Generate Parenthesis (Backtracking)

Given a number n, return all well formed parenthesis.

Example : n=3

output : ["((()))", "(()())", "(())()", "()(())", "()()()"]

Approach :

Generate all combinations using recursion include only those solution in which opening and closing braces are balanced.

```
recursive ( n, string S, open, close)
  if (close == n)
      include S and return.

  if (open > close)
      append ')';
      recursive ( n, S, open, close+1)
      backtrack.
  if (open < n)
      append '(';
      recursive ( n, S, open+1, close)
      backtrack.
```

Search in Rotated Sorted array

Sorted array is rotated at pivot i. Search an element in this array.

Approach

1- Find pivot index using binary search.

Pivot will be, where

$$arr[i-1] > arr[i].$$

Now binary search in both halfs.

Activity Selection problem

Greedy

Given start time and end time of meetings.

output :- maximum ~~number~~ number of meetings in a single room.

Approach

Sort w.r.t end time.

Select lowest end time process ~~in~~ possible in line.

Page No.
Date

Job Scheduling GFG (Greedy)

Given N jobs with deadline and profit.
Each job takes 1 unit of time.
Find maximum profit

Approach:- Jo kaam jitna jaruri hai use sabse pehle nipta lo iss tarah se ki wo bhi zaroor jaye magar baakiyo ki jagah bhi na khaye.

Hint

Each job takes 1 unit of time and start time jaisa kuch nahi hai.

Make n slots!

Minimum Number of Platforms required for a railway station

Given n trains, arrival and dept.
Find minimum number of platforms required.

Method 1:- Time O(n) space O(2400)
Check occupancy at every instant of time by creating an occupancy array.

Q:- Maximum Profit job Scheduling LC Hard

~~Same as~~ Same as "projects" DP CSES

Q:- Maximum trains for which stoppage can be provided.

Given n platforms, m trains. Each train has arr. time, dept. time and expected platform. Count maximum count of trains that can be provided stoppage. Trains without stoppage will simply pass through.

Sol^{n}:- Consider each platform as a separate activity selection problem.

Method-2 :- Time complexity $O(n \log n)$, Space $O(1)$

(Greedy)

→ sort both arrival and dept times.

→ Now using two pointer approach if(arr[i] <= dep[j]){ platforms++; i++; }

→ compute max answer at each step. else if(arr[i] > dep[j]){ platforms--; j++; }

1st Choice

Page No.

Maximize sum of k negations

Given an array of n and a number k operations. Maximize sum of array in exactly k operations. In each operation we can replace arr[i] by -arr[i].

Solution

Count number of negatives.

if(neg >= k).

{ fork(arr)

flip ~~minimum~~ k elements first

take sum

return;

}

(neg < k) // all elements can be made +ve

Take sum of all absolutes.

k -= neg;

if(k%2 == 0)

return sum

else

{ return sum - 2*(minimum absolute element)

Maximum Product Subarray (RatLo!) — Submitted on GFG

Time O(N), space: O(1).

Variation of Kadane.

Given an array, return maximum possible subarray sum.

Approach

ans ← inf, mx ← 1 (to store +ve max), mn ← 1 (to store -ve max).

```
loop(0, n-1)
  if(arr[i] == 0){
    ans = max(ans, 0)
    mx = 1   // resetting mx and mn because
    mn = 1   // subarray product should be contiguous.
  }
  else{ if(arr[i] > 0){
    mx = max(mx*arr[i], arr[i])
    mn = min(mn*arr[i], arr[i]) }
  if(arr[i] < 0){ temp ← mx.
    mx = max(mn*arr[i], arr[i])
    mn = min(temp*arr[i], arr[i]) }
  ans = max(ans, mx); }
```

Imp case (beside if(arr[i] < 0))

Minimum Lights to Activate - IB.

Given
Array size N. of 0 and 1. B → power of bulb.
0 → bulb is faulty.
1 → bulb is ok!.

Corridor is from [0 to N-1].
Find minimum ~~num~~ number of bulbs to ~~turn~~ light up whole corridor. return -1 if not possible.

Example: Arr = [0, 0, 1, 1, 1, 0, 0, 1].
B = 3
Ans = 2.

My Approach: O(n²), space O(n). (TLE on IB.)

First turn all the non-faulty bulbs on.
If there's any point un-lit return -1.

Turn unimportant bulbs off by checking its field of vision is ≥ 2.
if every point in

Greedy Approach $O(n^2)$, $O(n)$ space

Iterate from left to right. If position is unlit, assign rightmost bulb that can cover this spot. Before as well as after that spot.

★ Minimize cash flow among friends ★

Given number of friends and transactions between them minimize cash flow and return the minimized transactions.

