### Total no of paths in matrix if right and down move allowed

<https://leetcode.com/problems/unique-paths/submissions/>

<https://leetcode.com/problems/unique-paths/discuss/22958/Math-solution-O(1)-space>

<https://leetcode.com/problems/unique-paths-ii/>(with some obstacles)

Dp[i][j] = dp[i-1][j] + dp[i][j-1]

Minimum path sum in matrix

* 1. <https://leetcode.com/problems/minimum-path-sum/>
  2. dp[i][j] = a[i][j] + min(dp[i-1][j], dp[i][j-1])

### Min stair cost

<https://leetcode.com/problems/min-cost-climbing-stairs/>

Each stair has given cost find min cost to reach at top

Input: cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]

Output: 6

dp[i] = cost[i] + min(dp[i-1],dp[i-2])

### Is subsequence

<https://leetcode.com/problems/is-subsequence/>

* 1. Using dp
     1. dp[i][j] = dp[i-1][j-1] if s[i] = t[j]

= false else

If one cell is true make entire row true

| s\t | h | a | x | b | c |
| --- | --- | --- | --- | --- | --- |
| a | F | T | T | T | T |
| b | F | F | F | T | T |
| c | F | F | F | F | T |

* 1. Using two pointer
     1. Scan s by i and inc j when s[i] == t[j]

if j == t.length() found string

### Robbing house

* 1. Each house has some value indicated by array element adjacent house can not be robbed find max amount that can be robbed
  2. dp[ i ] = max( dp[ i - 1 ], value[ i ] + dp[ i - 2 ] )

Don’t rob current house Rob current house

* 1. Or using inc exc property

* 1. 2, 1, 1, 2 => (2 + 2 = 4)

1, 2, 3 ,4 => (2 + 4 = 6)

* 1. Circular Robbing house
     1. <https://leetcode.com/problems/house-robber-ii/submissions/>
     2. return max(robhouse(nums, 0, nums.size()-2),robhouse(nums, 1,nums.size()-1));

### Longest valid Parentheses

<https://leetcode.com/problems/longest-valid-parentheses/solution/>

| ( | ) | ( | ( | ) | ) |
| --- | --- | --- | --- | --- | --- |
| 0 | 2 | 0 | 0 | 2 | 6 |

If s[ i ] == ‘ ( ‘

Dp[ i ] = 0

Else

\\s[ i ] = = ‘ ) ‘

If s[ i - 1 ] == ‘ ( ‘

S[ i ] = s[ i - 2 ] + 2

Else s[ i - 1 ] == ‘ ) ‘

if( s[ i - dp[ i - 1 ] - 1 ] == ' ( ')

dp[i] = dp[i-1] + 2 + dp[i-dp[i-1]-2];

else

dp[i] = 0;

### Total no of unique binary search tree

[https://leetcode.com/problems/unique-binary-search-trees/discuss/31666/DP-Solution-in-6-lines-with-explanation.-F(i-n)-G(i-1)-\*-G(n-i)](https://leetcode.com/problems/unique-binary-search-trees/discuss/31666/DP-Solution-in-6-lines-with-explanation.-F(i-n)-G(i-1)-*-G(n-i))

Series : 1 2 5 14 42 132

G(n) = F(1, n) + F(2, n) + ... + F(n, n). // possible roots

G(n) = G(0) \* G(n-1) + G(1) \* G(n-2) + … + G(n-1) \* G(0)

G(n) = E( G(x-1) \* G(n-x) ) where 1<= x <= n

Eg : 1 2 3 4 5 6 if 3 i root left side 2 right side 3 element 1,2,3,4,5,6

G[0] = G[1] = 1;

for(int i=2; i<=n; ++i) {

for(int j=1; j<=i; ++j) {

G[i] += G[j-1] \* G[i-j];

}

}

return G[n];

### No of ones in binary representation of number

* 1. f[i] = f[i >> 1] + (i & 1);
  2. Explaination.
  3. Take number X for example, 10011001.
  4. Divide it in 2 parts:
  5. <1>the last digit ( 1 or 0, which is " i&1 ", equivalent to " i%2 " )
  6. <2>the other digits ( the number of 1, which is " f[i >> 1] ", equivalent to " f[i/2] " )

