CS514: Design and Analysis of Algorithms

Recursion: Dynamic Programming

Arijit Mondal

Dept of CSE

arijit@iitp.ac.in

https://www.iitp.ac.in/~arijit/

Recursive modeling

- Classical Sanskrit poetry distinguishes between two types of syllables *(aksara)*: light *(laghu)* and heavy *(guru)*. In one class of meters, each line of poetry consists of a fixed number of "beats" *(matra)*, where each light syllable lasts one beat and each heavy syllable lasts two beats. *Pingala* observed that there are exactly five 4-beat meters: $--$, $-\cdot\cdot$, $\cdot\cdot-$, $\cdot-\cdot$, and $\cdot\cdot\cdot\cdot$. How many n-beat meters are possible?

CS514

Recursive modeling

- Classical Sanskrit poetry distinguishes between two types of syllables *(aksara)*: light *(laghu)* and heavy *(guru)*. In one class of meters, each line of poetry consists of a fixed number of "beats" *(matra)*, where each light syllable lasts one beat and each heavy syllable lasts two beats. *Pingala* observed that there are exactly five 4-beat meters: $--$, $-\cdot\cdot$, $\cdot\cdot-$, $\cdot-\cdot$, and $\cdot\cdot\cdot\cdot$. How many n-beat meters are possible?
- Consider implementation of cell towers along a straight highway. There are n possible locations $(c_1, \ldots, c_n)$ available. The i-th location can serve p_i number of people. You can build cell towers in any location as long as you don't build towers in adjacent locations. What is the largest number of people you can cover?

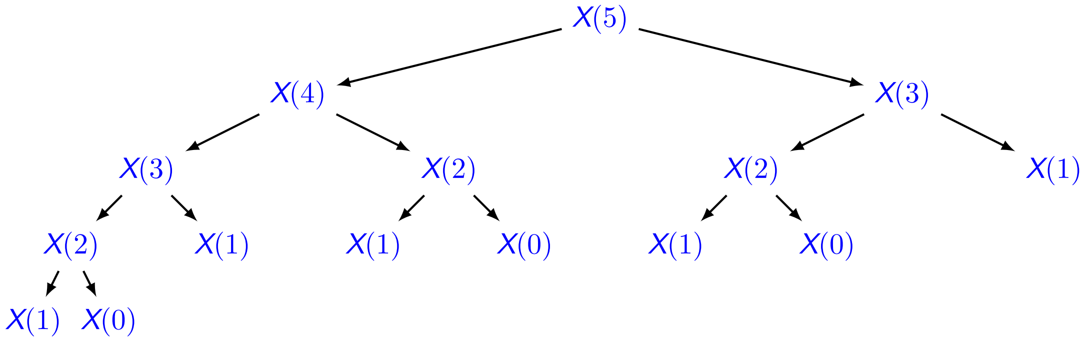
C	1	2	3	4	5
P	49	42	85	140	60

Recursive modeling

- n beats: $T(n) = T(n-1) + T(n-2)$, $T(0) = 1, T(1) = 1$
- Cell tower: $S(n) = \max\{S(n-1), p_n + S(n-2)\}$, $T(0) = 0, T(1) = p_1$

Recursive modeling

- n beats: $T(n) = T(n-1) + T(n-2)$, $T(0) = 1$, $T(1) = 1$
- Cell tower: $S(n) = \max\{S(n-1), p_n + S(n-2)\}$, $T(0) = 0$, $T(1) = p_1$



Dynamic Programming

- **Overlapping subproblems** — Different branches of the recursion will reuse each other's work.
- **Optimal substructure** — The optimal solution for one problem instance is formed from optimal solutions for smaller problems.
- **Polynomial subproblems** — The number of subproblems is small enough to be evaluated in polynomial time.
- A **dynamic programming** algorithm is one that evaluates all subproblems in a particular order to ensure that all subproblems are evaluated only once.

