

Solving problems by Dynamic Programming

Dynamic programming (DP) is a technique for efficiently computing recurrences by storing partial results and re-using them when needed.

We trade space for time, avoiding to repeat the computation of a subproblem.

Solving problems by Dynamic Programming

Dynamic programming (DP) is a technique for efficiently computing recurrences by storing partial results and re-using them when needed.

We trade space for time, avoiding to repeat the computation of a subproblem.

Guideline to implement DP:

1. Characterize the recursive structure of an optimal solution
2. Define recursively the value of an optimal solution
3. Compute, bottom-up, the cost of a solution
4. Construct an optimal solution

Dynamic Programming Examples

1. **Minimum cost from Sydney to Perth**
2. **Economic Feasibility Study**
3. **0/1 Knapsack problem**
4. **Traveling Salesman Problem**
5. **Sequence Alignment problem**

Minimum Cost from Sydney to Perth

Based on M. A. Rosenman: Tutorial - Dynamic Programming Formulation

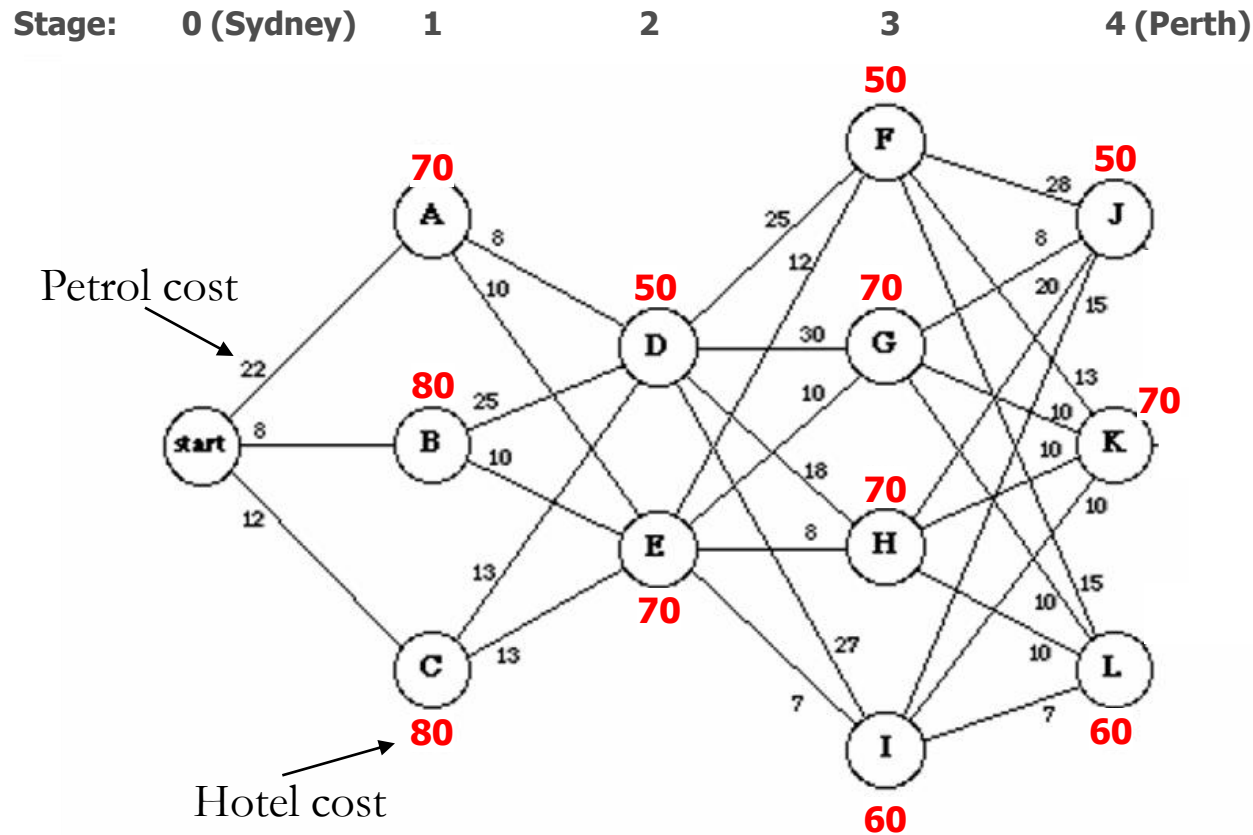
<http://people.arch.usyd.edu.au/~mike/DynamicProg/DPTutorial.95.html>

- **Problem definition**

- Travelling from home in Sydney to a hotel in Perth.
- Three stopovers on the way
 - a number of choices of towns for each stop,
 - a number of hotels to choose from in each city.
- Each trip has a different distance resulting in a different cost (petrol).
Hotels have different costs.
- The goal is to select a route to and a hotel in Perth so that **the overall cost of the trip is minimized.**

Minimum Cost from Sydney to Perth

- Diagrammatic representation of the problem



Minimum Cost from Sydney to Perth

- **Recursive definition of solution** in terms of sub-problem solutions

Optimal function:

$$V(i, n) = \min_{j, k} [hotel_cost(k) + petrol_cost(j, i) + V(j, n-1)]$$

with base case $V(i, 0) = 0$

where

n is the stage (stopover) number,

i is a city on stopover n ,

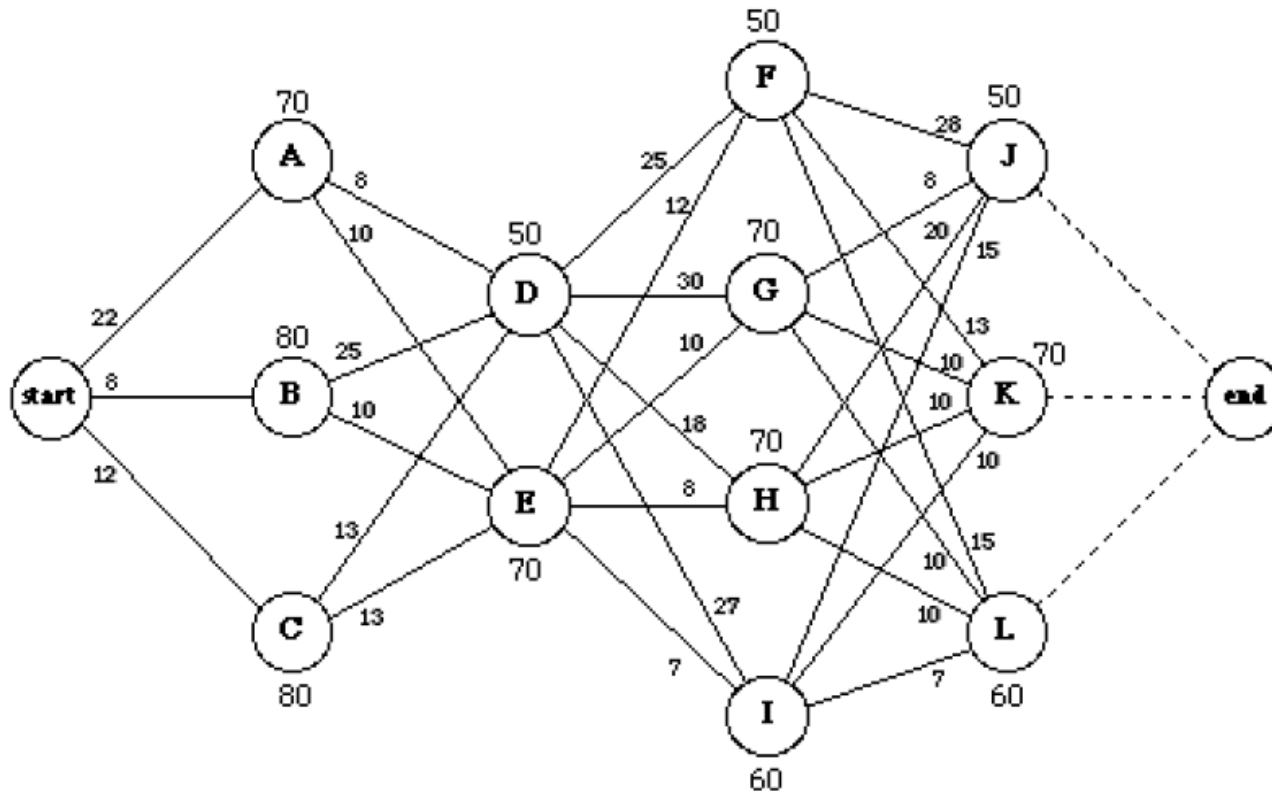
j is a city from which we can arrive at i ,

k is a hotel in city i

Minimum Cost from Sydney to Perth

Stage 1

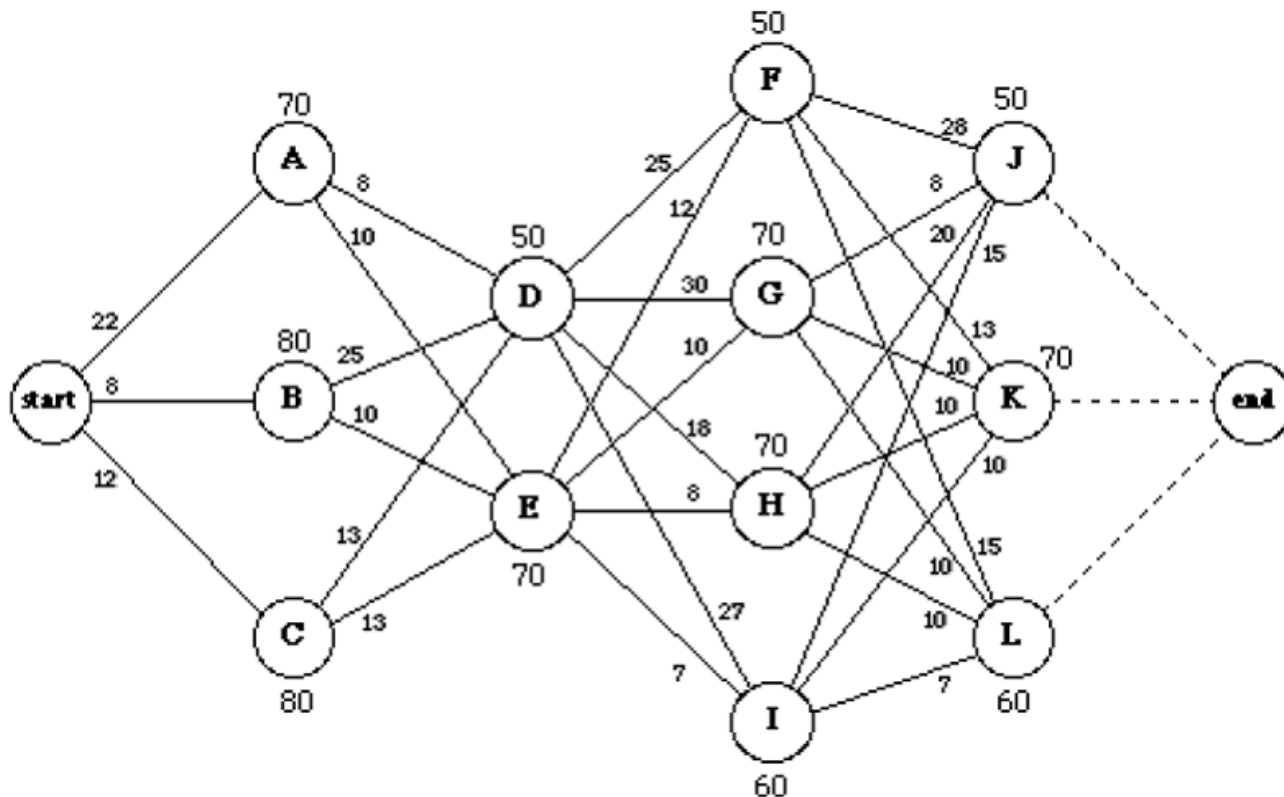
	start	cost	from
A	$22 + 70$	92	S
B	$8 + 80$	88	S
C	$12 + 80$	92	S



