Algorithms and Data Structures 2
Exam Notes
Week 3: Dynamic Programming II

Mads Richardt

1 General Methodology and Theory

Dynamic Programming Principles

- **Subproblems & optimal substructure.** Define states so each depends only on smaller states.
- **Recurrence → evaluation order.** Choose bottom-up or memoized top-down consistent with dependencies.
- **Reconstruction.** Save decisions or backtrack by comparing neighboring states.
- **Complexity.** time ≈ #states × work/state;   space ≈ #states stored.

2 Knapsack Problem (Detailed & Space-Optimized)

Problem

Given items with weights w_i and values v_i and capacity W, choose at most one of each to maximize $\sum v_i$ s.t. $\sum w_i \le W$.

2D DP Recurrence (0–1 Knapsack)

Let $OPT(i, w)$ be the best value using the first i items within capacity w:

$$OPT(i, w) = \begin{cases} OPT(i-1, w), & w < w_i, \\ \max\big(OPT(i-1, w),\ v_i + OPT(i-1, w - w_i)\big), & w \ge w_i. \end{cases}$$

Bottom-up algorithm

```
Array M[0..n][0..W];  M[0][w] = 0
For i = 1..n:
  For w = 0..W:
    if w < wi: M[i][w] = M[i-1][w]
    else:      M[i][w] = max(M[i-1][w], vi + M[i-1][w-wi])
return M[n][W]
```

Time $O(nW)$, space $O(nW)$.

Linear-Space Optimization ($O(W)$ space)

Observation: Row i depends only on row $i-1$ ⇒ reuse one array $D[0..W]$.

Correct 1D algorithm (iterate weights *backwards*)

```
D[0..W] := 0
for i = 1..n:
  for w = W down to wi:            # descending!
    D[w] = max(D[w], vi + D[w - wi])
return D[W]
```

Why backwards? In $OPT(i, w)$ the term $OPT(i-1, w-w_i)$ must come from the *previous* row. If you iterate $w = 0..W$ (forwards), then $D[w-w_i]$ may already include item i (same pass), allowing multiple uses of the same item (unbounded knapsack). Iterating $w = W, W-1, \dots, w_i$ preserves $D[w-w_i]$ as row $i-1$ until it is read.

Induction invariant (proof sketch). At the start of the pass for item i, $D[w] = OPT(i-1, w)$. For $w < w_i$, $D[w]$ unchanged $\Rightarrow OPT(i, w) = OPT(i-1, w)$. For $w \geq w_i$,

$$D[w] \leftarrow \max\big(OPT(i-1, w),\ v_i + OPT(i-1, w-w_i)\big) = OPT(i, w),$$

because $D[w]$ and $D[w-w_i]$ still hold *row* $i-1$ values when looping backwards.

Worked 1D Trace (tiny)

$W = 5$, items $(w, v) = (3, 4), (2, 3)$, $D = [0, 0, 0, 0, 0, 0]$.

```
Item (3,4): w=5..3 -> D = [0,0,0,4,4,4]
Item (2,3): w=5..2 -> D = [0,0,3,4,4,7]
```

Answer $D[5] = 7$ (take both).

3 Notes from Slides and Textbook (concise)

- **Knapsack (0–1)**: pseudo-polynomial $O(nW)$ DP; space can be $O(W)$ via backward 1D update.
- **Sequence alignment**: gap penalty δ, mismatch costs α_{pq}; $O(mn)$ DP; shortest-path view on grid.
- **DP recipe**: define states, prove optimal substructure, base cases, evaluation order, reconstruction.

4 Solutions to Problem Set

1. Knapsack Table (by hand)

Items $(w_i, v_i) = (5, 7), (2, 6), (3, 3), (2, 1)$, capacity $W = 6$. Fill $M[i, w]$.

$i \backslash w$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	0	7	7
2	0	0	6	6	6	7	7
3	0	0	6	6	6	9	9
4	0	0	6	6	7	9	9

Optimal value $M[4, 6] = 9$ (choose items $(2, 6)$ and $(3, 3)$ with total weight 5).

Linear-space note. The same instance can be solved with the 1D algorithm above using $O(W)$ space by looping $w = 6 \downarrow w_i$ for each item.

2. Sequence Alignment (APPLE vs PAPE)

Alphabet $\{A, E, L, P\}$, penalty matrix P:

	A	E	L	P
A	0	1	3	1
E	1	0	2	1
L	3	2	0	2
P	1	1	2	0

gap penalty $\delta = 2$.

Let $A[i,j]$ be min-cost to align $X[1..i]$ with $Y[1..j]$; $A[i,0]=2i$, $A[0,j]=2j$, and

$$A[i,j]=\min\left(\alpha_{x_iy_j}+A[i-1,j-1,\ \delta+A[i-1,j],\ \delta+A[i,j-1]\right).$$

For $X=$ "APPLE" ($m=5$), $Y=$ "PAPE" ($n=4$), the filled table A is:

	0	1	2	3	4
0	0	2	4	6	8
1	2	1	2	4	6
2	4	2	2	2	4
3	6	4	3	2	3
4	8	6	5	4	4
5	10	8	7	6	$\boxed{4}$

Minimum cost $=4$. One optimal alignment (via backtracking):

$$\begin{array}{ccccc} A & P & P & L & E \\ P & A & P & - & E \end{array}$$

(*Cost check:* $1+1+0+2+0=4$.)

3. Book Shop

Prices h_i, pages s_i, budget x, each book at most once.

Recurrence.

$$OPT(i,x)=\begin{cases} OPT(i-1,x), & x<h_i,\\ \max\left(OPT(i-1,x),\ s_i+OPT(i-1,x-h_i)\right), & x\ge h_i.\end{cases}$$

2D DP (baseline).

1. Initialize $M[0][x]=0$ for $x=0..X$.
2. For $i=1..n$, for $x=0..X$, apply the recurrence.
3. Answer $M[n][X]$.

Time $O(nX)$, space $O(nX)$.

Linear-space version ($O(X)$).

```
D[0..X] := 0
for i = 1..n:
  for x = X down to h[i]:
    D[x] = max(D[x], s[i] + D[x - h[i]])
return D[X]
```

Backward iteration ensures 0–1 usage (no repeated purchase of the same book).

4. Longest Palindromic Subsequence (LPS)

For string $S[1..n]$, let $L(i,j)$ be the LPS length in $S[i..j]$:

$$L(i,j)=\begin{cases} 1, & i=j,\\ 2, & i+1=j\ \wedge\ S[i]=S[j],\\ \max\left(L(i+1,j),L(i,j-1)\right), & S[i]\neq S[j],\\ 2+L(i+1,j-1), & S[i]=S[j].\end{cases}$$

Bottom-up: fill by increasing interval length $\ell=1..n$. Return $L(1,n)$. Time $O(n^2)$, space $O(n^2)$. (*Reconstruction*: follow choices that achieved the max.)

5. Defending Zion (KT 6.8)

Given arrivals $x_1, \ldots, x_n$ and recharge function f, EMP used at time k after j idle secs kills $\min(x_k, f(j))$. Let $D[t]$ = max robots destroyed up to time t. The DP:

$$D[t] = \max\Big(D[t-1],\ \max_{1 \le j \le t} \big\{D[t-j] + \min(x_t,\ f(j))\big\}\Big), \quad D[0] = 0.$$

Evaluation: For $t = 1..n$, scan $j = 1..t$ (time $O(n^2)$). If f has special structure (e.g. concave), optimizations may apply. (*Schedule-EMP* greedy fails in general; counterexamples exist.)

Puzzle of the Week: The Blind Man

Take any 10 face-up/face-down cards as pile B; flip all of pile B. The number of face-up cards in A equals that in B (invariant: "ups in B after flip" = "ups in A initially among those moved").

