# Dynamic Programming

## Longest Path in DAG

Input: $G(V,E)$. DAG  Output: $l$ = longest path in $G$.

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

Recurrence Relation: $L(v) = \begin{cases} 1 + \max_{(w,v)\in E} L(w) \\ 0 \text{ if } v \text{ has no incoming edges} \end{cases}$

Algo. (Runtime $O(|V|+|E|)$)

- Sort in topological order — $i$ is the $i^{th}$ vertex in topological ordering s.t. $\forall (i,j)\in E.\ j>i$
- For all $i$, set $L(i)=0$
- For all $i=1\cdots n$, set $L(i) = \begin{cases} 1+\max_{(w,i)\in E} L(w) \\ 0 \text{ if no incoming edges} \end{cases}$

## Longest Increasing Sequences

Input: $a_1 a_2 \cdots a_n$  Output: $l$ = length of longest increasing sequences.

⇒ Reduces to finding longest path in a graph.

$G=(V,E)$ $V=\{1,2\cdots n\}$. $E=\{(i,j) \mid i<j \text{ & } a_i<a_j\}$.

Approach: 1) Define appropriate subproblems
2) Write a recurrence relation
3) Determine order of computation
↳ induces a DAG structure.

## Edit Distance

Input: $x[1,2\cdots n]$. $y[1,2\cdots n]$

GOAL: find minimum # of keystroke (add, remove, substitute) needed to edit $x$ to $y$

1) Define subproblems: $E(i,j) = E(x[1\cdots i].\ y[1\cdots j])$
= Edit distance to edit $x[1\cdots i]$ to $y[1\cdots j]$

2) Recurrence Relation: $E(i,j) = \min \begin{cases} 1+E(i-1,j) \\ 1+E(i,j-1) \\ \text{diff}(x[i],y[j]) + E(i-1,j-1) \end{cases}$

Base Case: $E(0,0)=0$
$\forall j$, $E(0,j)=E(j,0)=j$

Algo.

Init: For all $i=1\cdots n$, $E(i,0)=i$
For all $j=1\cdots m$, $E(0,j)=j$  Runtime $O(nm)$
For all $i=1\cdots n$
For all $j=1\cdots m$, $E(i,j) = \min \begin{cases} 1+E(i-1,j) \\ 1+E(i,j-1) \\ \text{diff}(x[i],y[j]) + E(i-1,j-1) \end{cases}$

## Knapsack Problem

Input: total weight capacity of a knapsack $W$
list of $n$ items with weights $w_1, w_2 \cdots w_n$
values $v_1, v_2 \cdots v_n$

GOAL: Find set of items that will maximize total value with total weight $\le W$

$k(C)$ = max value that can be packed in a bag of capacity $C$

$k(C) = \max_{i:c\ge w_i} \{v_i + k(C-w_i)\}$.  Base case: $k(0)=0$

Algo.

Input $W$, $v[1\cdots n]$, $w[1\cdots n]$
$k(0)=0$
For $C=1$ to $W$  (exponential in $\log N$)
$k(C) = \max_{i:w_i\le c} \{v_i + k(C-w_i)\}$.
Output $k(W)$

Runtime $O(nW)$

Memorization (if $W$ is really large)
Initialize: hash table. initially empty
holds $k(w)$ indexed by $w$
knapsack($w$)
if $w$ in hash table return $k(w)$
$k(w) = \max\{\text{knapsack}(w-w_i)+v_i : w_i\in w\}$
insert $k(w)$ into hash table with key $w$
return $k(w)$

## Knapsack without repetition

$k(C,j)$: Maximum value that can be carried in a bag of capacity $C$ using items $1,2\cdots j$

$k(C,j) = \begin{cases} k(C,j-1) & \text{if } C<w_j \\ \max\{k(C,j-1),\ v_j + k(C-w_j,j-1)\}, & \text{if } C\ge w_j \end{cases}$

Base Case: $k(0,j)=0$ $\forall j$  runtim $O(nW)$

Algo.