Minimum Cost from Sydney to Perth

Stage 2

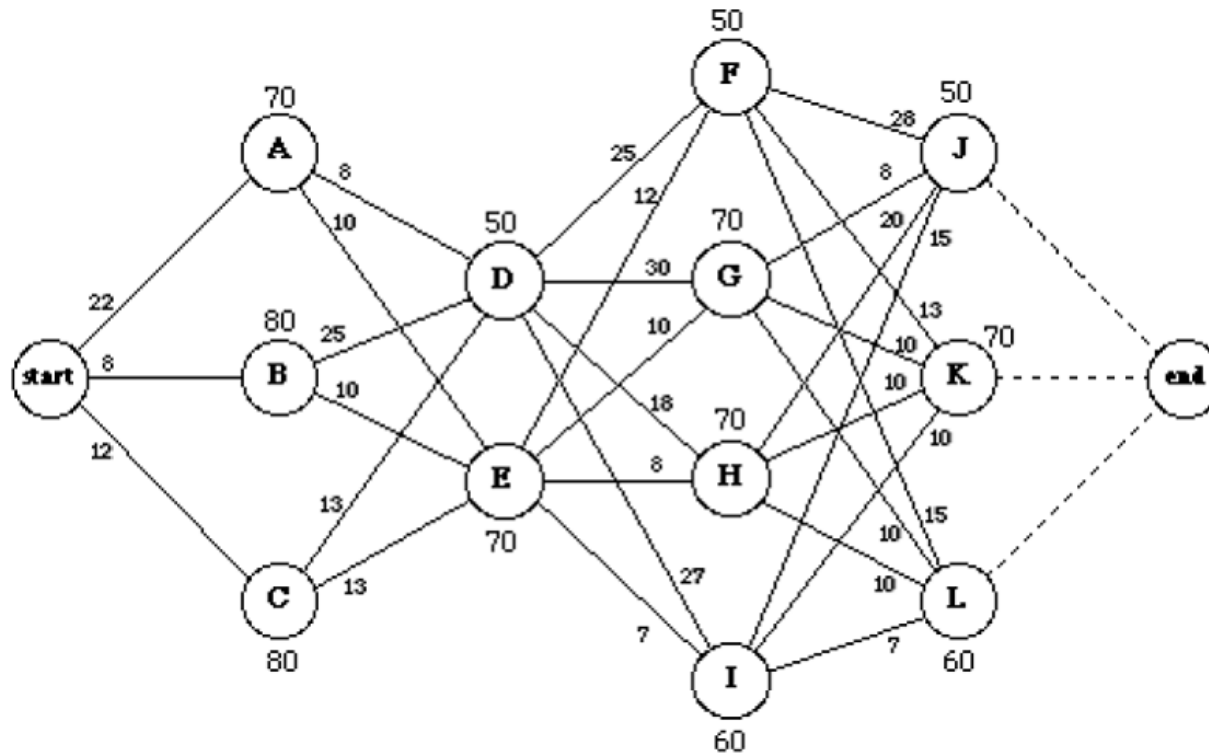
	A	B	C	cost	from
D	$92+8+50=150$	$88+25+50=163$	$92+13+50=155$	150	A
E	$92+10+70=172$	$88+10+70=168$	$92+13+70=175$	168	B



Minimum Cost from Sydney to Perth

Stage 3

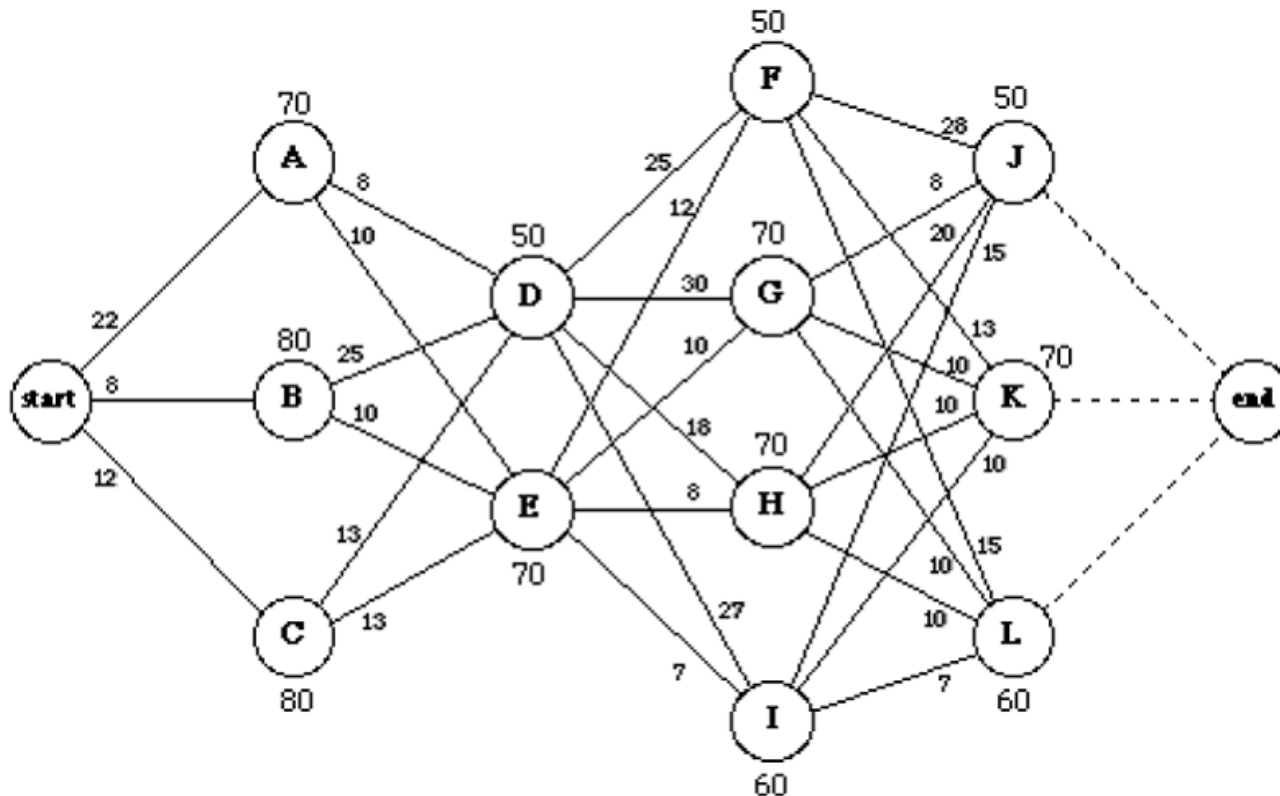
	D	E	cost	from
F	$150+25+50=225$	$168+12+50=230$	225	D
G	$150+30+70=250$	$168+10+70=248$	248	E
H	$150+18+70=238$	$168+8=70=246$	238	D
I	$150+27+60=237$	$168+7+60=235$	235	E



Minimum Cost from Sydney to Perth

Stage 4

	F	G	H	I	cost	from
J	$225+28+50=303$	$248+8+50=306$	$238+20+50=308$	$235+15+50=300$	300	I
K	$225+13+70=308$	$248+10+70=328$	$238+10+70=318$	$235+10+70=315$	308	F
L	$225+15+60=300$	$248+10+60=318$	$238+10+60=308$	$235+7+60=302$	300	F