### Edit distance problem

* 1. Min no of operation to convert word1 to word2
  2. dp[i][j] = dp[i-1][j-1] if word1[i-1] = word2[j-1]
  3. min( dp[i-1][j], dp[i][j-1],dp[i-1][j-1] ) + 1 else

<https://leetcode.com/problems/edit-distance/discuss/25846/C%2B%2B-O(n)-space-DP>

In above link 3 approaches to solve dp using O(n2) O(2n) and O(n)

### 0/1 Knapsack Problem

* 1. <https://practice.geeksforgeeks.org/problems/0-1-knapsack-problem/0>
  2. Here limited supply of item is given so

if (i==0 || w==0)

K[i][w] = 0;

else if (wt[i-1] <= w)

K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);

else

K[i][w] = K[i-1][w];

### Coin change problem

* 1. <https://practice.geeksforgeeks.org/problems/coin-change/0>

|  | 0 | 1 | 2 | 3 | 4 |
| --- | --- | --- | --- | --- | --- |
| 0 0 | 1 | 0 | 0 | 0 | 0 |
| 1 1 | 1 | 1 | 1 | 1 | 1 |
| 2 2 | 1 | 1 | 2 | 2 | 3 |
| 3 3 | 1 | 1 | 2 | 3 | 4 |

if(a[i-1] > j)

dp[i][j] = dp[i-1][j];

else

dp[i][j] = dp[i-1][j] + dp[i][j-a[i-1]];

* 1. <https://leetcode.com/problems/coin-change/submissions/>

|  | 0 | 1 | 2 | 3 | 4 |
| --- | --- | --- | --- | --- | --- |
| 0 0 | 0 | INF | INF | INF | INF |
| 1 1 | 0 | 1 | 2 | 3 | 4 |
| 2 2 | 0 | 1 | 1 | 2 | 2 |
| 3 3 | 0 | 1 | 2 | 1 | 2 |

if(a[i-1] > j)

dp[i][j] = dp[i-1][j];

else

dp[i][j] = min( dp[i-1][j] , dp[i][j-a[i-1]] + 1); take INT = 99999 to avoid overflow

* 1. <https://leetcode.com/problems/combination-sum-iv/>

Find all combinations of numbers having sum = target

Use backtracking approach and covert it to top down and bottom up approach

### Longest Inc subsequence nlogn

#### LCS

#### <https://leetcode.com/problems/longest-increasing-subsequence/discuss/74848/9-lines-C%2B%2B-code-with-O(NlogN)-complexity>

| vector<int> res; for(int i=0; i<nums.size(); i++) {  int ind = lower\_bound(res.begin(), res.end(), nums[i]) - res.begin();  if(ind == res.size()) res.push\_back(nums[i]);  else res[ind] = nums[i]; } return res.size(); |
| --- |
|  |

#### Stack Book

#### <https://cses.fi/problemset/task/1073/>

| for(int i=0;i<n;i++){  int index = upper\_bound(dp.begin(), dp.end(), a[i]) - dp.begin();  if(index == dp.size())  dp.push\_back(a[i]);  else  dp[index] = a[i];  }  cout << dp.size() <<en |
| --- |

#### 

#### Maximum nested doll possible

* 1. <https://leetcode.com/problems/russian-doll-envelopes/>
  2. Sort by width asc and when equal des by height
  3. Apply lis on height
  4. Reason : (6, 1), (6,2), (6, 3) here if we don't sort by height will get ans = 3 so.

