Linear Programming

LP in canonical form

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.:} \quad & \sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = 1, \cdots, m \\ & x_j \ge 0, \quad j = 1, \cdots n \end{aligned}$$

n variables $x_1, \cdots, x_n$

$n+m$ constraints

Matrix Notation

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax \le b \\ & x \geq 0 \end{aligned}$$

$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ $c = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}$

$b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$ $A = (a_{ij}) \in \mathbb{R}^{m \times n}$

More general constraints

$$\begin{aligned} \max \quad & c^T x \\ & (Ax)_i \le b_i \quad \text{for } i \in I \subseteq [1:m] \\ & (Ax)_i = b_i \quad \text{for } i \notin I \\ & x_j \geq 0 \quad \text{for } j \in N \subseteq [n] \end{aligned}$$

Feasible Region Polytop of $x \in \mathbb{R}^n$ s.th. x satisfies all constraints

Vertex: Point x in the feasible region such that n of the constraints are tight, i.e., satisfied as equalities

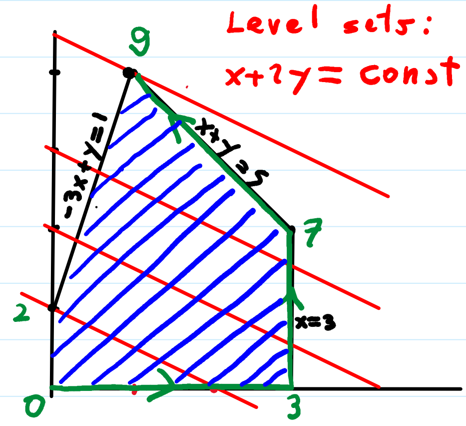
Neighbor of a vertex x: vertex x' which differs in 1 of the n equality constraints

Value: $c^T x$

Fact: The max of an LP is attained at a vertex

Simplex Algorithm

- Start at vertex x^*
- Find neighbor y^* with maximal value
- If value$(y^*) >$ value(x^*) move to y^*.
- Repeat



Running time

- $O(n^3 \cdot nm)$ per step
- Worst case exponentially many steps

Polynomial time alg: Elipsoid, interior point

LP-Duality

Goal: Derive best possible upper bound in terms of linear combinations of the constraints

Primal LP

$$\boxed{\begin{array}{ll} \max & c^T x \\ \text{s.t.} & Ax \le b \\ & x \geq 0 \end{array}}$$

Upper Bd $b^T y$

if $A^T y \geq c$

$y \geq 0$

Dual LP

$$\boxed{\begin{array}{ll} \min & b^T y \\ \text{s.t.} & A^T y \geq c \\ & y \geq 0 \end{array}}$$

More General Duality

Primal:

$$\boxed{\begin{array}{lll} \max \; c^T x & & \\ \text{s.t.} & (Ax)_i \le b_i & i \in I \\ & (Ax)_i = b_i & i \notin I \\ & x_j \geq 0 & j \in P \end{array}}$$

Dual

$$\boxed{\begin{array}{lll} \min \; b^T y & & \\ \text{s.t.} & (A^T y)_j \geq c_j & j \in P \\ & (A^T y)_j = c_j & j \notin P \\ & y_i \geq 0 & i \in I \end{array}}$$

Thm [Weak Duality]

Opt Primal $\leq$ Opt Dual

Thm [Strong Duality]

If primal LP is bounded

Opt Primal = Opt Dual

Proof of weak duality in the more general form

[not exam material, just for fun!]

First constraint + $y_i \geq 0$ for $i \in I$ $\Rightarrow$ $\sum_j y_i A_{ij} x_j \leq b_i y_i$ if $i \in I$

Second — " — $+ y_i \in \mathbb{R}$ for $i \notin I$ $\Rightarrow$ $\sum_j y_i A_{ij} x_j = b_i y_i$ if $i \notin I$.

$$\sum_j (A^T y)_j x_j = \sum_{i,j} y_i A_{ij} x_j \leq \sum_i b_i y_i = b^T y \quad (*)$$

We want the LHS to be an upper bd on $c^T x = \sum_j c_j x_j$. This follows from the dual constraints + $x_j \geq 0$ for $j \in P$

