DYNAMIC PROGRAMMING

1) Longest path in a DAG \( G = (V, E) \)

Subproblem: \( L(v) = \) length of longest path ending in \( v \)

Recurrent: \( L(v) = \max_{(u,v) \in E} (L(u) + 1) \) (where an empty max equals 0)

Order to compute: topological sorted order of \( E \)

Run Time: \( O(|V| + |E|) \)

2) Longest Increasing Subsequence of sequence \( a_1, \ldots, a_n \)

Subproblem: \( L(i) = \) length of longest increasing subsequence ending in \( a_i \)

Recurrent: \( L(i) = 1 + \max_{j < i} L(j) \) if \( a_j < a_i \)

Order to compute: \( i = 1, 2, 3, \ldots \)

Run Time: \( O(n^2) \)

3) Edit Distance of two strings \( x[1:n], y[1:m] \)

\( E(x[1:n], y[1:m]) = \min \left\{ \# \text{ of deletes, inserts, substitutes to transform } x[1:n] \text{ into } y[1:m] \right\} \)

Subproblem: \( E(x[1:j], y[1:j]) \)

Recurrent: \( E(x[1:j], y[1:j]) = \min \left\{ \begin{array}{l} E(x[1:j], y[1:j-1]) + 1 \\ E(x[1:j-1], y[1:j]) + 1 \\ E(x[1:j-1], y[1:j-1]) + 1, \text{ if } x_j \neq y_j \end{array} \right\} \)
Order to compute:

<table>
<thead>
<tr>
<th>$\emptyset$</th>
<th>$s$</th>
<th>$n$</th>
<th>$o$</th>
<th>$w$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$s$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n$</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$o$</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$w$</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y$</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Run Time: $O(nm)$  $O(nm)$  $O(nm)$  in parallel

Space: $O(n)$  $O(n)$  $O(n+m)$

4) Knapsack

**Input:** Capacity $W$ (integer)

- weights $w_1, \ldots, w_n$ (integers)
- values $v_1, \ldots, v_n$ (integers)

**Goal:** Find set of item with

$$\max \text{ total value, with } \text{ total weight } \leq W$$

4a) Knapsack with Replacement

We looked optimal solution

$$v_i + \cdots + v_{i_k} = v_i + \cdots + v_{i_{k-1}} + v_{i_k} = K(w-w_{i_k}) + v_{i_k}$$

Subproblem:

$$K(C) = \max \text{ total value with total weight } \leq C$$

$$C = 0, 1, \ldots, W$$

**Recurrence:**

$$K(C) = \max_{i: w_i \leq C} \left( v_i + K(C-w_i) \right)$$

Order to compute

$$C = 0, 1, \ldots, \text{ starting with } K(0) = 0$$
Algorithm:

\[
\begin{align*}
\text{Input: } & W, V[1:n], W[1:n] \\
K(0) &= 0 \\
\text{For } C = 1 \text{ to } W \\
& \quad K(C) = \max_{i: w_i \leq C} V_i + K(C - w_i) \\
\text{Output: } & K(W)
\end{align*}
\]

Run-time Space
\[O(nW) \quad O(W+n)\]
Exponential in \[\log W\]

4.b) Knapsack w/o replacement

1) Subproblem

Optimal solution: items \(i_1, \ldots, i_k\), \(\sum w_{i_k} \leq W\)

\[\text{Value} = V_{i_1} + \ldots + V_{i_k} = V_{i_1} + \ldots + V_{i_{k-1}} + V_{i_k}\]

Guess: \(\tilde{K}(W - w_{i_k})\)

\(\tilde{K}(C)\) optimum w/o replacement at capacity \(C\)

Problem: \(i_1, \ldots, i_{k-1}\) must be different from \(i_k\), but in \(\tilde{K}(W - w_{i_k})\) \(i_k\) could be used again

2nd Try:

\[\tilde{K}(k) = \text{Knapsack}\left(V_{i_1}, \ldots, V_{i_k}, w_1, \ldots, w_n\right)\]

\[\tilde{K}(n) = \max \{\tilde{K}(n-1), V_n + \tilde{K}(n-1)\}\]

Problem: We are not keeping track of the budget

3rd Try:

\[K(C, k) = \text{Optimum with total weight } \leq C\]

using a subset of items \(1, \ldots, k\)

2) Recursion

\[K(C, k-1) \quad \text{if } V_k > C\]
2) Recursion

\[
K(c, b) = \begin{cases} 
K(c, b - 1) & \text{if } v_b > c \\
\max \{ K(c, b - 1), K(c - w_b, b - 1) + v_b \} & \text{if } v_b \leq c
\end{cases}
\]

3) Order

\[\begin{array}{c|cccc}
\text{c} & 0 & 1 & \cdots \\
\hline
0 & 00000 & \cdots \\
1 & \text{...} & \text{...} \\
2 & \text{...} & \text{...} \\
& \text{...} & \text{...} \\
\end{array}\]

\[W(c, 0) = 0\]

Algorithm:

```
Input: W, v[1:n], w[1:n]
For c = 0, 1, \cdots W
    K(c, 0) = 0
For b = 1, \cdots n
    "Calculate W(b, c)"
    For c = 1, \cdots W
        K(c, b) = K(c, b - 1)
        IF v_b \leq c and v_b + K(c - w_b, b - 1) > K(c, b - 1)
            K(c, b) = v_b + K(c - w_b, b - 1)

Output: K(W, n)
```

5) Single Source Shortest Path

Input: Weighted graph G = (V, E, \ell), source s

Output: \forall v \in V, \text{dist}(v) = \text{length of shortest path } s \rightarrow v