Rod cutting-1

- Given a rod of length n inches and a table of prices p_i for $i = 1, \ldots, n$, determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces
- Example:

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Rod cutting-1

- Given a rod of length n inches and a table of prices p_i for $i = 1, \ldots, n$, determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces
- Example:

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- Recursive definition: $r_n = \max\{p_i + r_{n-i} : 1 \le i \le n\}$

Rod cutting: Top-down-1

- Cut-Rod(p, n)

1. if $n = 0$ return 0
2. $q = -\infty$
3. for i=1 to n
4. $\quad q = \max\{q, p_i + \text{Cut-Rod}(p, n - i)\}$
5. return q

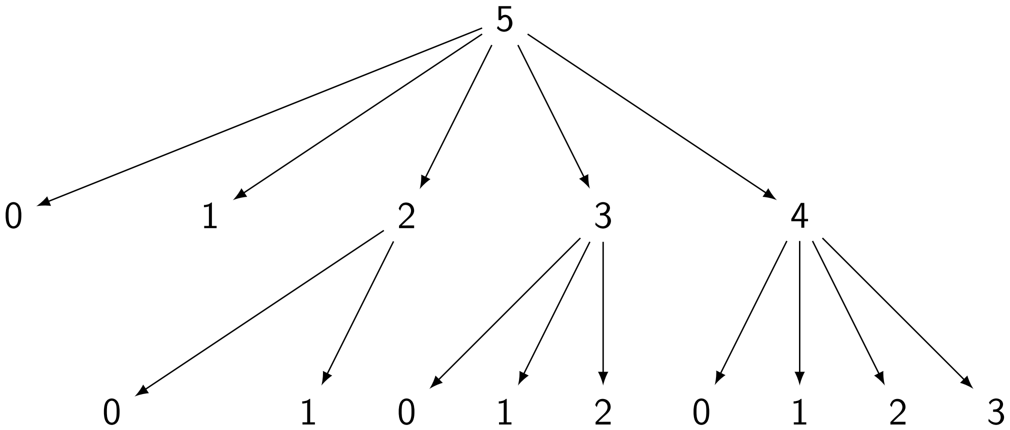
Rod cutting: Top-down-2

- Initialize $LUT[i] = -\infty \; \forall i = 1, \ldots, n$
- Cut-Rod-Memoized(p, n, LUT)

1. if $LUT[n] \geq 0$ return $LUT[n]$
2. if $n == 0$ then $q = 0$
3. else
4. $\quad q = -\infty$
5. $\quad$ for i=1 to n
6. $\qquad q = \max\{q, p_i + \text{Cut-Rod-Memoized}(p, n-i, LUT)\}$
7. $LUT[n] = q$
8. return q

Rod cutting: Recursion Tree

Rod cutting: Recursion Tree



Rod cutting: Bottom-up-1

- Cut-Rod-Bottom-up(p, n)

1. Let $r[n]$ be an array, r[0]=0;
2. for j=1 to n
3. $\quad q = -\infty$
4. $\quad$ for i=1 to j
5. $\quad\quad q = \max\{q, p_i + r[j-i]\}$
6. $\quad r[j] = q$
7. return $r[n]$

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

index	
revenue	
cut-point	

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

index	1
revenue	1
cut-point	1

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

index	1	2
revenue	1	5
cut-point	1	2

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

index	1	2	3
revenue	1	5	8
cut-point	1	2	3

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

index	1	2	3	4
revenue	1	5	8	10
cut-point	1	2	3	2

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

index	1	2	3	4	5
revenue	1	5	8	10	13
cut-point	1	2	3	2	2

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

index	1	2	3	4	5	6
revenue	1	5	8	10	13	17
cut-point	1	2	3	2	2	6

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

index	1	2	3	4	5	6	7
revenue	1	5	8	10	13	17	18
cut-point	1	2	3	2	2	6	1

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

index	1	2	3	4	5	6	7	8
revenue	1	5	8	10	13	17	18	22
cut-point	1	2	3	2	2	6	1	2

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

index	1	2	3	4	5	6	7	8	9
revenue	1	5	8	10	13	17	18	22	25
cut-point	1	2	3	2	2	6	1	2	3

Rod cutting: Example

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

index	1	2	3	4	5	6	7	8	9	10
revenue	1	5	8	10	13	17	18	22	25	30
cut-point	1	2	3	2	2	6	1	2	3	10

Rod cutting: Bottom-up-2

- Cut-Rod-Bottom-up(p, n)