Approach: Greedy

Settle all amounts of one person and recurs for remaining $n-1$ persons.
Apply heuristic to picking the first person.

⇒ Find the person who is about to receive maximum amount of money in total.
2) Find the person who is about to pay maximum amount of money in total.

Between these two persons.

if(amount to pay < amount to recive)
    amount to pay will settle.
else if (amount to recive <= amount to pay)
    amount to recive will settle.

Repeat this until all amounts are settled.

Minimum Cost to cut a board into squares.

Q:- A board of length m and width n. Break this square into m*n squares such that cost of breaking is minimum.

Cost of each m-1 and n-1 cut lines is given as two separate arrays.

Cost of each cut is calculated as:- cost of line * number of segment it will cut.

Example :-

Cost = 42.

Approach :-

When we make a horizontal cut, number of vertical segments will increase by 1. And similarly for 1 vertical cut, horizontal cut gets incremented by 1.

A vertical cut passes through "all" horizontal segments. Also, horizontal cut passes through "all" vertical segments.

Cost can be denoted by $C = a_1w_1 + a_2w_2 + \dots + a_kw_k$.

coefficients for all vertical and horizontal cuts are fixed to be 1, 2, 3 ..., n, increasing order. We just have to order these cuts such that coefficient * cost comes to be minimum.

Hence, we sort both the arrays in decending order and apply cut according to maximum cost first approach. (use two pointer)

1st Choice

Page No.
Date 10/12/21

Maximum sum of absolute difference of permutation of array:-

Given an array, rearrange it such that sum of absolute differences between adjacent elements is maximum.

Its circular, abs[last - first] will also get added.

Approach

Rearrange it as:-

~~[highest, lowest, 2nd highest, 2nd lowest ...]~~

~~[lowest, highest, 2nd lowest, 2nd highest ...]~~

Create two patterns :-

[highest, lowest, 2nd highest, 2nd lowest ...]

[lowest, highest, 2nd lowest, 2nd highest ...]

Example:- N=4

4 1 3 2

3 2 1 = 6

2 4 1 3 = 7

4 3 2

Example:- N=5

5 1 4 2 3 = 10

4 3 2 1

3 5 1 4 2 = 11

5 4 3 2

★ Maximum sum of differences of adjacent elements.

Given a number n.

Find maximum sum of a permutations of $\{1, 2, \ldots n\}$. Maximum sum will be sum of absolute difference of adjacent elements in array.

★ (Not Circular) ★

For non-circular array,

Example :-

1, 5, 2, 4, 3

Arranged in pattern from prev que.

Sum in this case = 4+3+2+1 = 10.

★ But maximum sum is obtained if we move 3 at the beginning i.e. 3, 1, 5, 2, 4.

Sum = 2+4+3+2 = 11.

Now, derive formula for numbers 1 to n, for this pattern

$$\text{Sum} = n(n-1)/2 - 1 + n/2.$$

1st Choice

Page No.
Date

Minimum sum absolute diff b/w two array

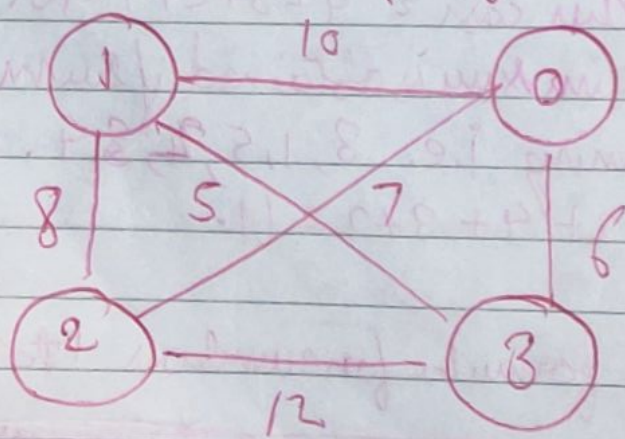
Given two arrays a and b of equal length n. pair elements of a with b such that pairwise absolute difference sum is minimum.

Sol^n :- Sort both the arrays and take sum of all $(a[i] - b[i])_{abs}$.

K-Centres Problem

Given n cities and distance b/w every pair of city. select k cities to place warehouses. such that maximum distance of city to warehouse is minimized.

Ex:



K=2

Ans: Two ATMs should be placed in city 2 and 3.

The 2-Approximate Greedy Algorithm.

Condition :- The distances between cities should follow triangular inequality.

Algorithm

- Choose first centre arbitrarily.
- Choose remaining k-1 centres using the following criteria :-

  Let $c_1, c_2, c_3 \dots c_i$ be the already chosen centres.