5 Summary

- **Knapsack 0–1:** $OPT(i, w) = \max(OPT(i-1, w), v_i + OPT(i-1, w-w_i))$. Space-optimal 1D update must iterate w *downwards*.
- **Sequence alignment:** $A[i, j] = \min(\alpha_{x_i y_j} + A[i-1, j-1],\ \delta + A[i-1, j],\ \delta + A[i, j-1])$.
- **LPS:** interval DP; match ends or drop one end.
- **Zion:** charge-length choice j each time t: $D[t] = \max(D[t-1], \max_j\{D[t-j] + \min(x_t, f(j))\})$.
- **Complexities:** Knapsack $O(nW)$ time, $O(W)$ space; Alignment $O(mn)$; LPS $O(n^2)$; Zion $O(n^2)$.

Dynamic Programming II

Inge Li Gørtz

KT section 6.4 and 6.6

Thank you to Kevin Wayne for inspiration to slides

Dynamic Programming

- Optimal substructure
- Last time
  - Weighted interval scheduling
- Today
  - Knapsack
  - Sequence alignment

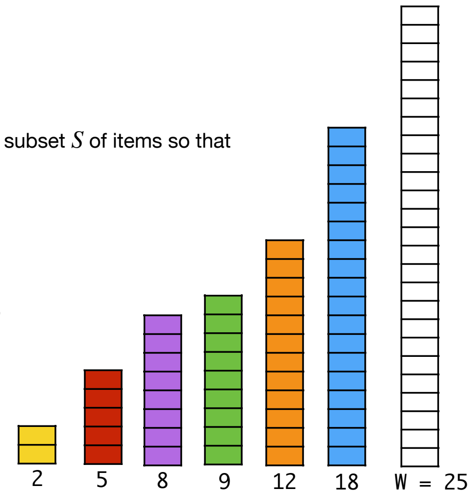
Subset Sum and Knapsack

Subset Sum

- Subset Sum
  - Given *n* items $\{1,\ldots,n\}$
  - Item i has weight w_i
  - Bound W
  - Goal: Select maximum weight subset S of items so that

$$\sum_{i \in S} w_i \leq W$$

- Example
  - {2, 5, 8, 9, 12, 18} and W = 25.
  - Solution: 5 + 8 + 12 = 25.

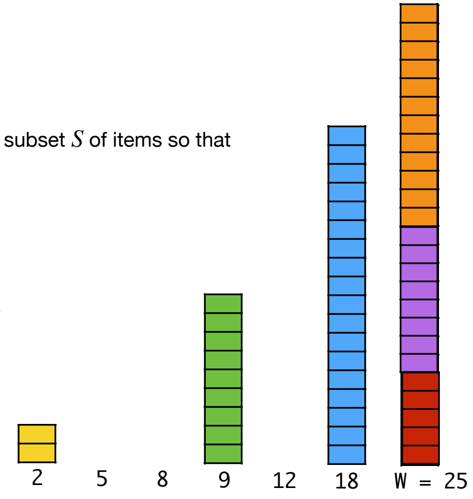


Subset Sum

- Subset Sum
  - Given n items $\{1,\ldots,n\}$
  - Item i has weight w_i
  - Bound W
  - Goal: Select maximum weight subset S of items so that

$$\sum_{i\in S} w_i \leq W$$

- Example
  - {2, 5, 8, 9, 12, 18} and W = 25.
  - Solution: 5 + 8 + 12 = 25.



Subset Sum

- $\mathcal{O}$ = optimal solution
- Consider element n.
  - Either in $\mathcal{O}$ or not.
    - $n \notin \mathcal{O}$: Optimal solution using items $\{1,\ldots,n-1\}$ is equal to $\mathcal{O}$.
    - $n \in \mathcal{O}$: Value of $\mathcal{O}$ = w_n + weight of optimal solution on $\{1,\ldots,n-1\}$ with capacity $W - w_n$.

- Recurrence
  - OPT(i, w) = optimal solution on $\{1,\ldots,i\}$ with capacity w.
  - From above:

$$\text{OPT}(n, W) = \max(\text{OPT}(n-1, W), w_n + \text{OPT}(n-1, W-w_n))$$

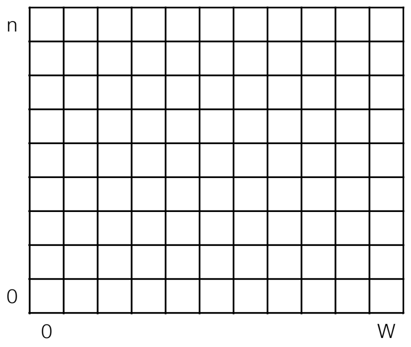
  - If $w_n > W$:

$$\text{OPT}(n, W) = \text{OPT}(n-1, W)$$

Subset Sum

- Recurrence:

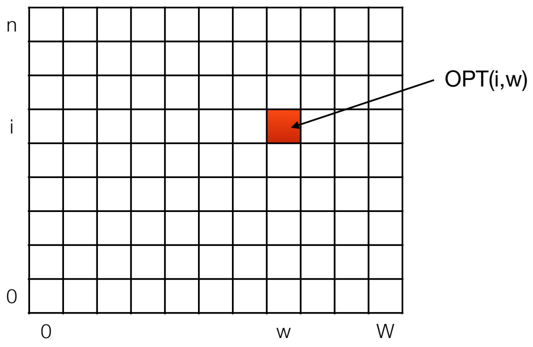
$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)) & \text{otherwise} \end{cases}$$



Subset Sum

- Recurrence:

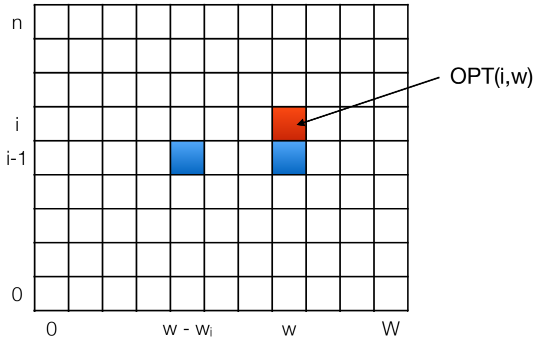
$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)) & \text{otherwise} \end{cases}$$



Subset Sum

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)) & \text{otherwise} \end{cases}$$



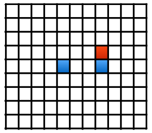
Subset Sum

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)) & \text{otherwise} \end{cases}$$

```
Array M[0…n][0…W]
Initialize M[0][w] = 0 for each w = 0,1,…,W
Subset-Sum(n,W)

Subset-Sum(i,w)
  if M[i][w] empty
    if w < wᵢ
      M[i][w] = Subset-Sum(i-1,w)
    else
      M[i][w] = max(Subset-Sum(i-1,w), wᵢ +
      Subsetsum(i-1,w-wᵢ))
  return M[i][w]
```



Subset Sum

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\underset{0}{\text{OPT}(i-1, w)}, \underset{1\ +\ 0}{w_i + \text{OPT}(i-1, w-w_i)}) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

9	5													?
8	4													?
5	3													?
2	2													?
1	1													1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

Subset Sum

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\underset{1}{\text{OPT}(i-1, w)}, \underset{2\ +\ 1}{w_i + \text{OPT}(i-1, w-w_i)}) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

9	5													?
8	4													?
5	3													?
2	2													3
1	1											1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

Subset Sum

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

9	5													?
8	4													?
5	3													?
2	2								?					3
1	1						1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

Subset Sum

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)) & \text{otherwise} \end{cases}$$

3 5 + 3

- Example
  - {1, 2, 5, 8, 9} and W = 12

9	5													?
8	4													?
5	3													8
2	2								3					3
1	1						1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

Subset Sum

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

9	5													?
8	4													?
5	3					?								8
2	2								3					3
1	1						1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

Subset Sum

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

9	5													?
8	4													?
5	3					?								8
2	2					?			3					3
1	1					?	1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

Subset Sum

- Recurrence:

$$\text{OPT}(i,w) = \begin{cases} \text{OPT}(i-1,w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1,w), w_i + \text{OPT}(i-1,w-w_i)) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

		0	1	2	3	4	5	6	7	8	9	10	11	12
9	5													?
8	4													?
5	3					?								8
2	2					?			3					3
1	1			?		1	1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Subset Sum

- Recurrence:

$$\text{OPT}(i,w) = \begin{cases} \text{OPT}(i-1,w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1,w), w_i + \text{OPT}(i-1,w-w_i)) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