1. Let $r[n]$, $s[n]$ be two arrays, r[0]=0;
2. for j=1 to n
3. $\quad q = -\infty$
4. $\quad$ for i=1 to j
5. $\quad\quad$ if $q < p_i + r[j-i]$
6. $\quad\quad\quad q = p_i + r[j-i]$; $s[j] = i$
7. $\quad r[j] = q$
8. return r and s

Rod cutting: Bottom-up-2

- Cut-Rod-Bottom-up(p, n)
  1. Let $r[n], s[n]$ be two arrays, r[0]=0;
  2. for j=1 to n
  3. $\quad q = -\infty$
  4. $\quad$ for i=1 to j
  5. $\quad\quad$ if $q < p_i + r[j-i]$
  6. $\quad\quad\quad q = p_i + r[j-i]$; $s[j] = i$
  7. $\quad r[j] = q$
  8. return r and s
- Print-Soln(p, n, s):
  1. while $n > 0$
  2. $\quad$ print $s[n]$
  3. $\quad n = n - s[n]$

Matrix Chain Multiplication

- Given a sequence (chain) $\langle A_1, \ldots, A_n \rangle$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parentesize the product $A_1 \times \ldots \times A_n$ in way that minimizes the number of scalar multiplications.

Matrix Chain Multiplication

- Given a sequence (chain) $\langle A_1, \ldots, A_n \rangle$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parentesize the product $A_1 \times \ldots \times A_n$ in way that minimizes the number of scalar multiplications.
- For $n = 4$ we have the following options

$$(A_1(A_2(A_3A_4))), \quad ((A_1A_2)(A_3A_4)), \quad ((A_1(A_2A_3))A_4)$$
$$(A_1((A_2A_3)A_4)), \quad (((A_1A_2)A_3)A_4)$$

Matrix Chain Multiplication

- Given a sequence (chain) $\langle A_1, \ldots, A_n \rangle$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parentesize the product $A_1 \times \ldots \times A_n$ in way that minimizes the number of scalar multiplications.
- For $n = 4$ we have the following options

$$(A_1(A_2(A_3A_4))), \quad ((A_1A_2)(A_3A_4)), \quad ((A_1(A_2A_3))A_4)$$
$$(A_1((A_2A_3)A_4)), \quad (((A_1A_2)A_3)A_4)$$

- Let the dimension of matrices are as follows - $5 \times 2, 2 \times 1, 1 \times 10, 10 \times 100$. Compute the number of multiplications for above cases.

Matrix Chain Multiplication

- Given a sequence (chain) $\langle A_1, \ldots, A_n \rangle$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parentesize the product $A_1 \times \ldots \times A_n$ in way that minimizes the number of scalar multiplications.
- For $n = 4$ we have the following options

  $(A_1(A_2(A_3A_4)))$, $((A_1A_2)(A_3A_4))$, $((A_1(A_2A_3))A_4)$
  $(A_1((A_2A_3)A_4))$, $(((A_1A_2)A_3)A_4)$
- Let the dimension of matrices are as follows - $5 \times 2, 2 \times 1, 1 \times 10, 10 \times 100$. Compute the number of multiplications for above cases.
- Recursive definition:

$$MCM(i,j) = \min_k\{p_{i-1}p_kp_j + MCM(i,k) + MCM(k+1,j)\}, \text{ if } i \leq k < j$$
$$= 0, \text{ if } i = j$$

Matrix Chain Multiplication

- Given a sequence (chain) $\langle A_1, \ldots, A_n \rangle$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parentesize the product $A_1 \times \ldots \times A_n$ in way that minimizes the number of scalar multiplications.
- For $n = 4$ we have the following options

  $(A_1(A_2(A_3A_4)))$, $((A_1A_2)(A_3A_4))$, $((A_1(A_2A_3))A_4)$
  $(A_1((A_2A_3)A_4))$, $(((A_1A_2)A_3)A_4)$

- Let the dimension of matrices are as follows - $5 \times 2, 2 \times 1, 1 \times 10, 10 \times 100$. Compute the number of multiplications for above cases.
- Recursive definition:

$$MCM(i, j) = \min_k\{p_{i-1}p_kp_j + MCM(i, k) + MCM(k + 1, j)\}, \text{ if } i \leq k < j$$
$$= 0, \text{ if } i = j$$