Minimum Cost from Sydney to Perth

Stage 4

	F	G	H	I	cost	from
J	$225+28+50=303$	$248+8+50=306$	$238+20+50=308$	$235+15+50=300$	300	I
K	$225+13+70=308$	$248+10+70=328$	$238+10+70=318$	$235+10+70=315$	308	F
L	$225+15+60=300$	$248+10+60=318$	$238+10+60=308$	$235+7+60=302$	300	F

- Generating solution routes by tracking backwards through the tables

Solution₁ = {I, J}

Minimum Cost from Sydney to Perth

Stage 3

	D	E	cost	from
F	$150+25+50=225$	$168+12+50=230$	225	D
G	$150+30+70=250$	$168+10+70=248$	248	E
H	$150+18+70=238$	$168+8+70=246$	238	D
I	$150+27+60=237$	$168+7+60=235$	235	E

- Generating solution routes by tracking backwards through the tables

Solution_1 = {E, I, J}

Minimum Cost from Sydney to Perth

Stage 2

	A	B	C	cost	from
D	$92+8+50=150$	$88+25+50=163$	$92+13+50=155$	150	A
E	$92+10+70=172$	$88+10+70=168$	$92+13+70=175$	168	B

- Generating solution routes by tracking backwards through the tables
Solution₁ = {B, E, I, J}
- **Question:** What is the second optimal solution?

Economic Feasibility Study

Based on M. A. Rosenman: Tutorial - Dynamic Programming Formulation

<http://people.arch.usyd.edu.au/~mike/DynamicProg/DPTutorial.95.html>

- **Problem definition**

- We are asked for an advice how to best utilize a large urban area in an expanding town.
- Envisaged is a mixed project of housing, retail, office and hotel areas.
- Rental income is a function of the floor areas allocated to each activity.
- The total floor area is limited to 7 units.
- **The goal is to find the mix of development which will maximize the return.**

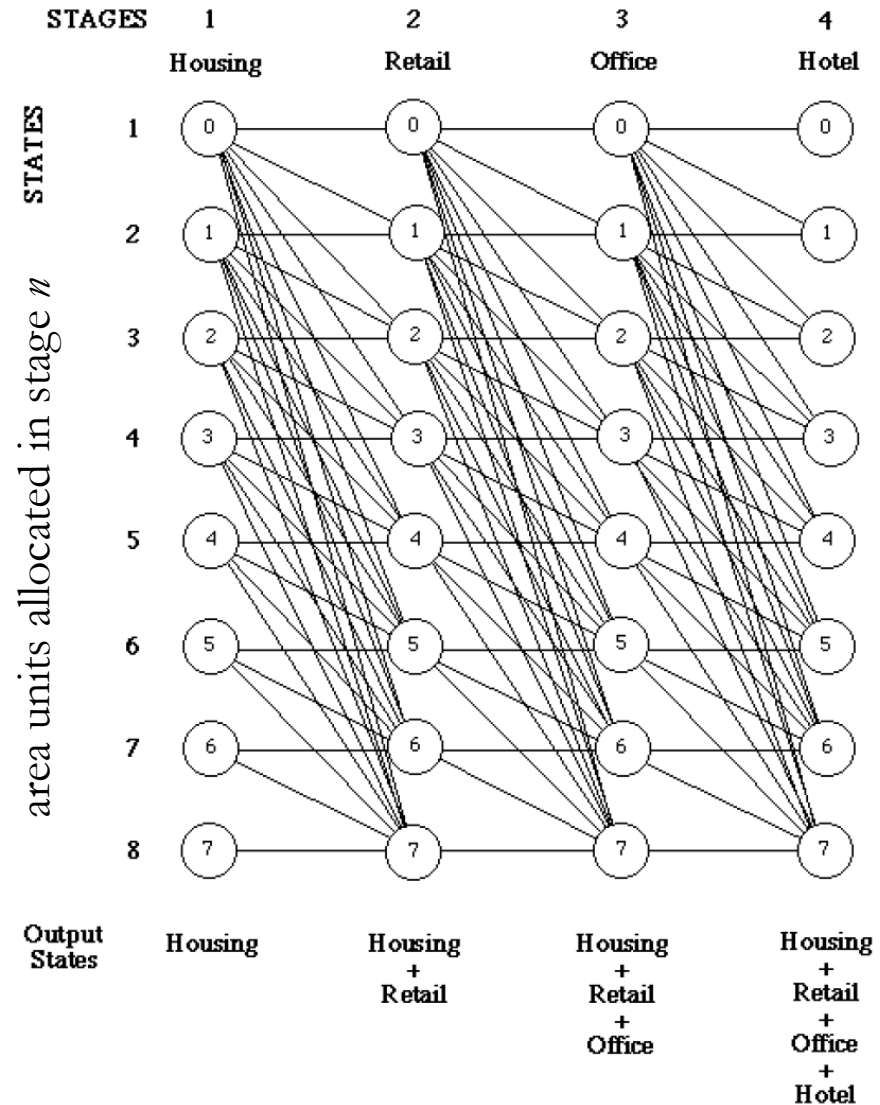
Economic Feasibility Study

Expected annual profit for areas of 0-7 units allocated to each development type in \$1000

AREA	Housing	Retail	Office	Hotel
0	0	0	0	0
1	2	6	1	8
2	4	9	1	12
3	6	9	2	20
4	8	10	1	16
5	10	11	15	12
6	12	12	12	10
7	14	13	20	4

Economic Feasibility Study

Diagrammatic representation:



Economic Feasibility Study

- **Recursive definition of solution** in terms of sub-problem solutions

Optimal function:

$$V(n, A) = \max_{a=0..A} [R(n, a) + V(n-1, A-a)]$$

with base case $V(n, 0) = 0$

where

n is the stage number corresponding to the development type

1...Housing

2...Retail

3...Office

4...Hotel

A is the total amount of area units allocated in stage n ,

a is the amount of area units allocated to development type n ,

R is the return generated by a units of development type n .

Economic Feasibility Study

STAGE 1 - Housing

$H^0 \backslash H$	0	R_1^*	D_1^*
0	0	0	0
1	2	2	0
2	4	4	0
3	6	6	0
4	8	8	0
5	10	10	0
6	12	12	0
7	14	14	0

STAGE 2 - Retail

$RH \backslash H$	0	1	2	3	4	5	6	7	R_2^*	D_2^*
0	0+0	-							0	0
1	0+6	2+0	-						6	0
2	0+9	2+6	4+0	-					9	0
3	0+9	2+9	4+6	6+0	-				11	1
4	0+10	2+9	4+9	6+6	8+0	-			13	2
5	0+11	2+10	4+9	6+9	8+6	10+0	-		15	3
6	0+12	2+11	4+10	6+9	8+9	10+6	12+0	-	17	4
7	0+13	2+12	4+11	6+10	8+9	10+9	12+6	14+0	19	5

R_i is the return generated by the total number of area units allocated in i -th stage.
 D_i is the number of area units allocated in $(i-1)$ -th stage.