		0	1	2	3	4	5	6	7	8	9	10	11	12
9	5													?
8	4													?
5	3					?								8
2	2					3			3					3
1	1			1		1	1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Subset Sum

- Recurrence:

$$\text{OPT}(i,w) = \begin{cases} \text{OPT}(i-1,w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1,w), w_i + \text{OPT}(i-1,w-w_i)) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

		0	1	2	3	4	5	6	7	8	9	10	11	12
9	5													?
8	4													?
5	3					3								8
2	2					3			3					3
1	1			1		1	1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0

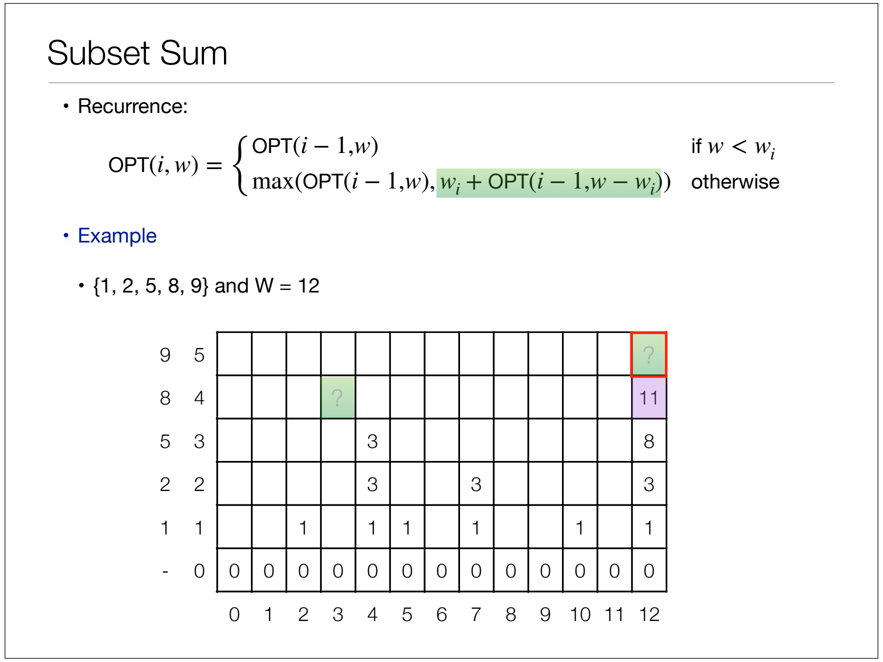
Subset Sum

- Recurrence:

$$\text{OPT}(i,w) = \begin{cases} \text{OPT}(i-1,w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1,w), w_i + \text{OPT}(i-1,w-w_i)) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

		0	1	2	3	4	5	6	7	8	9	10	11	12
9	5													?
8	4													11
5	3					3								8
2	2					3			3					3
1	1			1		1	1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0



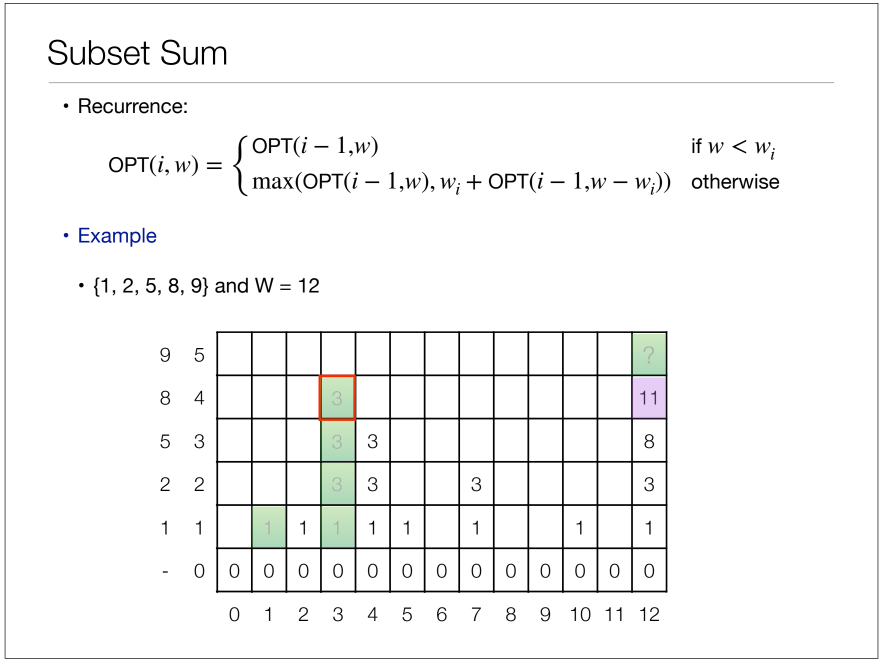
Subset Sum

- Recurrence:

$$\text{OPT}(i,w) = \begin{cases} \text{OPT}(i-1,w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1,w), w_i + \text{OPT}(i-1,w-w_i)) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

9	5													?
8	4				?									11
5	3					3								8
2	2					3			3					3
1	1			1		1	1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12



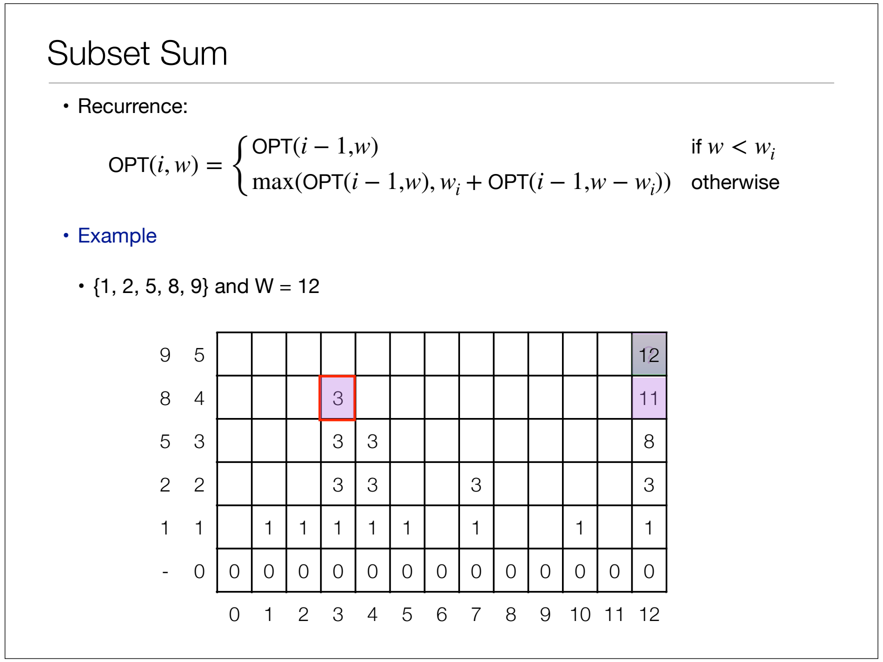
Subset Sum

- Recurrence:

$$\text{OPT}(i,w) = \begin{cases} \text{OPT}(i-1,w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1,w), w_i + \text{OPT}(i-1,w-w_i)) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

9	5													?
8	4				3									11
5	3				3	3								8
2	2				3	3			3					3
1	1		1	1	1	1	1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12



