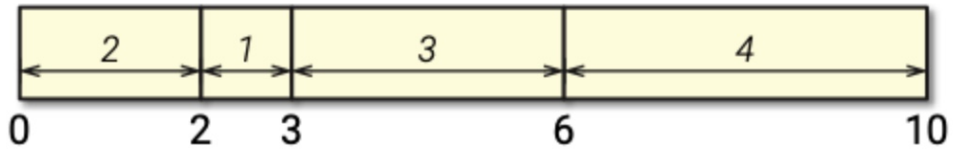
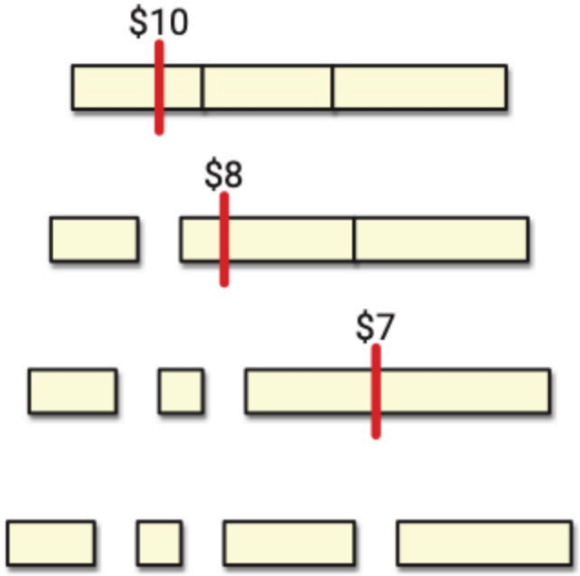
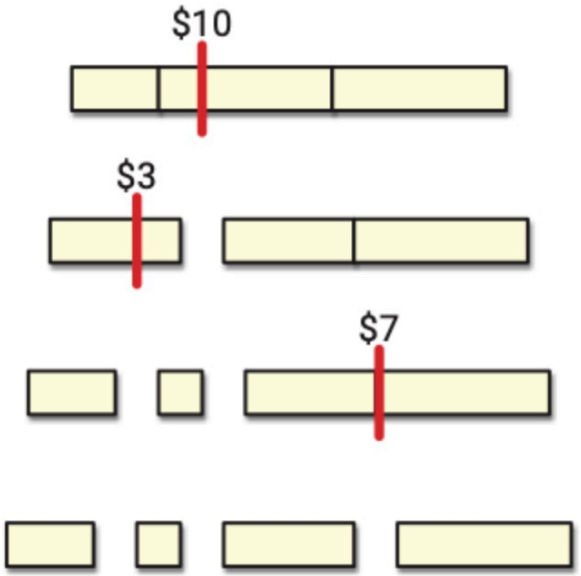
Given a long plank of wood. Need to cut it at marked locations, Charged length of a subplank to cut it.

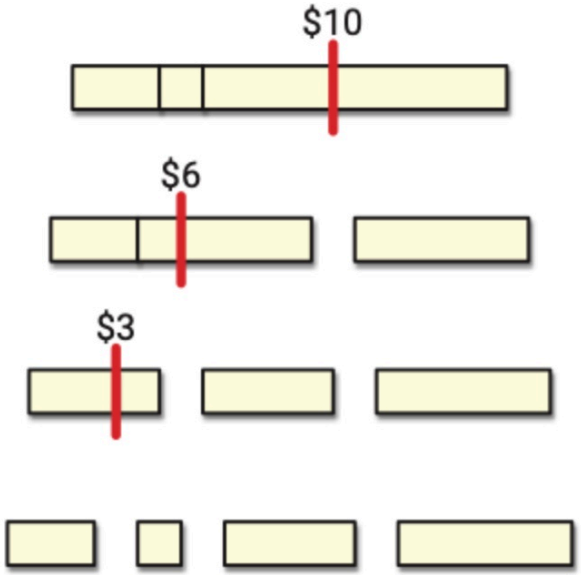




$25



$20



$19

Len[1..n] where Len[i]: desired length of ith board from left.

↑ n = 4 Len = [2, 1, 3, 4].

Subproblems are subarrays!

MinCost(i, k): Min total cost to decompose a subplank consisting of boards i through k (with lengths from Len[i..k]).

