

0 – 1 Knapsack summary

- The problem: We are given a knapsack of capacity W and a set of n items, where item i is worth $v[i]$ and has weight $w[i]$ pounds. The weights $w[i]$ and the total weight W are **integers**. Fill the knapsack so that the overall value of all items in the knapsack is maximized.
- Notation and choice of subproblem: Let $optknapsack(k, w)$ return the maximal value for a knapsack of size w using items among items 1 through k . To solve our problem we call $optknapsack(n, W)$.

for simplicity, we'll assume that the weight and value arrays $w[], v[]$ are global

- Recursive definition of $optknapsack(k, w)$:

```
@returns the max value to pack a knapsack of capacity  $w$  using the first  $k$  items
OPTKNAPSACK( $k, w$ )
1  if ( $w == 0$ ): return 0
2  if ( $k \leq 0$ ): return 0
3  if ( $w[k] \leq w$ ):  $with = v[k] + OPTKNAPSACK(k - 1, w - w[k])$ 
4  else:  $with = 0$ 
5   $without = OPTKNAPSACK(k - 1, w)$ 
6  RETURN  $\max \{ with, without \}$ 
```

- Why correct? It tries both possibilities for each item (with or without) and recurses on the rest, which is correct bec. it has optimal substructure (why?).
- Dynamic programming, recursive (top-down) with memoization:

```
Create a table a  $table[1..n][1..W]$ , where  $table[i][w]$  will store the result of  $optknapsack(i, w)$ .
Initialize all entries in the table as 0.
Call  $optknapsackDP(n, W, table)$  and return the result.
```

```
@returns the max value to pack a knapsack of capacity  $w$  using the first  $k$  items
OPTKNAPSACKDP( $k, w, table$ )
1  if ( $w == 0$ ): return 0
2  if ( $k == 0$ ): return 0
3  IF ( $table[k][w] \neq 0$ ): RETURN  $table[k][w]$ 
4  IF ( $w[k] \leq w$ ):  $with = v[k] + OPTKNAPSACKDP(k - 1, w - w[k], table)$ 
5  ELSE:  $with = 0$ 
6   $without = OPTKNAPSACKDP(k - 1, w, table)$ 
7   $table[k][w] = \max \{ with, without \}$ 
8  RETURN  $table[k][w]$ 
```

Running time for $optknapsackDP(n, W)$: $O(n \cdot W)$

- Dynamic programming, iterative (bottom-up):

```

@returns the max value to pack a knapsack of capacity  $W$  using the  $n$  items
OPTKNAPSACKDP_ITERATIVE()
1  create table[0..n][0..W] and initialize all entries to 0
2  for  $k = 1; k \leq n; k++$ 
3      for  $w = 1; w \leq W; w++$ 
4          if ( $w[k] \leq w$ ) with =  $v[k] + table[k - 1][w - w[k]]$  else with = 0
5          without =  $table[k - 1][w]$ 
6           $table[k][w] = \max \{ \text{with}, \text{without} \}$ 
7  RETURN  $table[n][W]$ 

```

Running time: $O(n \cdot W)$

- Computing full solution (without storing additional information while filling the table):

```

@param: The table  $table[1..n][1..W]$  as computed above, where  $table[i][x]$  stores the max
value to pack a knapsack of weight  $x$  using items  $1..i$ .
@return: prints the set of items corresponding to  $table[n][W]$ 
FINDITEMS(table)
1   $i = n, w = W$ 
2  while ( $i > 0$  and  $w > 0$ )
3      // is the value  $table[i][w]$  achieved with item  $i$  or without?
4      if  $table[i][w] == v[i] + table[i - 1][w - w[i]]$  //WITH item i
5          output item  $i$ 
6           $w = w - w[i], i = i - 1$ 
7      else // WITHOUT item i
8           $i = i - 1$ 

```

Running time: $O(n)$, with no extra space