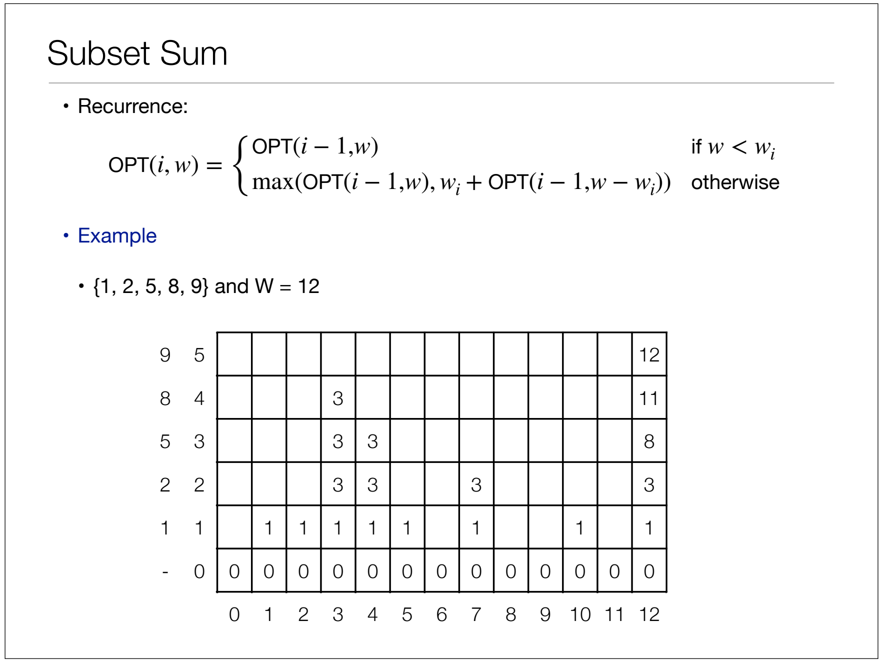
Subset Sum

- Recurrence:

$$\text{OPT}(i,w) = \begin{cases} \text{OPT}(i-1,w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1,w), w_i + \text{OPT}(i-1,w-w_i)) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

9	5													12
8	4				3									11
5	3				3	3								8
2	2				3	3			3					3
1	1		1	1	1	1	1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12



Subset Sum

- Recurrence:

$$\text{OPT}(i,w) = \begin{cases} \text{OPT}(i-1,w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1,w), w_i + \text{OPT}(i-1,w-w_i)) & \text{otherwise} \end{cases}$$

- Example
  - {1, 2, 5, 8, 9} and W = 12

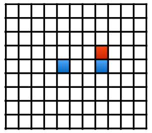
9	5													12
8	4				3									11
5	3				3	3								8
2	2				3	3			3					3
1	1		1	1	1	1	1		1			1		1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

Subset Sum

- Recurrence:

$$\text{OPT}(i,w) = \begin{cases} \text{OPT}(i-1,w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1,w), w_i + \text{OPT}(i-1,w-w_i)) & \text{otherwise} \end{cases}$$

```
Subset-Sum(n,W)
  Array M[0…n][0…W]
  Initialize M[0][w] = 0 for each w = 0,1,…,W
  for i = 1 to n
    for w = 0 to W
      if w < w_i
        M[i][w] = M[i-1][w]
      else
        M[i][w] = max(M[i-1][w], w_i + M[i-1][w-w_i])
  return M[n,W]
```

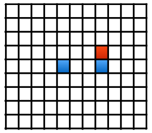


Subset Sum

- Recurrence:

$$\text{OPT}(i,w) = \begin{cases} \text{OPT}(i-1,w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1,w), w_i + \text{OPT}(i-1,w-w_i)) & \text{otherwise} \end{cases}$$

- Running time:
  - Number of subproblems = nW
  - Constant time on each entry $\Rightarrow O(nW)$
  - *Pseudo-polynomial time.*
    - Not polynomial in input size:
      - whole input can be described in O(n log n + n log w) bits, where w is the maximum weight (including W) in the instance.



Knapsack

- Knapsack
  - Given *n* items $\{1,\ldots,n\}$
  - Item i has weight w_i and value v_i
  - Bound W
  - Goal: Select maximum *value* subset S of items so that

$$\sum_{i \in S} w_i \leq W$$

- Example

Optimal solution: {vol 2, vol 3} has value 40

Capacity 12

value	1	6	18	22	28
weight	2	3	5	6	9

Knapsack

Knapsack

- $\mathcal{O}$ = optimal solution
- Consider element n.
  - Either in $\mathcal{O}$ or not.
    - $n \notin \mathcal{O}$: Optimal solution using items $\{1,\ldots,n-1\}$ is equal to $\mathcal{O}$.
    - $n \in \mathcal{O}$: Value of $\mathcal{O}$ = v_n + value on optimal solution on $\{1,\ldots,n-1\}$ with capacity $W - w_n$.

- Recurrence
  - OPT(i, w) = optimal solution on $\{1,\ldots,i\}$ with capacity w.

$$\text{OPT}(i,w) = \begin{cases} \text{OPT}(i-1,w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1,w), v_i + \text{OPT}(i-1,w-w_i)) & \text{otherwise} \end{cases}$$

- Running time $O(nW)$

Dynamic programming

- **First formulate the problem recursively.**
  - Describe the *problem* recursively in a clear and precise way.
  - Give a recursive formula for the problem.

- **Bottom-up**
  - Identify all the subproblems.
  - Choose a memoization data structure.
  - Identify dependencies.
  - Find a good evaluation order.

- **Top-down**
  - Identify all the subproblems.
  - Choose a memoization data structure.
  - Identify base cases.
  - Remember to save results and check before computing.

Sequence Alignment

Sequence alignment

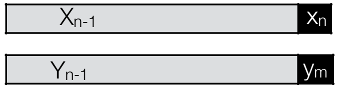
- How similar are ACAAGTC and CATGT.
- Align them such that
  - all items occurs in at most one pair.
  - no crossing pairs.
- Cost of alignment
  - gap penalty δ
  - mismatch cost for each pair of letters α(p,q).
- Goal: find minimum cost alignment.
- Input to problem: 2 strings X nd Y, gap penalty δ, and penalty matrix α(p,q).

```
A C A A G T C          A C A A - G T C
- C A T G T -          - C A - T G T -

1 mismatch, 2 gaps      0 mismatches, 4 gaps
```

Sequence Alignment

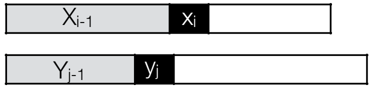
- Subproblem property.



- In the optimal alignment either:
  - x_n and y_m are aligned.
    - OPT = price of aligning x_n and y_m + minimum cost of aligning X_{i-1} and Y_{j-1}.
  - x_n and y_m are not aligned.
    - Either x_n and y_m (or both) is unaligned in OPT. Why?
    - OPT = δ + min(min cost of aligning X_{n-1} and Y_{m},
      min cost of aligning X_n and Y_{m-1})

Sequence Alignment

- Subproblem property.



- $SA(X_i,Y_j)$ = min cost of aligning strings X[1…i] and Y[1…j].

- Case 1. Align x_i and y_j.
  - Pay mismatch cost for x_i and y_j + min cost of aligning X_{i-1} and Y_{j-1}.
- Case 2. Leave x_i unaligned.
  - Pay gap cost + min cost of aligning X_{i-1} and Y_j.
- Case 3. Leave y_j unaligned.
  - Pay gap cost + min cost of aligning X_i and Y_{j-1}.

Sequence alignment

$$SA(X_i,Y_j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha(x_i,y_j) + SA(X_{i-1},Y_{j-1}), \\ \delta + SA(X_i,Y_{j-1}), \\ \delta + SA(X_{i-1},Y_j)\} \end{cases} & \text{otherwise} \end{cases}$$

		A	C	A	A	G	T	C
C								
A								
T						←		
G								
T								

$SA(X_5,Y_3)$

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0

Sequence alignment

$$SA(X_i,Y_j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha(x_i,y_j) + SA(X_{i-1},Y_{j-1}), \\ \delta + SA(X_i,Y_{j-1}), \\ \delta + SA(X_{i-1},Y_j)\} \end{cases} & \text{otherwise} \end{cases}$$