#### d) Maximum no of longest Increasing Subsequence

* 1. <https://leetcode.com/problems/number-of-longest-increasing-subsequence/>

| if (nums[i] > nums[j]) {  if (len[j]+1 > len[i]) {  len[i] = len[j]+1;  cnt[i] = cnt[j];  }  else if (len[j]+1 == len[i])   cnt[i] += cnt[j]; |
| --- |

### 

### Palindrome

* 1. Longest Palindromic subsequence
     1. <https://leetcode.com/problems/longest-palindromic-subsequence/submissions/>

if(s[i] == s[j])

dp[i][j] = dp[i+1][j-1] + 2;

else

dp[i][j] = max(dp[i][j-1],dp[i+1][j]);

* + - 1. A X B B A B

A 1 1 1 2 4 4

B 0 1 1 2 2 3

X 0 0 1 2 2 3

B 0 0 0 1 1 3

A 0 0 0 0 1 1

B 0 0 0 0 0 1

* 1. Longest Palindromic Substring :
     1. For each character traverse both side and find odd len palindrome
     2. Similarly traverse both side and find even len palindrome
     3. <https://leetcode.com/problems/longest-palindromic-substring/>
     4. If s[i] == s[j]

dp[i][j] =dp[i+1][j-1]

Else

dp[i][j] = false

A B A A B A

A 1 0 1 0 0 1

B 1 0 0 1 0

A 1 1 0 0

A 1 0 1

B 1 0

A 1

* 1. Count palindromes
     1. <https://practice.geeksforgeeks.org/problems/count-palindrome-sub-strings-of-a-string/0>

s = '@' + s + '#';

int n = s.size();

for(int mid = 1; mid <= n-1; mid++){

i = mid-1, j = mid+1;

while(s[i] == s[j])

count++, i--, j++;

i = mid-1, j = mid;

while(s[i] == s[j])

count++, i--, j++;

}

cout << count << endl;

* 1. Min insertion to make string palindrome
     1. <https://www.geeksforgeeks.org/minimum-insertions-to-form-a-palindrome-dp-28/>
     2. table[l][h] = (str[l] == str[h]) ? table[l + 1][h - 1] : (min(table[l][h - 1], table[l + 1][h]) + 1);
     3. Find longest palindromic subsequence. Ans = Total length-length of LPS
  2. .<https://www.geeksforgeeks.org/minimum-number-deletions-make-string-palindrome/>
     1. table[l][h] = (str[l] == str[h]) ? table[l + 1][h - 1] : (min(table[l][h - 1], table[l + 1][h]) + 1);
     2. Find longest palindromic subsequence. Ans = Total length-length of LPS
  3. <https://www.geeksforgeeks.org/find-if-string-is-k-palindrome-or-not/>  
      Find longest palindromic subsequence. Check length-length of LPS <= K
  4. Count Pal substring in range i to j
     1. <https://www.geeksforgeeks.org/count-of-palindromic-substrings-in-an-index-range/>
     2. https://www.geeksforgeeks.org/count-palindrome-sub-strings-string/

### Longest Repeating Subsequence

* 1. <https://www.geeksforgeeks.org/longest-repeating-subsequence/>
  2. Using dp

if (str[i-1] == str[j-1] && i != j)

dp[i][j] = 1 + dp[i-1][j-1];

else

dp[i][j] = max(dp[i][j-1], dp[i-1][j]);

a b b a

0 0 0 0 0

a 0 0 1 1 1

b 0 1 1 1 1

c 0 1 1 1 2

d 0 1 1 2 2

* 1. Without using dp O(n) for check existence of repeating subsequence
     1. <https://www.interviewbit.com/problems/repeating-subsequence/>
     2. If any char occurs more than 2 times return true

Now make a string where each char repeats 2 times

If this string is palindrome than then return 0

Else return 1

### Number of Distinct subsequences t in s

* 1. <https://leetcode.com/problems/distinct-subsequences/>
  2. if(t[i-1] == s[j-1])

dp[i][j] = dp[i-1][j-1] + dp[i][j-1];

else

dp[i][j] = dp[i][j-1];

B A B G B A G

1 1 1 1 1 1 1 1

B 0 1 1 2 2 3 3 3

A 0 0 1 1 1 1 4 4

G 0 0 0 0 1 1 1 5

Interviewbit

### Longest inc dec sequence

* 1. <https://www.interviewbit.com/problems/length-of-longest-subsequence/>
     1. inc[i] stores Longest Increasing subsequence ending with A[i]
     2. dec[i] stores Longest Decreasing subsequence ending with A[i]
     3. Now we need to find the maximum value of (inc[i] + dec[i] - 1)