  Choose (i+1)th center by picking the farthest from already selected centres, i.e. point p which has the following value as maximum

$$\min\left(dist(p, c_1), dist(p, c_2) \dots dist(p, c_i)\right)$$

Note that the greedy algorithm doesn't give best solution for k=2 as this is just an approximate algorithm with bound as twice of optimal.

The problem is originally NP-hard.

1st Choice

Page No.
Date

Reorganize String - Leetcode

Pnarom

Q:- Rearrange characters in a string such that no two adjacent are same.

Given a string with repeated characters, rearrange characters such that no two adjacent are same.

Input: aaabc
Output: abaca

Input: aa
Output: ""

Approach I O(N(logN))

1. Build a pq that stores characters and frequencies
2. Create a temporary that will be used as the previously used element. {Initialize:- freq -1, "#"}
3. While pq is not empty
   - Pop an element and add it to the result
   - Decrease freq of element by 1.
   - Push the previous element back to pq. if freq > 0.
   - Make the current element as prev element for next iteration.
4. If length of resultant is not n, then it's not possible.

Approach-2 O(N)

Fill all even positions of the string with the highest freq character. Fill all the even positions, then fill odd positions ~~the similar way~~ after that normally sequencially. normally.

SCRAMBLED STRING

→ Aditya Verma se Rat Lo!

Base condt. if (a==b)
return true
if(a.length<=1)
return false;

Approach MCM

loop(i=1 to <= n-1)

if(solve(a.substr(0,i), b.substr(0,i)) && solve(a.(substr(i, n-i), b.substr(i,n-i)))

return true;

elseif (solve(a.substr(0,i), b.substr(n-i,i))
&&
solve(a.substr(i, n-i), b.substr(0, n-i)))

return true.

}

return false

1st Choice

Page No.
Date

Egg Dropping Problem

⇒ Adikyaverma se rat Lo!

Palindrome Partitioning - I (Leetcode)

Given a string s, Return all palindrome partitioning of s.

Input: s = "aab".

Output: [["a","a","b"], ["aa","b"]]

Use backtracking, ⇒ Partitioning when we get palindrome on left, recurring for right part. And then backtrack!

```
void solve(string &s, int l, vector<string> &current){
    if(l >= s.length){
        result.pb(current);
        return
    }
    loop(i=l; i<s.length; i++){
        if(isPalindrome(s, l, i)){
            current.pb(s.substr(l, i-l+1));
            solve(s, i+1, current);
            current.pop_back();
        }
    }
}
```

Parsing a boolean expression (Leetcode)

Evaluate expression:

- "t" evaluating to true
- "f" " " false
- "!(expr)" " NOT(expr)
- "&(expr1, expr2, expr3...)" → AND of all expr.
- "|(expr1, expr2, expr3...)" → OR of all expr.

example: "|(&(t,f,t), !(t))"

Approach

1. Use a stack to store chars except ',' and ')'

2. If we find a ')', keep popping out the chars from stack till find a '('; add the popped out into a set.

3. Pop out the operator after '(' pop. Push result into the stack according to operator.

4. Repeat till the end, and the remaining is the result.

Palindrome Partitioning - II

Minimum cuts needed for a palindrome partitioning of S.

Optimisation in Aditya Verma's solution.

In MCM approach, partition only if left partition is a palindrome.

No need to check all combinations of partitions.

If left part is palindrome, recur for right part.

Time complexity :- $O(n^2)$, space complexity $O(n)$.

```
loop(i=l; i<r; i++)
    if(isPalindrome(l, i))
        mn = min(mn, 1 + Solve(i+1, r));
}
```

Course Schedule - III

Given courses[i] = [duration, lastday_i].
course should be taken continuously for duration_i days and must be finished before lastday_i.

Start of 1 on 1st day and cannot take two courses simultaneously.

Find Maximum number of courses you can take.

Example: [[100,200], [200,1300], [1000,1250], [2000,3200]]

Output: 3.

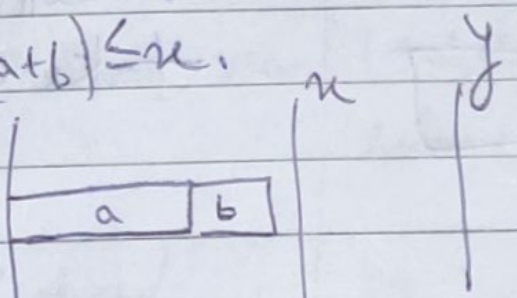
Approach-1

Greedy:- Sort courses w.r.t. end day.