1) Subproblems

Optimal solution:

\[v_i = \emptyset, v_1, \ldots, v_k = v\]
Optimal Solution:

\[ u_1, u_2, \ldots, u_k = v \]

\[ \text{dist}(v) = \text{dist}(v_{k-1}) + \ell(v_{k-1}, v) \]

\[ \Rightarrow \text{dist}(v) = \min_u (\text{dist}(u) + \ell(u, v)) \quad \text{not really a subproblem!} \]

Idea: recurse on # of edges \( k \) in path

\[ \text{dist}(v, k) = \text{length of shortest path } A \to v \text{ using } k \text{ edges} \]

\[ k = 0, 1, \ldots \]

Recursion

\[ \text{dist}(v, k) = \min \left\{ \text{dist}(v, k-1), \min_{u \in E} \left( \text{dist}(u, k-1) + \ell(u, v) \right) \right\} \]

Dependencies

\[ \text{dist}(v, k) \text{ depends on } \text{dist}(u, k-1) \text{ for } u \in E \]

Algorithm

```
For \( v \in V \): \text{dist}(v, 0) = \infty
\text{dist}(v, 0) = 0
For \( k = 1, \ldots, n-1 \)
For all \( v \in V \)
\text{dist}(v, k) = \text{dist}(v, k-1)
For all \( (u, v) \in E \)
If \( \text{dist}(v, k) > \text{dist}(u, k-1) + \ell(u, v) \)
\text{dist}(v, k) = \text{dist}(u, k-1) + \ell(u, v)
\forall v \in V \text{ Output } \text{dist}(v, n-1)
```

Run Time

\[ O(|V| (|E| + |V|)) \approx O(|E| |V|) \]

Remark: This is very similar to Bellman Ford, in fact, in the worst case, it is identical. (sometimes Bellman Ford just needs one run, e.g. if \( G = K_{n,n} \), in which case B.F. will be done after one run through all edges, while D.P. needs \( n-1 \) runs)
6) All Pairs Shortest Path

Input: \( G = (V, E) \), \( E: E \rightarrow R \)

Output: \( \text{dist}(u, v) = \text{length of shortest path from } u \text{ to } v \) for all \( u, v \in V \)

Idea 1: Run Bellman-Ford IVI times \( \text{Runtime } O(IVI^2EI) \)

Today: DP Solution Floyd Warshall \( \rightarrow O(IVI^3) \)

Optimal Solution:
\[
\text{dist}(u, v) = \ell(u, v) + \ell(v, v_2) + \cdots + \ell(v_n, v)
\]

Idea 1:
\[
\text{dist}(u, v, k) \quad \# \text{ of edges used}
\]
\[
\# \text{ of intermediate vertices + 1}
\]

Problem: by iteration on the number of intermediate edges, we effectively run the algorithm from 5 for all sources \( n \), getting again \( O(IVI^2EI) \) run-time.

Idea 2:
Iterate on which vertices are used

Subproblem
\[
V = \{1, 2, \ldots, n\}
\]
\[
\text{dist}(i, j; k) \quad \text{uses intermediate vertices } \leq k
\]

Base Case
\[
\text{dist}(i, j; 0) = \ell(i, j)
\]

Recursion:
Assume \( \text{dist}(i, j, k-1) \) is known
\[
\text{dist}(i, j, k) = \begin{cases} 
\ell(i, j) & \text{if } k \leq 0 \\
\text{dist}(i, j, k-1) & \text{if } k > 0
\end{cases}
\]
Case 1: \( x \) is not used

\[ d(i,j,\lambda-1) \]

Case 2: \( x \) is used

\[ \lambda \leq \lambda-1 \leq j = ? \]

\[ d(i,j,\lambda) = \min \{ d(i,j,\lambda-1), d(i,\lambda,\lambda-1) + d(\lambda,j,\lambda-1) \} \]

Algorithm:

\[ FV(E, \lambda) \]

\[
\text{For } \lambda, \lambda = 1, \ldots, n \text{ } d(i,j,0) = \infty \\
\text{For } (i,j) \in E \text{ } d(i,j,0) = \ell(i,j) \\
\text{For } \lambda = 1, \ldots, n \text{ } d(i,i,0) = 0 \\
\text{For } \lambda = 1, \ldots, n \text{ } \\
\text{For } j = 1, \ldots, n \text{ } \\

dist(i,j,\lambda) = \min\{ \text{dist}(i,j,\lambda-1), \text{dist}(i,\lambda,\lambda-1) + \text{dist}(\lambda,j,\lambda-1) \} \\
\text{Output dist}(i,j,\lambda) \]

7) Travelling Salesman Problem (TSP)

Given: \( n \) cities, distances \( d_{ij} \) \( i \neq j \)

Goal: Find path of minimal length,

starting at \( 1 \), ending at \( 1 \), visiting every city once

Naive Running Time \( n! = O(2^n) \)

Goal: Find optimal tour in time \( O(2^n) \)

Idea: Consider subproblem on

set of cities \( S \subseteq \{1, \ldots, n\} \), \( l \in S \)

Idea 1:

\[ c(S) = \text{cost of tour in } S \]
Idea 1:

\[ C(S) = \text{cost of tour in } S, \text{ starting and ending in } 1 \]

Problem: \[ S \rightarrow S \cup \{2\} \]

\[ a \quad 1 \quad b \]

\[ x \quad 1 \quad j \]

Idea 2:

\[ C(S) = \min_{j \neq 1} \text{Cost (Tour } 1 \rightarrow j \text{ in } S) + d_{j \rightarrow 1} \]

Subproblem:

Let \( i, j \in S, j \neq 1 \)

\[ C(S, j) = \text{length of shortest path from } 1 \text{ to } j \]

visiting every \( i \in S \) once