$x_j \geq 0$ and $(A^T y)_j \geq c_j$ $\Rightarrow$ $(A^T y)_j x_j \geq c_j x_j$ if $j \in P$

$x_j \in \mathbb{R}$ and $(A^T y)_j = c_j$ $\Rightarrow$ $(A^T y)_j x_j = c_j x_j$ if $j \notin P$

$$\sum_j (A^T y)_j x_j \geq \sum_j c_j x_j = c^T x$$

Together with (*), we got the weak duality claim

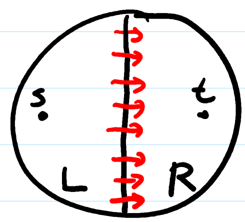
$c^T x \leq b^T y$ whenever x, y are feasible ∎

Min Cut = Max Flow

Input:
- a directed graph $G=(V,E)$
- capacities $c_e \geq 0$, $e \in E$
- a source $s \in V$, a sink $t \in V$

Def s-t cut: a partition $V = L \cup R$ s.th. $s \in L$, $t \in R$

$$\text{capacity}(L,R) = \sum_{\substack{uv \in E \\ u \in L, v \in R}} c_{uv}$$



$$\text{MinCut} = \min_{(L,R)} \text{capacity}(L,R)$$

Maximum Flow Problem

Max size(f) s.t. f is a flow from s to t

$$\boxed{\begin{aligned} &\max \sum_{v:\, sv \in E} f_{sv} \\ &\text{s.t. } 0 \leq f_e \leq c_e \quad \text{for all } e \in E \\ &\qquad \sum_{uv \in E} f_{uv} = \sum_{vw \in E} f_{vw} \quad \text{for all } v \neq s,t \end{aligned}}$$

$$\text{MaxFlow} = \max_{f \text{ is a flow}} \text{size}(f)$$

Thm: If the capacities are integers

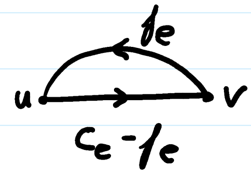
$$\boxed{\text{MinCut} = \text{MaxFlow}} = \text{size}(f)$$

where f is the output of the Ford-Fulkerson algor.

Residual Graph

Given a directed graph $G=(V,E)$ and a fow f, the residual graph G_f is obtained by

- replacing the capacities c_e by $c_e - f_e$
- adding backedges with capacities f_e



$e = uv$

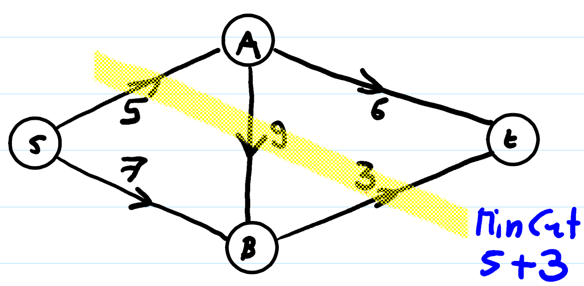
Ford-Fulkerson Algorithm

1) Find not yet saturated path P from s to t in G_f
2) Send additional flow $\Delta f(P) = \min_{e \in P} c_e(G_f)$ along P
3) Repeat until $\Delta f(P) = 0$ for all paths from s to t in G_f

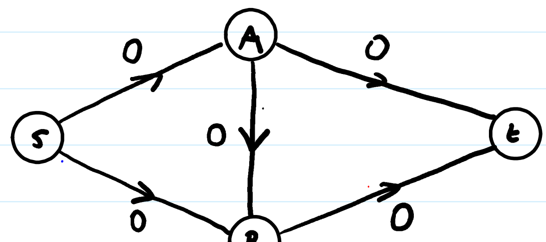
Rem: If all capacities are integers, this outputs an integer flow, and Run Time $= O(\text{Max Flow} \cdot (m+n))$

