

Algorithms and Data Structures 2

Exam Notes

Week 3: Dynamic Programming II

Mads Richardt

1 General Methodology and Theory

Dynamic Programming Principles

- **Optimal substructure:** Break problem into overlapping subproblems.
- **Recurrence relation:** Express solution to larger subproblem in terms of smaller subproblems.
- **Evaluation order:** Fill DP table bottom-up or use memoized recursion (top-down).
- **Time/space complexity:** Usually proportional to number of subproblems.

Subset Sum and Knapsack

Problem: Given n items with weight w_i and value v_i , and capacity W , maximize

$$\sum_{i \in S} v_i \quad \text{s.t.} \quad \sum_{i \in S} w_i \leq W.$$

Recurrence:

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

Algorithm (bottom-up):

```
Array M[0..n][0..W]
Initialize M[0][w] = 0 for all w
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]
```

Complexity: $O(nW)$ time, $O(nW)$ space (can be optimized to $O(W)$).

Sequence Alignment

Problem: Given strings $X = x_1 \dots x_m$ and $Y = y_1 \dots y_n$, gap penalty δ , mismatch costs α_{pq} , find minimum-cost alignment.

Recurrence:

$$OPT(i, j) = \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1), \\ \delta + OPT(i-1, j), \\ \delta + OPT(i, j-1). \end{cases}$$

Algorithm:

```

Array A[0..m][0..n]
A[i][0] = i*, A[0][j] = j*
For i=1..m:
  For j=1..n:
    A[i][j] = min( [xi,yj]+A[i-1][j-1],
                  + A[i-1][j],
                  + A[i][j-1] )
Return A[m][n]

```

Complexity: $O(mn)$ time, $O(mn)$ space. Backtracking reconstructs alignment.

Longest Palindromic Subsequence

Let $S[1..n]$ be input string. Define $L(i, j)$ = length of longest palindromic subsequence in $S[i..j]$.

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

Algorithm builds $n \times n$ table.

Supporting Math Tools

- Max/min operators, recurrence solving.
- Pseudo-polynomial runtime: $O(nW)$ depends on W .
- Backtracking in DP tables for solution reconstruction.

2 Notes from Slides and Textbook

- Knapsack: dynamic programming with recurrence (6.11), $O(nW)$ time.
- Sequence alignment: recurrence (6.16), $O(mn)$ time and space, shortest-path graph interpretation.
- DP design steps:
 1. Formulate recursively.
 2. Define subproblems.
 3. Identify base cases.
 4. Choose memoization structure and evaluation order.

3 Solutions to Problem Set

Exercise 1: Knapsack

Items: (5, 7), (2, 6), (3, 3), (2, 1), capacity $W = 6$.

We fill DP table $M[i, w]$. Optimal solution: pick items (2,6) and (3,3), weight = 5, value = 9.

Exercise 2: Sequence Alignment

Strings: APPLE vs PAPE, gap penalty $\delta = 2$, mismatch costs as given.

DP table constructed; minimal alignment cost = 3. Alignment:

A	P	P	L	E
A	P	-	P	E

Exercise 3: Book Shop

Recurrence:

$$OPT(i, x) = \max(OPT(i-1, x), s_i + OPT(i-1, x - h_i)).$$

Algorithm:

1. Input: n , budget x , arrays h_i, s_i .
2. Initialize $D[0..x] = 0$.
3. For each book i : for $j = x$ down to h_i : $D[j] = \max(D[j], s_i + D[j - h_i])$.
4. Answer = $D[x]$.

Time: $O(nx)$, **Space:** $O(x)$.

Exercise 4: Longest Palindrome Subsequence

Recurrence: See general section. Build DP table of size $n \times n$. **Pseudocode:**

1. Initialize $L[i, i] = 1$.
2. For subseq length $\ell = 2..n$, fill table using recurrence.
3. Answer: $L[1, n]$.

Time: $O(n^2)$, space $O(n^2)$.

Exercise 5: Defending Zion (KT 6.8)

Let $D[t] = \max$ robots destroyed up to time t . Recurrence:

$$D[t] = \max(D[t-1], \max_{j \leq t} \{D[t-j] + \min(x_t, f(j))\}).$$

Compute in $O(n^2)$. Possible optimizations depending on $f(\cdot)$.

Puzzle: The Blind Man

Divide 52 cards into two piles: take any 10 cards, flip them over. Each pile then has same number of face-up cards.

