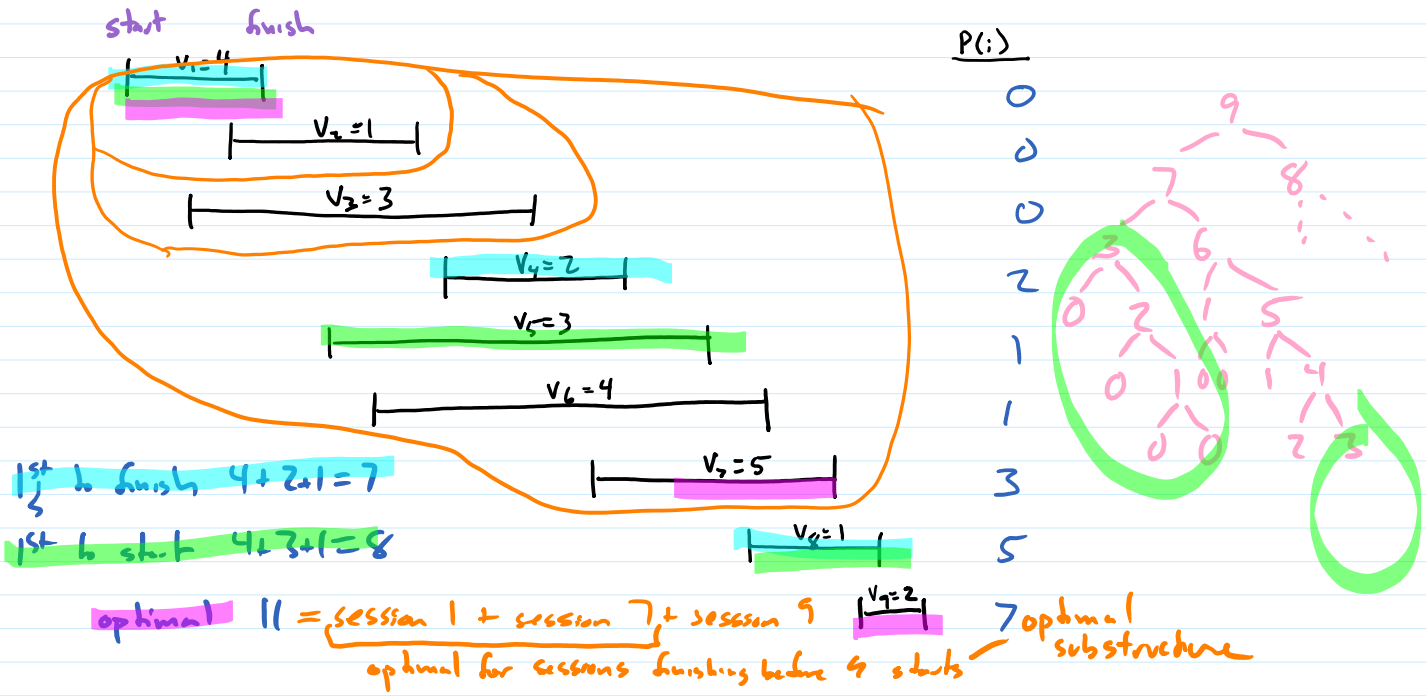
Weighted Interval Selection



OPT(i) = value of optimal selection using activities $\leq i$

= OPT(subproblem w/ last piece removed + other laps rewarded) + last piece

$$OPT(i) = \begin{cases} 0 & \text{if } i = 0 \\ \max\left(OPT(P(i)) + v_i,\ OPT(i-1)\right) & \end{cases}$$