- Subproblems can be identified by two indices i, j

Matrix Multiplication

- MatMult($A_{p\times q}$, $B_{q\times r}$, $C_{p\times r}$)

1. Initialize $C_{ij} = 0, \ \forall i, j$
2. for $i = 1, \ldots, p$
3. for $j = 1, \ldots, r$
4. for $k = 1, \ldots, q$
5. $C_{ij} = C_{ij} + A_{ik} \times B_{kj}$

- Time complexity is $O(n^3)$. However, we do not need to compute the product.
- We can determine the number of multiplications, pqr, in $O(1)$ time

MCM: Top-down

- Let us assume $Count[n, n]$ stores number of multiplications and $LUT[n, n]$ stores break-up point
- Initialize $Count[i, j] = \infty$ if $i \neq j$ and 0 if $i = j$
- $MCM(i, j, Count, LUT)$

1. if $i = j$ then $Count[i, i] = 0$; $LUT[i, i] = 0$; return 0;
2.
3. for $k = i, \ldots, j - 1$
4. $\quad q_k = MCM(i, k) + MCM(k + 1, j) + p_{i-1} p_k p_j$
5. $q_{min} = \min_k \{q_k\}$; $bp = \arg\min_k \{q_k\}$
6.
7. return q_{min}

MCM: Top-down

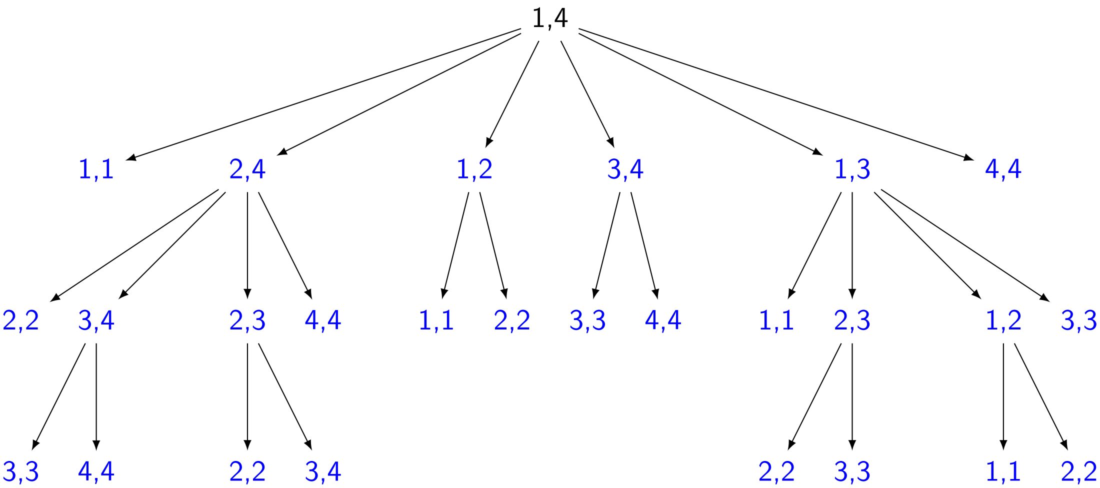
- Let us assume $Count[n, n]$ stores number of multiplications and $LUT[n, n]$ stores break-up point
- Initialize $Count[i, j] = \infty$ if $i \neq j$ and 0 if $i = j$
- $MCM(i, j, Count, LUT)$

1. if $i = j$ then $Count[i, i] = 0$; $LUT[i, i] = 0$; return 0;
2. if $Count[i, j] < \infty$ return $Count[i, j]$
3. for $k = i, \ldots, j - 1$
4. $\quad q_k = MCM(i, k) + MCM(k + 1, j) + p_{i-1} p_k p_j$
5. $q_{min} = \min_k \{q_k\}$; $bp = \arg\min_k \{q_k\}$
6. $Count[i, j] = q_{min}$; $LUT[i, j] = bp$
7. return q_{min}

MCM: Recursion Tree

1,4

MCM: Recursion Tree



MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0					
0	0				
0	0	0			
0	0	0	0		
0	0	0	0	0	
0	0	0	0	0	0