4 Summary

- **Knapsack recurrence:** $OPT(i, w) = \max(OPT(i-1, w), v_i + OPT(i-1, w - w_i))$.
- **Sequence alignment recurrence:** $OPT(i, j) = \min(\alpha_{x_i y_j} + OPT(i-1, j-1), \delta + OPT(i-1, j), \delta + OPT(i, j-1))$.
- **Longest palindrome subsequence:** $L(i, j) = 2 + L(i+1, j-1)$ if $S[i] = S[j]$, else $\max(L(i+1, j), L(i, j-1))$.
- **Defending Zion:** $D[t] = \max(D[t-1], \max_{j \leq t} \{D[t-j] + \min(x_t, f(j))\})$.
- **Complexities:** Knapsack $O(nW)$, Sequence 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

1

Dynamic Programming

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

2

Subset Sum and Knapsack

3

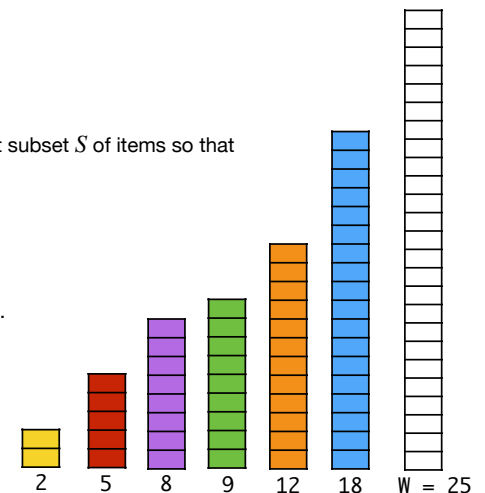
Subset Sum

- Subset Sum
 - Given n items $\{1, \dots, n\}$
 - 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

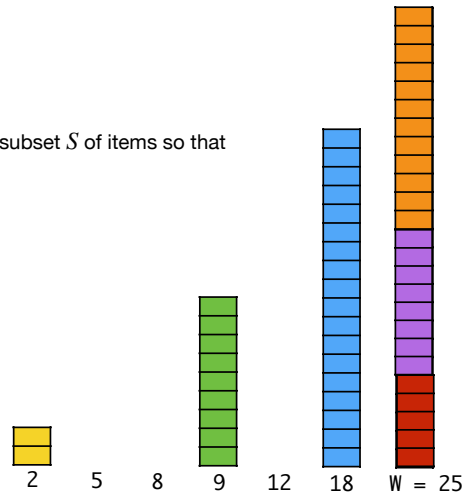
• Subset Sum

- Given n items $\{1, \dots, n\}$
- 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 $\{1, \dots, n-1\}$ is equal to \mathcal{O} .

• $n \in \mathcal{O}$: Value of $\mathcal{O} = w_n$ + weight of optimal solution on $\{1, \dots, n-1\}$ with capacity $W - w_n$.

• Recurrence

• $\text{OPT}(i, w)$ = optimal solution on $\{1, \dots, i\}$ with capacity w .

• From above:

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

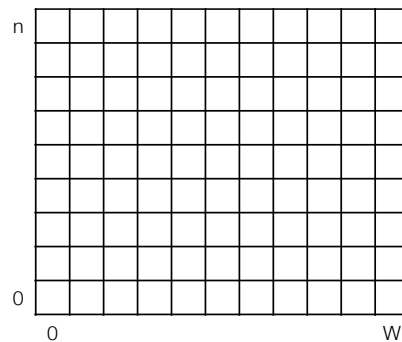
• If $w_n > W$:

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

Subset Sum

• Recurrence:

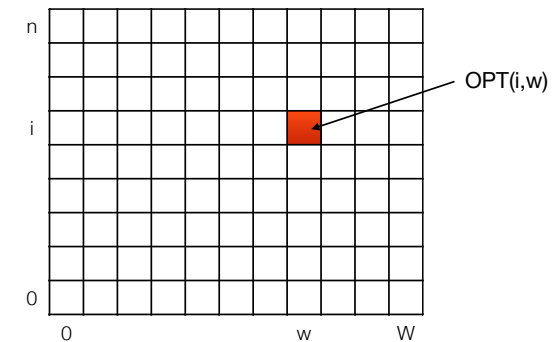
$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)) & \text{otherwise} \end{cases}$$



Subset Sum

• Recurrence:

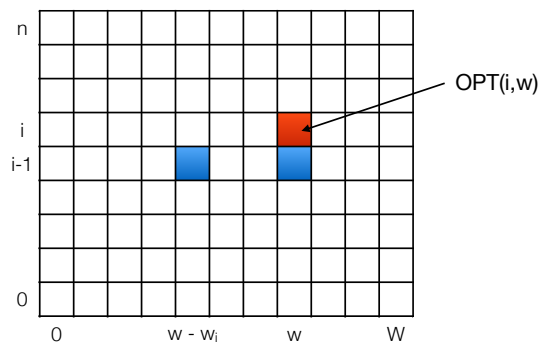
$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)) & \text{otherwise} \end{cases}$$



