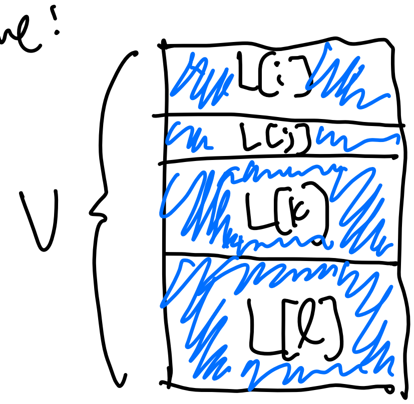
CS 331, Fall 2024
Lecture 6 (9/16)

Today:
- Knapsack
- LCS & friends
- Palindromes
- Game theory

Knapsack (Part III, Section 3.3)

Recall subset sum from last time:



Input: L: list of n natural #'s

$V \in \mathbb{N}$: target value

Output: True/False, $\exists S \subseteq [n]$, $\sum_{i \in [n]} L[i] = V$?

Key idea: 2D memoization, prefix × value

General strategy applies to integer-constrained optimization problems, e.g. knapsack

Input: W: n natural #'s

$B \in \mathbb{N}$: target value

Output: $\exists\, S \subseteq [n],\ \sum_{i \in S} W[i] = B$?

Subset sum

Input: W: n natural #'s (weights)

$B \in \mathbb{N}$: weight budget

V: n real positive #'s (values)

Output: $\max_{\substack{S \subseteq [n] \\ \sum_{i \in S} W[i] \leq B}} \sum_{i \in S} V[i]$

0-1 knapsack

Applications: budget-constrained decision making problems.

Before: $S[j][b]$ = can you hit target b
$j \in [n], b \in [B]$ using first j items?

Intuition: decision to take/not take $L[j]$
affects remaining value, must store

Now: $S[j][b]$ = max value w/ weight
$j \in [n], b \in [B]$ budget b using first j items

DP formula:

$$S[j][b] = S[j-1][b] \text{ OR } S[j-1][b-W[j]]$$

Subset sum

$$S[j][b] = \max\big(S[j-1][b],\ V[j] + S[j-1][b-W[j]] \big)$$

0-1 knapsack

Also runs in $O(nB)$ time. (row-by-row)

$\boxed{\text{Extension}}$ Unbounded knapsack

You can take multiple copies of any item.

Goal: maximize $\sum_{i \in [n]} c_i V[i]$ s.t.

$$c \in \mathbb{Z}^n_{\geq 0} \quad (\text{counts})$$

$$\sum_{i \in [n]} c_i W[i] \leq B$$

DP: $S[b]$ = max achievable w/ budget b

$$S[b] = \max\Big(\underbrace{0}_{\text{take nothing}}, \max_{\substack{i \in [n] \\ W[i] \leq b}} \underbrace{S[b - W[i]] + V[i]}_{\text{take item } i} \Big)$$

Runtime: $\underbrace{O(B)}_{\#\text{ subprobs}} \times \underbrace{O(n)}_{\text{time/subprob}} = O(nB)$

Longest Common Subsequence (Part III, Section 4.1)

$\boxed{\text{Strings}}$ Ω = universe

e.g. $\{ \underbrace{'a', 'b', \ldots, 'z', '1', \ldots, '0'}_{\text{characters}} \}$

String of length n: ordered list of n characters from Ω

"algorithms" $= \{ 'a', 'l', \ldots 'm', 's' \}$

Substring: contiguous sublist "algo"

Subsequence: delete any characters, concatenate the rest "arms"

LCS: natural distance measure on strings

Input: X, length-m string (e.g. DNA)

Y, length-n string

Output: $|Z|$, largest possible length of common subsequence Z of X & Y

$\boxed{\text{Example}}$

X = "algorithms"

Y = "complexity"

$$LCS(X,Y) = 3$$

Conclusion: they are both $\underbrace{\text{"lit"}}_{\arg\max_Z |Z|}$

Key idea: 2-D DP again

$S[i][j] = LCS(X[:i], Y[:j])$

prefixes

2 cases: Can we match $X[i]$, $Y[j]$?

Case 1: No ($X[i] \neq Y[j]$)

e.g. "algor", "compl" ($i=5, j=5$)

What is last char of Z?

$X[i]$, $Y[j]$, or neither. (not both)

If not $X[i]$, Plan A: $S[i-1][j]$

If not $Y[j]$, Plan B: $S[i][j-1]$

Case 2: Yes $\left(X[i] = Y[j]\right)$

e.g. "al", "compl" $(i=2, j=5)$

Now we can make $X[i] = Y[j]$ last character in Z.

