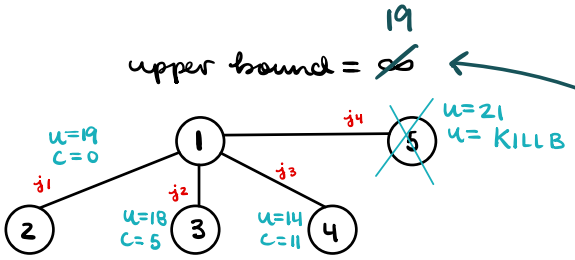
↳ worst case deadline

ex: Cost if only select $job_3$ and $job_4$?

penalty of $job_1$ + penalty of $job_2$
5 + 6
= 11

We minimize the upper bound

1. $u = \infty$
$\hat{c} = 0$




assume including $job_1$

$u = 19$ ← sum of all penalties excluding $job_1$
$\hat{c} = 0$ ← didn't leave out any jobs before $job_1$

2. $u = 19$
$\hat{c} = 0$

18
upper bound = ~~19~~

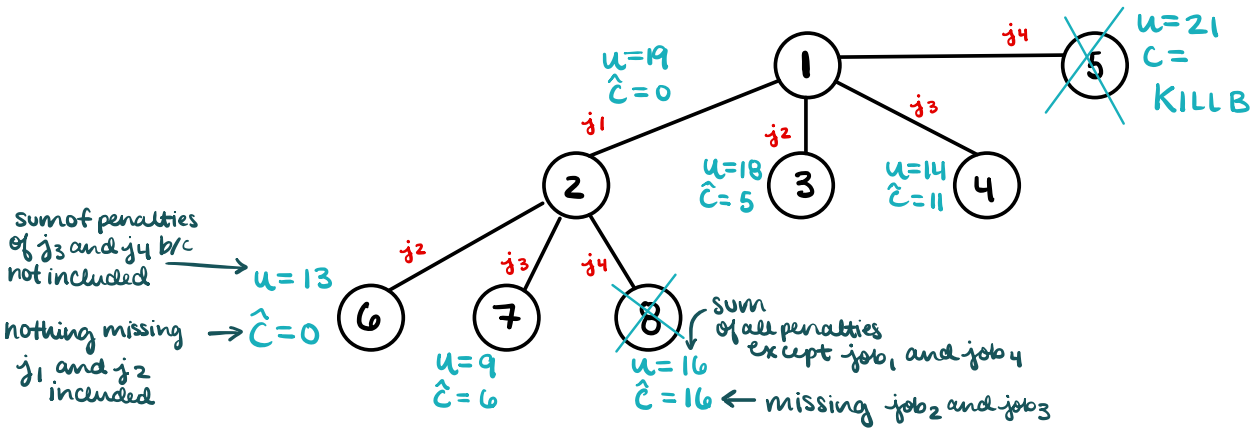
assume including $job_2$

$u = 18$ ← sum of all penalties excluding $job_1$
$\hat{c} = 5$ ← $job_1$ excluded, $\hat{c}$ = penalty

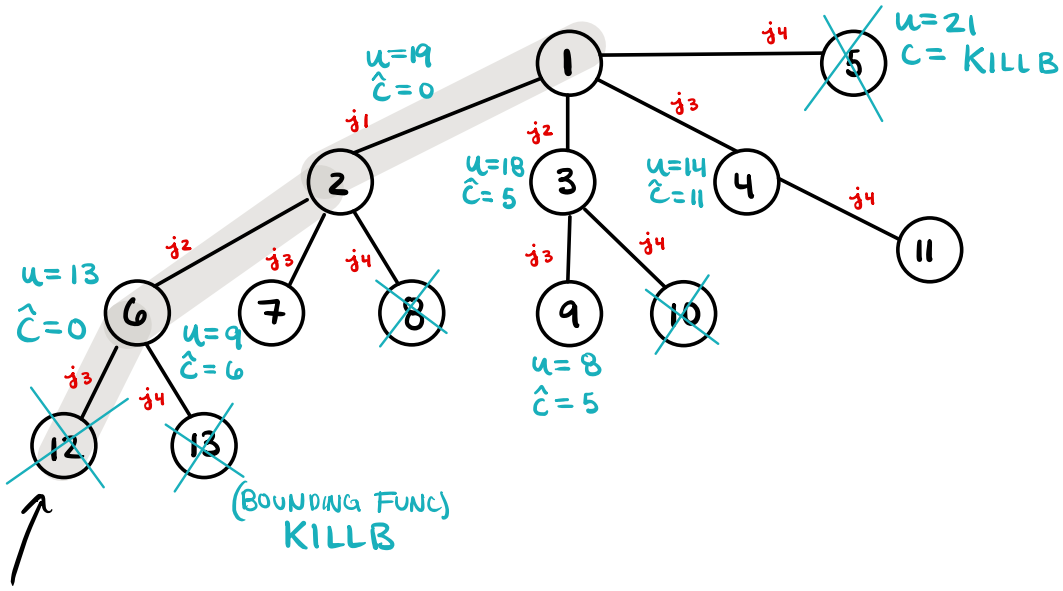
- and so on ($job_3$ & $job_4$)