		A	C	A	A	G	T	C
C								
A								
T						🟥		
G								
T								

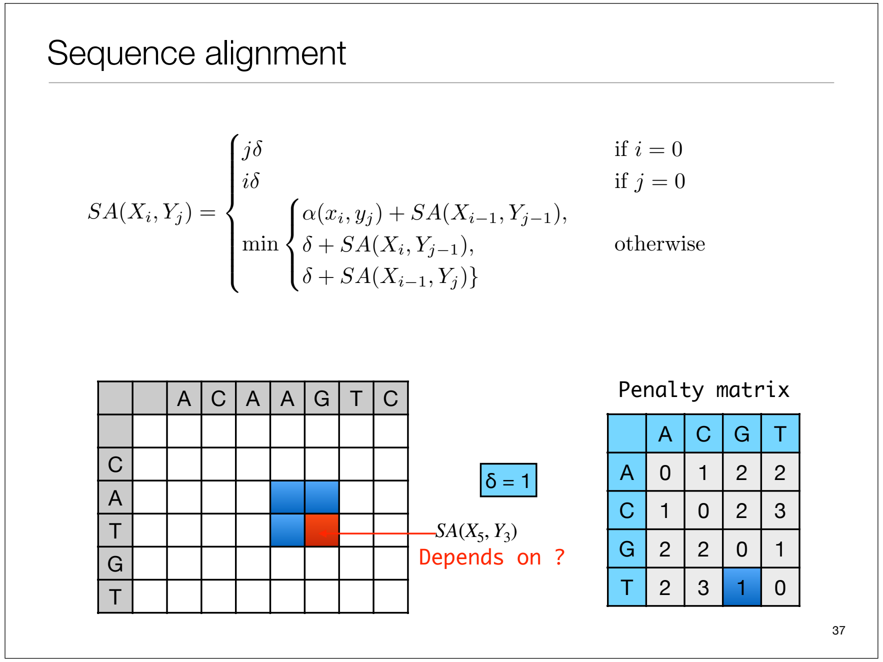
$SA(X_5,Y_3)$

Depends on ?

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

$$SA(X_i,Y_j) = \begin{cases} j\delta & \text{if } i=0 \\ i\delta & \text{if } j=0 \\ \min\begin{cases}\alpha(x_i,y_j)+SA(X_{i-1},Y_{j-1}),\\ \delta+SA(X_i,Y_{j-1}),\\ \delta+SA(X_{i-1},Y_j)\}\end{cases} & \text{otherwise}\end{cases}$$

		A	C	A	A	G	T	C
C								
A					🟦	🟦		
T					🟦	🟥		
G								
T								

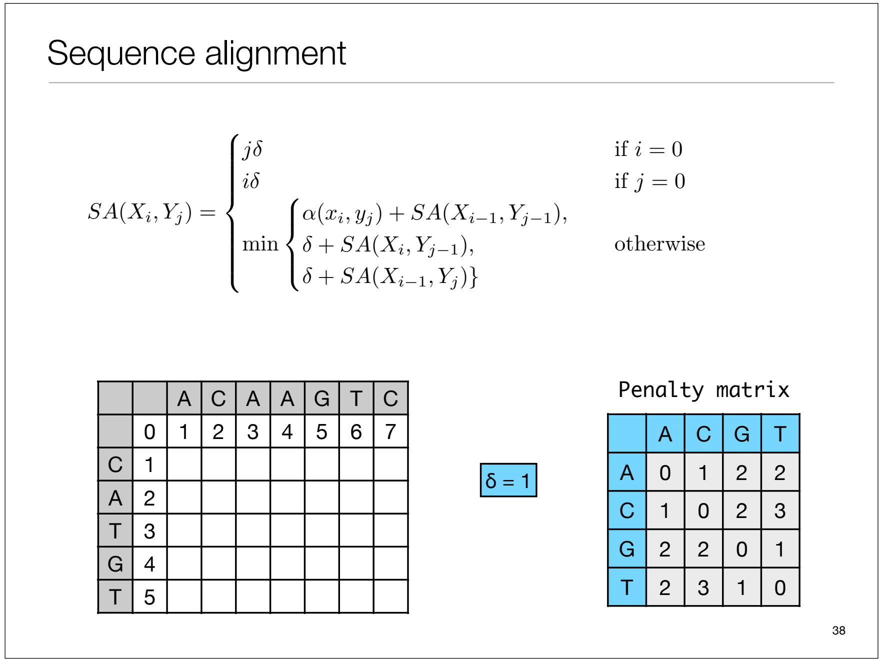
$SA(X_5, Y_3)$

Depends on ?

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

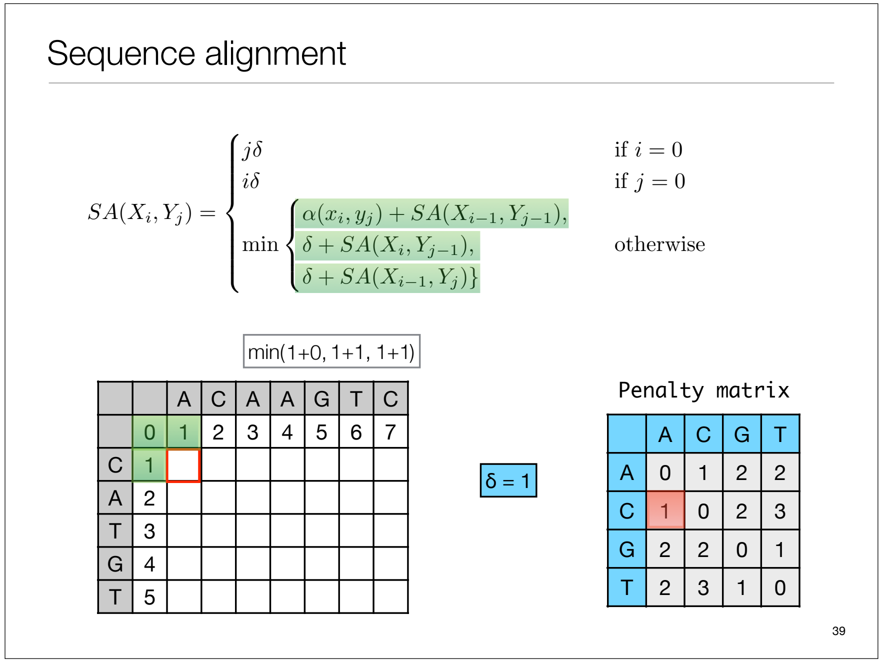
$$SA(X_i,Y_j) = \begin{cases} j\delta & \text{if } i=0 \\ i\delta & \text{if } j=0 \\ \min\begin{cases}\alpha(x_i,y_j)+SA(X_{i-1},Y_{j-1}),\\ \delta+SA(X_i,Y_{j-1}),\\ \delta+SA(X_{i-1},Y_j)\}\end{cases} & \text{otherwise}\end{cases}$$

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1							
A	2							
T	3							
G	4							
T	5							

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

$$SA(X_i,Y_j) = \begin{cases} j\delta & \text{if } i=0 \\ i\delta & \text{if } j=0 \\ \min\begin{cases}\alpha(x_i,y_j)+SA(X_{i-1},Y_{j-1}),\\ \delta+SA(X_i,Y_{j-1}),\\ \delta+SA(X_{i-1},Y_j)\}\end{cases} & \text{otherwise}\end{cases}$$

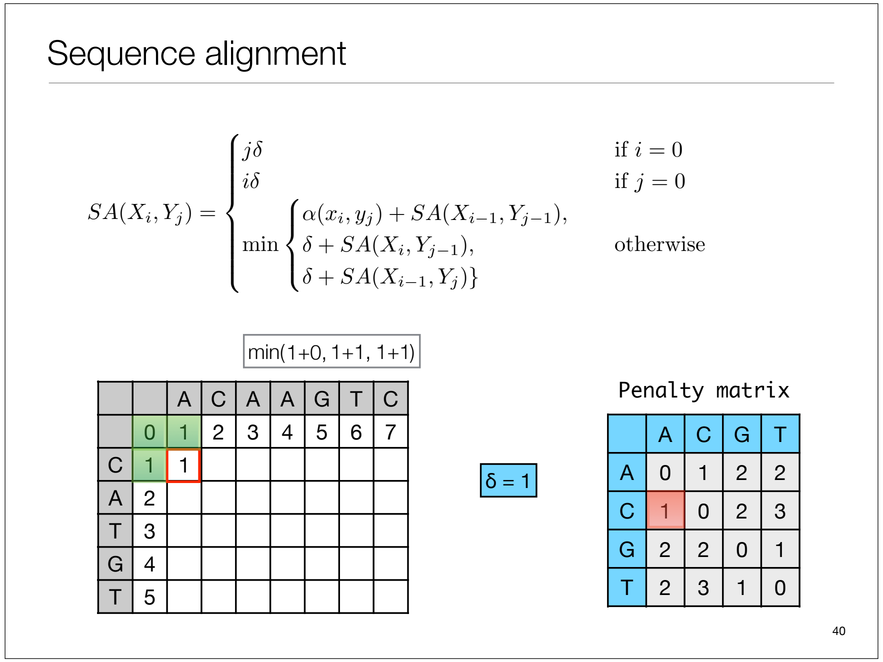
min(1+0, 1+1, 1+1)