Input $W, v[1\cdots n]$. $w[1\cdots n]$
For $C=0\cdots W$.
$k(C,0)=0$
For $l=1\cdots n$
For $C=1\cdots W$  (If $C-w_l \ge 0$)
$k(C,l) = \max\{k(C,l-1),\ v_l + k(C-w_l, l-1)\}$.
Output $k(W,n)$

## Chain Matrix Multiplication

Input: $A_1$ ($m_0 \times m_1$), $A_2$ ($m_1 \times m_2$), $A_3$ ($m_2 \times m_3$) $\cdots$ $A_n$ ($m_{n-1}\times m_n$)

Output: Minimum number of multiplications needed

$(A\times B)_{ij} = \sum_{j=1}^{m_2} A_{ij} B_{jk}$  ($A_{m\times n}$. $B_{n\times o}$)

Subproblem: $M(i,j)$ = the minimum number of multiplications needed to multiply $A_i \times A_{i+1} \cdots A_j$

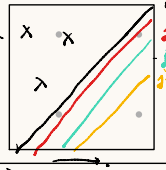
$M(1,n)$ is what we want

Recurrence Relation:

$M(i,j) = \min_{i\le k\le j} \{M(i,k) + M(k+1,j) + m_{i-1} m_k m_j\}$.

Base Case: $M(i,i)=0$. for all $j\in\{1,2\cdots n\}$

Order of computation




Runtime $O(n^2 \times n) = O(n^3)$

Algo.

For $i=1\cdots n$  $M(i,i)=0$
For $s=1\cdots n-1$
For $i=1\cdots n-s$  Runtime $O(n^2\times n) = O(n^3)$
$j=i+s$
$M(i,j) = \min_{k=i\cdots j} \{M(i,k) + M(k+1,j) + m_{i-1} m_k m_j\}$.
Return $M(1,n)$

## Subproblem Structure

(i) input $x_1\cdots x_n$ and a subproblem is $x_1\cdots x_i$

$x_1 \cdots x_i \quad x_{i+1}\cdots x_n$

(ii) input $x_1\cdots x_n$ & $y_1\cdots y_m$ and a subprob is $x_1\cdots x_i$ $y_1\cdots y_j$

$x_1\cdots x_i \quad x_{i+1}\cdots x_n$
$y_1\cdots y_j \quad y_{j+1}\cdots y_m$

(iii) input $x_1\cdots x_n$ and a subproblem is $x_i\cdots x_j$

$x_1\cdots \boxed{x_i \cdots x_j} \cdots x_n$

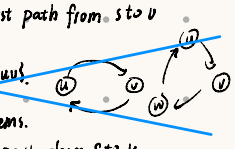
## SSSP (Single Source Shortest Path)

Input: Weighted graph $G=(V,E)$. We for edge $e$ (WeteR)
source $s$

Output: For every $v$, dist(v) = shortest path from $s$ to $v$

① ~~dist(v) for $v\in V$~~

② ~~dist(v) = $\min_{(u,v)\in E}$ {dist(s,u) + $w_{uv}$}~~




③ Need to redefine the subproblems.

① dist(v,k) = distance of shortest path from $s$ to $v$ involving upto $k$ edges

∴ dist(v,k) ← dist(u,k-1), ⋮, dist(u₁,k-1)

Base case:
dist(s,0)=0
dist(v,0)=∞ $\forall v\ne s$

② $dist(v,k) = \min\{dist(v,k-1),\ \min_{(u,v)\in E}\{dist(u,k-1) + w_{uv}\}\}$

③ Nice order to compute k=1, k=2 ⋯

Algo.