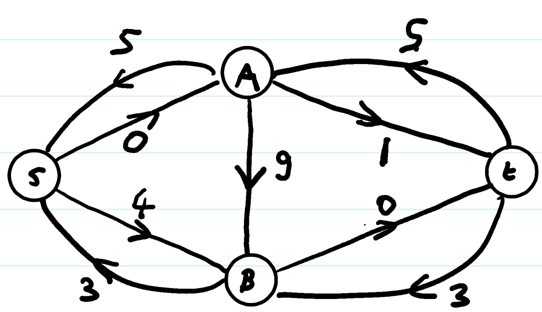
Example:

Residual Graph

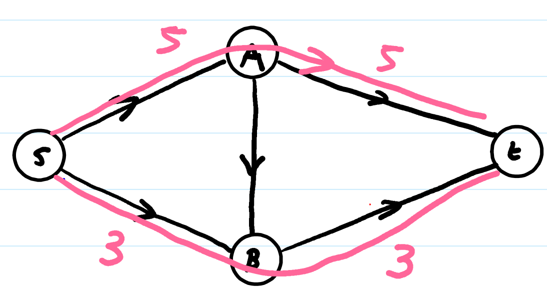


Flow

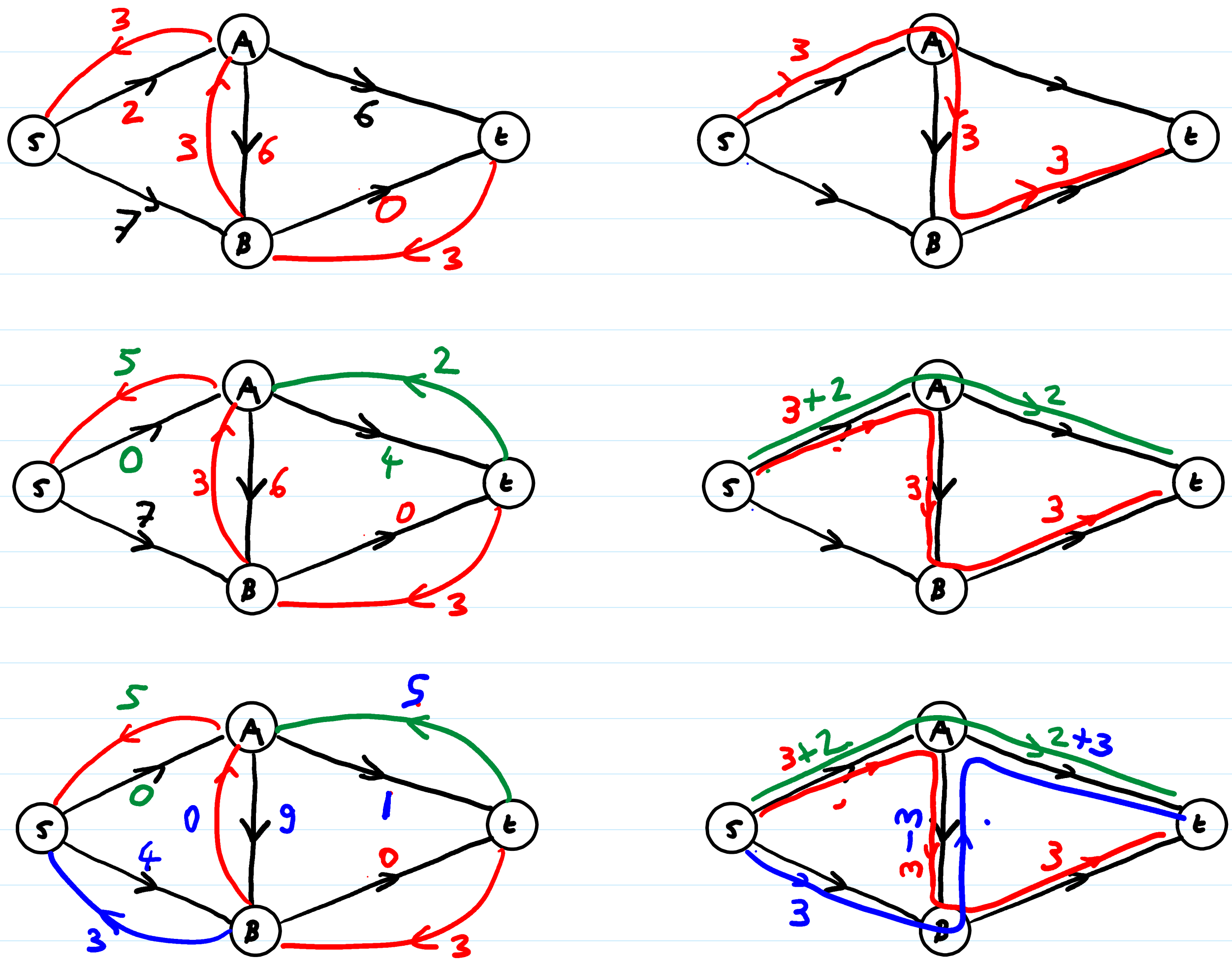




Max Flow = 8



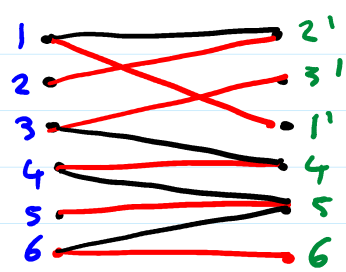
Intermediate steps



Bipartite Perfect Matching

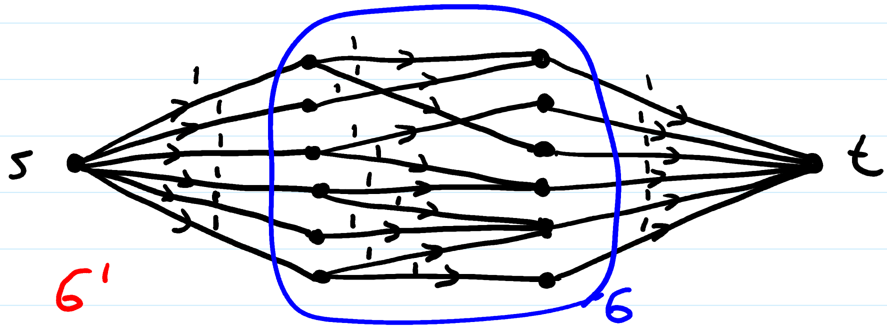
Input: Bipartite graph $G=(L,R,E)$ $\quad |L|=|R|=n$

Output: A perfect matching M from L to R



$M \subseteq E$

Solution via Max-Flow on new graph G'



Thm: G has perfect matching $\Leftrightarrow$ G' has integer flow with size(f) = n

$\Rightarrow$ Run Ford-Fulkerson on G' to find matching, if it exists

Two Player – Zero Sum Games

Input: Payoff Matrix M

	rock	paper	siss.
rock	0	-1	1
paper	1	0	-1
sciss.	-1	1	0

Row Player: picks row r
Col. Player: picks col c

Payoff $\begin{cases} M[r,c] \\ -M[r,c] \end{cases}$

Pure Strategy: a single row/column

Mixed Strategy: probability distribution over pure strategies