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1							
A	2							
T	3							
G	4							
T	5							

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

$$SA(X_i,Y_j) = \begin{cases} j\delta & \text{if } i=0 \\ i\delta & \text{if } j=0 \\ \min\begin{cases}\alpha(x_i,y_j)+SA(X_{i-1},Y_{j-1}),\\ \delta+SA(X_i,Y_{j-1}),\\ \delta+SA(X_{i-1},Y_j)\}\end{cases} & \text{otherwise}\end{cases}$$

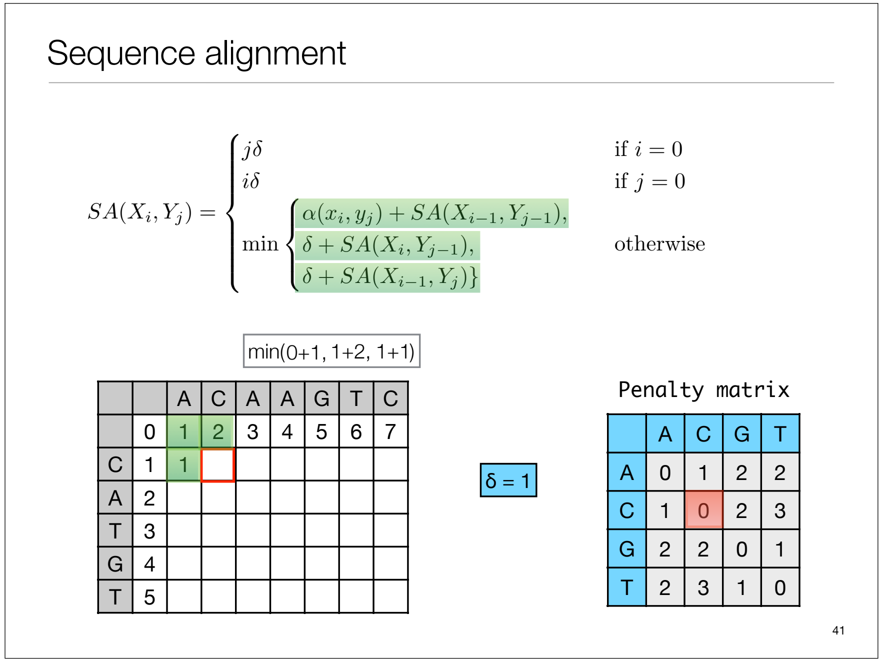
min(1+0, 1+1, 1+1)

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1	1						
A	2							
T	3							
G	4							
T	5							

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

$$SA(X_i, Y_j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha(x_i, y_j) + SA(X_{i-1}, Y_{j-1}), \\ \delta + SA(X_i, Y_{j-1}), \\ \delta + SA(X_{i-1}, Y_j)\} \end{cases} & \text{otherwise} \end{cases}$$

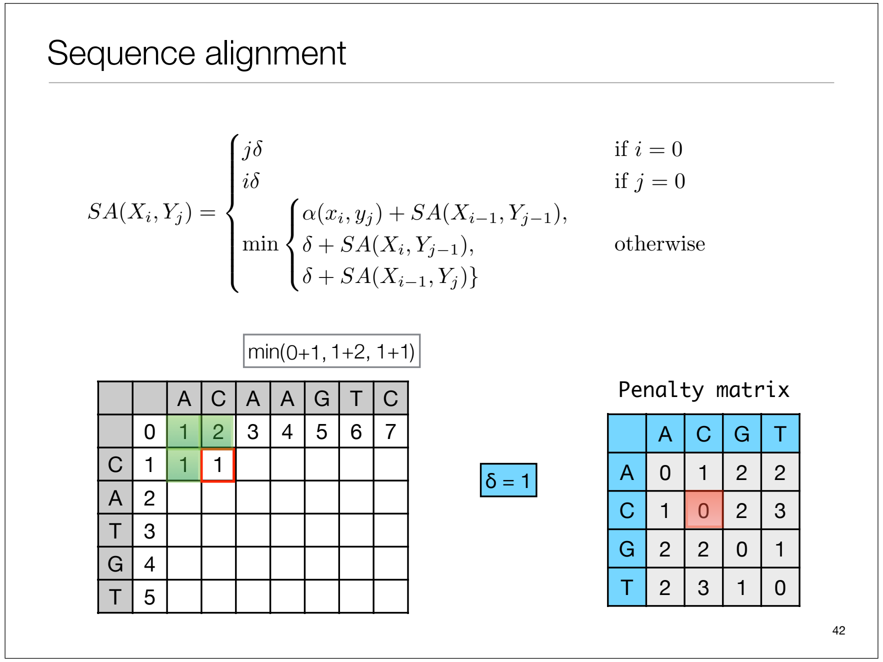
min(0+1, 1+2, 1+1)

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1	1						
A	2							
T	3							
G	4							
T	5							

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

$$SA(X_i, Y_j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha(x_i, y_j) + SA(X_{i-1}, Y_{j-1}), \\ \delta + SA(X_i, Y_{j-1}), \\ \delta + SA(X_{i-1}, Y_j)\} \end{cases} & \text{otherwise} \end{cases}$$

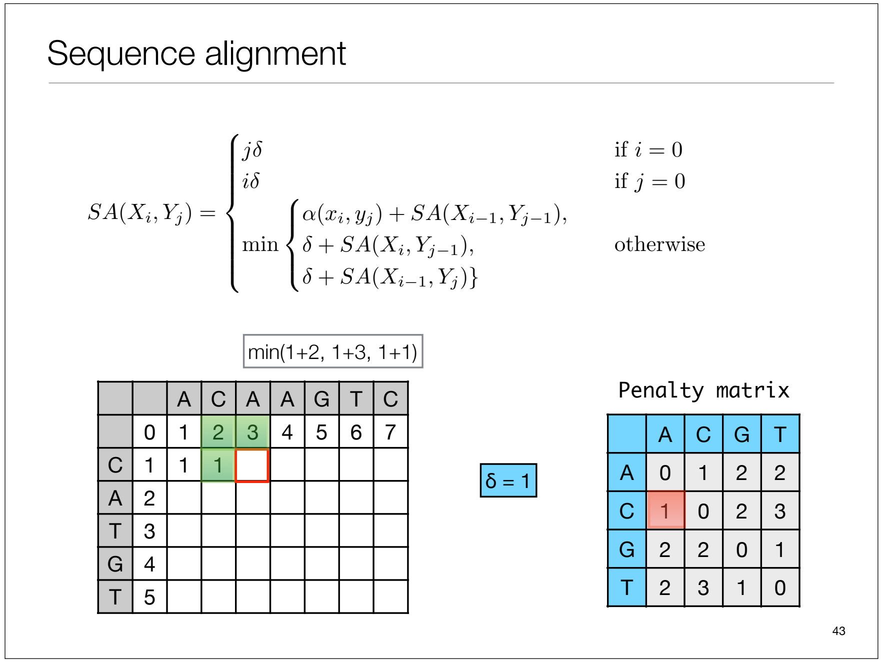
min(0+1, 1+2, 1+1)

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1	1	1					
A	2							
T	3							
G	4							
T	5							

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

$$SA(X_i, Y_j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha(x_i, y_j) + SA(X_{i-1}, Y_{j-1}), \\ \delta + SA(X_i, Y_{j-1}), \\ \delta + SA(X_{i-1}, Y_j)\} \end{cases} & \text{otherwise} \end{cases}$$