For $v\in V$. dist(v,0) = ∞.
dist(s,0) = 0
For all $k=1\cdots n-1$  $n\cdot(n+m) \to O(nm)$
For all $v\in V$
dist(v,k) = dist(v,k-1)
For all $(u,v)\in E$
if dist(v,k) > dist(u,k-1) + $w_{uv}$
then dist(v,k) = dist(u,k-1) + $w_{uv}$
Output $\forall v\in V$. dist(v,n-1)

## All Pairs Shortest Path

Input: Weighted graph $G=(V,E)$. We for edge $e$ (WeteR)

Output: For every u,v dist(u,v)=shortest path from u to v

① Run BF $n$ times to get $u$ to $v$ paths for all $u$  $O(nm)$

② dist(u,v,k) → shortest path from $u$ to $v$ with almost $k$ edges  $O(n^2m)$

③ dist(u,v,k) → shortest path from $u$ to $v$ that takes vertex in $\{1,\cdots k\}$ only.

claim: On the shortest path from $u$ to $v$, not vertex $w$ happens twice.

$dist(u,v,k) = \min\{dist(u,v,k-1),\ dist(u,k,k-1) + dist(k,v,k-1)\}$

Algo.

For $i,j=1\cdots n$  $dist(i,j,0) = \infty$  Runtime $O(n^3)$
For $(i,j)\in E$.  $dist(i,j,0) = w_{ij}$
For $i=1\cdots n$  $dist(i,i,0) = 0$
For $k=1\cdots n$
For $i=1\cdots n$
For $j=1\cdots n$
$dist(u,v,k) = \min\{dist(u,v,k-1),\ dist(u,k,k-1) + dist(k,v,k-1)\}$
Output dist(i,j,k) for all $i,j\in V$.

## Travelling Salesman Problem

Input: $n$ cities & distances $d_{ij}$ ($i\ne j$)

Goal: Find minimum distance path from city 1 back to city 1 visiting each city exactly once.

$C(S,j)$ → the least cost path that ① starts at 1 ② visits all nodes in set $S$ ③ ends in node $j$

Base Case: $|S|=2$  $\forall j\ne 1$.  $C(S,j) = d_{1j}$  ($S=\{1,j\}$)

For $|S|>2$.  $C(S,j) = \min_{i\in S\setminus\{j\}} \{C(S\setminus\{j\}, i) + d_{ij}\}$.

Algo.

C({1}, 1) = 0
for $s=2$ to $n$
for all subsets $S\subseteq\{1\cdots n\}$ of size $s$ and containing 1:
$C(S,1)=\infty$
for all $j\in S, j\ne 1$
$C(S,j) = \min_{\text{if } s\ne j} \{C(S-\{j\}, i) + d_{ij}\}$.  Runtime
return $\min_j \{C(\{1\cdots n\}, j) + d_{j1}\}$.  $2^n\times n\times n \to O(2^n n^2)$

# Max Flow

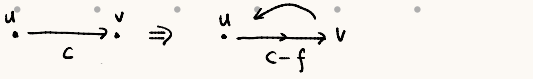
**Definition (s-t flow)**: An s-t flow is an assignment $f: E \to \mathbb{R}^+$ such that

1) Capacity Constraint: For each edge $e$, flow on $e$ ≤ capacity of $e$: $f_e \le c_e$

2) Conservation Constraint: For each vertex $v\ne s/t$. Flow coming into $v$ = Flow leaving $v$: $\sum_{v\to u} f_{v\to u} = \sum_{u\to w} f_{u\to w}$

Algorithm for Max Flow  $O(|E|)f$
REPEAT:
* FIND A PATH P from s to t with non-zero capacity in Residual graph
[Terminate if NO PATH P exists]
* Add flow along P to the current flow.