Economic Feasibility Study

STAGE 3 - Office

RH ORH	0	1	2	3	4	5	6	7	R₃[*]	D₃[*]
0	0+0	-							0	0
1	0+1	6+0	-						6	1
2	0+1	6+1	9+0	-					9	2
3	0+2	6+1	9+1	11+0	-				11	3
4	0+1	6+2	9+1	11+1	13+0	-			13	4
5	0+15	6+1	9+2	11+1	13+1	15+0	-		15	0,5
6	0+12	6+15	9+1	11+2	13+1	15+1	17+0	-	21	1
7	0+20	6+12	9+15	11+1	13+2	15+1	17+1	19+0	24	2

Economic Feasibility Study

STAGE 4 - Hotel

$\begin{matrix} \text{ORH} \\ \text{HtORH} \end{matrix}$	0	1	2	3	4	5	6	7	R_4^*	D_4^*
0	0+0	-							0	0
1	0+8	6+0	-						8	0
2	0+12	6+8	9+0	-					14	1
3	0+20	6+12	9+8	11+0	-				20	0
4	0+16	6+20	9+12	11+8	13+0	-			26	1
5	0+12	6+16	9+20	11+12	13+8	15+0	-		29	2
6	0+10	6+12	9+16	11+20	13+12	15+8	17+0	-	31	3
7	0+4	6+10	9+12	11+16	13+20	15+12	17+8	24+0	33	4

- **Question:** Can you generate the optimal solution by tracking backwards through the tables?

Economic Feasibility Study

STAGE 4 - Hotel

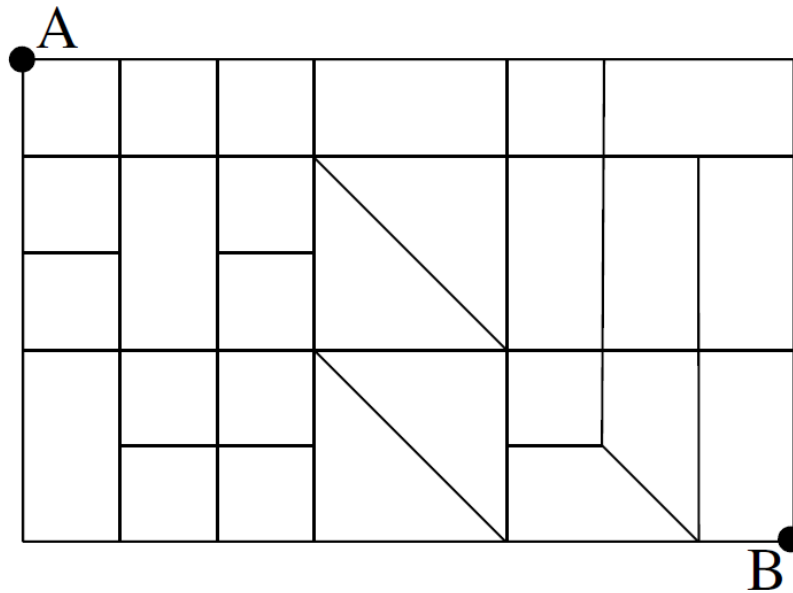
ORH H ORH	0	1	2	3	4	5	6	7	R ₄ *	D ₄ *
0	0+0	-							0	0
1	0+8	6+0	-						8	0
2	0+12	6+8	9+0	-					14	1
3	0+20	6+12	9+8	11+0	-				20	0
4	0+16	6+20	9+12	11+8	13+0	-			26	1
5	0+12	6+16	9+20	11+12	13+8	15+0	-		29	2
6	0+10	6+12	9+16	11+20	13+12	15+8	17+0	-	31	3
7	0+4	6+10	9+12	11+16	13+20	15+12	17+8	24+0	33	4

- **Solution:** Housing=2, Retail=2, Office=0, Hotel=3
Maximal return = 33

Counting Paths

How many ways are there to walk from A to B on this graph, if you are not allowed to “backtrack”?

That is, all steps should be toward either the East, the South or the Southeast.



1/0 Knapsack problem

Based on Dr. Shieu-Hong Lin lecture notes:

<http://csci.biola.edu/csci480spring03/knapsack.pdf>

- **Problem definition**

- Input: a set of $S = \{s_1, \dots, s_n\}$ of n items where each s_i has its value v_i and weight w_i , and a knapsack capacity W .
- **The goal is to choose a subset O of S such that the total weight of the items chosen does not exceed W and the sum of items v_i in O is maximal with respect to any other subset that meets the constraint.**
- Note that each item s_i either is or is not chosen to O .

Question: What would be a successful strategy for finding the optimal solution if we were allowed to add a fraction x_i of each item to the knapsack?

1/0 Knapsack problem

- **Decompose the problem into smaller problems.**

Let us assume the sequence of items $S = \{s_1, s_2, s_3, \dots, s_n\}$.

Suppose the optimal solution for S and W is a subset $O = \{s_2, s_4, s_k\}$, in which s_k is the highest numbered item in the sequence of items:

$$S = \{s_1, s_2, s_3, s_4, \dots, s_{k-1}, s_k, \dots, s_n\}$$

Then, $O - \{s_k\}$ is an optimal solution for the sub-problem $S_{k-1} = \{s_1, \dots, s_{k-1}\}$ and capacity $W - w_k$.

The value of the complete problem S would simply be the value calculated for this sub-problem S_{k-1} plus the value v_k .

1/0 Knapsack problem

- **Recursive definition of solution** in terms of sub-problem solutions.

We construct a **matrix** $V[0..n, 0..W]$, where W is the maximal allowed weight.

For $0 \leq i \leq n$, and $0 \leq w \leq W$, the entry $V[i, w]$ will store the maximum (combined) value of any subset of items $\{1, 2, \dots, i\}$ of (combined) weight at most w .

Each value $V(i, w)$ represents the optimal solution to this sub-problem: What would the value be if our knapsack weight was just w and we were only choosing among the first k items?

If we can compute all the entries of this array, then the array entry $V(n, W)$ will contain the maximal value of items selected from the whole set S of combined weight at most W , that is, the solution to our problem.