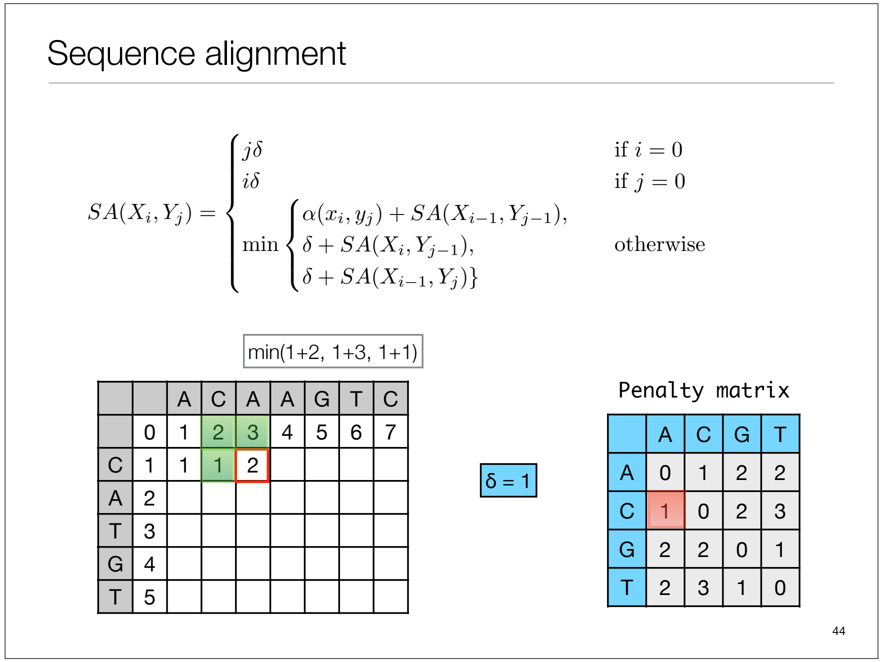
min(1+2, 1+3, 1+1)

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1	1	1					
A	2							
T	3							
G	4							
T	5							

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

$$SA(X_i, Y_j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha(x_i, y_j) + SA(X_{i-1}, Y_{j-1}), \\ \delta + SA(X_i, Y_{j-1}), \\ \delta + SA(X_{i-1}, Y_j)\} \end{cases} & \text{otherwise} \end{cases}$$

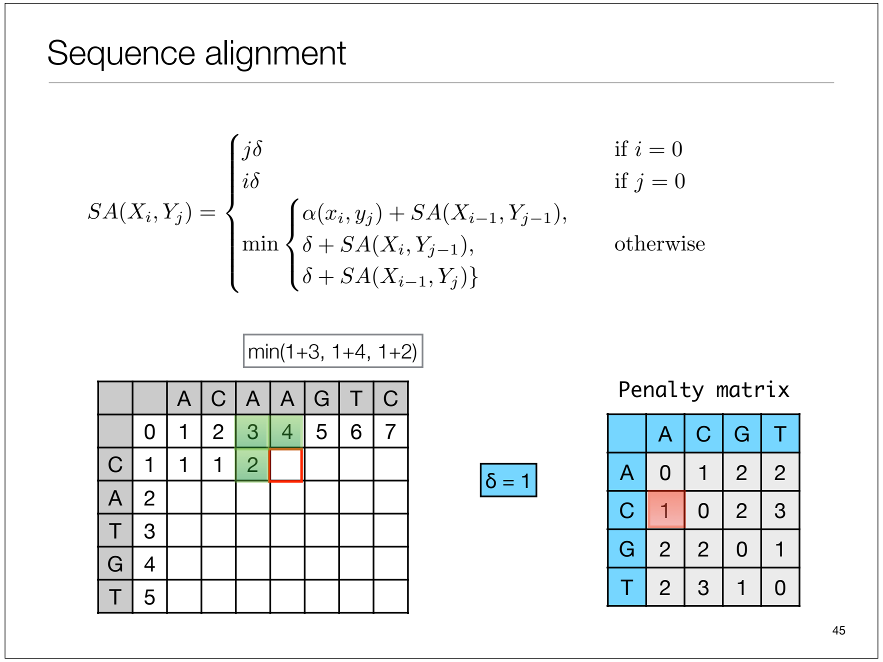
min(1+2, 1+3, 1+1)

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1	1	1	2				
A	2							
T	3							
G	4							
T	5							

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

$$SA(X_i, Y_j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha(x_i, y_j) + SA(X_{i-1}, Y_{j-1}), \\ \delta + SA(X_i, Y_{j-1}), \\ \delta + SA(X_{i-1}, Y_j)\} \end{cases} & \text{otherwise} \end{cases}$$

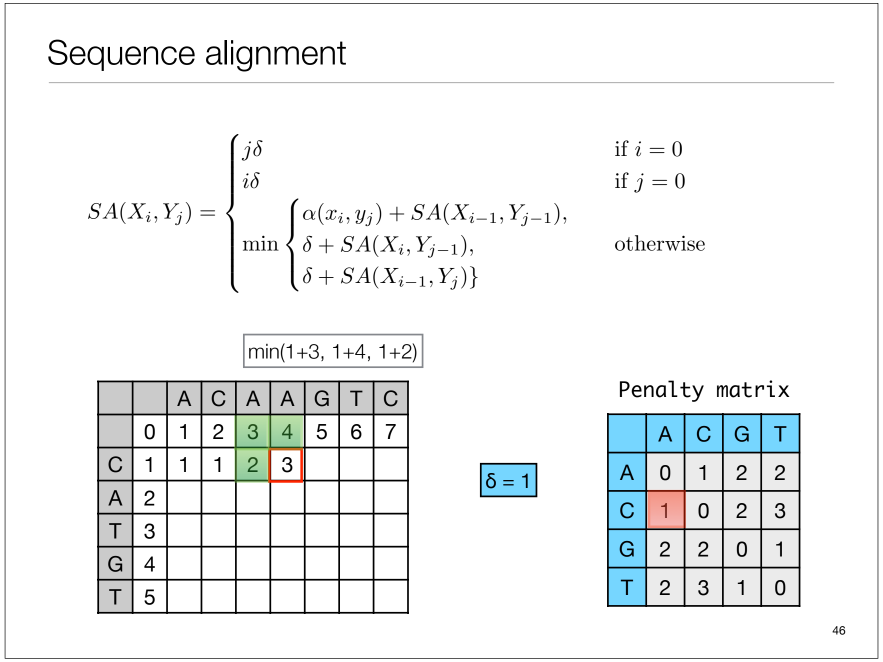
min(1+3, 1+4, 1+2)

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1	1	1	2				
A	2							
T	3							
G	4							
T	5							

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

$$SA(X_i, Y_j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha(x_i, y_j) + SA(X_{i-1}, Y_{j-1}), \\ \delta + SA(X_i, Y_{j-1}), \\ \delta + SA(X_{i-1}, Y_j)\} \end{cases} & \text{otherwise} \end{cases}$$

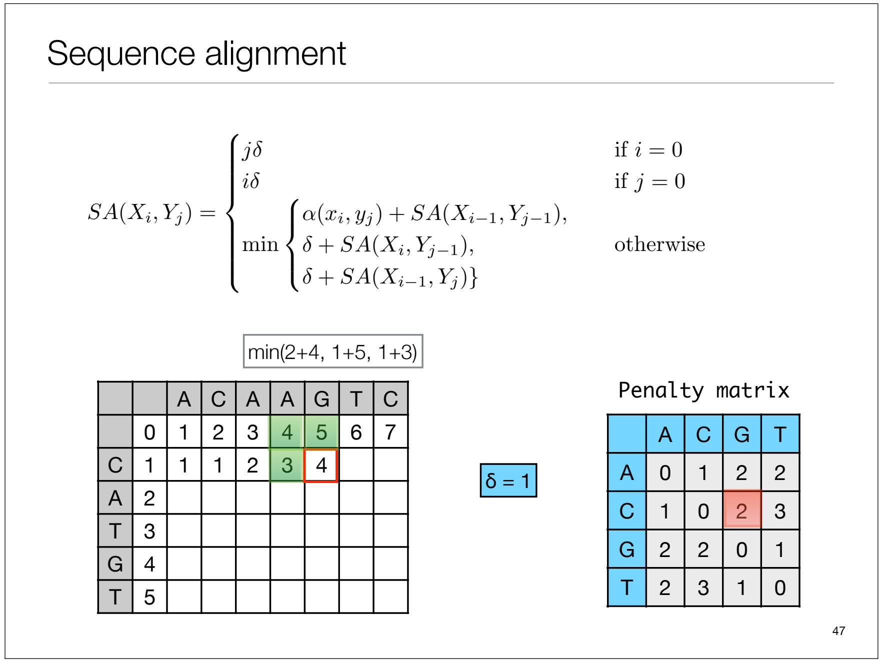
min(1+3, 1+4, 1+2)

