**1.** Find longest increasing subsequence of the given sequence. Use the DP method, construct the DP table of the subsequence lengths and the table of predecessors.

a) 5 8 11 13 9 4 1 2 0 3 7 10 12 6

b) 6 7 5 15 10 9 11 18 19 8 12 1 3 4 13 14 0 17 2 16

**Solution** [subsequences: a) 1,2,3,7,10,12 b) 6, 7, 9, 11, 12, 13, 14, 17/16 ]

**2**. Modify the DP method of finding the longest increasing subsequence to find

a) longest decreasing subsequence

b) longest non-increasing subsequence

c) longest constant subsequence

d) longest alternating subsequence

In case c) also try to find method assymptotically faster than the DP approach.

In case d), alternating subsequence a[1], a[2], ..., a[n] satisfies

(a[k] − a[k−1])\* (a[k+1] − a[k]) < 0, for k = 2, 3, ..., *n*−1.

**Solution** a) The problem is identical to the original one, for example you can multiply all elements by −1.

b) The problem is identical to the original one, only the inequality which compares the elements is changed from the strict one to the one which is not strict.

c) Same as b), substitute the inequality by equality. Non-DP approch may consist of sorting the sequence

which can be done in O(n\*log(n)) time (sometimes even in Θ(n) time) and then finding the longest segment of constant values in one pass in time Θ(n). Alternatively, a hash map may be utilized to store and update the number of occurences of each element in the sequence and the problem my be then solved in one pass in expected time Θ(n).

d) During the search, register two longest subsequences found so far: The one which ends with a decreasing pair of elements a[x] > a[y] for some indices x < y and the one which ends with a decreasing pair of elements a[t] < a[w] for some indices t < w. In the current step k try to extend the first subsequence if a[k] > a[w] or try to extend the second subsequence if a[k] < a[y]. Nonetheless, there is a very simple and more effective approach which does not depend on DP and which solves the task in one pass and which you are encouraged to find yourself.

**3.** In how many ways can the be the matric product parenthesized? (Different parenthesizations result in different progress of the product calculation. Parenthesization (X) and ((X)) are identical in this problem.)

a) A × B × C × D

b) A × B × C × D × E

**Solution** [5 a 14]

**4.** Let A and B be real matrices, A ∈ **R***r*×*s* a B ∈ **R***s*×*t.*  Supose we need exactly *r*∙*s*∙*t* operations to compute the product A × B. Determine how many operations must be performed to compute the product (A × B) × C and how many to compute the product A × (B × C) when:

a) A ∈ **R**2×3 , B ∈ **R**3×5 , C ∈ **R**5×4

b) A ∈ **R**3×4 , B ∈ **R**4×5 , C ∈ **R**5×2

c) A ∈ **R***n*×4 , B ∈ **R**4×2*n* , C ∈ **R**2*n*×3

**Solution** [a) 70 a 84 b) 90 a 64 c) 14*n*2 a 36*n* ]

**5.** For which values of *n* is it more efficient to compute the product (A × B) × C then the product A × (B × C)?

a) A ∈ **R***n*×2 , B ∈ **R**2×3 , C ∈ **R**3×4

b) A ∈ **R**5×*n* , B ∈ **R***n*×4 , C ∈ **R**4×*n*

c) A ∈ **R***n*×*n* , B ∈ **R***n*×100 , C ∈ **R**100×*n*

**Solution** [a) 18*n* < 8*n+*24, n < 3 b) 40*n* < 9*n*2, 4 < *n* c) 200*n*2 < *n*3+100*n*2*, 100 < n* ]

**6.** The dimensions of matrices A, B, C, D, E, are (in this order) 2 × 5, 5 × 3, 3 × 6, 6 × 2, 2 × 4.

Apply the DP method to determine parenthesization of the product A × B × C × D × E which minimizes the number of multiplications in the process of calculating the final product. What is the minimum number of operations?

**Solution**

((A × B) × (C × D)) × E

Number of operations: 94.

DP tables:

Operations count:

0 30 66 78 94

0 0 90 66 106

0 0 0 36 60

0 0 0 0 48

0 0 0 0 0

reconstruction table

0 A B B D

0 0 B B D

0 0 0 C D

0 0 0 0 D

0 0 0 0 0

**7.** We do some HW/SW benchmarks and we want to multiply matrices A, B, C, D, E in the previous problem in such way that the number of multiplications is maximized.

In which way can you modify the Matrix chain multiplication algorithm to sove this problem?

**Solution** The structure of the problem is identical, the only difference is that when building the DP table we should choose the maximum instead of the minimum values.]