Subset Sum

- Recurrence:

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



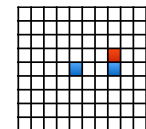
Subset Sum

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



Subset Sum

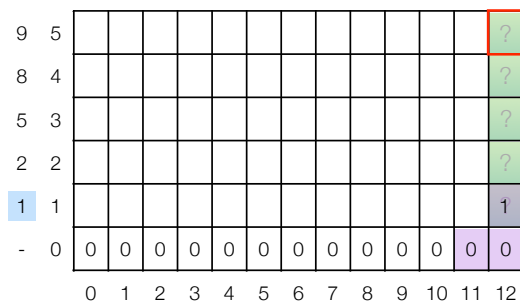
- Recurrence:

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

$\underset{0}{\text{OPT}(i-1, w)}$
 $\underset{1+0}{w_i + \text{OPT}(i-1, w - w_i)}$

- Example

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



Subset Sum

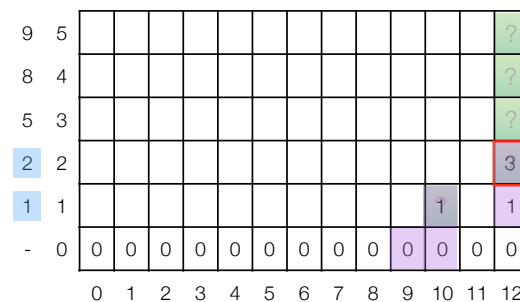
- Recurrence:

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

$\underset{1}{\text{OPT}(i-1, w)}$
 $\underset{2+1}{w_i + \text{OPT}(i-1, w - w_i)}$

- Example

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



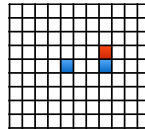
Subset Sum

- 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 < wi
        M[i][w] = M[i-1][w]
      else
        M[i][w] = max(M[i-1][w], wi + M[i-1][w-wi])
  return M[n, W]
    
```



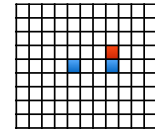
Subset Sum

- 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 $\{1, \dots, n\}$
- 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



Capacity 12

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

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

Knapsack



Knapsack

- \mathcal{O} = optimal solution
- Consider element n .
 - Either in \mathcal{O} or not.
 - $n \notin \mathcal{O}$: Optimal solution using items $\{1, \dots, n-1\}$ is equal to \mathcal{O} .
 - $n \in \mathcal{O}$: Value of $\mathcal{O} = v_n$ + value on optimal solution on $\{1, \dots, n-1\}$ with capacity $W - w_n$.
- **Recurrence**
 - $\text{OPT}(i, w)$ = optimal solution on $\{1, \dots, i\}$ 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 $a(p, q)$.
- Goal: find minimum cost alignment.
- Input to problem: 2 strings X and Y , gap penalty δ , and penalty matrix $a(p, q)$.

A C A A G T C

- C A T G T -

1 mismatch, 2 gaps

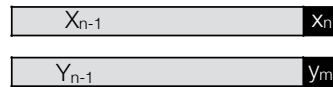
A C A A - G T C

- C A - T G T -

0 mismatches, 4 gaps

Sequence Alignment

- Subproblem property.



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

33

Sequence Alignment

- Subproblem property.



- $\text{SA}(X_i, Y_j) = \text{min cost of aligning strings } X[1 \dots i] \text{ and } Y[1 \dots j].$
- 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} .

34

Sequence alignment

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

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

$\delta = 1$

$\text{SA}(X_5, Y_3)$

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

35

Sequence alignment

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

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

$\delta = 1$

$\text{SA}(X_5, Y_3)$
Depends on ?

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

36

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								

$\delta = 1$

$SA(X_5, Y_3)$
Depends on ?

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

37

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							

$\delta = 1$

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

38

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							

$\delta = 1$

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

39

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							

$\delta = 1$

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

40

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							

$$\delta = 1$$

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

41

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							

$$\delta = 1$$

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

42

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							

$$\delta = 1$$

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

43

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							

$$\delta = 1$$

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

44

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							

$\delta = 1$

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

45

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						

$\delta = 1$

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

46

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						

$\delta = 1$

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

47

Sequence alignment

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

$\delta = 1$

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

48

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: $\Theta(mn)$
- Space: $\Theta(mn)$

49

Sequence alignment: Finding the solution

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

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

$\delta = 1$

		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	↑	↑	↑	↑	↖	↑	↖	←

50

Sequence alignment

- Use dynamic programming to compute an optimal alignment.
 - Time: $\Theta(mn)$
 - Space: $\Theta(mn)$
- 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).

51

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 given 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 $\Sigma = \{A, E, L, P\}$ 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 bcadfbba.

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?