0					
0	0				
0	0	0			
0	0	0	0		
0	0	0	0	0	
0	0	0	0	0	0

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0	15750				
0	0				
0	0	0			
0	0	0	0		
0	0	0	0	0	
0	0	0	0	0	0

0	1				
0	0				
0	0	0			
0	0	0	0		
0	0	0	0	0	
0	0	0	0	0	0

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0	15750				
0	0	2625			
0	0	0			
0	0	0	0		
0	0	0	0	0	
0	0	0	0	0	0

0	1				
0	0	2			
0	0	0			
0	0	0	0		
0	0	0	0	0	
0	0	0	0	0	0

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0	15750				
0	0	2625			
0	0	0	750		
0	0	0	0		
0	0	0	0	0	
0	0	0	0	0	0

0	1				
0	0	2			
0	0	0	3		
0	0	0	0		
0	0	0	0	0	
0	0	0	0	0	0

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0	15750				
0	0	2625			
0	0	0	750		
0	0	0	0	1000	
0	0	0	0	0	
0	0	0	0	0	0

0	1				
0	0	2			
0	0	0	3		
0	0	0	0	4	
0	0	0	0	0	
0	0	0	0	0	0

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0	15750				
0	0	2625			
0	0	0	750		
0	0	0	0	1000	
0	0	0	0	0	5000
0	0	0	0	0	0

0	1				
0	0	2			
0	0	0	3		
0	0	0	0	4	
0	0	0	0	0	5
0	0	0	0	0	0

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0	15750	7875			
0	0	2625			
0	0	0	750		
0	0	0	0	1000	
0	0	0	0	0	5000
0	0	0	0	0	0

0	1	1			
0	0	2			
0	0	0	3		
0	0	0	0	4	
0	0	0	0	0	5
0	0	0	0	0	0

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0	15750	7875			
0	0	2625	4375		
0	0	0	750		
0	0	0	0	1000	
0	0	0	0	0	5000
0	0	0	0	0	0

0	1	1			
0	0	2	3		
0	0	0	3		
0	0	0	0	4	
0	0	0	0	0	5
0	0	0	0	0	0

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0	15750	7875			
0	0	2625	4375		
0	0	0	750	2500	
0	0	0	0	1000	
0	0	0	0	0	5000
0	0	0	0	0	0

0	1	1			
0	0	2	3		
0	0	0	3	3	
0	0	0	0	4	
0	0	0	0	0	5
0	0	0	0	0	0

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0	15750	7875			
0	0	2625	4375		
0	0	0	750	2500	
0	0	0	0	1000	3500
0	0	0	0	0	5000
0	0	0	0	0	0

0	1	1			
0	0	2	3		
0	0	0	3	3	
0	0	0	0	4	5
0	0	0	0	0	5
0	0	0	0	0	0

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0	15750	7875	9375	11875	15125
0	0	2625	4375	7125	10500
0	0	0	750	2500	5375
0	0	0	0	1000	3500
0	0	0	0	0	5000
0	0	0	0	0	0

0	1	1	3	3	3
0	0	2	3	3	3
0	0	0	3	3	3
0	0	0	0	4	5
0	0	0	0	0	5
0	0	0	0	0	0

MCM: Example

- Let us assume: $p = [30 \quad 35 \quad 15 \quad 5 \quad 10 \quad 20 \quad 25]$

0	15750	7875	9375	11875	15125
0	0	2625	4375	7125	10500
0	0	0	750	2500	5375
0	0	0	0	1000	3500
0	0	0	0	0	5000
0	0	0	0	0	0

0	1	1	3	3	3
0	0	2	3	3	3
0	0	0	3	3	3
0	0	0	0	4	5
0	0	0	0	0	5
0	0	0	0	0	0

$$(A_1(A_2A_3))((A_4A_5)A_6)$$

MCM: Bottom-up

- Let us assume $Count[n, n]$ stores number of multiplications and $LUT[n, n]$ stores break-up point
- MCM-Bottom-Up$(i, j, Count, LUT)$