- then next level
See tree above for level 1!

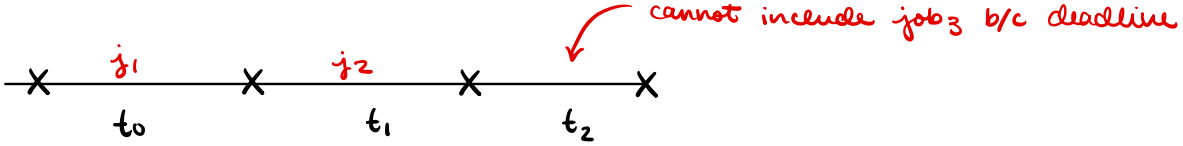
upper bound = _ ← updated each time



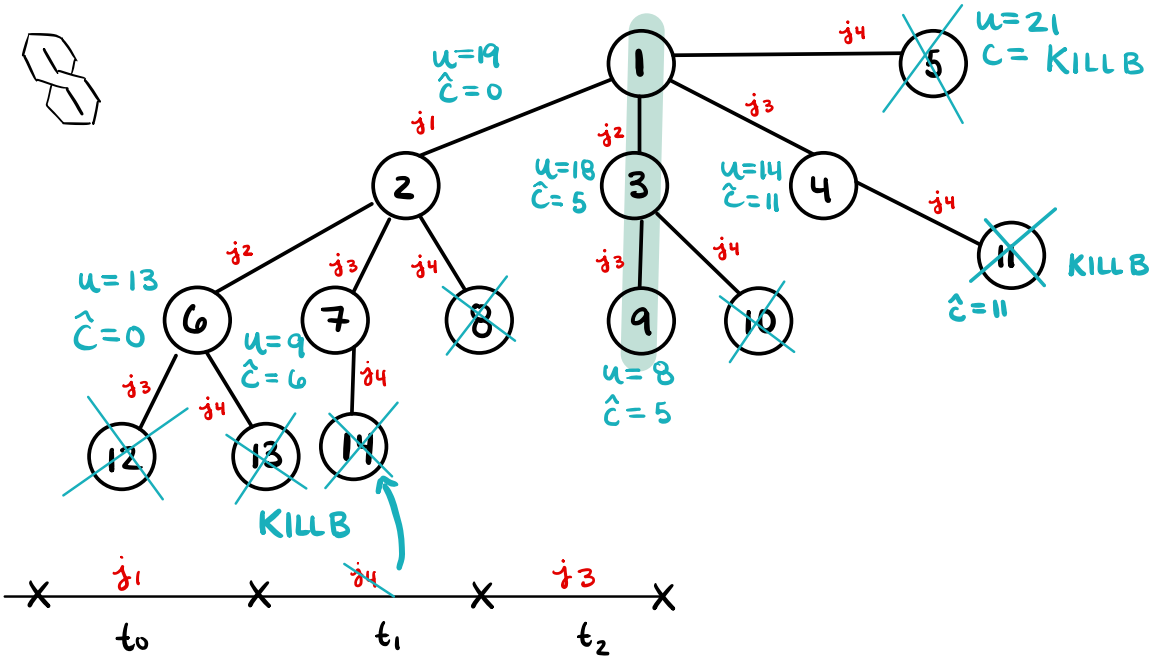

node killed b/c $u = \hat{c}$ (violates bounding function)




proceed or kill node?



8




$j_2 \rightarrow j_3 \rightarrow$ minimum = 8 penalty