1/0 Knapsack problem

- **Recursive definition of solution** in terms of sub-problem solutions

Optimal function:

$$V(i, w) = \max \left[\overset{\text{leave item } i}{V(i-1, w)}, \overset{\text{take item } i}{v_i + V(i-1, w - w_i)} \right]$$

$$\text{for } 1 \leq i \leq n, 0 \leq w \leq W$$

with base cases

$$V(0, w) = 0$$

$$V(i, w) = 0 \text{ for } w < 0$$

1/0 Knapsack problem

- **Bottom-up computation:**

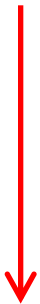
$$V(0, w) = 0 \quad \text{for all } 0 \leq w \leq W, \quad V(i, 0) = 0 \quad \text{for } 1 \leq i \leq n$$

Compute the table using

$$V(i, w) = \max \left[V(i-1, w), v_i + V(i-1, w - w_i) \right], \quad \text{for } 1 \leq i \leq n, \quad 0 \leq w \leq W$$

row by row.

$V[i, w]$	$w=0$	1	2	3	W
$i=0$	0	0	0	0	0
1	→						
2	→						
⋮	→						
n	→						



1/0 Knapsack problem

- Example: Let $W=10$ and

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

$V[i, w]$	0	1	2	3	4	5	6	7	8	9	10
$i = 0$	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0	0	0	40	40	40	40	40	50	50
3	0	0	0	0	40	40	40	40	40	50	70
4	0	0	0	50	50	50	50	90	90	90	90

- The final output is $V(4, 10)=90$.

Question: How do we find the optimal subset of items?

1/0 Knapsack problem

- Recovering the items that produce the optimal value:

- Start at $V[n, W]$ and track backwards through the table.

Recall:
$$V(i, w) = \max_{1 \leq i \leq n, 0 \leq w \leq W} [V(i-1, w), v_i + V(i-1, w-w_i)]$$

- If $V(i, w) = V(i-1, w)$ then item s_k was not added to the knapsack.

Continue the trace at $V[i-1, w]$.

- If $V(i, w) > V(i-1, w)$ then item s_k was added to the knapsack.

Continue the trace one row higher at $V(i-1, w-w_i)$.

$V[i, w]$	0	1	2	3	4	5	6	7	8	9	10
$i = 0$	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0	0	0	40	40	40	40	40	50	50
3	0	0	0	0	40	40	40	40	40	50	70
4	0	0	0	50	50	50	50	90	90	90	90

- The optimal subset is $O = \{2, 4\}$ in this case.

Traveling Salesman Problem

- **Problem definition**

- Given n cities and the distances d_{ij} between any two of them, we wish to find the shortest tour going through all cities and back to the starting city.

Usually the TSP is given as a $G = (V, D)$ where $V = \{1, 2, \dots, n\}$ is the set of cities, and C is the adjacency distance matrix, with $\forall i, j \in V, i \neq j, c_{i,j} > 0$, the problem is to find the tour with minimal distance weight, that starting in 1 goes through all n cities and returns to 1.

- **Known to be an NP-hard problem.**

It can be solved in $O(n!)$ steps.

Traveling Salesman Problem

- **Stages** – stage t represents t cities visited .
- **Decisions** – where to go next.
- **States** – we need to know where we have already gone, so states are represented by a pair (i, S) , where S is the set of t cities already visited and i is the last city visited.
- $f(i, S)$ – length of the shortest path from city i to city 1 through all the cities in S (in any order).

Ex.:

$f(4, \{5, 2, 3\})$ – length of the shortest path from city 4 to city 1 through cities 5, 2, and 3.

- Recursive transition from smaller problems to bigger ones:

$$f(i, S) = \min_{j \in S} (\text{dist}(i, j) + f(j, S - \{j\}))$$

- TSP solution reduces to finding $f(1, V - \{1\})$

Traveling Salesman Problem: Example

Distance matrix:

$$C = \begin{bmatrix} 0 & 2 & 9 & 10 \\ 1 & 0 & 6 & 4 \\ 15 & 7 & 0 & 8 \\ 6 & 3 & 12 & 0 \end{bmatrix}$$

t=1: $f(3, \{2\}) = c_{32} + f(2, \{\}) = c_{32} + c_{21} = 7+1 = 8;$
 $f(4, \{2\}) = c_{42} + f(2, \{\}) = c_{42} + c_{21} = 3+1 = 4;$
 $f(2, \{3\}) = c_{23} + f(3, \{\}) = c_{23} + c_{31} = 6+15 = 21;$
 $f(4, \{3\}) = c_{43} + f(3, \{\}) = c_{43} + c_{31} = 12+15 = 27;$
 $f(2, \{4\}) = c_{24} + f(4, \{\}) = c_{24} + c_{41} = 4+6 = 10;$
 $f(3, \{4\}) = c_{34} + f(4, \{\}) = c_{34} + c_{41} = 8+6 = 14;$

$$\begin{aligned} succ(3, \{2\}) &= 2 \\ succ(4, \{2\}) &= 2 \\ succ(2, \{3\}) &= 3 \\ succ(4, \{3\}) &= 3 \\ succ(2, \{4\}) &= 4 \\ succ(3, \{4\}) &= 4 \end{aligned}$$

Traveling Salesman Problem: Example

Distance matrix:

$$C = \begin{bmatrix} 0 & 2 & 9 & 10 \\ 1 & 0 & 6 & 4 \\ 15 & 7 & 0 & 8 \\ 6 & 3 & 12 & 0 \end{bmatrix}$$

t=1:

$f(3, \{2\}) = c_{32} + f(2, \{\}) = c_{32} + c_{21} = 7+1 = 8;$	$succ(3, \{2\}) = 2$
$f(4, \{2\}) = c_{42} + f(2, \{\}) = c_{42} + c_{21} = 3+1 = 4;$	$succ(4, \{2\}) = 2$
$f(2, \{3\}) = c_{23} + f(3, \{\}) = c_{23} + c_{31} = 6+15 = 21;$	$succ(2, \{3\}) = 3$
$f(4, \{3\}) = c_{43} + f(3, \{\}) = c_{43} + c_{31} = 12+15 = 27;$	$succ(4, \{3\}) = 3$
$f(2, \{4\}) = c_{24} + f(4, \{\}) = c_{24} + c_{41} = 4+6 = 10;$	$succ(2, \{4\}) = 4$
$f(3, \{4\}) = c_{34} + f(4, \{\}) = c_{34} + c_{41} = 8+6 = 14;$	$succ(3, \{4\}) = 4$

t=2:

$f(4, \{2,3\}) = \min \{c_{42} + f(2, \{3\}), c_{43} + f(3, \{2\})\} = \min \{3+21, 12+8\} = 20;$	$succ(4, \{2,3\}) = 3$
$f(3, \{2,4\}) = \min \{c_{32} + f(2, \{4\}), c_{34} + f(4, \{2\})\} = \min \{7+10, 8+4\} = 12;$	$succ(3, \{2,4\}) = 4$
$f(2, \{3,4\}) = \min \{c_{23} + f(3, \{4\}), c_{24} + f(4, \{3\})\} = \min \{6+14, 4+27\} = 20;$	$succ(2, \{3,4\}) = 3$

Traveling Salesman Problem: Example

Distance matrix:

$$C = \begin{bmatrix} 0 & 2 & 9 & 10 \\ 1 & 0 & 6 & 4 \\ 15 & 7 & 0 & 8 \\ 6 & 3 & 12 & 0 \end{bmatrix}$$

t=1:

$$\begin{aligned} f(3, \{2\}) &= c_{32} + f(2, \{\}) = c_{32} + c_{21} = 7+1 = 8; & succ(3, \{2\}) &= 2 \\ f(4, \{2\}) &= c_{42} + f(2, \{\}) = c_{42} + c_{21} = 3+1 = 4; & succ(4, \{2\}) &= 2 \\ f(2, \{3\}) &= c_{23} + f(3, \{\}) = c_{23} + c_{31} = 6+15 = 21; & succ(2, \{3\}) &= 3 \\ f(4, \{3\}) &= c_{43} + f(3, \{\}) = c_{43} + c_{31} = 12+15 = 27; & succ(4, \{3\}) &= 3 \\ f(2, \{4\}) &= c_{24} + f(4, \{\}) = c_{24} + c_{41} = 4+6 = 10; & succ(2, \{4\}) &= 4 \\ f(3, \{4\}) &= c_{34} + f(4, \{\}) = c_{34} + c_{41} = 8+6 = 14; & succ(3, \{4\}) &= 4 \end{aligned}$$

t=2:

$$\begin{aligned} f(4, \{2,3\}) &= \min \{c_{42} + f(2, \{3\}), c_{43} + f(3, \{2\})\} = \min \{3+21, 12+8\} = 20; & succ(4, \{2,3\}) &= 3 \\ f(3, \{2,4\}) &= \min \{c_{32} + f(2, \{4\}), c_{34} + f(4, \{2\})\} = \min \{7+10, 8+4\} = 12; & succ(3, \{2,4\}) &= 4 \\ f(2, \{3,4\}) &= \min \{c_{23} + f(3, \{4\}), c_{24} + f(4, \{3\})\} = \min \{6+14, 4+27\} = 20; & succ(2, \{3,4\}) &= 3 \end{aligned}$$

t=3:

$$\begin{aligned} f(1, \{2,3,4\}) &= \min \{c_{12} + f(2, \{3,4\}), c_{13} + f(3, \{2,4\}), c_{14} + f(4, \{2,3\})\} = \min \{2+20, 9+12, 10+20\} = 21 \\ succ(1, \{2,3,4\}) &= 3 \end{aligned}$$

Traveling Salesman Problem: Example

Distance matrix:

$$C = \begin{bmatrix} 0 & 2 & 9 & 10 \\ 1 & 0 & 6 & 4 \\ 15 & 7 & 0 & 8 \\ 6 & 3 & 12 & 0 \end{bmatrix}$$

t=1:

$$\begin{aligned} f(3, \{2\}) &= c_{32} + f(2, \{\}) = c_{32} + c_{21} = 7+1 = 8; & succ(3, \{2\}) &= 2 \\ f(4, \{2\}) &= c_{42} + f(2, \{\}) = c_{42} + c_{21} = 3+1 = 4; & succ(4, \{2\}) &= 2 \\ f(2, \{3\}) &= c_{23} + f(3, \{\}) = c_{23} + c_{31} = 6+15 = 21; & succ(2, \{3\}) &= 3 \\ f(4, \{3\}) &= c_{43} + f(3, \{\}) = c_{43} + c_{31} = 12+15 = 27; & succ(4, \{3\}) &= 3 \\ f(2, \{4\}) &= c_{24} + f(4, \{\}) = c_{24} + c_{41} = 4+6 = 10; & succ(2, \{4\}) &= 4 \\ f(3, \{4\}) &= c_{34} + f(4, \{\}) = c_{34} + c_{41} = 8+6 = 14; & succ(3, \{4\}) &= 4 \end{aligned}$$

t=2:

$$\begin{aligned} f(4, \{2,3\}) &= \min \{c_{42} + f(2, \{3\}), c_{43} + f(3, \{2\})\} = \min \{3+21, 12+8\} = 20; & succ(4, \{2,3\}) &= 3 \\ f(3, \{2,4\}) &= \min \{c_{32} + f(2, \{4\}), c_{34} + f(4, \{2\})\} = \min \{7+10, 8+4\} = 12; & succ(3, \{2,4\}) &= 4 \\ f(2, \{3,4\}) &= \min \{c_{23} + f(3, \{4\}), c_{24} + f(4, \{3\})\} = \min \{6+14, 4+27\} = 20; & succ(2, \{3,4\}) &= 3 \end{aligned}$$

t=3:

$$\begin{aligned} f(1, \{2,3,4\}) &= \min \{c_{12} + f(2, \{3,4\}), c_{13} + f(3, \{2,4\}), c_{14} + f(4, \{2,3\})\} = \min \{2+20, 9+12, 10+20\} = 21 \\ succ(1, \{2,3,4\}) &= 3 \end{aligned}$$

Optimal tour: **1 → 3 → 4 → 2 → 1**

Sequence Alignment Problem

Based on Advanced Dynamic Programming Tutorial by Eric C. Rouchka

http://www.avatar.se/molbioinfo2001/dynprog/adv_dynamic.html

- **Problem definition**

- Given two sequences X and Y of length m and n , respectively, composed of characters from alphabet A.
- The goal is to find the best alignment between the two sequences considering the following scoring scheme:
 - $S_{i,j}=2$ if the residue at position i of sequence X is the same as the residue at position j of sequence Y (match score),
 - $S_{i,j}=-1$ if the residue at position i of sequence X is not the same as the residue at position j of sequence Y (mismatch score),
 - Penalty $w=-2$ if either the residue at position i of sequence X or the residue at position j of sequence Y is a space '-' (gap penalty).

