CS F364
Design & Analysis of Algorithms

Algorithm Design Techniques

0/1 Knapsack Problem: Dynamic Programming Algorithm:

- **Time Complexity**
- **Pseudo-Polynomial Time Algorithms**
- **Space Complexity**
- **Limitations**
- **Problem Variants**

EXAMPLE – 0/1 KNAPSACK – DP SOLUTION

Known (Atomic) Solutions: $P(0, w)=0$ forall w and $P(k, 0)=0$ forall k

Recursive structure: $P(k,w) =$

$$\begin{cases} P(k-1, w) & \text{if } w_k > w \\ \max\{ P(k-1, w), P(k-1, w-w_k) + p_k \} & \text{otherwise} \end{cases}$$

```
Profit(k,w)
 // assume output array Pf[0..N][0..Wmax]
//  assume array wt[1..N] of weights and p[1..N] of prices
  {    for (k=0; k<=N; k++)  Pf[k,0] = 0;
       for (w=0; w<=Wmax; w++) Pf[0,w] = 0;
       for (k = 1; k<=N; k++)
           for (w=1; w<=Wmax; w++)
              Pf[k,w] = (wt[k] > w) ? Pf[k-1,w] :
                          max(Pf[k-1,w], Pf[k-1,w-wt[k]]+p[k]);
}
```

EXAMPLE – 0/1 KNAPSACK – DP SOLUTION

```
Profit(k,w)
 // assume output array Pf[0..N][0..Wmax]
//  assume array wt[1..N] of weights and p[1..N] of prices
 {   for (k=0; k<=N; k++)  Pf[k,0] = 0;
     for (w=0; w<=Wmax; w++) Pf[0,w] = 0;
     for (k = 1; k<=N; k++)
        for (w=1; w<=Wmax; w++)
            Pf[k,w] = (wt[k] > w) ? Pf[k-1,w] :
                        max(Pf[k-1,w], Pf[k-1,w-wt[k]]+p[k]);
}
```

- Time Complexity: O (N*Wmax)
  - Is this polynomial time? Why or Why not?
  - What if Wmax is $O(2^N)$?
- Pseudo-polynomial time algorithms
  - Complexity is defined in terms of max. input size
    - e.g. N*Wmax is polynomial in the size of the set of items

Example – 0/1 Knapsack – DP Solution

```
Profit(k,w)
 // assume output array Pf[0..N][0..Wmax]
//  assume array wt[1..N] of weights and p[1..N] of prices
  {   for (k=0; k<=N; k++)  Pf[k,0] = 0;
      for (w=0; w<=Wmax; w++) Pf[0,w] = 0;
      for (k = 1; k<=N; k++)
          for (w=1; w<=Wmax; w++)
              Pf[k,w] = (wt[k] > w) ? Pf[k-1,w] :
                              max(Pf[k-1,w], Pf[k-1,w-wt[k]]+p[k]);
}
```

- Space Complexity: O(N*Wmax)
  - Can this be reduced? If so, how? If not why not?
- P[k,_] is dependent only on P[k-1]
  - At any time only 2 rows (index k and k-1) are needed.
- Exercise: *Rewrite the procedure after pruning unwanted rows in the profit matrix.*

EXAMPLE – 0/1 KNAPSACK – DP SOLUTION

```
Profit(k,w)
 // assume output array Pf[0..N][0..Wmax]
//  assume array wt[1..N] of weights and p[1..N] of prices
 {   for (k=0; k<=N; k++)  Pf[k,0] = 0;
      for (w=0; w<=Wmax; w++) Pf[0,w] = 0;
      for (k = 1; k<=N; k++)
           for (w=1; w<=Wmax; w++)
                Pf[k,w] = (wt[k] > w) ? Pf[k-1,w] :
                                 max(Pf[k-1,w], Pf[k-1,w-wt[k]]+p[k]);
}
```

- P[k,_] is dependent only on P[k-1]
  - At any time only 2 rows (index k and k-1) are needed.
- What about columns ? Can they be pruned?
  - Number of columns needed at any time: $1+\max_j w_j$
- Exercise: *Rewrite the procedure after pruning unwanted rows in the profit matrix*

EXAMPLE – 0/1 KNAPSACK – DP SOLUTION

- Validity of assumptions:
  - What if weights are not integers?
  - Rational numbers? Real numbers?

Example – 0/1 Knapsack – DP Solution

- Validity of assumptions:
  - What if weights are not integers?
  - Rational numbers? Real numbers?
- Consider weights to be rationals
  - i.e. normalized fractions of the form (p_j / q_j)
  - Multiply all weights by lcm_j (q_j)
    - All (scaled) weights are integers:
      - Scaling weights does not affect profits.
  - Impact on complexity:
    - Time : N * ($lcm_j(q_j)$ * Wmax)
    - Space: 2 * (1+$max_j(p_j)$) * $lcm_j(q_j)$

EXAMPLE – 0/1 KNAPSACK – DP SOLUTION

- Integer weights can also be normalized (i.e. scaled)
  - If Integer weights are divided by $gcd_j(w_j)$ the time and space complexities can be reduced by the same factor.
    - When is this useful?
  - Are there ways reducing the complexity factor dependent on weights?
    - Relook at the recurrence relation.