Row players average score = - col player average score

$$\text{Score}(p,q) = \sum_{r,c} p_r q_c M[r,c]$$

when row plays mixed strat. p and col plays mixed strat. q

1) Row plays first

1. Row player announces mixed strategy p
2. Col player responds w/ mixed strategy q

Row player choose strategy, anticipating best response

$$\max_p \min_q \text{Score}(p, q) = \max_p \min_c \sum_r p_r M[r, c]$$

$\hookleftarrow$ pure strategy

2) Col plays first

$$\min_q \max_p \text{Score}(p, q) = \min_q \max_r \sum_c q_c M[r, c]$$

$\hookleftarrow$ pure strategy

Min Max Theorem

$$\max_p \min_q \text{Score}(p, q) = \min_q \max_p \text{Score}(p, q)$$

Example

	1	2
1	3	-1
2	-2	1

Row player first
Col. — " — 2nd

$\max_p \min_q \text{Score}(p, q)$

solution of LP

$$\begin{aligned} &\max \quad x \\ &\text{s.th.} \quad x \le 3p_1 - 2p_2 \\ &\qquad\quad x \le -p_1 + p_2 \\ &\qquad\quad p_1 + p_2 = 1 \\ &\qquad\quad p_1, p_2 \geq 0 \end{aligned}$$

