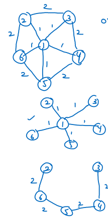


optional.
$\rightarrow S = \{1\} \ldots\ldots$ (100)

$V \setminus S = \{2, 3, 4, 5, 6\}$
↓
mandatory

⇐ Steiner Tree
cost : 5

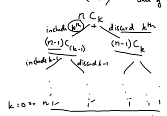
⇐ Steiner Tree
cost : 8

Dijkstra's

Dynamic Programming

1. Computing the binomial Coefficient

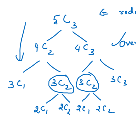
nC_k $\binom{n}{k}$: to select k objects out of 'n' objects.



$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 & k = 0 \text{ or } k = n \\ 0 & \text{otherwise} \end{cases}$$

```
int select(n, k) {
    if k == 0 or k == n
        return 1;
    else
        return (select(n-1, k-1) +
                select(n-1, k));
}
```

$T(n,k) = O(nC_k)$

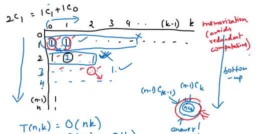


⇐ redundancy in computations

Overlapping sub-problems

Top-down

$2C_1 = 1C_1 + 1C_0$



memorization (avoids redundant computations)

bottom-up

$T(n,k) = O(nk)$
$S(n,k) = O(nk)$ $O(k)$
↑ You need only the previous row to compute the current row.

⇑
A Dynamic Programming Algorithm.

Divide and Conquer	Dynamic Programming
- Top down	. Bottom up
. Redundant sub-problems are possible	- subvert redundant sub-problems through memorization
- Call-Stack (implicit)	- Memory needed for storing memorization table
- Recursive	- Iterative
. .	. . .

II. Tendering Change :

$D = \{1, 4, 6\}$ ⇐ denomination

$n = 8$ units

Greedy: (6) (1) (1) = 8
3 coins.

Dynamic prg. : (4) (4) = 8
2 coins

	0	1	2	3	4	5	6	7	8
$d_1 = 1$	0	1	2	3	4	5	6	7	8
$d_2 = 4$	0	1	2	3	1	2	3	4	2
$d_3 = 6$	0	1	2	3	1	2	1	2	2

(amount →; (1,4), 1+(2,0) = 1, (1,8), (2,4), (2,8), (3,8), (3,2))

$[i,j]$: optimal number of coins to tender a change for amount 'j' using coins of denomination only from d_1 to d_i

$i = 1 / j < d_i \Rightarrow +\infty$ ⇒ All out of bounds entries = $+\infty$

$T[i,j] \leftarrow \min(T[(i-1), j], 1 + T[i, j-d_i])$
(excluding coinage i) (including coinage i)

$T[1,2] \leftarrow \min(T[0,2], 1+T[1,2-1])$
$\leftarrow \min(+\infty, 1+T[1,1])$
$\leftarrow \min(+\infty, 1+1)$ ← from table
$\leftarrow 2$

$T[2,4] \leftarrow \min(T[1,4], 1+T[2,0])$
$\leftarrow \min(4, 1)$
$T[2,4] \leftarrow 1$

```
tenderChange(amount, D) {
    T[1...|D|, 0.....amount]
    for i <- 1 to |D|
        T[i,0] <- 0

    for i <- 1 to |D| {
        for j <- 1 to amount {
            if (i == 1 and j < D(i))
                return (+inf)
            else if (i == 1) {
                T(i,j) <- 1 + T[i, j-D(i)]
            else if (j < D(i)) {
                T(i,j) <- T[i-1, j]
            else
                T(i,j) <- min(T(i-1,j), 1+T(i,j-D(i)))
        }
    }
    return T[|D|, amount];
}
```

$T(|D|, amount) = O(|D| \times amount)$

$S(|D|, amount) = O(|D| \times amount)$