## Residual Graph




**Definition (Capacity of an s-t cut)**:

Capacity (L,R) = Total capacity of edges from L to R = $\sum_{u\to v} c_{u\to v}$ (u∈L, v∈R)

**CLAIM**: For every s-t cut L,R and for every s-t flow f
Size(f) ≤ CAPACITY(L,R)
⇒ Maximum Flow ≤ Minimum Cut

**Theorem**: In any graph G,
Maximum flow s-t = Minimum s-t cut

# Linear Programming

## Terminology:

Feasible: A point $x \in \mathbb{R}^n$ is "feasible" for a linear program, if it satiesfies all the constraints.

Optimal: A point $x$ is optimal if it minimizes/maximizes the objective value.

Polytope: Feasible region of a linear program.

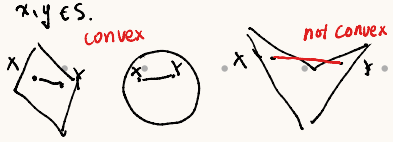
Vertex/Corner: A point $x$ in the feasible region that lies at intersection of "n" hyperplanes (a.k.a. n faces). Specified by constraints

## FACT:

① Feasible region of a linear program is always convex

② ∀ linear program, there is a optimal solution, which is a "vertex"/corner.

Convex Set: A set of points $S \subseteq \mathbb{R}^d$ is convex if $\forall x, y \in S \Rightarrow$ line segment joining $x, y \in S$.




## Simplex Algorithm

* Start at some vertex
* keep moving to neighboring vertex to increase objective

Remark: 1) Simplex can take exponential time in general, but is very different and widely used in practice.

2) LP can be solved in polynomial time by using 1) Ellipsoid Algorithm 2) Interior point methods

## Listing all vertices

Given a LP $\{\sum a_{ij} x_j \le b_j : j = 1, 2, \dots, m\}$.

For each subset of n constraints:

→ Solve for point of intersection $x^*$ (solving linear system/ Gaussian Elimination)

→ If $x^*$ is feasible (satiesfies all remaining constraint), then $x^*$ is a vertex.

Therefore, { # of vertices of an LP with n variables ≥ m constraints } can be as large as $\binom{m}{n}$ ≈ exponential in n.

Example: 6 constraints {A.B.C.D.E.F} and 3 variables

consider neighbors of A.B.C

{ A.B.D  A.D.C  D.A.C
  A.B.E  A.E.C  E.A.C
  A.B.F  A.F.C  F.A.C }

In general, for a vertex v of an LP with n variables and m constraints, # neighbors ≤ (m−n)·n

(# of choices of constraints to add) × (# of choices of constraints to remove)

## General Form

Variable: $x_1, x_2, \dots, x_n \in \mathbb{R}^n$  Maximize: $c_1x_1 + c_2x_2 + \dots + c_nx_n$ ($c^T x$)

Constraints: $\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \le b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \le b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \le b_m \end{cases}$ i.e. $Ax \le b$

## Changing Form of LP

≤ constraints to ≥ constraints

$\sum_i a_i x_i \ge b_i \iff -\sum_i a_i x_i \le -b_i$

= constraints to ≤ constraints

$\sum a_i x_i = b_i \iff \{ \sum a_i x_i \le b_i,\ -(\sum a_i x_i) \le -b_i \}$

Maximization to Minimization

$\max \sum a_i x_i \iff \min -\sum a_i x_i$

Duality

LP (PRIMAL)
$(2x_1 + x_2 \le 100) \cdot y_1$
$(x_1 \le 30) \cdot y_2$
$(x_2 \le 60) \cdot y_3$
$x_1 \ge 0$, $x_2 \ge 0$
$(2y_1 + y_2)\cdot x_1 + (y_1 + y_3)\cdot x_2 \le 100y_1 + 30y_2 + 60y_3$
Maximize $5x_1 + 4x_2$

LP* (DUAL)
Minimize $100y_1 + 30y_2 + 60y_3$
$y_1, y_2, y_3 \ge 0$
$2y_1 + 1\cdot y_2 + 0\cdot y_3 \ge 5$
$1\cdot y_1 + 0\cdot y_2 + 1\cdot y_3 \ge 4$