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1	1	1	2	3			
A	2							
T	3							
G	4							
T	5							

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

$$SA(X_i, Y_j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha(x_i, y_j) + SA(X_{i-1}, Y_{j-1}), \\ \delta + SA(X_i, Y_{j-1}), \\ \delta + SA(X_{i-1}, Y_j)\} \end{cases} & \text{otherwise} \end{cases}$$

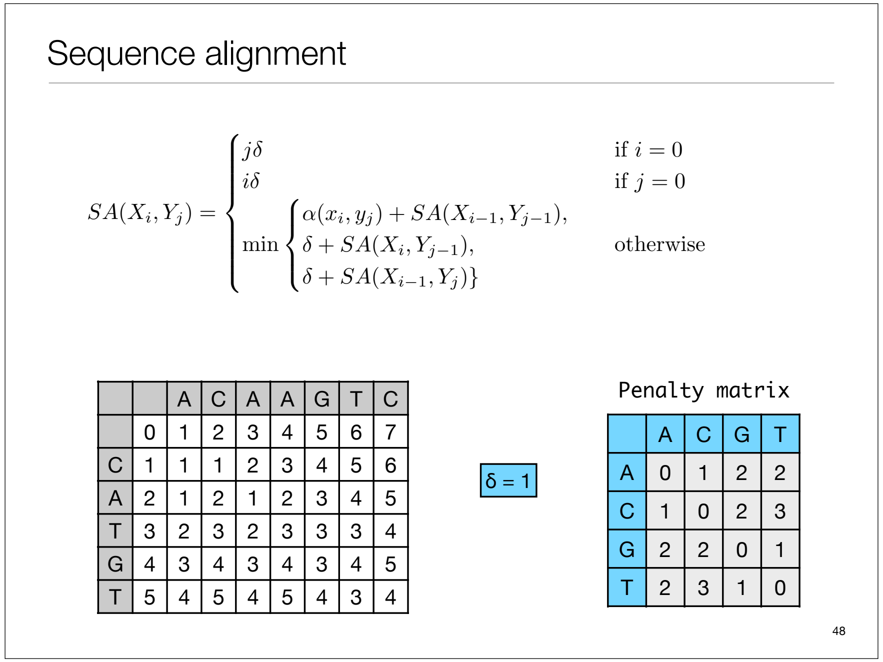
min(2+4, 1+5, 1+3)

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1	1	1	2	3	4		
A	2							
T	3							
G	4							
T	5							

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0



Sequence alignment

$$SA(X_i, Y_j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha(x_i, y_j) + SA(X_{i-1}, Y_{j-1}), \\ \delta + SA(X_i, Y_{j-1}), \\ \delta + SA(X_{i-1}, Y_j)\} \end{cases} & \text{otherwise} \end{cases}$$

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1	1	1	2	3	4	5	6
A	2	1	2	1	2	3	4	5
T	3	2	3	2	3	3	3	4
G	4	3	4	3	4	3	4	5
T	5	4	5	4	5	4	3	4

δ = 1

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0

Sequence alignment

```
SA(X[1…m],Y[1…n],δ,A){
  for i=0 to m
    M[i,0] := iδ

  for j=0 to n
    M[0,j] := jδ

  for i=1 to m
    for j = 1 to n
      M[i,j] := min{ A[i,j] + M[i-1,j-1],
                     δ + M[i-1,j],
                     δ + M[i,j-1]}

  Return M[m,n]
}
```

- Time: Θ(mn)
- Space: Θ(mn)

Sequence alignment: Finding the solution

$$SA(X_i,Y_j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min\begin{cases} \alpha(x_i,y_j) + SA(X_{i-1},Y_{j-1}), \\ \delta + SA(X_i,Y_{j-1}), \\ \delta + SA(X_{i-1},Y_j)\} \end{cases} & \text{otherwise} \end{cases}$$

Penalty matrix

	A	C	G	T
A	0	1	2	2
C	1	0	2	3
G	2	2	0	1
T	2	3	1	0

δ = 1

		A	C	A	A	G	T	C
	0	1	2	3	4	5	6	7
C	1	1	1	2	3	4	5	6
A	2	1	2	1	2	3	4	5
T	3	2	3	2	3	3	3	4
G	4	3	4	3	4	3	4	5
T	5	4	5	4	5	4	3	4

		A	C	A	A	G	T	C
		←	←	←	←	←	←	←
C	↑	↖	↖	←	←	←	←	↖
A	↑	↖	↖	↖	↖	←	←	←
T	↑	↑	↑	↑	↖	↖	↖	←
G	↑	↑	↖	↑	↖	↖	↖	↖
T	↑	↑	↑	↑	↖	↑	↖	←

Sequence alignment

- Use dynamic programming to compute an optimal alignment.
  - Time: Θ(mn)
  - Space: Θ(mn)
- Find actual alignment by backtracking (or saving information in another matrix).
- Linear space?
  - Easy to compute value (save last and current row)
  - How to compute alignment? Hirschberg. (not part of the curriculum).

Reading material

At the lecture we will continue with dynamic programming. We will talk about the knapsack problem and sequence alignment. You should read KT section 6.4 and 6.6.

Exercises

1 [w] **Knapsack** Solve the following knapsack problem by filling out the table below. The items are givens as pairs (w_i, v_i): $(5,7),(2,6),(3,3),(2,1)$. The capacity $W = 6$.

4							
3							
2							
1							
0							
$i \setminus w$	0	1	2	3	4	5	6

2 [w] **Sequence alignment** Consider the strings `APPLE` and `PAPE` over the alphabet $\Sigma = \{\texttt{A},\texttt{E},\texttt{L},\texttt{P}\}$ and a penalty matrix P:

	A	E	L	P
A	0	1	3	1
E	1	0	2	1
L	3	2	0	2
P	1	1	2	0

Compute the sequence alignment of the two strings when the penalty for a gap $\delta = 2$. Fill the dynamic programming table below, and explain how the minimum cost sequence alignment is found in it.

	j	0	1	2	3	4	5
i			A	P	P	L	E
0							
1	P						
2	A						
3	P						
4	E						

3 **Book Shop** You are in a book shop which sells n different books. You know the price h_i and number of pages s_i of each book $i \in \{1,\dots,n\}$. You have decided that the total price of your purchases will be at most x and you will buy each book at most once.

3.1 Give an algorithm that computes the maximum number of pages you can buy. Analyse the time and space usage of your algorithm.

3.2 Modify your algorithm to only use $O(x)$ space (if it doesn't already). It is possible to solve the problem using only a single 1-dimensional array D.

3.3 Implement your linear space algorithm on CSES: https://cses.fi/problemset/task/1158

4 Longest palindrome subsequence A *palindrome* is a (nonempty) string over an alphabet Σ that reads the same forward and backward. For example are `abba` and `racecar` palindromes. A string P is a *subsequence* of string T if we can obtain P from T by removing 0 or more characters in T. For instance, `abba` is a subsequence of `bcadfbbba`.

Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. To do this first give a recurrence for the problem and then write pseudo code for an algorithm based on your recurrence and dynamic programming. Argue that your recurrence is correct and analyse the running time and space usage of your algorithm.

5 Defending Zion Solve KT 6.8

Puzzle of the week: The Blind Man A blind man was handed a deck of 52 cards with exactly 10 cards facing up. How can he divide it into two piles, each of which having the same number of cards facing up?