Col. player first
Row — " — 2nd

$\min_q \max_p \text{Score}(p, q)$

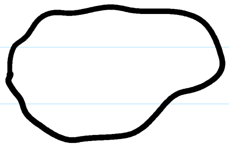
solution of dual LP

$$\begin{aligned} &\min \quad y \\ &\text{s.th.} \quad y \geq 3q_1 - q_2 \\ &\qquad\quad y \geq -2q_1 + q_2 \\ &\qquad\quad q_1 + q_2 = 1 \\ &\qquad\quad q_1, q_2 \geq 0 \end{aligned}$$

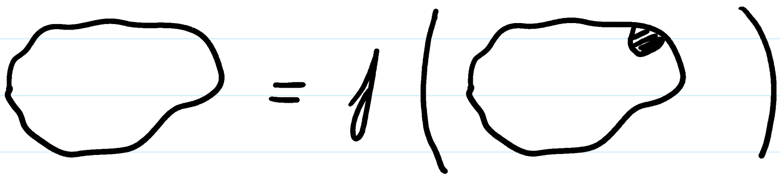
Strong LP Duality: max min = min max

DYNAMIC PROGRAMMING

General Recepie

Problem P 

- Try to reduce it to a slightly smaller subproblem



- Introduce sequence of smaller and smaller problems
- Calculate bottom up

Formally:

1) Define subproblems
2) Derive recursion relation
3) Determine dependencies / calculation order

7) Travelling Salesperson Problem (TSP) $O(n^2 2^n)$

Given: n cities, distances d_{ij} $\quad i \neq j$

Goal: Find path of minimal length,

starting at 1, ending at 1, visiting every city once

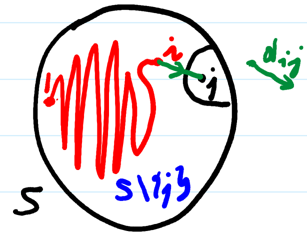
Subproblem: For every $S \subseteq \{1, \cdots n\}$ containing 1, $\forall j \in S, j \neq 1$

$C(S, j)$ = Length of shortest path from 1 to $j \in S$

visiting every $i \in S$ once

Recurrence:

$$C(S,j) = \min_{\substack{i \in S \\ i \neq 1, j}} C(S\setminus\{j\}, i) + d_{ij}$$



Calculation Orders: All sets S of size 2,

all sets S of size 3, ...

Initialization:

$C(S,1) = \infty$ if $|S| > 1$ and $C(\{1\}) = C(\{1\},1) = 0$

Algorithm:

$C(\{1\},1) = 0$

For $k = 2, \cdots, n$

  For all $S \subseteq \{1, \cdots n\}$, $1 \in S$, $|S| = k$

  $C(S,1) = \infty$

  For all $j \in S\setminus\{1\}$

    $C(S,j) = \min_{i \in S, i \neq j} C(S\setminus\{j\}, i) + d_{ij}$

Output $\min_{j \in \{1,\cdots,n\}} C(\{1,\cdots,n\}, j) + d_{j1}$

1) Longest path in a DAG $O(n+m)$

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

Recursion

$$L(v) = \begin{cases} \max_{u: uv \in E} (L(u) + 1) & \text{if exist } u \text{ with } uv \in E \\ 0 & \text{otherwise} \end{cases}$$

Calculation order topological sort

2) Longest Increasing Subsequence $O(n^2)$

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

3) Edit Distance $O(nm)$

Subproblem: $E(x[1:i], y[1:j])$ = edit distance between prefixes

4) Knapsack (capacity W, items with weights $w_1, \cdots w_n$ & values $v_1, \cdots v_n$)

4a) Knapsack with Replacement $O(n|W|)$

Subproblem:

$K(C)$ = max total value with capacity C $C = 0, 1, \cdots, W$

4b) Knapsack w/o replacement $O(n|W|)$

Subproblem

$K(C, k)$ = Optimum with total weight $\leq C$ while only using items in $\{1, \cdots, k\}$ $k = 0, 1, 2, \cdots$