$5x_1 + 4x_2 \le (2y_1 + y_2)x_1 + (y_1 + y_3)x_2 \le 100y_1 + 30y_2 + 60y_3$

Primal LP

$\begin{bmatrix} A \end{bmatrix}\begin{bmatrix} x \end{bmatrix} \le \begin{bmatrix} b \end{bmatrix}$

Max $\begin{bmatrix} c^T \end{bmatrix}\begin{bmatrix} x \end{bmatrix}$

$\begin{bmatrix} x \end{bmatrix} \ge 0$

Dual LP

$\begin{bmatrix} A^T \end{bmatrix}\begin{bmatrix} y \end{bmatrix} \ge \begin{bmatrix} c \end{bmatrix}$

Min $\begin{bmatrix} b^T \end{bmatrix}\begin{bmatrix} y \end{bmatrix}$

$\begin{bmatrix} y \end{bmatrix} \ge 0$

Primal Solutions ——— Dual Solutions

objective value

Weak Duality: (Any solution to Primal LP) ≤ (Any solution to Dual LP)

Strong Duality: (Optimal value of primal LP) = (Optimal value of Dual LP)

## Zero-Sum Game

Value of game = payoff of row player if all play optimally

GAME A: * Row player announces row r
* column player responds

GAME B: * Column player announces column c
* row player responds

$A: \max_r [\min_c A[r,c]] = \max_r [\min A[1,c], \min_c A[2,c]]$

$= \max[10, -30] = 10$

$B: \min_c [\max_r A[r,c]]$

$= \min_c [\max A[1,c], \max_r A[2,c]] = \min[20, 40] = 20$

| | 1 | 2 |
|---|---|---|
| 1 | 20 | -30 |
| 2 | 10 | 40 |

## Strategies

pure strategy: player picks a particular row or column

mixed strategy: player picks a probability distribution over rows/columns.

payoff: expected payoff of the player

Game C: Row and column players play with mixed strategy.

Eg. $[P[r=1] = \frac{1}{4},\ P[r=2] = \frac{3}{4}\ P[c=1] = \frac{2}{5},\ P[c=2] = \frac{3}{5}]$

Expected Payoff: (game C)

$A[1,1]\cdot(\frac{1}{4}\cdot\frac{2}{5}) + A[1,2]\cdot(\frac{1}{4}\cdot\frac{3}{5}) + A[2,1]\cdot(\frac{3}{4}\cdot\frac{2}{5}) + A[2,2]\cdot(\frac{3}{4}\cdot\frac{3}{5})$

Expected payoff: $E[p, q] = p_1 q_1 A[1,1] + p_1 q_2 A[1,2] + p_2 q_1 A[2,1] + p_2 q_2 A[2,2]$

game D: $\max_p [\min_q E[p,q]]$ (LP)

game E: $\min_q [\max_p E[p,q]]$ (LP*)

* LP and LP* are dual of each other

By strong duality, $OPT(LP) = OPT(LP^*)$

⇒ Value of game D = value of game E

⇒ order of game doesn't matter (with mixed strategy)

FACT for game D:

Maximize [row mixed strategies $p_1, p_2$] [Minimize [column mixed strategies $q_1, q_2$] $\begin{bmatrix} 20p_1q_1 + 10p_1q_2 \\ +10p_2q_1 - 30p_2q_2 \end{bmatrix}$] = Maximize [row mixed strategies $p_1, p_2$ $p_1+p_2=1$] [Min [col] $\begin{bmatrix} 20p_1q_1 + \\ 10p_1q_2 \\ +10p_2q_1 \\ -30p_2q_2 \end{bmatrix}$]