### Largest Rectangle with 1’s , swapping of columns allowed

* 1. <https://www.interviewbit.com/problems/largest-area-of-rectangle-with-permutations/>

### Decode Ways

* 1. <https://leetcode.com/problems/decode-ways/submissions/>

if(x < 27)

dp[i] = dp[i+1] + dp[i+2];

else

dp[i] = dp[i+1];

### Interleaving Strings

* 1. <https://www.interviewbit.com/problems/interleaving-strings/>

if(i >= 0 && A[i] == C[k])

a = isInterleaving(A,B,C,i-1,j,k-1);

if(j >= 0 && B[j] == C[k])

b = isInterleaving(A,B,C,i,j-1,k-1);

Return a || b

### Wild card character

* 1. Use map <pair,bool> of dp to avoid memory limit errror
  2. <https://www.interviewbit.com/problems/regular-expression-match/> (both top down and bottom up)

if(text[i-1] == pat[j-1] || pat[j-1] == '?')

dp[i][j] = dp[i-1][j-1];

else if(pat[j-1] == '\*')

dp[i][j] = dp[i-1][j] || dp[i][j-1];

else

dp[i][j] = false;

### Regular Expression

* 1. <https://www.interviewbit.com/problems/regular-expression-ii/> ( both top down and bottom up)
  2. Top down approach

if(text[i-1] == pat[j-1] || pat[j-1] == '.')

dp[i][j] = dp[i-1][j-1];

else if(pat[j-1] == '\*')

{

if(pat[j-2] == text[i-1] || pat[j-2] == '.')

dp[i][j] = dp[i][j-2] || dp[i-1][j-2] || dp[i-1][j];

Else

dp[i][j] = dp[i][j-2];

}

### Scramble String

* 1. <https://www.interviewbit.com/problems/scramble-string/>

if(Scramble(A.substr(0,i),B.substr(0,i)) && Scramble(A.substr(i),B.substr(i)) ||

Scramble(A.substr(0,i),B.substr(n-i)) && Scramble(A.substr(i), B.substr(0,n-i)))

return mp[A+B] = true;

### Longest Arithmetic Sequence

* 1. <https://leetcode.com/problems/longest-arithmetic-sequence/submissions/>
  2. A,B,C are in AP then A = 2 B - C
     1. dp[i][j] no of items in arithmetic series ends with a[i], a[j]
     2. int target = 2\*a[i] - a[j];
     3. if(index.find(target) == index.end())
     4. dp[i][j] = 2;
     5. else
     6. dp[i][j] = dp[index[target]][i] + 1;

### Min palindrome partitioning

* 1. <https://www.interviewbit.com/problems/palindrome-partitioning-ii/> (bottam up) O(n^3)
     1. if(ispal[i][j])

dp[i][j] = 0;

else{

for(int k=i;k<j;k++){

dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j] +1);

}

}

* 1. abcb ⇒ a | bcb , ab | cb, abc | b
  2. <https://leetcode.com/submissions/detail/314502291/> (top down)
     1. for(int i=start;i<end;i++){
     2. ans = min(ans, 1+findMincut(A,start,i) + findMincut(A,i+1,end));
     3. }
  3. <https://leetcode.com/problems/palindrome-partitioning-ii/submissions/> O(n^2)

### Word Break Problem

* 1. <https://leetcode.com/submissions/detail/314851563/> (top down)
     1. for(int i=1;i<=maxlen;i++)

if(dict.find(s.substr(start,i)) != dict.end() && canBreak(s,start+i))

return true;

* 1. <https://leetcode.com/submissions/detail/314912714/> (bottom up) O(n^3)

if(dict.find(s.substr(i,len))!=dict.end())

dp[i][j] = true;

else{

for(int k=i;k<j;k++){

dp[i][j] = dp[i][k] && dp[k+1][j];

if(dp[i][j]) break;