Plan C: $1 + S[i-1][j-1]$

Summary:

$$S[i][j] = \max\begin{pmatrix} S[i-1][j], & \text{no } X[i] \\ S[i][j-1], & \text{no } Y[j] \\ S[i-1][j-1] + \mathbb{1}(X[i] = Y[j]) & \text{both} \end{pmatrix}$$

Runtime:
$O(mn)$

Extension Multiple strings

$LCS(W, X, Y)$: longest mutually common subsequence

lengths ℓ m n

Same idea.

$$S[i][j][k] = LCS(W[:i], X[:j], Y[:k])$$

$$= \max\big(S[i-1][j][k],\ S[i][j-1][k],\ S[i][j][k-1],\ S[i-1][j-1][k-1] + \mathbb{1}(W[i]=X[j]=Y[k])\big)$$

Extension Edit distance

How many ops needed to turn X to Y?

Ops: Insert, Delete, Substitute

Example

X = "kitten"
Y = "sitting"

3 steps: "kitten" → "kittin"
→ "sittin" → "sitting"

Observation: Suppose no substitutions.
optimal edit sequence:

$$X \xrightarrow[\text{(deletions)}]{} Z$$

$$Z \xrightarrow[\text{(insertions)}]{} Y$$

Z is LCS of X & Y

Proof:
1) All deletions from X in optimal moves.
2) Rearrange so all deletions first.
3) $X \to Z \to Y$ shortest if Z longest.

Edit distance: Same idea.

All moves are 1) Delete from X

2) Insert from Y

3) Substitute X to Y

or sequence can be made shorter.

$S[i][j]$ = Edit distance of $X[:i]$, $Y[:j]$

$$S[i][j] = \min\Big(S[i][j-1] + 1 \quad \text{insert } Y[j]$$

$$S[i-1][j] + 1 \quad \text{delete } X[i]$$

$$S[i-1][j-1] + \mathbb{1}(X[i] \neq Y[j]) \Big) \quad \text{sub } X[i] \to Y[j]$$

not necessary if equal

Runtime: Again $O(mn)$.

Longest palindromic substring (Part III, Section 4.2)

Palindrome: "RACECAR" (odd length)

center

"TATTARRATTAT" (even length)

center x2

Input: String X

Output: |Z|, largest possible length of palindromic substring Z of X

$\boxed{\text{Example}}$

$X = \text{"b}\underbrace{\text{anana}}_{\text{longest possible } Z}\text{"}$

$X = \text{"b}\underbrace{\text{anaana}}_{\text{longest possible } Z}\text{"}$

Idea 1: DP

To determine $\underbrace{\text{whether } X[i:j] \text{ is palindrome}}_{:= S[i][j]}$

either:
- just check $(j \le i+1)$
- $S[i][j] = S[i+1][j-1]$ AND $(X[i] == X[j])$

$O(n^2)$ time

Idea 2 (or 1?): center growing

1) guess center c or cc $O(n)$

2) grow left & right pointers until can't $\times O(n)$

$= O(n^2)$

Seems to match DP.

Manacher's algo: $O(n)$ (see notes)

Game theory (Part III, Section 5.1)

Consider two-player win-lose game.

Alice vs. Bob. E.g. Tic-tac-toe, chess, ...

Move 1 Alice

Move 2 Bob

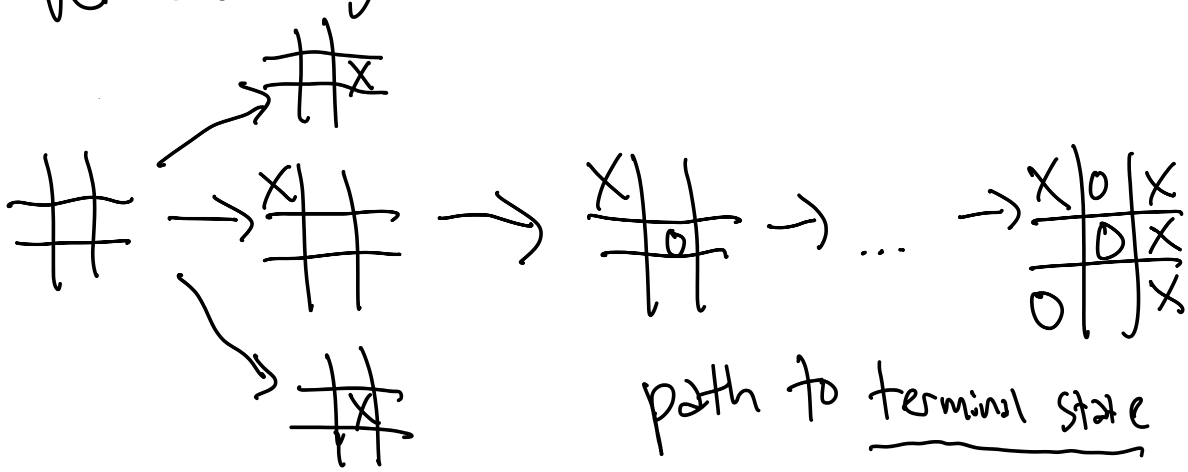
⋮

Move k (game over, Alice wins)

Game graph

(potentially huge, pruning in practice)

Vertices: game states

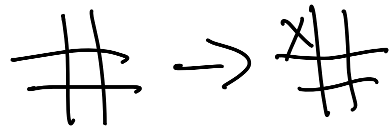


Terminal state: Vertex where game is over.

