CS 341: Algorithms
Module 6: Dynamic Programming

Armin Jamshidpey, Eugene Zima

Based on lecture notes by many previous CS 341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Spring 2019

Integer Knapsack

Problem specification:
We are given n objects and a knapsack.
Each object i has a positive weight w_i and a positive value v_i.
The knapsack can carry a weight not exceeding W. Fill the knapsack so that the value of objects in the knapsack is maximized.
Brute force:
Try all possibilities. An object can be in or out and we sum weights to be sure we are not over W. This has complexity $\Theta(n2^n)$.
Greedy:
At each step add the object with the highest v_i/w_i ratio. Does not work. Counterexample?

Integer Knapsack - DP

Recall that objects are numbered from 1 to n.

Definition of a subproblem

Let $V[i, j]$ be the maximum value of the objects, selected from the first i objects, that can fit into a knapsack with upper weight limit j (the optimal value will be found in $V[n, W]$).

Key observation:

We either use object i in the optimal solution or we do not.

Suppose object i is not in the Knapsack. Then there is no difference between $V[i-1, j]$ and $V[i, j]$.

Suppose object i is in the Knapsack. Our claim, for this case, is that $V[i, j] = V[i-1, j-w_i] + v_i$.

Consider an optimal selection extracted from the first $i-1$ objects with a weight limitation of $j - w_i$.

Integer Knapsack: Derivation of the Recurrence

Looking at only these first $i-1$ objects, we can assume we have an optimal selection that is not more valuable than those chosen from the first $i-1$ objects as used in $V[i,j]$.

This is true because:

A more valuable selection from objects 1 to $i-1$ could be extended with object i and we would get a total value in excess of $V[i,j]$ in contradiction of the fact that $V[i,j]$ is optimal. So the value of $V[i,j]$ must be v_i plus the optimal solution for the first $i-1$ objects with a weight limitation of $j-w_i$.

Considering the above facts we are able to make up the following recurrence for $V[i,j]$:

$$V[i,j] = \max\{V[i-1,j], v_i + V[i-1,j-w_i]\}$$

Base case: $V[0,j] = 0$.

Order of computation:

Use row-order from top-left down to the bottom-right corner.

Knapsack Problem: Pseudo-code for DP

```
for j := 0 to W do
  V[0,j]:=0;
for i := 1 to n do
  for j := 1 to W do
    sol := V[i-1, j];
    if (w[i] <= j) then
      othersol := V[i-1, j-w[i]] + v[i];
      if (othersol > sol) then
        sol := othersol;
    V[i, j] := sol;
return V[n, W];
```

Complexity? $\Theta(nW)$. Is it good or bad???

Integer Knapsack: Notes on Pseudo-code

Note

We can make the program more memory efficient.
Note that to compute value $V[i,j]$, we need only the cells from the previous line and to the left of $V[i-1,j]$ (including $V[i-1,j]$).

```
for j := 0 to W do
  V[j] := 0;
for i := 1 to n do
  for j := W downto 1 do
    sol := V[j];
    if (w[i] <= j) then
      othersol := V[j-w[i]] + v[i];
      if (othersol > sol) then
        sol := othersol;
    V[j] := sol;
return V[W];
```

Integer Knapsack: Notes on Pseudo-code

More simplifications..

```
for j := 0 to W do
  V[j] := 0;
for i := 1 to n do
  for j := W downto 1 do
    if (w[i] <= j) then
      othersol := V[j-w[i]] + v[i];
      if (othersol > V[j]) then
        V[j] := othersol;
return V[W];
```

Integer Knapsack: Notes on Pseudo-code

Recovery of the solution added

```
for j := 0 to W do
   V[j] := 0; D[j] := 0;
for i := 1 to n do
  for j := W downto 1 do
    if (w[i] <= j) then
      othersol := V[j-w[i]] + v[i];
      if (othersol > V[j]) then
        V[j] := othersol; D[j]:= i;
print V[W];
\\ recover the items in knapsack
j:=W;
while (j>0) and (D[j]>0) do
  print(D[j]); j:=j-w[D[j]];
```

Minimum Length Triangulation

Problem 4.4

Minimum Length Triangulation v1
Instance: n points $q_1, \cdots, q_n$ in the Euclidean plane that form a convex $n - gon$ P.
Find: A triangulation of P such that the sum S_c of the lengths of the $n - 3$ chords is minimized.