}

}

* 1. O(n2) <https://leetcode.com/problems/word-break/discuss/43790/Java-implementation-using-DP-in-two-ways>

### Word break 2

* 1. <https://www.interviewbit.com/problems/word-break-ii/>
  2. Dp[i] = string of vector of solution from 0 to i
  3. For i in 0 to n-1
     1. For j in 0 to i

Stg = stg(j , i)

If ( dict contains stg and dp[j-1] > 0)

For x in dp[j-1]

Dp[i] += stg + ‘ ‘ + x

### Dungeon Princesses

* 1. <https://www.interviewbit.com/problems/dungeon-princess/>

### Buy and Sell

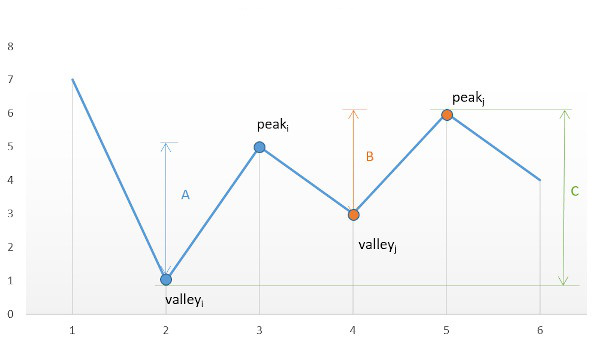
* 1. Any no of times

<https://www.interviewbit.com/problems/best-time-to-buy-and-sell-stocks-ii/>

Option 1: Buy stock at valley[i] then sell at peak[i] makes profit A (peak[i] - valley[i]). Then buy stock at valley [j] and sell at peak[j] makes profit B (peak[j] - valley[j]). So the total profit of this trade option is A + B.

Option 2: Skip the intermediate trades, i.e,, we buy stock at valley[i] then sell at peak[j]. In this case, the total profit will be C (peak[j]-valley[i]).

Based on the graph shown below, A + B > C (if not, peak[i] and valley[j] won't exist). So in order to maximize the profit, we can buy stock at valleys and then sell stock at peaks.

* 1. 
  2. At Most two
     1. <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/>
  3. Exact k transactions
     1. <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/>
     2. Either dont do a transaction on jth day or do do transaction with maximum profit
     3. If k >= n/2 than maxprofit (a)

Else

dp[i][j] = max(dp[i][j-1]); // i transactions j days

max(prices[j] - prices[m] + dp[i-1][m]); 0<= m < j

* + 1. Optimization using maxvalue refer link
  1. <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/discuss/108870/Most-consistent-ways-of-dealing-with-the-series-of-stock-problems>
     1. <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown>
     2. <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee>

### Maximum Product subarray

* 1. [www.interviewbit.com/problems/max-product-subarray/](http://www.interviewbit.com/problems/max-product-subarray/)
  2. Dpmax[i] = max subarray ending at index i
  3. Dpmin[i] = min subarray ending at index i
  4. Dpmax = max(A[i],dpmax\*A[i],dpmin\*A[i])
  5. Dpmax = min(A[i],dpmax\*A[i],dpmin\*A[i])

### Smallest k no whose prime factors are only Primes p1 p2 and p3

* 1. <https://www.interviewbit.com/problems/smallest-sequence-with-given-primes/>

| dp[0] = 1;  for(int i=1;i<D+1;i++){  int mine = min(dp[p1]\*A, min(dp[p2]\*B, dp[p3]\*C));  if(mine == dp[p1]\*A) p1++;  if(mine == dp[p2]\*B) p2++;  if(mine == dp[p3]\*C) p3++;  dp[i] = mine;  } |
| --- |
|  |
|  |
|  |

### Flip min no so resultant array is min no negative

* 1. <https://www.interviewbit.com/problems/flip-array/>
  2. S1 + s2 = sum

S1 - s2 = 0

So s2 = sum/2

Using knapsack find no of items with max target possible