**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

**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.

**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

**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

**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*

**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?

**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?

**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 $\frac{1}{n}\left(\begin{matrix}2n-2\\n-1\end{matrix}\right)$, which is the (*n*−1)-th Catalan number defined as $\frac{1}{n+1}\left(\begin{matrix}2n\\n\end{matrix}\right)$ for positive integer n.

**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

**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)

**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

**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

**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

**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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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?

**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.

**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)

**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)