$= \{\max_{p_1+p_2=1} [\min[20p_1 + 10p_2,\ 10p_1 - 30p_2]]\}$ ⇔ LP

value of game D =

Maximize $z$
$z \le 20p_1 + 10p_2$
$z \le 10p_1 - 30p_2$
$p_1 + p_2 = 1$

Minimize [column mixed strategies $q_1, q_2$] [Maximize [row mixed strategies $p_1, p_2$] $\begin{bmatrix} 20p_1q_1 + 10p_1q_2 \\ +10p_2q_1 - 30p_2q_2 \end{bmatrix}$] = Maximize [col mixed strategies $q_1, q_2$ $q_1+q_2=1$] [Min [row pure strategy $p_1=1, p_2=0$ / $p_1=0, p_2=1$] $\begin{bmatrix} 20p_1q_1 + \\ 10p_1q_2 \\ +10p_2q_1 \\ -30p_2q_2 \end{bmatrix}$]

$= \min_{q_1+q_2=1} [\max[20q_1 + 10q_2,\ 10q_1 - 30q_2]]$ ⇔ $LP^*$

value of game E =

Min $z$
$20q_1 + 10q_2 \le z$
$10q_1 - 30q_2 \le z$
$q_1 + q_2 = 0$, $q_1, q_2 \ge 0$

## Independent Set

Definition: Graph $G = (V, E)$, an independent set is a set $I \subseteq V$ such that $\forall u, v \in I$, we have that $(u, v) \notin E$

Maximum Independent Set for Tree

Given: Tree $G = (V, E)$  Find $Ind(G) = \max\{|I| : I$ is an independent set$\}$

$I(v) = \max\{\sum_{u \in C(v)} I(u),\ 1 + \sum_{u \in G(v)} I(u)\}$

$C(v)$: children of $v$
$G(v)$: grandchildren of $v$

Algo  Input Tree on $V = \{1, \dots, n\}$  $\pi(v)$ = parent of $v$. $\pi(r) = r$

Topologically sort $V$ (s.t. $\pi(v) > v$. $\forall v \ne r$)

For $v \in V$ if $\pi(v) \ne v$. then add $v$ to $C(\pi(v))$

if $\pi(\pi(v)) \ne \pi(v)$. then add $v$ to $G(\pi(\pi(v)))$

For $v \in V$ if $C(v) = \emptyset$ then $I(v) = 1$  Runtime $O(n)$

For all $v \in V$

$I(v) = \max\{\sum_{u \in C(v)} I(u),\ 1 + \sum_{u \in G(v)} I(u)\}$.

Output $I(v)$

## Coin Denomination Problem

Input: $x_1 \cdots x_n$ ; $V$  Goal: Possible to make change of $v$ using given coins?

$k(v)$ = minimum # of coin needed to give change for $v$ (use set this to $\infty$ if no possible solution)

$k(v) = \min\{\min_{i:\, x_i \le v}\{k(v - x_i) + 1\},\ \infty\}$  Base case: $k(0) = 0$

Algo. Input $V$. $x_1 \cdots x_n$

$k(0) = 0$  $O(V)$

For $C = 1$ to $V$

$k(C) = \min\{\infty, \min_{i:\, x_i \le C}\{1 + k(C - x_i)\}\}$. ↘ $n$

Output $k(V)$

∴ Runtime $O(Vn)$

line of optima: if there are 2 distinct optimal solutions then the objective function is also optimal on the line.

$ax + by \le c$

$\min_{\text{Rowcol}}[\max_{\text{Colrow}} P] = \max_{\text{Colrow}}[\min_{\text{Rowcol}} P]$  $\min_{\text{Rpure}}[\max_{\text{Cpure}} P] \ge \min_{\text{Rmixed}}[\max_{\text{Cpure}} P]$  $\min_{\text{Rpure}}[\max_{\text{Cpure}} P] \ge \max_{\text{Cpure}}[\min_{\text{Rpure}} P]$  $\min_{\text{Rmixed}}[\max_{\text{Cpure}} P] = \min_{\text{Rmixed}}[\max_{\text{Cmixed}} P]$  Zero sum game  Try: $\begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix}$