1. for $i = 1, \ldots, n$ do $Count[i, i] = 0, LUT[i, i] = 0$
2. for $l = 2, \ldots, n$
3.   for $i = 1, \ldots, n - l + 1$
4. $j = i + l - 1;\ Count[i, j] = \infty$
5.   for $k = 1, \ldots, j - 1$
6. $q = Count(i, k) + Count(k + 1, j) + p_{i-1} p_k p_j$
7.   if $Count[i, j] > q$ then $count[i, j] = q; LUT[i, j] = k$
8. return $Count, LUT$

MCM: Parenthesization

- MCM-Print(i, j, LUT)

1. if $i = j$ print "A_i"
2. else
3. print "("
4. MCM-Print$(i, LUT[i, j], LUT)$
5. MCM-Print$(LUT[i, j] + 1, j, LUT)$
6. print ")"

Edit distance-1

- When a spell checker encounters a possible misspelling, it looks in its dictionary for other words that are close by. What is the appropriate notion of **closeness** in this case?

Edit distance-1

- When a spell checker encounters a possible misspelling, it looks in its dictionary for other words that are close by. What is the appropriate notion of **closeness** in this case?
- The **edit distance** between two strings is the minimum number of letter insertions, letter deletions, and letter substitutions required to transform one string into the other
- Example: `SNOWY` and `SUNNY`

```
S  --  N  O  W  Y        --  S  N  O  W  --  Y
S  U   N  N  -- Y        S   U  N  -- --  N  Y
     Cost: 3                    Cost: 5
```

Edit distance-1

- When a spell checker encounters a possible misspelling, it looks in its dictionary for other words that are close by. What is the appropriate notion of **closeness** in this case?
- The **edit distance** between two strings is the minimum number of letter insertions, letter deletions, and letter substitutions required to transform one string into the other
- Example: `SNOWY` and `SUNNY`

S	--	N	O	W	Y		--	S	N	O	W	--	Y
S	U	N	N	--	Y		S	U	N	--	--	N	Y
Cost: 3							Cost: 5						

- Problem definition: Given two strings $x[1..m]$ and $y[1..n]$, and opertions (a) insert in y, (b) delete from x, (c) substitute, find the minimum number of edits needed to transform the first string to second. Assume cost of operations are as α, β, γ.

Edit distance-2

- Recursive definition:

  Edit(i,j) $=$ min$\{\alpha+$Edit(i,j-1), $\beta+$Edit(i-1,j), diff(i,j)$+$Edit(i-1,j-1)$\}$,

  where diff(i,j)$=$0 if $x_i = y_j$ and γ otherwise

Edit distance-2

- Recursive definition:

  Edit(i,j) $=$ min$\{\alpha+$Edit(i,j-1), $\beta+$Edit(i-1,j), diff(i,j)$+$Edit(i-1,j-1)$\}$,

  where diff(i,j)$=$0 if $x_i = y_j$ and γ otherwise
- Edit(i,0)$=$??, Edit(0,j)$=$??

Edit distance-2

- Recursive definition:

  Edit(i,j) $=$ min$\{\alpha$+Edit(i,j-1), β+Edit(i-1,j), diff(i,j)+Edit(i-1,j-1)$\}$,

  where diff(i,j)=0 if $x_i = y_j$ and γ otherwise
- Edit(i,0)=??, Edit(0,j)=??
- Assuming x="INTENTION", y="EXECUTION" and $\alpha = \beta = 1, \gamma = 2$

Edit distance-3

- Assuming x="INTENTION", y="EXECUTION" and $\alpha = \beta = 1, \gamma = 2$

	#	E	X	E	C	U	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
I	1	2	3	4	5	6	7	6	7	8
N	2	3	4	5	6	7	8	7	8	7
T	3	4	5	6	7	8	7	8	9	8
E	4	3	4	5	6	7	8	9	10	9
N	5	4	5	6	7	8	9	10	11	10
T	6	5	6	7	8	9	8	9	10	11
I	7	6	7	8	9	10	9	8	9	10
O	8	7	8	9	10	11	10	9	8	9
N	9	8	9	10	11	12	11	10	9	8