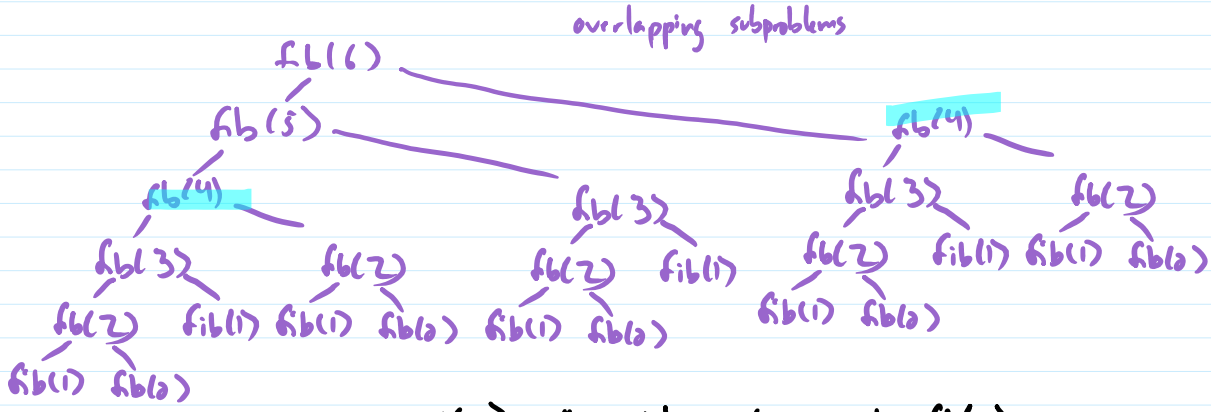
$OPT(P(i)) + v_i$: use interval i; $OPT(i-1)$: don't use interval i

```
COMPUTE-OPT-REC(n, v, P)
    return COMPUTE-OPT-HELPER(n, v, P)

COMPUTE-OPT-HELPER(i, v, P)
    if i = 0
        return 0
    if i is in memo                      ← memoized
        return memo[i]
    else memo[i] = max(OPT(P(i)) + v_i, OPT(i-1))      ← Θ(1) [assuming seen subproblems] per subprob
    return memo[i]

COMPUTE-OPT(n, v, P)                     ← Dynamic Programming
    memo  OPT ← (n+1)-elt array          Θ(n) subprobs to solve
    OPT[0] ← 0                           Θ(n) total
    Θ(n) steps → for i = 1 to n
        OPT[i] = max(OPT[P(i)] + v_i, OPT[i-1])
        CHOICE[i] = T if ... gave max, F otherwise
    Θ(n) total ← Θ(1) per iteration

SELECT(OPT, i)
    if i = 0 return []
    else if OPT[i] = OPT[P(i)] + v_i return SELECT(OPT, P(i)) + [i]
```

memo

or CHOICE[i] = "use interval i" if you've recorded CHOICE

```
SELECT(OPT, i)
    if i = 0 return []
    else if OPT[i] = OPT[p(i)] + v_i return SELECT(OPT, p(i)) + [i]
    else return SELECT(OPT, j-1)
```



$T(n)$ = # additions to compute $fib(n)$

$T(0) = T(1) = 0$

$T(n) = 1 + T(n-1) + T(n-2)$

adds in $fib(n-1)$ + # adds in $fib(n-2)$

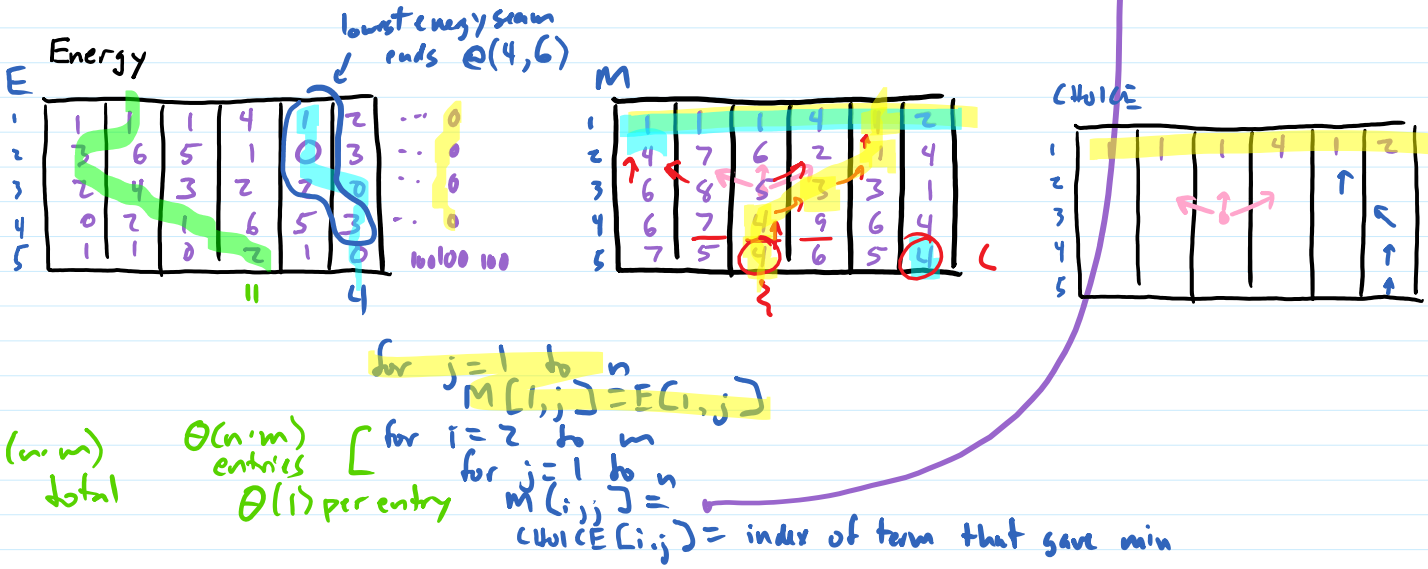
$$T(n) > Fib(n) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