Example:

X = G A A T T C A G T T A

Y = G G A T C G A



X' : G A A T T C A G T T A

Y' : G G A _ T C _ G _ _ A

Sequence Alignment

Example

X = G A A T T C A G T T A

Y = G G A T C G A

$m=11$ and $n=7$

One of the optimal alignments for these sequences is

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

+	-	+	-	+	+	-	+	-	-	+
2	1	2	2	2	2	2	2	2	2	2

with score $S = 2-1+2+2-2+2-2+2-2-2+2=3$

Sequence Alignment

- **The idea:**
Find an **optimal alignment between each pair of prefixes** for each of the two strings.
- **Decompose the problem into smaller recursive problems.**

Let:

- C1 be the right-most character of sequence X,
- C2 be the right-most character of sequence Y,
- X' be X with C1 “chopped-off”,
- Y' be Y with C2 “chopped-off”.

Then there are **three recursive sub-problems:**

- $S1 = \text{align}(X', Y)$
- $S2 = \text{align}(X, Y')$
- $S3 = \text{align}(X', Y')$

The solution to the original problem is whichever of these is the biggest:

- **S1-2**
- **S2-2**
- **S3+2** if C1 equals C2, or **S3-1** if C1 and C2 mismatch.

Sequence Alignment

- **Needleman-Wunsch algorithm:**
 1. **Matrix initialization and fill step.**

Construct a score matrix M in which you build up partial solutions.
 2. **Traceback step.**

Determine the actual alignment using the score matrix.

Sequence Alignment

- **Matrix initialization step.**

Create a matrix with $m+1$ columns and $n+1$ rows.

The first row and first column can be initially filled with 0.

	G	A	A	T	T	C	A	G	T	T	A
0	0	0	0	0	0	0	0	0	0	0	0
G	0										
G	0										
A	0										
T	0										
C	0										
G	0										
A	0										

The idea is that the table will be build from top to bottom, and from left to right, and **each cell will contain** a number that is **the best alignment score of the two sequence prefixes up to that row and column.**

Sequence Alignment

- **Matrix fill step.**

In order to find $M_{i,j}$ for any i,j values $M_{i-1,j}$, $M_{i,j-1}$ and $M_{i-1,j-1}$ are used.

For each position, $M_{i,j}$ is defined to be the maximum score at position i,j :

$$M_{i,j} = \max \left[\begin{array}{ll} M_{i-1,j-1} + S_{i,j} & \text{(match/mismatch in the diagonal)} \\ M_{i,j-1} + w & \text{(gap in sequence Y)} \\ M_{i-1,j} + w & \text{(gap in sequence X)} \end{array} \right]$$

- **Interpretation**

- Moving horizontally corresponds to aligning a letter in string 1 with a gap in string 2
- Moving vertically corresponds to aligning a letter in string 2 with a gap in string 1
- Moving diagonally corresponds to aligning the two characters, one from each string.

Sequence Alignment

- We continue filling in the cells of the scoring matrix using the same reasoning. Note, there is also an arrow placed back into the cell that resulted in the maximum score.

In case that there are two different ways to get the maximum score, the pointers are placed back to all of the cells that produce the maximum, $M[3,2]$.

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	2	0	-1								
G	0	2	1	-1								
A	0	0	4									
T	0	-1	2									
C	0	-1	0									
G	0	2	0									
A	0	0	4									

Sequence Alignment

- **The complete score matrix.**

The maximum global alignment score for the two sequences is 3.

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	2	0	-1	-1	-1	-1	-1	2	0	-1	-1
G	0	2	1	-1	-2	-2	-2	-2	1	1	-1	-2
A	0	0	4	3	1	-1	-3	0	-1	0	0	1
T	0	-1	2	3	5	3	1	-1	-1	1	2	0
C	0	-1	0	1	3	4	5	3	1	-1	0	1
G	0	2	0	-1	1	2	3	4	5	3	1	-1
A	0	0	4	2	0	0	1	5	3	4	2	3

Sequence Alignment

- **Traceback step, constructing the actual alignment strings – X' and Y' .**

We use the pointers back to all possible predecessors. At each cell, we look to see where we move next according to the pointers.

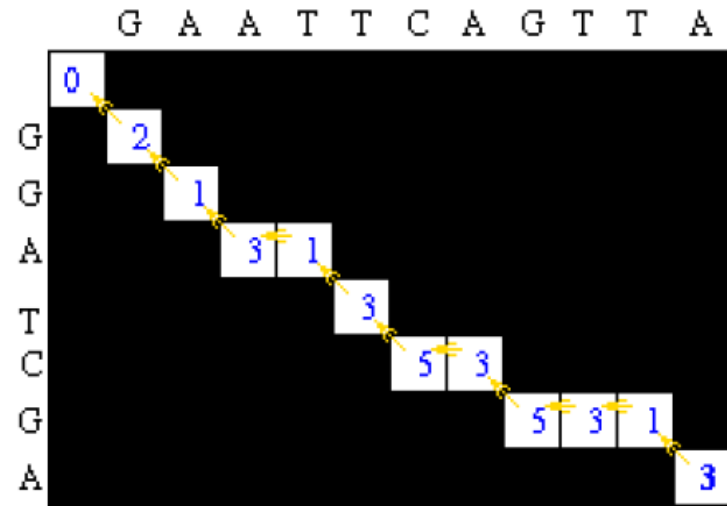
Begin at position $M[m, n]$ and just follow the pointers:

- going up corresponds to adding the character to the left from Y to Y' while adding a space to X' ,
- going left corresponds to adding the character above from X to X' while adding a space to Y' ,
- going diagonally up and to the left means adding a character from X and Y to X' and Y' , respectively.

If there are two possible neighbors, one of them is arbitrarily chosen.

Sequence Alignment

	G	A	A	T	T	C	A	G	T	T	A	
0	0	0	0	0	0	0	0	0	0	0	0	
G	0	2	0	-1	-1	-1	-1	-1	2	0	-1	-1
G	0	2	1	-1	-2	-2	-2	-2	1	1	-1	-2
A	0	0	4	3	1	-1	-3	0	-1	0	0	1
T	0	-1	2	3	5	3	1	-1	-1	1	2	0
C	0	-1	0	1	3	4	5	3	1	-1	0	1
G	0	2	0	-1	1	2	3	4	5	3	1	-1
A	0	0	4	2	0	0	1	5	3	4	2	3



- Final alignment

```

X': G A A T T C A G T T A
   |   |   | |   |   |
Y': G G A _ T C _ G _ _ A
  
```