Need to compute: MinCost(1, n).

$$MinCost(i,k) = \begin{cases} 0 & i = k \\ \left(\sum_{j=i}^{k} Len[j]\right) + \min_{i < r \le k} \left\{ MinCost(i, r-1) + MinCost(r, k) \right\} & \text{otherwise} \end{cases}$$

$$TotLen(i,k) := \sum_{j=i}^{k} Len[j]$$

```
InitTotLen(Len[1..n]):
  for i ← 1 to n
      TotLen[i, i−1] ← 0
      for k ← i to n
          TotLen[i,k] ← TotLen[i,k−1] + Len[k]
```

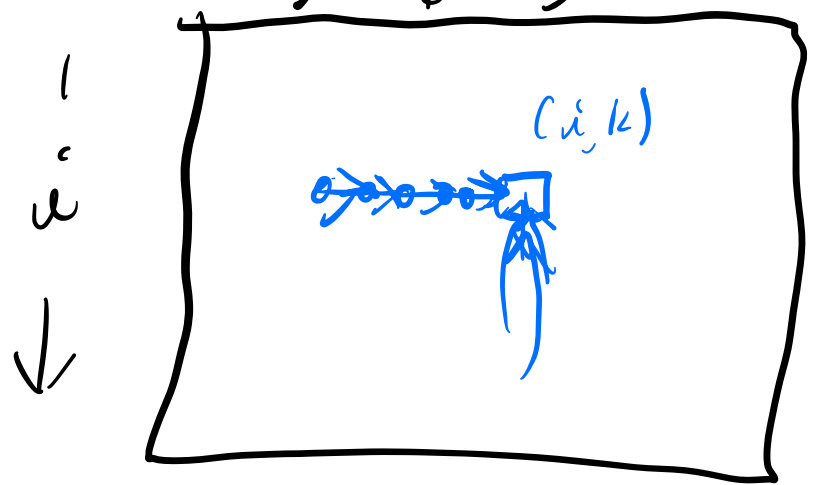
$$MinCost(i,k) = \begin{cases} 0 & \text{if } i = k \\ TotLen[i,k] + \min_{i<r\le k} \{ MinCost(i,r-1) + MinCost(r,k) \} & \text{otherwise} \end{cases}$$

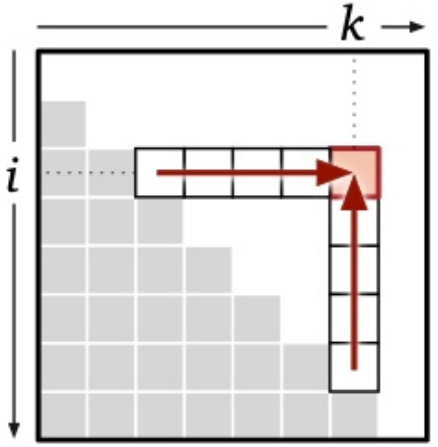
Subproblems: $1 \le i \le n$

$1 \le k \le n$, but really $1 \le i \le k \le n$.

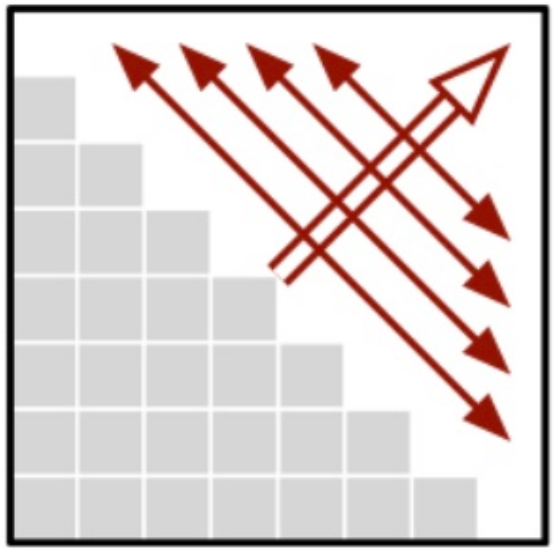
Memoization: 2D array $MinCost[1..n, 1..n]$.