5) Single Source Shortest Path $O(nm)$

Subproblems

$\text{dist}(v, k)$ = length of shortest path $s \rightsquigarrow v$ using $\leq k$ edges

Gives modified version of Bellman Ford

6) All Pairs shortest path (Floyd Warshall Algorithm) $O(n^3)$

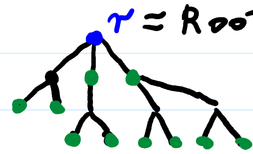
Subproblem

$V = \{1, 2, \cdots n\}$

$dist(i, j; k)$ uses only vertices in $\{1, 2, \cdots, k\}$
as intermediate vertices

8) Maximal Independent Set for Trees $O(n)$

Def: Given $G = (V, E)$, an independent set is a set $I \subseteq V$ s.th. no pair of vertices $\{x, y\} \subseteq I$ is an edge in E



Define Subproblem:

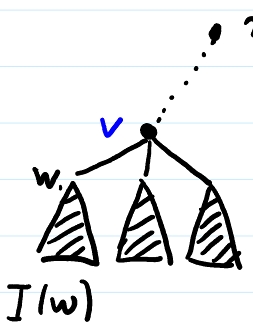
T_v = subtree under v

$I(v)$ = size of largest independent set I in T_v

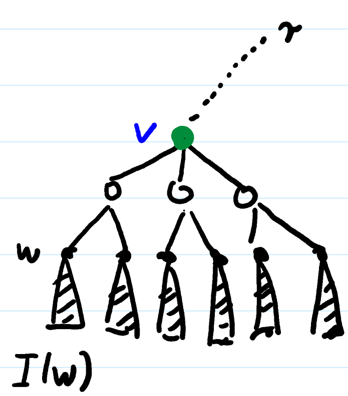
Recursion

$$I(v) = \max\left\{ \sum_{\substack{w \in C(v) \\ \text{children}}} I(w),\; 1 + \sum_{\substack{w \in G(v) \\ \text{grand children}}} I(w) \right\}$$

Case $v \notin I$



Case $v \in I$



$I(w)$

Base Case: v is a leaf

$$I(v) = 1$$

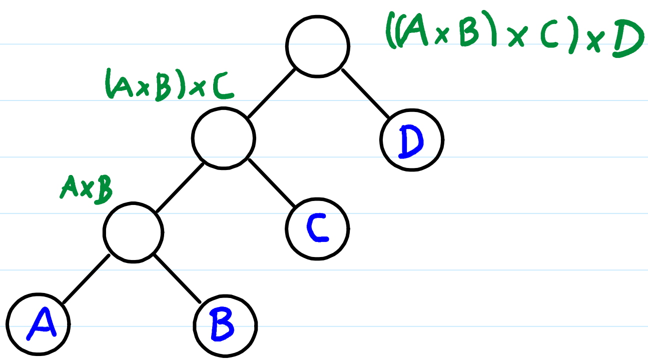
Calculation Order: Leaves to root

9) Chain Matrix Multiplication

Input: Matrices $A_1, \cdots, A_n$, where A_1 is $m_0 \times m_1, \cdots$
A_n is $m_{n-1} \times m_n$

Question: Best order to calculate

Binary tree representation:



Subproblem

$$A_i \times A_{i+1} \times \cdots \times A_j$$

$C(i,j)$ run time for optimal way to put parentheses

Recursion:

$$(A_i \times \cdots \times A_k) \times (A_{k+1} \times \cdots \times A_j)$$

Cost:

$$C(i,k) + C(k+1,j) + m_{i-1} m_k m_j$$

Best way

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

Order of Calculations

$s = |i-j| = 0, 1, 2, \cdots$

Algorithm

$\text{For } i = 1, \cdots n \qquad C(i,i) = 0$

$\text{For } s = 1, \cdots n-1$

$\quad \text{For } i = 1, \cdots, n-s$

$\qquad \text{set } j = i+s$

$$C(i,j) = \min_{i \le k \le j} \left(C(i,k) + C(k+1) + m_{i-1}\, m_k\, m_j \right)$$

$\text{Return } C(1,n)$