* Reasoning:-

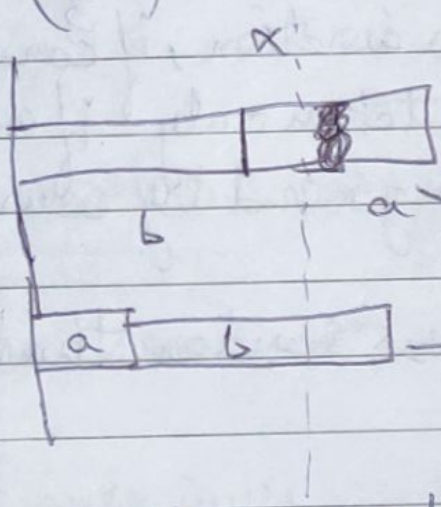
Consider two courses (a, x) & (b, y). Let's assume $y > x$ and consider cases.

1. $(a+b) \leq x$.



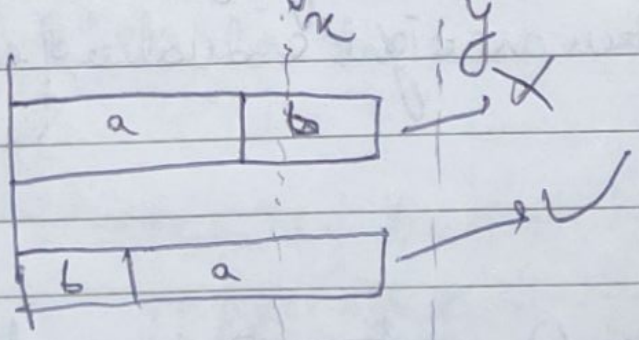
Both can be taken irrespective of order

2. $(a+b) > x$ && $(a+b) \leq y$ && $b > a$



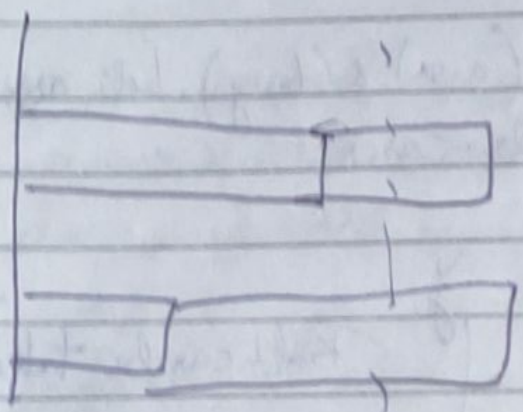
courses can be taken by only taking a before b.

3. $x < (a+b) \leq y$, $a > b$



courses can be taken by only taking b before a.

Q. (a+b) > y.



irrespective of order we can only take one.

So, this can be concluded, that if the course with a smaller duration, if can be taken, can surely be taken only if we consider it before larger end day course

Hence, we sort courses based on their end day.

Now we can formulate the problem as knapsack. duration is taken as weight and value of each process is 1.

In main:

sort (arr.begin(), arr.end());

```
int solve( arr, int i, int time)
  if(i>= arr.size())
    return 0;
  if( time+ arr[i].duration <= arr[i].endday)
  { return max(1+ solve(arr, i+1, time+ arr[i].duration),
                solve(arr, i+1, time));
  } else{
      return solve(arr, i+1, time);
  }
```

Approach-2 - Using priority-queue.

- Sort arr w.r.t. end day.

Priority queue pq; → Store durations in it.

```
int time=0;
for( arr)
{
   time += duration[i]
   pq.push(duration[i])
   if( time > c[i]) time-= pq.top(); pq.pop();
}
return pq.size();
```

The crux of the solution :

★ Once you make sure a course fits in, you can remove it any time later and the other courses you have added after would still fit. So it is always safe to remove any course in the past.

Time complexity $O(n \log w)$, Space $O(n)$.

Set Zero (IB)

Given a matrix of 1s and 0s. If an element is 0, set its entire row and entire column to 0.

Constraint Time ($O(n^2)$) Space ($O(1)$)

~~★~~ Approach 1

- First check rowwise, only rows.
  in each iteration for row, if found 0, set bool setzero = true.

if (setZero == true)
    set all the 1s in this row to 2.

- Now check in similar fashion columnwise.
  in each iteration for col, if found 0, set
  bool setZero = true
  if (setZero == true)
  { set all the 1's in this col to 2.

- Finally, traverse matrix and set all 2s to 0.

Approach 2

Use first row and first column to store info about setting whole row or whole column to 0.
and use two booleans for these two ~~rows~~ separately.

Check first row and first col.
Traverse matrix and mark at first row and first col.

Traverse matrix and replace according to A[i][0] A[0][i]
Check bools for first row and first col and
replace.