Problem 4.5

Minimum Length Triangulation v2
Instance: n points $q_1, \cdots, q_n$ in the Euclidean plane that form a convex $n - gon$ P.
Find: A triangulation of P such that the sum S_p of the perimeters of the $n - 2$ triangles is minimized.

Let L denote the perimeter of P. Then we have that $S_p = L + 2S_c$. Hence the two versions have the **same optimal solutions**.

Problem Decomposition

We consider version 2 of the problem.
The edge $q_n q_1$ is in a triangle with a third vertex q_k, where $k \in 2, \cdots, n-1$.
For a given k, we have:

1. the triangle $q_1 q_k q_n$,
2. the polygon with vertices $q_1, \cdots, q_k$,
3. the polygon with vertices $q_k, \cdots, q_n$.

The optimal solution will consist of optimal solutions to the **two subproblems** in (2) and (3), along with the triangle in (1).

Recurrence Relation

For $1 \leq i < j \leq n$, let $S[i, j]$ denote the optimal solution to the subproblem consisting of the polygon having vertices $q_i, \cdots, q_j$.
Let $\Delta(q_i, q_k, q_j)$ denote the perimeter of the triangle having vertices q_i, q_k, q_j.
Then we have the recurrence relation

$$S[i, j] = \min \{\Delta(q_i, q_k, q_j) + S[i, k] + S[k, j] : i < k < j\}$$

the base cases are given by

$$S[i, i+1] = 0$$

for all i.
We compute all $S[i, j]$ with $j - i = c$, for $c = 2, 3, \cdots, n-1$.

Weighted Interval Scheduling

Problem 4.6

Problem: Weighted Interval Scheduling.
Instance: A set I of n intervals $[s_1, f_1], \cdots, [s_n, f_n]$ with weights $\omega_1, \cdots, \omega_n$.
Question: Find subset S of disjoint intervals that maximizes $\sum_{i \in S} \omega_i$.

Greedy approach does not work (example?)

Denote: $OPT(I)$ - optimum set S; $\omega_{OPT(I)}$ - corresponding weight.
The structure of optimal solution:
Consider interval i: it is either in $OPT(I)$ or not.
If $i \in OPT(I)$ then $OPT(I) = \{i\} \bigcup OPT(I')$, where I' denotes intervals disjoint from i.
If $i \notin OPT(I)$ then $OPT(I) = OPT(I - \{i\})$. Therefore

$$\omega_{OPT(I)} = \max\left\{\omega_{OPT(I-\{i\})}, \omega_i + \omega_{OPT(I')}\right\}$$

Using this directly one ends up with exponential running time (solving subproblems for 2^n subsets of I).

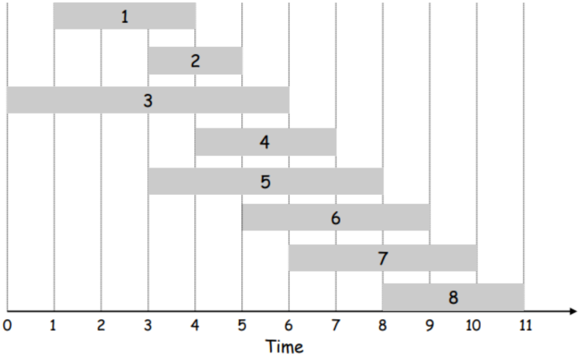
Rename the intervals, by sorting if necessary, so that $f_1 \leq f_2 \leq \cdots \leq f_n$.
Denote $p(j)$ the largest index $i < j$ such that interval i is disjoint from the interval j.

Let $opt(j)$ be the weight of optimal solution that considers intervals $1, 2, \cdots, j$.

Then $opt(0) = 0$ and

$$opt(j) = \max\{\omega_j + opt(p(j)), opt(j-1)\}$$

Ex: $p(8) = 5$, $p(7) = 3$, $p(2) = 0$.



j	p(j)
0	-
1	0
2	0
3	0
4	1
5	0
6	2
7	3
8	5

```
Sort intervals according to finish time
Compute p[j] for each j
opt[0]=0
for j from 1 to n
    opt[j]= max{opt[j-1], opt[p[j]]+w[j]}
Output opt[n]
```

Complexity?

Solution recovery ...

```
j = n
while (j>=0) do
if (opt[p[j]]+w[j] > opt[j-1])
print j
j = p[j]
else
j = j-1
```