1) Alice wins

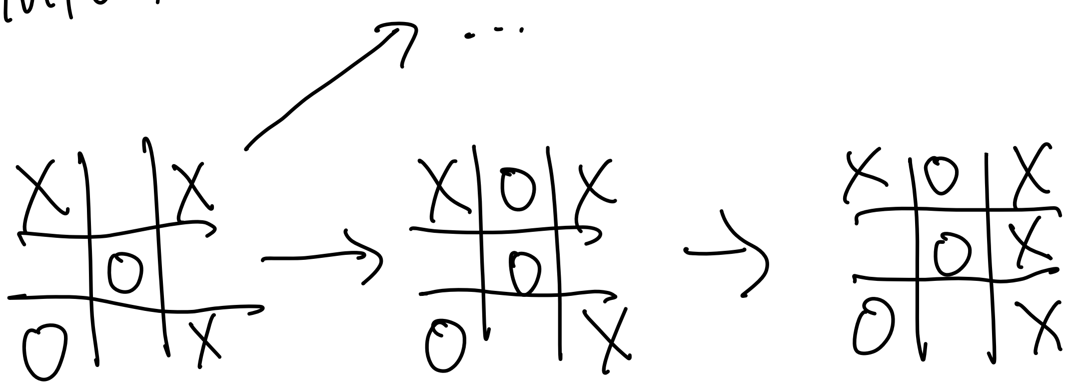
2) Bob wins

3) Tie (we'll mostly ignore)

Game moves: directed edges 

Q: Can Alice always force a win?

Intuition:



"forced win"

Whatever Bob does, Alice can win.

DP solution: label all vertices v True "forced win"

False

Work backwards from leaves = terminal states
(next class)

(with ties, label them False.)

$$S(v) = \begin{cases} \underset{\text{edge }(v,u)}{\text{OR}}\left(S(u)\right) & \text{Alice plays} \\ \underset{\text{edge }(v,u)}{\text{AND}}\left(S(u)\right) & \text{Bob plays} \end{cases}$$

Alice wins if she can move to another winning state.

Bob wins if he can move to another losing state.

So Alice needs all possible moves to be True.

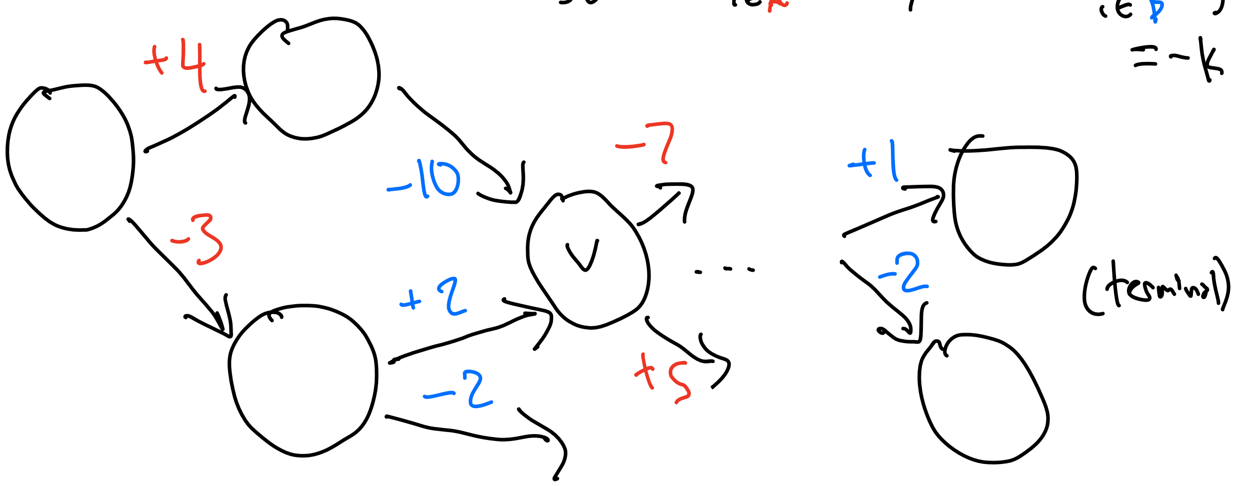
Extension: Zero-sum games

Alice and Bob have scores.

@ end of game, sum = 0.

e.g. win-lose (score at end is winner +1, loser −1)

dividing items ($v_1 + \dots + v_n = T$ split to A, B
score: $\sum_{i \in A} v_i = k, \; -T + \sum_{i \in B} v_i = -k$)



$$S[v] = \begin{cases} \max\limits_{\text{edge}(v,u)} S[u] + \text{score change}(v,u) & \text{Alice} \\ \min\limits_{\text{edge}(v,u)} S[u] + \text{score change}(v,u) & \text{Bob} \end{cases}$$

max guaranteed score if dropped in at state v.