Seam Carving

https://www.youtube.com/watch?v=6NcIJXTlugc

$M[i,j]$ = cost of min-cost seam from top to row i, col j

$$= \begin{cases} E[i,j] & \text{if } i = 1 \\ \min\left(M[i-1,j-1], M[i-1,j+1], M[i-1,j]\right) + E[i,j] \end{cases}$$



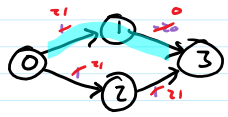

```
for j = 1 to n
    M[1,j] = E[1,j]

for i = 2 to m
    for j = 1 to n
        M[i,j] = ...
        choice[i,j] = index of term that gave min
```

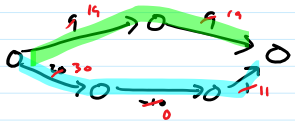
$\Theta(m \cdot n)$ total

$\Theta(n \cdot m)$ entries, $\Theta(1)$ per entry

Negative Weights

Negative weights → Dijkstra





simple reweighting doesn't work - paths w/more edges penalized more

If $s, v_1, v_2, \ldots, v_k$ is a shortest path $s \rightsquigarrow v_k$ then — Optimal Substructure

for each v_i, v_j $v_i, v_{i+1}, \ldots, v_j$ is shortest path $v_i \rightsquigarrow v_j$ — subpaths of shortest paths are shortest paths

$d_s(v)$ = tot weight of min-weight path $s \rightsquigarrow v$

if we know next-to-last vertex is u then $d_s(v) = d_s(u) + \ell(u,v)$
if we don't know, minimize over all possibilities

$$d_s(v) = \min_u d_s(u) + \ell(u,v)$$

but this isn't making subproblems smaller ☹

introduce extra parameter : # edges

Let $d_s(i, v)$ = total weight of min-weight path $s \rightsquigarrow v$ using $\le i$ edges

so $d_s(n-1, v)$ gives weight of min-weight path with any # verts (since $n-1$ is most shortest path could have)

$$= \begin{cases} 0 & \text{if } v = s \\ \infty & \text{if } i = 0 \text{ and } v \neq s \quad (\infty \text{ to mean no path}) \\ \min\left(\min_{u \in V}\left(d_s(i-1,u) + \ell(u,v)\right),\ d_s(i-1,v)\right) \end{cases}$$

SHORTEST-PATHS(n, s) — dynamic programming — $s = v_0$


```
M ← n×n array
for v = 0 to n-1   M[0,v] ← ∞        (base case)
    M[0,s] ← 0
for i = 1 to n-1                      (row-by-row)
    for v = 0 to n-1
        M[i,v] = min( M[i-1,v], min_{u=0..n-1} M[i-1,u] + ℓ(u,v) )
```

∞ if no edge

$\Theta(n^3)$
$\Theta(n^2)$ space

SHORTEST-PATHS(n, s)

```
M ← n×n array
for v = 0 to n-1   M[0,v] ← ∞
    M[0,s] ← 0

for i = 1 to n-1
    for v = 0 to n-1
        M[i,v] = min( M[i-1,v], min_{u=0..n-1} M[i-1,u] + ℓ(u,v) )
```

$\Theta(n^3)$ time
$\Theta(n)$ space

only need 1 row above
also only need $M[i,v]$ = weight of some path $s \rightsquigarrow v$
and $M[i,v] \le$ weight of shortest $s \rightsquigarrow v$ with $\le i$ edges

SHORTEST-PATHS(n, s)
(BELLMAN-FORD)

```
M ← n×n array
for v = 0 to n-1   M[0,v] ← ∞
    M[0,s] ← 0

for i = 1 to n-1
    for each edge (u,v)
```

skip edges that don't exist; doesn't matter what order we do them in

$\Theta(n \cdot m)$

M[0,s] ← 0

for i = 1 to n-1

~~for v = 0 to n-1~~ for each edge (u, v)

M[v] = min(m[v], ~~min~~ M[u] + ℓ(u,v))

↓ don't exist; doesn't matter

what order we do them in

O(n·m)