**8.**  Modify the idea of solution of the chain matrix multiplication probem to solve a more simple problem:

In how many different ways can be the product of n terms be parenthesized? Will you need a 2D or a 1D table? Verify the solution with a few small values of *n*, the result should be equal to , which is the (*n*−1)-th Catalan number defined as for positive integer n.

**Solution**  Use 1D table T with the recursive rule

T[n] = T[1]\*T[*n*−1] + T[2]\*T[*n*−2]+ ... + T[*n*−2]\*T[2] + T[*n*−1]\*T[1].

The recursive rule is often cited together with the Catalan numbers definition.

**9.** Optimal binary search tree

1. maximizes the depth of the tree
2. maximizes costs of the nodes
3. maximizes number of leaves
4. minimizes the time of search operation

e) minimizes length of the path from the root to any leaf

**Solution**  d).

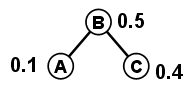
**10.** There are n keys and with each key is associated the probability that this key will be queried. The complexity of construction of the optimal BST using the given keys is

O(log(n))

1. Θ(n)
2. O(n·log(n))
3. Ω(n2)
4. Ω(2n)

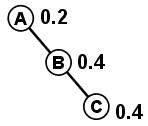
**Solution**  d).

**11a.** The probablility of a particular key to be queried is written at the particular node associated with the key in the picture. Suppose that only the keys which are present in the tree are queried in long time run. The average number of the nodes visited during one single query is then

1. 0.5
2. 1.0
3. 1.25
4. 1.5
5. 1.75

**Solution**  d).

**11b.** The probablility of a particular key to be queried is written at the particular node associated with the key in the picture. Suppose that only the keys which are present in the tree are queried in long time run. The average number of the nodes visited during one single query is then



1. 0.2
2. 1.0
3. 2.15
4. 2.2
5. 2.5

**Solution**  d).

**12.** There are two binary search trees containing the same keys. The probablility of a particular key to be queried is listed in the table bellow. Find out which of the trees is more search effective, that is, in which of the trees the long term average cost of operation FIND is smaller. The cost of the operation is equal to the number of nodes visited during that operation. (We suppose that the tree contents and shape do not change over time.)

A: 0.10

B: 0.20

C: 0.25

D: 0.05

E: 0.10

F: 0.25

G: 0.05

**Solution**  The cost of each node is multiplied by its depth (the depth of the root is 1 in this case)

and the results are added in each tree separately:

Left tree: 4\*0.1 + 3\*0.20 + 4\*0.25 + 2\*0.05 + 3\*0.10 + 1\*0.25 + 2\*0.05 = 2.75

Right tree: 2\*0.1 + 1\*0.20 + 4\*0.25 + 3\*0.05 + 4\*0.10 + 2\*0.25 + 3\*0.05 = 2.60

The right tree is more search effective.

**13.**  Determine the shape of the optimalBST, constructed for the given 7 keys and their corresponding relative query frequencies:

a) E 0.04 F 0.05 G 0.22 H 0.04 I 0.06 J 0.05 K 0.15

b) A 0.10 B 0.10 C 0.25 D 0.35 E 0.10 F 0.05 G 0.05

**Solution**  a)

**0 0.04 0.13 0.44 0.52 0.68 0.83 1.28**

**0 0 0.05 0.32 0.40 0.56 0.71 1.16**

**0 0 0 0.22 0.30 0.46 0.61 1.06**

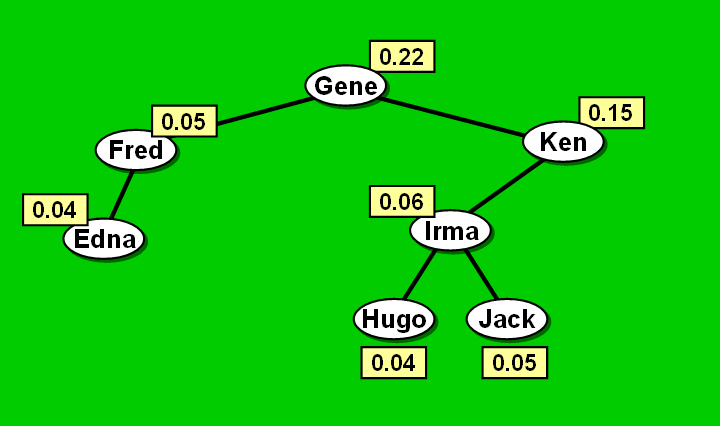
**0 0 0 0 0.04 0.14 0.24 0.54**

**0 0 0 0 0 0.06 0.16 0.42**

**0 0 0 0 0 0 0.05 0.25**

**0 0 0 0 0 0 0 0.15**

**0 0 0 0 0 0 0 0**



**5-E 6-F 7-G 8-H 9-I 10-J 11-K**

**0 5 6 7 7 7 7 7**

**0 0 6 7 7 7 7 7**

**0 0 0 7 7 7 7 7**

**0 0 0 0 8 9 9 11**

**0 0 0 0 0 9 9 11**

**0 0 0 0 0 0 10 11**

**0 0 0 0 0 0 0 11**

**0 0 0 0 0 0 0 0**

**Solution**  b)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D | E | F | G |  |  |  | A | B | C | D | E | F | G |
| A | 0 | 0.1 | 0.3 | 0.75 | 1.45 | 1.75 | 1.9 | 2.1 |  | A | - | A | AB | C | C | CD | D | D |
| B | 0 | 0 | 0.1 | 0.45 | 1.15 | 2.03 | 1.5 | 1.7 |  | B | - | - | B | C | CD | D | D | D |
| C | 0 | 0 |  | 0.25 | 0.85 | 1.05 | 1.2 | 1.4 |  | C | - | - |  | C | D | D | D | D |
| D | 0 | 0 |  |  | 0.35 | 0.55 | 0.7 | 0.9 |  | D | - | - |  |  | D | D | D | D |
| E | 0 | 0 |  |  |  | 0.1 | 0.2 | 0.35 |  | E | - | - |  |  |  | E | E | EFG |
| F | 0 | 0 |  |  |  |  | 0.05 | 0.15 |  | F | - | - |  |  |  |  | F | FG |
| G | 0 | 0 | 0 |  |  |  |  | 0.05 |  | G | - | - |  |  |  |  |  | G |

The values in yellow cells may be obtained in more than one way, therefore the shape of the tree may vary.

Example: Table value of the subtree BCD in the cell t[B,D] is computed:

0.1+0.25+0.35 + min(0+0.85, 0.1+0.35, 0.45+0) = 0.7 + min( 0.85, 0.45, 0.45) = 1.15

One of the possible tree shapes is shown in the picture:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | 10 | 11 | 23 | 22 | 23 |
| 27 | 25 | 29 | 12 | 29 | 24 |
| 18 | 21 | 11 | 27 | 14 | 24 |
| 30 | 17 | 26 | 29 | 23 | 22 |
| 12 | 25 | 23 | 13 | 28 | 16 |
| 20 | 24 | 10 | 14 | 30 | 15 |

**14.** We start anywhere in the first column of the given matrix and the we proceed step by step each time by one column in any of the N, NE and E direction. The journey stops in the last column. The cost of the journey is the sum of the values in all visited cells during the journey. What is the minimum possible cost?

**Solution**  The path goes through the cells with values 12 - 17 - 11 - 12 - 14 - 22, total minimum cost is 88.

**15.** We travel through the matrix according to the same rules. The cost of one step this time is the absolute values of the difference of the current and the previous visited cell. The problem remains the same: Find the cheapest journey.

**Solution** The path goes through the cells with values 27 - 25 - 29 - 23 - 22 - 23, total minimum cost is 14.

**16.**  Both two given strings are of length n. Longest common subsequence of the strings can be found in time

1. Θ(log(n))
2. Θ(n)
3. Θ(n·log(n))
4. Θ(n2)
5. Θ(n3)

**Solution**  d).

**17.** Find the longest common subsequence of the pairs of strings:

a)

A: 11101001000

B: 00010010111 (B = A backwards)

b)

A: 1100110011001100

B: 1010101010101010

c)

A: 110100100010000100001000001

B: 001011011101111011110111110 (B = complement of A)

**Solution**

a) 0001000

b) All 70 solutions in lexicographical order:

100100100100

100100100110

100100101100

100100110010

100100110100

100100110110

100101001100

100101100100

100101100110

100101101100

100110010010

100110010100

100110010110

100110011010

100110100100

100110100110

100110101100

100110110010

100110110100

100110110110

101001001100

101001100100

101001100110

101001101100

101011001100

101100100100

101100100110

101100101100

101100110010

101100110100

101100110110

101101001100

101101100100

101101100110

101101101100

110010010010

110010010100

110010010110

110010011010

110010100100

110010100110

110010101100

110010110010

110010110100

110010110110

110011001010

110011010010

110011010100

110011010110

110011011010

110100100100

110100100110

110100101100

110100110010

110100110100

110100110110

110101001100

110101100100

110101100110

110101101100

110110010010

110110010100

110110010110

110110011010

110110100100

110110100110

110110101100

110110110010

110110110100

110110110110

c) 110100101110 is one of the possibilities, others (if they exist) remain unknown to this day.