Outline

Dynamic Programming

Shortest Paths
Knapsack Warmup

Problem

- Input: $n$ items each with value $w_i \in \mathbb{N}$ and a capacity $W \in \mathbb{N}$
- Output: Subset $S$ that maximizes $\sum_{i \in S} w_i$ subject to $\sum_{i \in S} w_i \leq W$

Example

Consider input $\{7,5,4\}$ and $W = 10$. Optimal is 9.
Try something like divide and conquer... 

**Definition**

Let $\text{knap}(i, j)$ be the optimal solution obtained by using only first $i$ items and capacity $j$ where $\text{knap}(i, j) = -\infty$ for $j < 0$

To compute $\text{knap}(i, j)$:

- If $i = 0$: $\text{knap}(i, j) = 0$
- Otherwise:
  - Compute $\text{knap}(i - 1, j)$ and $\text{knap}(i - 1, j - w_i)$
  - $\text{knap}(i, j) = \max(\text{knap}(i - 1, j), \text{knap}(i - 1, j - w_i) + w_i)$

**Claim**

*The above recursive algorithm will return $\text{knap}(n, W)$ correctly.*

But it’s very inefficient because evaluating both $\text{knap}(i - 1, j)$ and $\text{knap}(i - 1, j - w_i)$ requires a lot of duplication of work.
Dynamic Programming Table

Construct a \((n + 1) \times (W + 1)\) table \(K\) where \(K_{i,j} = \text{knap}(i,j)\):

- Fill in “0” for each entry of first row
- To fill in \(i\)-th row use entries of \((i - 1)\)-th row:

\[
K_{i,j} = \begin{cases} 
\max(K_{i-1,j}, K_{i-1,j-w_i} + w_i) & \text{if } j \geq w_i \\
K_{i-1,j} & \text{if } j < w_i
\end{cases}
\]

Claim

Running time is \(O(nW)\) and space required is \(O(W)\).

Easy to tweak algorithm to find \(S\) and not just \(\sum_{i \in S} w_i\)

Actually Knapsack is NP-complete, have we proved that \(P = NP\)?
When to use dynamic programming...

- *Optimal Substructure*: The solution to the problem can be found using solutions to smaller sub-problems.

- *Overlap of Sub-Problems*: By taking advantage of the fact that many identical sub-problems are created, a dynamic programming algorithm may be more efficient than a divide and conquer algorithm.
Outline

Dynamic Programming

Shortest Paths
Shortest Paths

Let $G = (V, E)$ be a directed graph with weights $w : E \rightarrow \mathbb{R}^+$. 

**Definition**

For path $p = (v_1, \ldots, v_k)$ be a path, define

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}).$$

The shortest path between $u$ and $v$ is

$$\delta(u, v) = \min\{w(p) : p \text{ is a path from } u \text{ to } v\}$$

if there is a path from $u$ to $v$ and $\infty$ otherwise.
Problem: Find $\delta(u, v)$ for all $u, v \in V$.

- Define sub-problems by limiting the set of intermediate nodes
- Let $d^{(k)}_{ij} =$ length of shortest path from $i$ to $j$ for which all intermediate vertices are in $\{v_1, \ldots, v_k\}$
- Easy: $d^{(0)}_{ij} = w(i, j)$ if $(i, j) \in E$ and $d^{(0)}_{ij} = \infty$ otherwise
- For $k \geq 1$:
  $$d^{(k)}_{ij} = \min(d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj})$$
Floyd-Warshall Algorithm

Algorithm

1. Let $d_{ij}^{(0)} = w(i, j)$ if $(i, j) \in E$ and $d_{ij}^{(0)} = \infty$ otherwise.

2. For $k = 1$ to $n$:
   2.1 For $i, j \in [n]$: let
   $$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

3. Return $d_{ij}^{(n)}$

Running Time: $\Theta(n^3)$ where $n = |V|$