Elements of Dynamic Programming

- What feature did we use?
- What should we look for to use again?

- "Optimal Substructure"
  Optimal solution contains optimal subproblems
- "Repeated Subproblems"
  The same subproblems arise in various ways

12

Knapsack (Subset-Sum) Problem

- Given:
  - integer K (knapsack size)
  - n object sizes $x_1, x_2, \ldots, x_n$
- Find:
  - Subset S of {1,…, n} such that

$$\sum_{i \in S} x_i = K$$

13

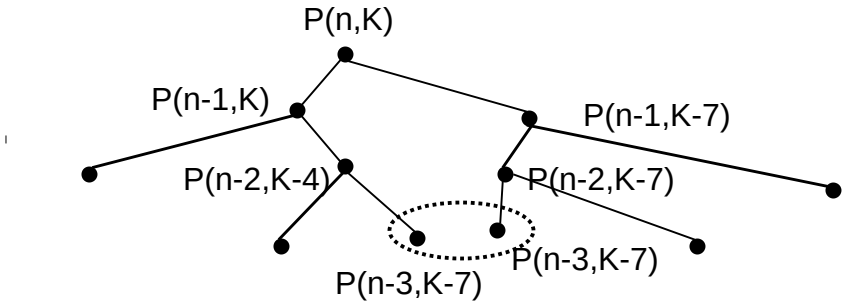
Recursive Algorithm

- Let P(n,K) denote the problem to solve for K and x_1, x_2, … , x_n
- For n>0,
  - P(n,K) has a solution iff either P(n-1,K) or P(n-1,K-x_n) has a solution
- For n=0
  - if K=0 then P(0,K) has a trivial solution of an empty set S
  - if K>0 then P(0,K) has no solution

14

Recursive calls

- Recursive calls on list …,3, 4, 7



15

Common Sub-problems

- Only sub-problems are P(i,k) for
  - i = 0,1,..., n
  - k = 0,1,..., K
- Dynamic programming solution
  - Table entry for each P(i,k)
    - **exist** flag - does a solution exist?
    - **belong** flag - is x_i a part of this solution?
  - Initialize P(0,k) for k=0,...,K
  - Compute all P(i,?) from P(i-1,?) for i>0

16

Dynamic Knapsack Algorithm

- P(0,0).exist = true
- for k=1 to K; P(0,k).exist = false; end for
- for i=1 to n do
  - for k=0 to K do
    - P(i,k).exist = false
    - if P(i-1,k).exist then
      - P(i,k).exist = true
      - P(i,k).belong = false
    - else if k >= x_i then
      - if P(i-1,k-x_i).exist then
        - P(i,k).exist = true
        - P(i,k).belong = true
  - end for
- end for

Time O(nK)

17

Sample execution on 2, 3, 4, 7 with K=15

18