Edit distance-4

- Assuming x="INTENTION", y="EXECUTION" and $\alpha = \beta = 1, \gamma = 2$

	#	E	X	E	C	U	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
I	1	2	3	4	5	6	7	6	7	8
N	2	3	4	5	6	7	8	7	8	7
T	3	4	5	6	7	8	7	8	9	8
E	4	3	4	5	6	7	8	9	10	9
N	5	4	5	6	7	8	9	10	11	10
T	6	5	6	7	8	9	8	9	10	11
I	7	6	7	8	9	10	9	8	9	10
O	8	7	8	9	10	11	10	9	8	9
N	9	8	9	10	11	12	11	10	9	8

Longest Increasing Subsequence

- For any sequence S, a **subsequence** of S is another sequence obtained from S by deleting zero or more elements, without changing the order of the remaining elements; the elements of the subsequence need not be contiguous in S
- Example: $S = 3, 1, 4, 2, 5, 9, 7, 13, 11, 19, 15, 18$, Increasing Subsequence – $S' = 3, 4, 5, 9, 13, 19$, $S'' = 1, 2, 5, 7, 11, 15, 18$
- Given a S, find LIS

Longest Increasing Subsequence

- For any sequence S, a **subsequence** of S is another sequence obtained from S by deleting zero or more elements, without changing the order of the remaining elements; the elements of the subsequence need not be contiguous in S
- Example: $S = 3, 1, 4, 2, 5, 9, 7, 13, 11, 19, 15, 18$, Increasing Subsequence – $S' = 3, 4, 5, 9, 13, 19$, $S'' = 1, 2, 5, 7, 11, 15, 18$
- Given a S, find LIS
- Steps: $LIS(n)$

1. for j=1,2,...,n
2. $LIS(j) = 1 + \max\{LIS(i) : S_i < S_j\}, i < j$
3. return $\max_j LIS(j)$

Knapsack

- During a robbery, a burglar finds much more loot than he had expected and has to decide what to take. His bag will hold a total weight of at most W kgs. There are n items to pick from, of weight $w_1, \ldots, w_n$ and INR value $v_1, \ldots, v_n$. What's the most valuable combination of items he can fit into his bag? Develop state-space exploration based approach.

Knapsack

- During a robbery, a burglar finds much more loot than he had expected and has to decide what to take. His bag will hold a total weight of at most W kgs. There are n items to pick from, of weight $w_1, \ldots, w_n$ and INR value $v_1, \ldots, v_n$. What's the most valuable combination of items he can fit into his bag? Develop state-space exploration based approach.
- What will happen if repetition is allowed?

Exercise

- **Longest Common Subsequence**:
  - A subsequence of a string is obtained by taking a string and possibly deleting zero or more elements.
  - If $x_1, \ldots, x_n$ is string and $1 \leq i_1 \leq \ldots \leq i_k \leq n$ is a strictly increasing sequence of indices, then $x_{i_1}, \ldots, x_{i_k}$ is a subsequence of x
    - For example, `art` is a subsequence of `algorithm`.
  - In the longest common subsequence problem, given strings x and y we want to find the longest string that is a subsequence of both
  - For example, `art` is the longest common subsequence of `algorithm` and `parachute`.

Exercise

- **Longest Common Subsequence**:
  - A subsequence of a string is obtained by taking a string and possibly deleting zero or more elements.
  - If $x_1, \ldots, x_n$ is string and $1 \leq i_1 \leq \ldots \leq i_k \leq n$ is a strictly increasing sequence of indices, then $x_{i_1}, \ldots, x_{i_k}$ is a subsequence of x
    - For example, `art` is a subsequence of `algorithm`.
  - In the longest common subsequence problem, given strings x and y we want to find the longest string that is a subsequence of both
  - For example, `art` is the longest common subsequence of `algorithm` and `parachute`.
- Recursive definition: length of x and y are i, j respectively

$$
\begin{aligned}
LCS(i,j) &= 0, \qquad\qquad \text{if } i = 0 \text{ or } j = 0 \\
&= \max\{LCS(i-1,j),\ LCS(i,j-1),\ LCS(i-1,j-1) + eq(x_i, y_j)\} \\
&\qquad\qquad eq(x_i, y_j) = x_i == y_j\ ?\ 1\ :\ 0;
\end{aligned}
$$

Thank you!