Dependencies:



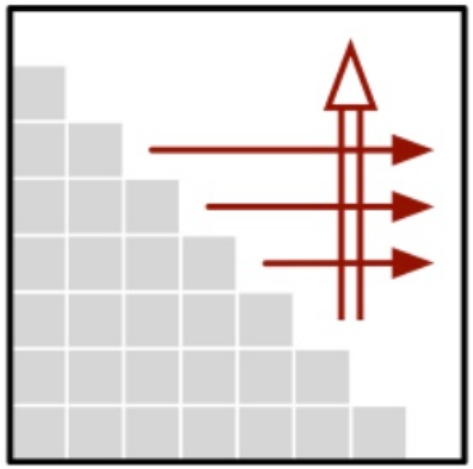


Eval order:

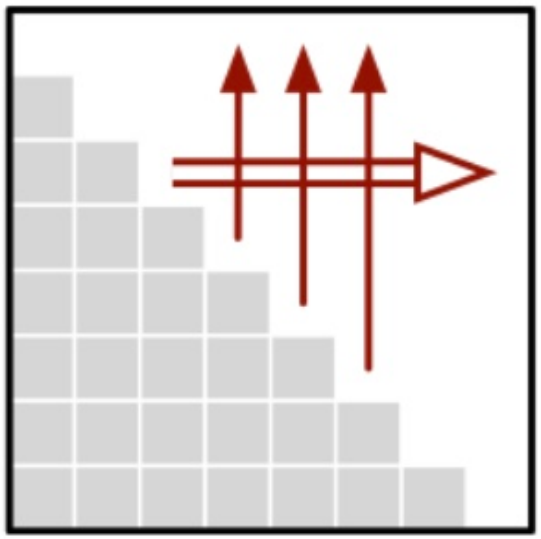


```
ComputeMinCost(i, k):
    MinCost[i, k] ← ∞
    for r ← i + 1 to k
        tmp ← MinCost[i, r − 1] + MinCost[r, k]
        if MinCost[i, k] > tmp
            MinCost[i, k] ← tmp
    MinCost[i, k] ← MinCost[i, k] + TotLen[i, k]
```

```
Woodcutter(Len[1 .. n]):
    InitTotLen(Len[1 .. n])
    for r ← 1 to n
        MinCost[i, i] ← 0
    for d ← 1 to n − 1
        for i ← 1 to n − d
            ComputeMinCost(i, i + d)
    return MinCost[1, n]
```



```
Woodcutter2(Len[1 .. n]):
    InitTotLen(Len[1 .. n])
    for i ← n down to 1
        MinCost[i, i] ← 0
        for j ← i + 1 to n
            ComputeMinCost(i, j)
    return MinCost[1, n]
```



```
WOODCUTTER3(Len[1 .. n]):
  INITTOTLEN(Len[1 .. n])
  for j ← 1 to n
      MinCost[j, i] ← 0
      for i ← j down to 1
          COMPUTEMINCOST(i, j)
  return MinCost[1, n]
```

O(n)

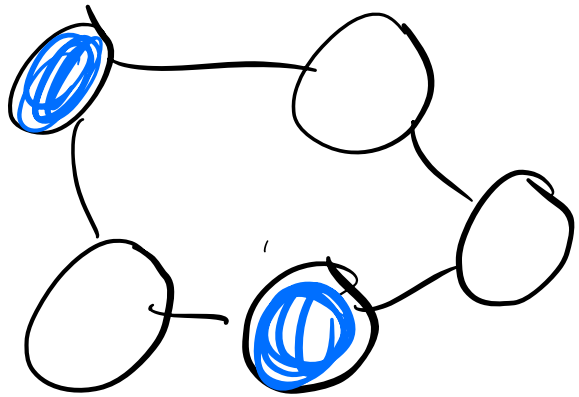
Time: $O(n^3)$

Really finding a best binary search tree.

Knuth: $O(n^2)$

Hu Tucker: $O(n \log n)$

Given a graph, $G = (V, E)$, an independent set $S \subseteq V$



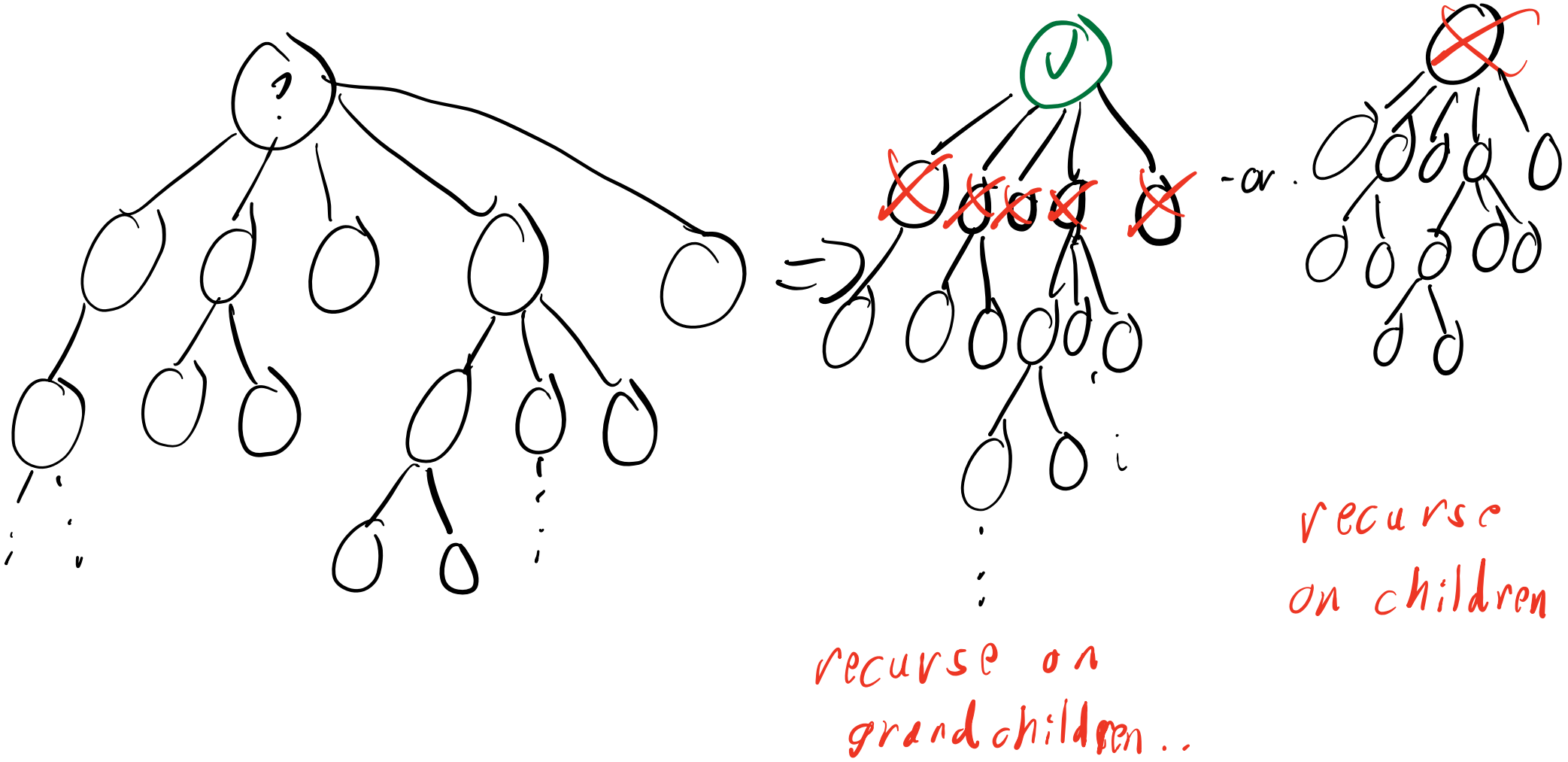
↑ subset of vertices

where no two vertices in S share an edge.

Problem: Find a maximum size independent set.

NP-hard :(

But what if G is a (rooted) tree T?



MIS(v): size of the largest independent set of v's subtree.

If T is rooted at r, want MIS(r).

$$MIS(v) = \max \left\{ \sum_{w \downarrow v} MIS(w),\ 1 + \sum_{w \downarrow v} \sum_{x \downarrow w} MIS(x) \right\}$$

$w \downarrow v$: w is a child of v

Subproblems: One per node v.

Memoization: $v.MIS$ for each node v.

Dependencies: children & grandchildren

Eval order: postorder

Code:

```
TreeMIS(T):
  r ← root of T
  for each node v in postorder:
    skip ← 0
    for each child w of v
      skip ← skip + w.MIS
    keep ← 1
      for each grandchild x of v
      keep ← keep + x.MIS
    v.MIS ← max{skip, keep}
  return r.MIS
```

Time: $O(n)$: each node is a child/grandchild once